Graphical user interfaces for Python programs

Python and Tkinter Programming

Python与Tkinter编程



[美] John E. Grayson 著

陈文志 高垒 缪瑾 崔广仁 蒋涛 译

国防工业出版社

MANNING

http://www.ndip.com.cn

经典译丛

PROGRAMMING / PYTHON

www.manning.com

责任编辑：欧阳光

Python 与 Tkinter 编程

John E. Grayson

Tkinter（原意为tea-kay-inter），捆绑上Python编程语言的图形用户界面包.是一个清晰的到Tcl/Tk图形工具包的面向对象界面。它使你能够在Unix Macintosh和Windows系统下快速编写GUI代码，运行起来有本地系统的外观和感觉。

《Python与Tkinter编程》仔细引导你进入编程的摩天大楼，它使Tkinter的功能可以为所有知道Python基本功能的人所用。我给出能够实际运行的真实代码－不是玩具般的例子，参考部分的Tkinter文档将对你很有帮助，用起来也很方便。

你要加入图形到你的应用程序中要做的一切都在这里，从简单介绍，到“努力”工作的例子，到全面的键盘参考资料。

本书主要内容：

- 一个大师对创建迷人界面的建议
- 图形、点、橡皮线、树、机器，线程
- 画界面、仿真面板、控制及更多
- 如何调试Python及性能调整
- 所有类、方法和选项的参考手册

"Packed with well-explained examples that teach good Python habits."

—Guido van Rossum

Inventor of Python

"This book is a winner.I'll be recommending it to a lot of readers."

—Cameron Laird

Columnist, *Sun World*

"Regular Expressions"

"Some very cool material is covered here,which takes one well beyond the 'yet another collection of the same old buttons and sliders' stuff."

—Garry Hodgson

Technology Consultant

"Using chapters from this book,interns here at General Dynamics were producing knockout GUIs in a week..."

—Gordon Smith

General Dynamics

John Grayson 能够背诵 *Monty Python and Holy Grail* 的所有场面，他是分子生物学博士，自1993年来就把Python推向应用。他使用Python为州政府、航空公司和远程通讯公司编写了大规模的应用程序及创新的图形界面。





ISBN 7-118-02781-2

9 787118 027815 >

ISBN 7-118-02781-2/TP·700
定价：66.00 元

编程经典译丛

Python 与 Tkinter 编程

Python and Tkinter Programming

John E. Grayson 著

陈文志 高垒 缪瑾 崔广仁 蒋涛 译

国防工业出版社

·北京·

著作权合同登记号　图字:军-2001-011 号

图书在版编目（CIP）数据

Python 与 Tkinter 编程／(美)格雷森(Grayson., J.E.)著;
陈文志等译. —北京：国防工业出版社,2002.9
（编程经典译丛）
书名原文：Python and Tkinter Programming
ISBN 7-118-02781-2

Ⅰ.P…　Ⅱ.①格…②陈…　Ⅲ.①PYTHON 语言—
程序设计②图形程序包，Tkinter　Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2002）第 004507 号

©2000 by Manning Publications Co. All rights reserved.
Published by arrangement with Manning Publications Co.,USA.
本书中文简体版由 Manning Publications Co.授予国防工业出版社（独家）出版。版权所有，侵权必究。

国防工业出版社出版发行
（北京市海淀区紫竹院南路 23 号）
（邮政编码　100044）
北京奥隆印刷厂 印刷
新华书店经售
*
开本 787×1092　1/16　印张 35 3/4　821 千字
2002 年 9 月第 1 版　2002 年 9 月北京第 1 次印刷
印数：1-3000 册　定价：66.00 元

（本书如有印装错误，我社负责调换）

前　言

我是在 1993 年加入位于 Rhode Island 的一个小公司时结识 Python 的。他们的主要产品是为 X/Motif 产生 C、C++、Ada 和 Python 编码的 GUI-builder。我的任务是为 X/Motif 和 Python 扩展面向对象界面。在这之前，我一直怀疑解释性语言的用途，因此，开始这工作时，我兴趣不大。两天之后，我上瘾了。使用 C 语言编码很复杂的界面，能够很容易地完成！不久，我更乐于选择使用 Python 而不是 C 来开发界面。

离开 Rhode Island 的公司之后，我开始使用 Tkinter 开发软件，这成了有名的 Python GUI。我说服了一家公司（我在那里做点合同工）来通过 Python 编码开发一个濒临超时和超出预算的大项目，项目很成功。四年以后，该公司有了很多 Python 程序员，他们的项目中相当的代码是通过 Tkinter 和 Python 来完成的。

正是这些经历，使得我编写这本书。早些时候，Python 可用文档资料很少。Tkinter Life Preserver 是第一本帮助人们把基本信息凑到一起的书。1997 年，Fredrik Lundh 在网上发布了不少优秀的 Python 类文档，这些为近两年 Python 编程起到极大的作用。我所见到的几个问题之一是：虽然有很多例程(伴随 Python 的发布不少)，但他们大部分内容太简洁，不能代表 Python 程序完整框架。当然，要把代码一个个连起来运转也不难，但是，当体系结构依赖于解释语言时，很容易得到一个在执行速度、美感、可维护性和可扩展性意义上差劲的产品。

因此，我写此书时别人问的第一个问题是："我如何编个 XXX？"我会给他一堆我写过的源代码。像大部分专业程序员一样，他们将详细研究。我相信，从一个完整程序着手是学习某一特定编程语言的好方法，能够达到特定的目标。

我练习"空手道"的时候常去新泽西 Shukokai 世界总部，和 Sensei Shigeru Kimura 练习。Sensei Kimura 经常告诉我们："我不能告诉你怎么做（某一特定技术）——你必须去'偷'。"我的学习 Tkinter 的办法也是一样的。如果社区里的人解决了某个问题，我们必须从他们那里"偷"。现在，我不是建议侵犯版权，我的意思是，我们从能得到的任何资料学习知识。我希望你把本书的例子作为程序的起点。在一些地方，我已经使用了其他程序员的代码或想法。如果是这样，我已经向原作者致谢了。如果你使用了这里的一些代码，我希望你能够对原作者致谢。毕竟，我们"偷"来的东西比我们自己的更有

价值，这是从 Sensei 那里得来的。

我对 Douglas A. Young 的 *The X Window System: Programming and Applications with Xt* 一书的风格印象很深刻。它有些老了，但是有一个完整的代码，其中许多可以做新应用程序的模板。《Python 与 Tkinter 编程》一书有些类似的东西。你会发现一些很长的、别的编程书籍中很少见到的例子。我希望许多这里的例子可以作为解决特定问题的程序员的模板或源码。

展示整个完整例子而不是一个程序段将使你们熟悉到我的编程风格。在《Python 与 Tkinter 编程》的审阅过程中，一些审阅者就提出一些别的风格。在可能的地方，我都采纳了他们的意见。因此，书中的例子包含了多人的风格。我希望读者在自己的编程实践中能有所提高。

我希望你发现《Python 与 Tkinter 编程》一书有用。如果它在你编程时能为你省下一两个小时，你用来阅读的几个小时也是值得的。

特 别 致 谢

《Python 与 Tkinter》一书是多人智慧的结晶。他们中的每人都为此书的出版尽智尽力。是他们的很多金玉良言，使此书更完美。

我要感谢技术审阅队伍：Fred L. Drake, Robin Friedrich, Alan Gauld, Bob Gibson, Lynn Grande Doug Hellmann, Gatrett G. Hodgson, Paul Kendrew, Andrew M.Kuchling, Cameron Laird Gregory A. Landrum, Ivan Van Ladingham, Burt Leavenworth, Ken McDonald, Frank McGeough, Robert Meegan, William Peloquin, Robert J. Roberts 和 Guido van Rossum。他们提供了详细的评论，使此书的内容、重点和准确性得到大大改进。

很多例子源代码是从其他人的代码中得来的。我要感谢这些代码编写者允许此书使用他们的代码。

Doug Hellman 在 Pmw 中写了个优秀的模块，GUIAppD.py，我改写后成为书中的 AppShell.py，书中还使用了很多例子。Doug 允许我使用其代码。如果你发现此代码可以在你的应用中使用的话，请感谢原作者。

Konrad Hinsen 写了 TkPlotCanvas.py，原本是在 Numpy，即数字操作最优的扩展模块下的。我将它改为不需要在 Numpy 下面，还增加了些图形功能。如果你发现能用到你工作中，感谢原作者。

第 8 章的树和点类是从 OpenChem 他们发布的公开码里得来的。你是否想看看他们组织将来发行的版本？因为本书中树控件的例子都来源于他们。

附录 B 把 Tk 手册作为 Tkinter 文档的起点。他们的版权所有者，加利福尼亚的 Regents 和 Sun Microsystems 允许从中导出些结果，向他们致谢。

我要感谢 General Dynamics 的 Gordon Smith，他在负责的许多项目中很有信心地使用 Python 和 Tkinter，看到他们的使用，是导致我写此书的重要原因。我曾把其中一些草稿章节给他们的员工和内部学员测试，来解决他们编程任务。

其次，我要感谢 Manning 出版社把我的想法变为此书的诸君。我和发行人 Marjan Bace 进行过长时间的谈话，从而使我完成了这一复杂任务，使得本书对读者有用。Ted Kennedy 协调了整个复审过程，提出了许多建设性意见。Mary Piergics 和技术编辑 Kristen Black，排字员 Dottie Marsico 一道负责此书的出版。Doug Hellman 是一位很好的技术编辑，纠正了不少打印结果中的代码错误。

最后，我要感谢我妻子 Allison，我的孩子 Nina、Chris、Jeff 和 Alana，他们懂得牺牲一个伴侣和父亲也意味着赢得一个作家。

致读者

《Python 与 Tkinter 编程》旨在为知道或熟悉 Python，而又想在书中加入图形界面的程序员（可能是使用 Manning 的 *Quick Python* 为指导）而写的。因为《Python 与 Tkinter 编程》提供了大量的功能例子和详细代码注解，有其他编程经验但没有 Python 经验的程序员会发现此书为他们立即解决实际问题很有帮助。

此书还可以作为 Tcl/Tk 脚本程序员把 Tcl/Tk 转变为 Python 和 Tkinter 的好指南。然而，并不打算从哲学的角度来讨论是否适合——我存有偏爱。

关于作者

John Grayson 是一个图形用户界面(GUI)方面的顾问。他为一家美国通讯大公司从事应用设计多年，致力于设计革新性的界面，并引入了 Python 和面向对象编程方法取代传统的开发方式。另外他还为商业用户提供 Python 与 Tkinter 的应用。

他获得应用生物学学士学位和分子生物学博士学位，但此决非一个障碍（尤其因为他 90%的论文是酶行为的计算机模型）。

在专攻用户界面之前，他是个操作系统专家，后来为 Stratus Computer 公司 Unix 开发支持方法。他在 Pacer 软件公司做了个 F77 编译器和 Unix 便携工具。他还是在美国和英国的 Prime Computer 公司的操作系统专家。

约　定

《Python 与 Tkinter 编程》一书中例子起到了极大的作用。不少编程书籍中例子的特征是短小简洁，把一两个要点很好地表述出来——但其实不起什么作用。本书中，例子可能适合你的应用或者正是需要的。很多例子的目的是单独运行，而不是交互式的。大部分例子包括标号以及对应的注解。例如：

```
def mouseDown (self, event):
    self.currentobject = None
    self.lastx = self.startx = self.canvas.canvasx (event.x)
    self.lasty = self.starty = self.canvas.canvasy (event.y)
    if not self.currentFunc:
        self.selobj = self.canvas.find_closest (self.startx,
                                    self.starty) [ 0 ]
        self.canvas.itemconfig (self.selobj, width=2)

        self.canvas.lift (self.selobj)
```



代码注解

❶ 鼠标按下方法反选任何当前已经选定对象。事件返回在屏幕中鼠标点击点的 x 和 y 坐标。画布控件的 canvasx 和 canvasy 方法…

❷ 如果绘画功能没有选中，则我们在选择模式下，并且在画布中对最近的对象定位，然后选中。该方法……

偶尔，我把部分代码变为黑体强调其在代码例子中的重要性。

有些地方代码跨越了多页，我已经把源代码散开，这样注释和源代码的距离可以保持比较近。任何例子中的标号都是连续的。

关于封面

本书封面是从 1805 年版的 Sylvain Maréchal 的四卷地区服饰习俗概述中得来的。这本书第一次于 1788 年在巴黎出版，是在巴黎革命前一年。单单标题就 30 多字。

Costumes Civils actuels de tous les peuples connus dessinés d'après nature gravées et coloriés, accompagnés d'une notice historique sur leurs coutumes, moeurs, religions, etc. , etc., redigés par M. Sylvain Maréchal

四卷包括图片的注解："gravé à la manière noire par mixelle d'après Desrais et colorié"。很明显，能将雕刻师和插图者姓氏罗列下来，已经不错了，因为毕竟他们只是艺术家而已。而手绘这些东西的人则默默无闻。

这一丰富多彩的艺术品集生动地向我们说明了 200 年前的城市和乡村文化如何远离而去。服饰不断变化，不同地区也差异甚远。当年的多样性渐已消失。现在，很难区分一个大陆和另一个大陆之间的差异。也许，我们已经用文化的多样化换取了人生的复杂——确实，多么丰富和激动的技术环境啊。在无法区分一本计算机书和另一本计算机书的时候，Manning 通过在封面上使用 200 年前的地区生活的多样性来庆祝计算机事业的发明和兴起，带回到 Marechal 图片的生活中。想一想，Marechal 的世界是一个远远不同于我们的世界，在那里，人们愿意花时间去阅读 30 个字的标题。

作者在线

《Python 与 Tkinter 编程》的购买包括了可以进入一个私人 Internet 论坛，在那里，你可以对此书做评论，提出技术问题以及从作者及其他读者那里获得帮助。你通过你的浏览器浏览 http://www.manning.com/grayson 进入论坛。那里，你可以订阅论坛，此站点还提供信息，告诉你一旦注册之后如何加入论坛，可以得到什么样的帮助以及论坛行为准则。

书中所有的例子在 Manning 主页上都有，URL 为 www.manning.com/grayson，包括了一个指向源代码的链接。

内 容 简 介

Thinter 是解释性脚本语言 Pyhton 的图形界面开发工具包，是一个清晰的、面向对象的图形界面开发工具，它可以让你快速地开发出可以运行在 Unix、Macintosh 和 Windows 等系统平台下的 Python 应用程序。

本书详细地介绍了 Python 语言的语法和函数及其图形界面的开发方法，重点介绍了在 Tkinter 可视化平台下的编程方法。

本书非常适合各个层次的 Python 程序员，同时也适合利用 Python 开发各种脚本的设计人员。

目　录

第1部分　基本概念

第1章　Python ····· 3

1.1　Python 编程及特点概述 ····· 3

1.1.1　为什么是 Python ····· 4

1.1.2　Python 能被用于何处 ····· 4

1.2　关键字数据类型：表、元组和字典 ····· 4

1.2.1　表 ····· 5

1.2.2　元组 ····· 6

1.2.3　字典 ····· 7

1.3　类 ····· 8

1.3.1　类是怎样描述对象的 ····· 8

1.3.2　定义类 ····· 8

1.3.3　Python 技巧 ····· 8

1.3.4　一个实例的初始化 ····· 9

1.3.5　方法 ····· 9

1.3.6　私有与公有变量和方法 ····· 9

1.3.7　继承性 ····· 10

1.3.8　多重继承性 ····· 10

1.3.9　混合类 ····· 10

第2章　Tkinter ····· 11

2.1　Tkinter 模块 ····· 11

2.1.1　Tkinter 是什么 ····· 11

2.1.2　性能如何 ····· 11

2.1.3　如何运用 Tkinter ····· 12

2.1.4　Tkinter 的特点 ····· 12

2.2　Tcl/Tk 对 Tkinter 的映像 ····· 13

2.3　Win32 和 Unix 下的 GUI ····· 13

2.4　Tkinter 类的层次 ····· 14

2.5　Tkinter 控件的外观 ····· 15

第3章　建立一个应用 ····· 16

3.1　计算器范例：关键特点 ····· 18

3.2 计算器范例：源代码 …… 19
3.3 检查应用结构 …… 25
3.4 应用的扩展 …… 26

第2部分 显 示

第4章 Tkinter控件 …… 29
4.1 Tkinter控件漫游 …… 29
4.1.1 顶层 …… 29
4.1.2 框架 …… 30
4.1.3 标签 …… 32
4.1.4 按钮 …… 32
4.1.5 输入 …… 34
4.1.6 单选按钮 …… 34
4.1.7 复选按钮 …… 35
4.1.8 主菜单 …… 35
4.1.9 消息 …… 38
4.1.10 文本 …… 39
4.1.11 画布 …… 40
4.1.12 滚动条 …… 41
4.1.13 列表框 …… 41
4.1.14 标尺 …… 42
4.2 字体和和颜色 …… 43
4.2.1 字体描述符 …… 43
4.2.2 X视窗系统字体描述符 …… 43
4.2.3 颜色 …… 44
4.2.4 设置宽应用默认字体和颜色 …… 44
4.3 Pmw大控件漫游 …… 45
4.3.1 关于框 …… 45
4.3.2 浮动图 …… 46
4.3.3 按钮框 …… 47
4.3.4 组合框 …… 47
4.3.5 组合对话框 …… 49
4.3.6 计数器 …… 49
4.3.7 计数对话框控件 …… 50
4.3.8 对话框 …… 51
4.3.9 输入域 …… 51
4.3.10 组 …… 52
4.3.11 标签控件 …… 53

4.3.12 菜单条 …… 54
4.3.13 消息栏 …… 54
4.3.14 消息对话 …… 55
4.3.15 记事本 R …… 56
4.3.16 记事本 S …… 57
4.3.17 记事本 …… 57
4.3.18 选项菜单 …… 58
4.3.19 窗格控件 …… 59
4.3.20 提示对话框 …… 60
4.3.21 单选选项 …… 60
4.3.22 滚动画布 …… 61
4.3.23 滚动区域 …… 62
4.3.24 滚动框架 …… 62
4.3.25 滚动列表框 …… 63
4.3.26 滚动文本 …… 64
4.3.27 选项对话 …… 64
4.3.28 文本对话 …… 65
4.3.29 时间计数 …… 66
4.4 创建新的大控件 …… 66
4.4.1 大控件的描述 …… 66
4.4.2 选项 …… 67
4.4.3 制作大控件类 …… 67
第 5 章 屏幕版面设计 …… 70
5.1 版面设计的介绍 …… 70
5.2 打包器 …… 71
5.2.1 使用展开选项 …… 74
5.2.2 使用填充选项 …… 74
5.2.3 使用 padx 和 pady 选项 …… 75
5.2.4 使用锚选项 …… 76
5.2.5 使用分层性包装 …… 76
5.3 网格 …… 77
5.4 放置器 …… 82
5.5 小结 …… 85
第 6 章 事件，捆绑和回叫 …… 87
6.1 事件驱动系统：评论 …… 87
6.1.1 事件是什么 …… 87
6.1.2 事件的传播 …… 88
6.1.3 事件类型 …… 88
6.2 Tkinter 事件 …… 89

6.3 回叫 …… 93
6.4 λ表达 …… 94
6.5 捆绑事件和回叫 …… 95
6.5.1 捆绑方法 …… 95
6.5.2 处理多重捆绑 …… 97
6.6 定时器和背景程序 …… 97
6.7 动态回叫管理者 …… 98
6.8 使事件运作 …… 98
6.8.1 捆绑动态数据与控件 …… 98
6.8.2 数据确认 …… 102
6.8.3 格式化（smart）控件 …… 107
6.9 小结 …… 109
第7章 使用类、控件和特殊控件 …… 110
7.1 创建发光二极管类 …… 110
7.1.1 再试试 …… 115
7.1.2 什么改变了 …… 119
7.2 构件类库 …… 119
7.2.1 将六边形螺帽加入类库 …… 121
7.2.2 创建开关类 …… 123
7.2.3 创建大控件 …… 126
7.3 小结 …… 129
第8章 对话框和窗体 …… 130
8.1 对话框 …… 130
8.1.1 标准对话框 …… 131
8.1.2 数据输入对话框 …… 132
8.1.3 简单表格 …… 135
8.1.4 Tkinter 变量 …… 140
8.2 标准窗体框架 …… 144
8.3 数据字典 …… 154
8.4 活页夹 …… 160
8.5 浏览器 …… 163
8.6 压缩程序 …… 172
8.7 图像映射 …… 180
8.8 小结 …… 187
第9章 面板与机器 …… 188
9.1 创建前端面板 …… 188
9.2 模块性 …… 189
9.3 实现前端面板 …… 190
9.4 GIF、BMP 与图层 …… 204

9.5 一个更完整的例子 …… 209
9.6 使用 POV-Ray 创建的虚拟设备 …… 222
9.6.1 现在看点完全不同的东西 …… 222
9.7 小结 …… 225
第 10 章 绘制点和标尺 …… 226
10.1 在画布上绘图 …… 226
10.1.1 移动画布对象 …… 231
10.2 一个更完整的绘图程序 …… 232
10.3 滚动画布 …… 240
10.4 标尺工具 …… 243
10.5 缩放画布对象 …… 246
10.6 一些已完成的小玩意 …… 251
10.7 快速绘制 …… 260
10.8 小结 …… 263
第 11 章 图形和图表 …… 264
11.1 简单图形 …… 264
11.2 图形控件 …… 266
11.2.1 添加条形图 …… 273
11.2.2 饼图 …… 276
11.3 三维图形 …… 279
11.4 带形记录图 …… 284
11.5 小结 …… 286
第 12 章 导航 …… 287
12.1 引言：导航方法 …… 287
12.2 鼠标导航 …… 287
12.3 键盘导航：“无鼠标导航” …… 288
12.4 建立应用程序的导航 …… 288
12.5 图像映射 …… 291
12.6 小结 …… 291
第 13 章 窗口管理器 …… 292
13.1 什么是窗口管理器 …… 292
13.2 几何方法 …… 293
13.3 可见性方法 …… 293
13.4 图标方法 …… 294
13.5 协议方法 …… 294
13.6 其他 wm 方法 …… 295

第3部分 组合起来

第14章 Python扩展 ····· 299
14.1 写Python扩展 ····· 299
14.2 建立Python扩展 ····· 301
14.2.1 在Unix下静态链接一个扩展 ····· 302
14.2.2 在Windows下静态链接一个扩展 ····· 302
14.2.3 在Unix下构建动态模块 ····· 303
14.2.4 在Windows下构建动态模块 ····· 303
14.2.5 安装动态模块 ····· 304
14.2.6 使用动态模块 ····· 304
14.3 在扩展中使用Python API ····· 304
14.4 在C++中构建扩展 ····· 306
14.5 格式化字符串 ····· 306
14.6 引用计数 ····· 309
14.7 嵌入式的Python ····· 309
14.8 小结 ····· 313
第15章 调试程序 ····· 314
15.1 为什么使用print表达式 ····· 314
15.2 一个简单的例子 ····· 314
15.3 如何调试 ····· 317
15.4 Tkinter开发器 ····· 318
15.5 pdb ····· 319
15.6 IDLE ····· 319
15.7 DDD ····· 320
第16章 设计有效的图形应用程序 ····· 321
16.1 友好界面设计的元素 ····· 321
16.2 人为因素 ····· 324
16.2.1 选择字体 ····· 325
16.2.2 在GUI中使用颜色 ····· 326
16.2.3 尺寸的考虑 ····· 327
16.3 可选择的GUI ····· 327
16.4 小结 ····· 328
第17章 性能编程 ····· 329
17.1 每日加速 ····· 329
17.1.1 程序组织 ····· 329
17.1.2 使用Python优化器 ····· 330
17.1.3 检查代码 ····· 330

17.2 Tkinter 性能 ····· 330
17.2.1 让它保持简短 ····· 331
17.2.2 删减本地变量 ····· 331
17.2.3 保持简单 ····· 332
17.2.4 快速初始化 ····· 332
17.2.5 节省事件 ····· 332
17.3 Python 技术 ····· 332
17.3.1 引用模块 ····· 333
17.3.2 字符串链接 ····· 333
17.3.3 正确使用嵌套的循环 ····· 334
17.3.4 减少模块引用 ····· 334
17.3.5 使用本地变量 ····· 335
17.3.6 使用异常处理 ····· 335
17.3.7 使用 map，filter 和 reduce ····· 336
17.4 程序剖析 ····· 337
17.5 Python 扩展 ····· 340
17.6 小结 ····· 340
第 18 章 线程和异步技术 ····· 341
18.1 线程 ····· 341
18.1.1 非图形用户界面下的线程 ····· 341
18.1.2 图形用户界面下的线程 ····· 344
18.2 “after”处理 ····· 348
18.3 小结 ····· 351
第 19 章 发布 Tkinter 程序 ····· 353
19.1 发布程序的一般问题 ····· 353
19.2 发布 Unix 程序 ····· 354
19.3 发布 Win32 程序 ····· 354
19.4 Python 发布工具 ····· 356

第 4 部分 附 录

附录 A Tk 到 Tkinter 映射 ····· 361
附录 B Tkinter 参考资料 ····· 392
附录 C Pmw 参考：Python megawidgets ····· 485
附录 D 创建和安装 Python,Tkinter ····· 539
附录 E 事件与键盘系统 ····· 545
附录 F 光标 ····· 549
附录 G 参考文献 ····· 551

第1部分

基本概念

在第 1 部分,我介绍 Python、Tkinter 和应用编程。既然我假设你已经在某种程度上熟悉 Python，第 1 章旨在介绍一些 Python 语言在全书中将要用到的重要特性。另外，我还介绍 Python 支持的面向对象编程，这样，你们中熟悉 C++或 Java 的可以知道你的经验也可以用到 Python 中。

第 2 章大致介绍了 Tkinter，说明了它是如何与 Tcl/Tk 相关的。你会在后面看到详细的 Tk 到 Tkinter 的映射，包括控件及外观介绍。

第 3 章通过两个计算器例子来说明 Python 的开发。第一个为简单无装饰计算器，演示了基本原理，第二个为部分完成的应用，告诉你 Python 与 Tkinter 能开发多么强大的程序。

第1章 Python

这一章定义了Python的主要特点。它使语言规范化以适应系统的快速标准化和完全的函数应用。《Python和Tkinter编程》一书并不希望成为Python初学者的学习资料，一些其他的出版物更能胜任。*Quick Python*、*Learning Python*、*Programming Python*、*Internet Programming in Python*和*Python Pocket Reference*都是优秀的课本。本书末参考文献部分提供了更多的信息。在这一章，将用代码的简明范例来强调Python的关键特点以阐明一些书中所用范例的构件块。

1.1 Python编程及特点概述

如前所述，本书不是直接用来学习Python的基础的。精通其他语言的程序员能够通过分析范例发现用Python编程的关键点。然而，若你相对于一般的编程而言还是个新手，那么你学习Python就将是个困难重重的奋斗过程了。

本章实际上对大多数读者来说并不是必须的，因为其内容早已为读者所熟悉。它的目的是给那些早期用Python工作的读者提供一个更新教程，给Tcl/Tk程序员，以及那些用过其他语言的程序员提供一个指南。

不熟悉面向对象的程序设计方法(OOP)的读者会发现1.3节非常有用，它对Python中所用的OOP作了介绍。需要了解Python的类是如何操作的C++或Java的程序员也可以从中受益。

我不打算阐述Python是为什么或什么时候被发展起来的，因为这些信息在其他各种关于Python的书中已被很好地提及了。我将叙述Python的创作者Guido van Rossum,自从约1990年他在荷兰的阿姆斯特丹的CWI发明了这种语言后他一直支持着它。目前他在美国维吉尼亚洲Reston的国家专利研究所（CNRI Corporation for National)。事实上发展了该语言成长的人已就它的稳定性和完美性作了大量的工作,无论他们用了什么方法使这一语言得到发展，Gudio将首先感谢所有作出贡献的人。

语言的名称可能比以上所说的任何信息更重要。这种语言跟蛇类没有任何关系。Python是以1969年到1974年BBC中的喜剧连续剧*Monty Python Flying Circus*来命名的。像20世纪70年代的许多大学生一样，我也深受Monty Python的影响，所以当我开始写本书的时候，除语言之外我不可抑制地加了少许的Python。现在，所有跳过本书枯燥无味的开始部分，或决定不需要读这段的人将大吃一惊。从范例中你将发现少许的Python。如果你没有经历过Monty Python，那么我只能给你以下忠告：如果范例中的有些东西看上去不可思议，那很可能是Python，像我南斯拉夫大学的一个朋友所常说的“你认为这

好笑吗？”

1.1.1 为什么是 Python

一些主要特点使 Python 成为广泛应用范围内的一种理想的语言，加上 Tkinter 的结合扩大了戏剧性的可能性。这是 Python 之所以为 Python 的重点：

- 比特码的自动编译
- 高级数据类型及操作
- 可移植结构
- 支持大范围的扩充
- 面向对象模式
- 理想规范系统
- 带有不同类 C 支持保持性的易读码
- 在 C 与 C++中易于扩展，内嵌于应用
- 大型贡献性的应用与工具库
- 优秀的文档

你可能注意到了我没有明确地提及翻译程序。Python 的一个特点是它是一个用 C 编写的比特码编码器。可扩充模块是用 C 编写的。粗心大意会妨碍你编写代码。你的大多数代码是通过编译 C 来运行的，因为许多操作是构建在系统中的。余下的代码是在比特码编码器中运行的。结果是这样一个系统：可作为脚本语言来展开，从某些系统管理脚本一直到一个复杂的 GUI 基础应用（用数据库、客户/服务器、CORBA 或其他技术）。

1.1.2 Python 能被用于何处

了解在哪里 Python 不是最佳选择是知道 Python 可用于何处的最佳途径。除了我刚刚说过比特码编码器，Python 带有翻译性质（解释性），所以如果你无法保持在 C 的范围内，在执行上就会有麻烦。因此，实时应用与高速很难匹配（实时性与快速性无法兼得）。Python 的一系列扩展被发展起来，特别是数字编程（参见附录 G 的 NumPy）。这些扩展支持计算设限应用，但是除非不考虑时间因素，否则 Python 并不是大型的集中计算应用的最佳选择。同样，包含实时观察的集中制图应用也不是个理想的选择（但参看 10.7 的 Speed drawing 的范例可知它能做什么）。

1.2 关键字数据类型：表、元组和字典

三个关键数据类型给了 Python 有效的应用能力：两个顺序类——表和元组——和映像类——字典，当它们被一起用时，它们能用一小段代码表达出令人吃惊的力量。

表和元组有很多共同之处，它们的主要不同在于表的元素能在适当处被修改而元组却是不可变的：你必须析构它并重新构建一个元组来改变个别元素。我们应当关注这个区别有几个很好的原因：如果你想用元组作为字典中的一个关键字，那么最好知道它不能被反复修改，元组的一个小小的优势是个便宜的资源，因为它们并不执行表

中的一些复杂操作。如果你想深入了解这些数据类型，看一下第 6 至 8 章的快速 Python。

1.2.1 表

让我们先看一下表，如果你以前没有学过 Python，记得看一下标准文档教程，点击 www.python.org 可见。

表的创建

表很容易创建及使用，初始化表：

```
lst = []    # Empty list
lst = ['a', 'b', 'c']                    # String list
lst = [1, 2, 3, 4]                       # Integer list
lst = [[1, 2, 3], ['a', 'b', 'c']]       # List of lists
lst = [(1, 'a'), (2, 'b'), (3, 'c')]     # List of tuples
```

表的追加

表可用以下方法追加：

```
lst.append ('e')
lst.append ((5, 'e'))
```

表的连接

连接表很方便：

```
lst = [1, 2, 3] + [4, 5, 6]
print lst
[1, 2, 3, 4, 5, 6]
```

元素的迭代

在表中的迭代很容易：

```
lst = ['first', 'second', 'third']
for str in lst:
print 'this entry is %s' % str
set = [(1, 'uno'), (2, 'due'), (3, 'tres')]
for integer,str in set:
print 'Numero "%d" in Italiano: è "%s"' % (integer, str)
```

排序与反转

建立一个排序表或反转表：

```
lst = [4, 5, 1, 9, 2]
lst.sort ()
print lst
[1, 2, 4, 5, 9]

lst.reverse ()
print lst
[9, 5, 4, 2, 1]
```

索引

在表中查找值：

```
lst = [1, 2, 4, 5, 9]
print lst.index (5)
3
```

元素

检查表中的元素非常方便：

```
if 'jeg' in ['abc', 'tuv', 'kie', 'jeg']:
    ...
if '*' in '123*abc':
    ...
```

修改元素

可在适当处修改表中的元素：

```
lst = [1, 2, 4, 5, 9]
lst [3] = 10
print lst
[1, 2, 4, 10, 9]
```

插入与删除元素

在表中插入一个元素：

```
lst = [1, 2, 3, 4, 10, 9]
lst.insert (4, 5)
print lst
[1, 2, 3, 4, 5, 10, 9]
```

删除一个元素：

```
lst = [1, 2, 3, 4, 10, 9]
del lst (4)
print lst
[1, 2, 3, 4, 9]
```

1.2.2 元组

与表相似，但它们是不可变的（意思是不可修改的），元组收集数据非常方便。它可作为单一实体被传送或存储在一张表或字典中（列表），当需要时实体即被打开。

元组的创建

除了元组包含的一个元素外，元组的创建方法同表类似（表与元组是相关顺序类型，极易互换）。

```
tp1=()                              #空元组
tp1=(1,)                            #单元素元组
tp1=('a','b','c')                   #字符串元组
tp1=(1, 2, 3, 4)                    #整数元组
tp1=([1, 2,3], ['a','b','c'])       #列表元组
tp1=((1,'a')(2,'b')(3,'c'))         #元组之元组
```

元组的迭代

```
for I in tp1:
...
for I, a in ((1, 'a'), (2, 'b'), (3, 'c')):
```

元组的更新（修改）

（但是你说过元组是不可变的！）

```
a = 1, 2, 3
a = a[0], a[1], 10, a[2]
a
(1, 2, 10, 3)
```

注意！你并没有修改原来的元组，将它与 a 捆绑。

1.2.3 字典

字典是以关键码作索引的数据数组，我认为它在设计只读（紧缩，压缩）系统中给了 Python 一个界限。如果你在列表中使用表与元组作为数据项，你将得到一个强有力的组合（不是说混合目标代码，字典与抽象目标并不有力）。

字典的创建

字典可根据提供的关键码：值对来创建。

```
dict = {}  # Empty dictionary
dict = {'a'':1, 'b':2, 'c':3} # String key
dict = {1: 'a', 2: 'b', 3: 'c'} # Integer key
dict = {1:[1,2,3],2:[4,5,6]} # List data
```

字典的修改

字典极易修改：

```
dict['a'] = 10
dict[10] = 'Larch'
```

字典的存取

Python 的最新版方便了没有关键码的查询。首先，旧的方法：

```
if dict.has_key('a'):
value = dict['a']
else:
Value = None
try:
value = dict['a']
except KeyError:
value = None
```

这是现在的方法：

```
value=dict.get('a',None)
```

输入的迭代

获得键，然后在其中迭代：

```
keys = dict.keys()
for key in keys:
    ...
```

字典的排序

字典顺序任意，所以你要按顺序存取时，必须排序。

```
keys = dict.keys().sort()
for key in keys:
    ...
```

1.3 类

本书包含了一小部分 Python 的类，这是可能需要学习 Python 应用工具的一些细节的 C++程序员们的，还有那些要在 Python 中实现面向对象程序员设计的程序员们。

1.3.1 类是怎样描述对象的

类提供了以下的对象描述：
对象的属性（数据-元素）
对象的行为（方法）
类的继承性行为（总纲）
继承与其他的行为在那儿？

说到所有这些，C++的程序员会证明这一点——但是不能持之以恒，Python 的类有一些有用的特点。其中的一些特点将会令那些用 Python 面向对象程序设计没有完全达到速度的人吃一惊。

本书中的大部分应用举例着重依赖于建立类库来创建一个广泛的对象范围，这些类是以多重格式的典型方式创建实例的（看第七章的发光二极管与开关），在我们创建这些对象前，让我们先回顾一下应用于 Python 的类中的规则和特点。

1.3.2 定义类

Python 的类是个（以类语句来定义的）用户自定义数据类型

```
class Aclass:
  statement
```

statement 是任何一种有效的定义属性与成员函数的 Python 语句，如我们在下一章所将看到的，（可以用任何一种）Python 语句，包括完整的周期语句，称类为一个函数创建一个类的范例。

```
AnInstanceOfAClass=Aclass()
```

1.3.3 Python 技巧

一个类例可以像 C 语言结构或 Pascal 记录一样被应用，但又不同于 C 和 pascal。结构中的元素不必在使用前定义——它们可以动态地创建，我们可利用它的这个特征在模块中反复存取数据对象（后面将会给出）；用类例来支持全局数据的例子。

```
class DummyClass:
pass
Colors = DummyClass()
Colors.alarm = 'red'
Colors.warning = 'orange'
Colors.normal = 'green'
```

如果上述几行被存储在一个文件里就叫做程序数据 Py(programdata.py),以下是可能的代码结果。

```
from programdata import Colors
...
Button(parent, bg=Colors.alarm, text='Pressure\nVessel',
       command=evacuateBuilding)
```

另一方面如果你应用一点 Python 是如何管理内部数据的知识，你就能运用以下结构。

```
class Record:
def_init_(self, **kw):
self._dict_.update(kw)
Colors = Record(alarm='red', warning='orange', normal='green')
```

1.3.4 一个实例的初始化

实例域（实例变量）可以通过在类实体里包含一个_init_的方法来定义，当一个类的新的实例创建时，此方法会自动执行。Python 按第一个参数传递实例。习惯上将它命名为 self（它在 C++内被调用）。此外，可以通过函数调用实现初始化。如果需要的话，这个类的继承的_init_方法也会被用到调用。

```
class ASX200(Frame):
    def_init_(self, master=None):
        Frame._init_(self, master)
        Pack.config(self)
        self.state = NORMAL
        self.set_hardware_data(FORE)
        self.createWidgets()
...
...
switch =ASX200()
```

注意 用到实例变量时你必须引用内含对象（目标）（在前一个例子中它是 switch.state,不是 self.state）。如果你以一个变量本身为参考，它在函数执行过程中是个局部变量而不是实例变量。

1.3.5 方法

我们已经见过创建实例时所采用的_init_方法，其他方法同 def 语句相似。方法可获得自变量（参数）：self 总是第一个或唯一的自变量（参数）。

你将看到大量的方法的例子，真正需要讨论的很少，注意在方法调用与函数调用时，除了按位（位置）Python 也接受命名的参数。这使得方法支持缺省值非常容易，因为参数的缺省结果是用缺省值来替代，注意当按位参数与命名参数结合时，用这种方法能很容易地在类库中介绍问题。

1.3.6 私有与公有变量和方法

除非你采用特殊方法，否则所有的变量和 methods 都是公共的和虚拟的，如果你使

用命名，你可以模拟私有变量和方法，你可用如下方法来命名，任何以双下划线开头的名字是私有的，不可输出到一个包含式的环境。任何以单下划线开头的名字表示习惯的私有名。这同 C++或 Java 的保护字相似，事实上，Python 通常比 C++或者其他语言更直观，因为只要一有私有变量或方法被引用就显现出来。

1.3.7 继承性

Python 的继承性规则真的相当简单：

- 类的继承行为来自头部详细说明的类和此类所有的父类。
- 实例继承行为来自创建它们的类及此类所有的父类。

Python 在直接的名称空间（实例）和每个更高层名称空间搜寻附注。引用的第一个具体值被运用，这意味着类能轻易再定义超类（superclasses）。若引用未被找到 Python 特别给出出错信息。

注意继承关系不会自动调用，创建一个基础类，基类必须明确调用_init_方法。

1.3.8 多重继承性

Python 的多重继承就是继承性的一个扩展，如果在类头部有不止一个类被详细说明，那这就是一个多重继承。与 C++不同，如果类的属性被重复定义，Python 并不显示出错信息。基本规则是第一个被建立的具体值就是所用值。

1.3.9 混合类

集合了若干普通方法及可由基类自由地继承的类通常被称作混合类（一些书上可能称之为基础的、概指的、简洁的类，但那不完全正确）。一些方法可能包含在 Python 模块中，但是运用一个混合类的优点是方法可以存取类的 self，因此，可以修改实例的动作。通过本书，我们可以看到混合类的例子。

第 2 章　Tkinter

这一章描述了 Tkinter 模块及其与 Tcl/Tk 关系的结构，为了帮助 Tcl/Tk 程序员将 Tcl/Tk 转换成 Tkinter，它阐述了 Tcl/Tk 结构到 Tkinter 的映像。我们将讨论国内适用于 Unix、Win32 和 Macintosh 补充的 GUIs 并强调关键体系结构的不同。字体和颜色的选择也将被介绍。我将在第 4 章的“Tkinter 控件”中更详细地涵盖这一部分内容。对不熟悉 Tkinter 的读者来说，这章阐述了它在 Python 应用中的重要性。

2.1 Tkinter 模块

2.1.1　Tkinter 是什么

Tkinter 给 Python 的应用提供了一个易编程的用户界面。Tkinter 支持这样的 TK 典型集合：它支持大部分的应用要求。Tkinter 是 Python 与 Tk 的接口，Tcl/Tk 的 GUI 工具组。Tcl/Tk 是 John Ousterhout 发展的书写和图形设备。John 原先就读于伯克利的加利福尼亚大学，后来在 Sun Microsystems 工作。最近，Scriptics 公司在开发和支持 Ousterhout 建立的 Tcl/Tk。Tcl/Tk 与其开发者在许多领域内拥有了一批重要的追随者，在 Unix 系统中取得了压倒性的成就，但是最近转移到了 Win32 系统和 MacOS 上。关于 Tcl/Tk 的第一本书，Ousterhout 的“TCL and the TK Toolkit”，虽然旧了，却仍是一本可用的参考文献（你可在附录“参考文献”中找到一些关于这个课题的好文章）。

Tcl/Tk 开始是设计在 X 视窗系统及其产品和类似于 Motif 的视窗（Windows）下运行的，捆绑与控制行为也设计成模拟 Motif 的。Tcl/Tk 的最新版本（特别是 8.0 版及其后发布的），控件相似于执行体系结构的本机的控件。事实上，许多控件是本机控件，而加入更多控件的趋势可能将持续下去。

如同 Python 的扩展。Tcl/Tk 是作为 C 语言库组件和模块来实现的，用于解释脚本及应用。Tkinter 的接口是作为一个 Python 组件 Tkinter.py 来实现的。Tkinter.py 一定是个利用这些相同的 Tcl/Tk 库的 C 扩展（_Tkinter）。在许多情况下，Tkinter 程序不需要关注 Tcl/Tk 的实现，因为 Tkinter 可作为独立的 Python 的扩展来看待。

2.1.2　性能如何

乍看起来，假定 Tkinter 表现不好是有道理的，然而 Python 的解释程序利用了 Tkinter

模块，反过来说，依赖于调用 Tcl 和 Tk 库及有时调用 Tcl 解释程序来连接组件的_Tkinter 接口，好了，这就是事实，但是在现代系统中，它不必如此麻烦。如果你按第 17 章的“性能编程”的指示，你会发现 Python 和 Tkinter 能够传递可行的应用，如你用 Python/Tkinter 的目的是为了发展应用模块，那么这一点是不切实际的；在 Python/Tkinter 中原型将发展得更快。

2.1.3 如何运用 Tkinter

Tkinter 由若干部分组成，先提到过的_Tkinter，是个 Tk 库的低层接口，它与 Python 链接起来。直到最近，在 Python 的建立中添加 Tkinter 还一直是程序员的责任，但是从 Python 的 1.5.2 版开始，Tkinter，Tcl 和 Tk 都是安装组件的一部分了，最少在 Win32 分布中是这样，在一些 Unix 可变部分和 Macintosh 中，建造一个包含 Tkinter 的 Python 仍是必要的，不过，要检查一下你的特殊平台上是否可用二进制版本。

一旦一个包含_Tkinter 的 Python 的版本被建立起来，作为一个共享库，dll 或动态链接，就需要输入 Tkinter 模块，也输入了其他任何必需的模块，比如 Tkconstants。

新建一个 Tkinter 窗口，打三行 Python 命令行（或将它们输一个文件并打：Python filename.py”）

```
from Tkinter import Label, mainloop                              ❶
Label(text='This has to be the\nsimplest bit of code').pack()    ❷
Mainloop()                                                       ❸
```

代码注解

❶ 首先，我们输入来自于 Tkinter 模块的部分（成分），用 from *module* import label,mainloop 我们可以避免参考存取包含在模块中的属性与方法的模块。

❷ 我们创建一个包含两行正文的标签并用 Pack 几何管理器来认识控件。

❸ 最后，我们调用 Tkinter 的 mainloop 来处理事件并使显示处于活动状态，这个例子不对特殊应用的事件作出反应，但我们需显示一个 mainloop，基础窗口管理是自动的。



图 2.1 简单例子

你将看到的会显示在图 2.1 中。现在，真的再也无法得到更简单的方法了。

2.1.4 Tkinter 的特点

Tkinter 在 Tk 上加了面向对象界面。Tcl/Tk 是个面向命令的脚本语言，使用正常的驱动 Tk 控件的方法是对控件标识符应用操作。在 Tkinter 中，控件的参考是 objects，而我们常用目标 methods(方法)及它们的属性来驱动控件。

结果，Tkinter 程序易读易懂，特别是对 C++或 Java 的程序员（虽然那完全是另一回事了）。

Tk 在 Tkinter 应用上的一个重要特点是：不必过于费心去选择字体和其他关联结构特点（功能），它将在无任何修饰的众多不同的 Unix、Win32 和 Macintosh 系统下运行，

无疑地，这些结构有些本质的不同，但是 Tkinter 已为应用提供了一个极好的关联结构图形平台。

这是个面向对象的特点，然而，这就确定地将 Tkinter 还分为一个理想的发展应用体系的平台，在本书中你将看不少范例，用相对少量的代码即可完成大量的应用。

2.2 Tcl/Tk 对 Tkinter 的映像

Tcl/Tk 的映像对 Tkinter 的命令与自变量（参数）真的是一个简单处理，短时间内写下 Tkinter 代码后，Tcl/Tk 程序互做转化就容易了，可能他将不必回头用 Tcl/Tk 了，让我们来看些例子：

Txl/Tk	Tkinter
label.myLabel	myLabel = Label(master)

控件（通常作为 master 控件来参考）在 Tkinter 中很明确：

Tcl/Tk	Tkinter
label.screen.for	label = Label(form)(screen is form's parent)

对配置选项，运用值与配置命令之后的关键参数，Tkinter 用关键字参数或在目标控件中参考配置方法选项的字典。

Tcl/Tk	Tkinter
label.myLabel-bg blue	myLabel = Label(master,bg="blue")
MyLabel configure-bg blue	MyLabel["bg"]="blue"
	MyLabel.configure(bg = "blue")

因为 Tkinter 控件对象有 methods，你可以加上合适的参数直接利用它们。

Tcl/Tk	Tkinter
Pack label-side left-fill y	Label.pack(side=LEFT,fill=Y)

Tk 对 Tkinter 的完全映像在附录 A 的“Tk 到 Tkinter 的映像”中可见。

2.3 Win32 和 Unix 下的 GUI

像我前面提到的那样，在 Win32，Unix 和 Macintosh 环境中发展 Tkinter 应用是合理的。Tcl/Tk 是可移植的，并可在建立在特殊平台上，如带_Tkinter C 单元的 Python，

运用提供一组可移植的复合控件和 loop Python 代码的 Pmw*（Python 大控件），在 Unix 系统或 Win32 或 Macintosh 系统中用比特码是可能的，你所不能控制的是字体的使用及在更小范围内操作系统利用的色彩设计。

如图 2.2 所示，这个应用用了 Pmw combobox 控件同在框架内安排的 Tkinter 按钮（button）及输入控件。此例所用的字体是 Arial、bold 和 16 point。除了边缘明显的 Win32 控制，这个窗口与在 Unix 中运行的图 2.3 相差甚微，在这个例子中，字体是 Helvetica、bold 和 16 point，窗口略大些，因为字体在出格规则和笔画的粗细度上有些许不同，又因为控件的尺寸与字体有关，所以结果版面就有一些不同，如果精确地校准与测量是绝对必须的，那么，可能就需要检查运行应用程序状态的平台并对已知的差异作调整。发现运行应用和调整已知的不同的平台是可能的。一般来说，最好是设计不对规划的小变化较敏感的应用程序。

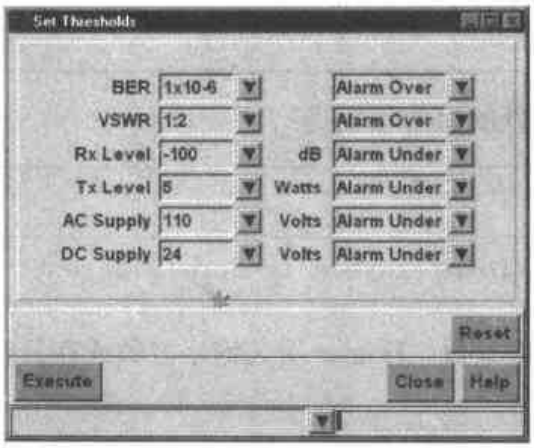


图 2.2 Win32 下的 Tkinter 与 Pmw



图 2.3 Unix 下的 Tkinter 与 Pmw

如果你仔细看，你也会注意到在顶端和底部要点上实行和关上按钮的差异，但它并不针对 Pmw 控件上的按钮。这是因为 Tk 是 Unix 的图画图形装饰而视窗 SDK 是 Win32 的装饰。

一般来说，只要你的应用不能用于特殊字体的平台，可能就要发展可移植的代码了。

2.4 Tkinter 类的层次

不同于许多视窗系统，Tkinter 的层次确实很简单，事实上，根本就没有层次。WM、Misc、Pack、Place 和 Grid 类混合在每一个控件类中，大部分程序员只需要知道树的最低层来完成日常操作且常常可以忽略纯理论的（观念的）“层次（hierarchy）”，见图 2.4。

* Pmw——Python 大控件提供从基本 Tkinter 基本控件构造而来的复杂控件，这扩展了可用控件，例如组合框、滚动帧和按钮框等。使用这些控件可使 GUI 开发人员在开发时有一个丰富的输入器件板。

0384786

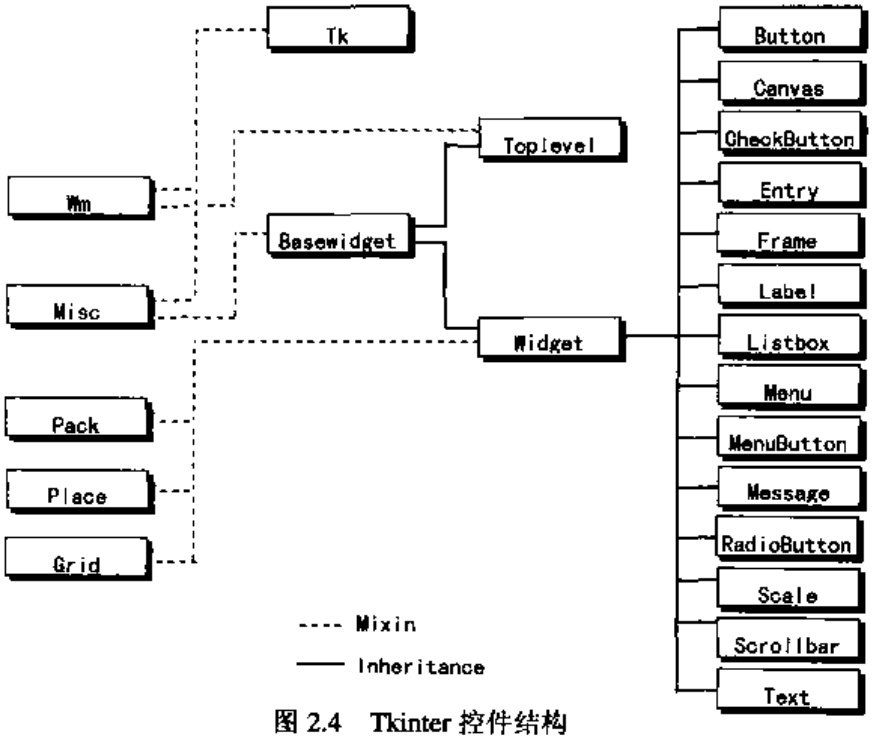


图 2.4 Tkinter 控件结构

2.5 Tkinter 控件的外观

为了包含最初的 Tkinter 介绍，让我快速浏览一下一个程序员能得到的控件的外观。

在这个例子里，我们看到控件的基础配置，它只显示了一个画布选项，我改变了框架的边缘以增加一些变化（多样性、种类），但是你们可以看到外观内缺省值的控件，见图 2.5。这里没有给出代码，可从网上得到。

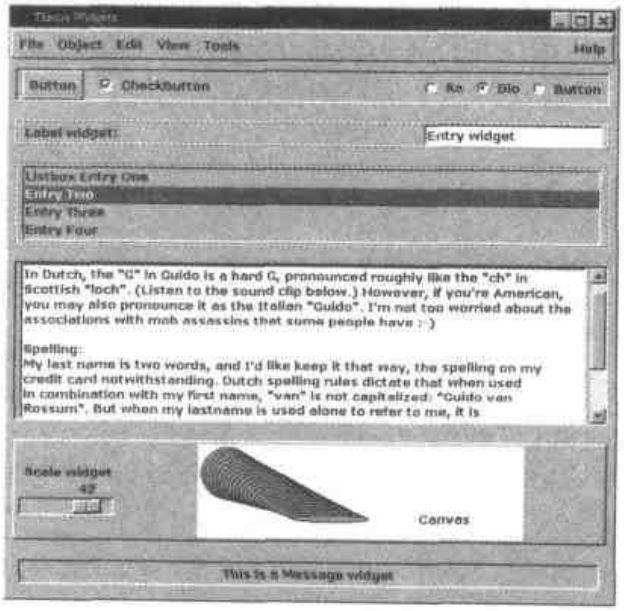


图 2.5 Tkinter 控件：大杂烩

第3章　建立一个应用

大部分关于编程的书效仿 Kernigan 和 Ritchie 的范例并且拿出了必看的“Hello World”这个例子来说明该语言很容易使用。含有 GUI 组件的书，基本上继承了这个传统，并且显示一个“Hello GUI World”或类似的东西。实际上，图 2.1 的那个三行的例子即为那类的例子。

最近图书趋向于使用计算器作为例子，本书我们打算以介绍一个如前述风格的简单的计算器的范例为开始，这个例子是用来阐述 Python 与 Tkinter 的特点及展示 Python 代码的简要性。



图 3.1　计算器例子

这个例子并不完全，因为它只能由鼠标输入。在一个完全的例子里，我们也将能用键盘输入。然而，此例能正常运行并且显示出这样一点：你并不需要一大堆代码使 Tkinter 窗口显示及运行。让我们看看支持这个屏幕的代码：

Calc1.py

```
from Tkinter import *

def frame(root, side):
    w = Frame(root)
    w.pack(side=side, expand=YES, fill=BOTH)
    return w

def button(root, side, text, command=None):
    w = Button(root, text=text, command=command)
    w.pack(side=side, expand=YES, fill=BOTH)
    return w

class Calculator(Frame):
    def __init__(self):
        Frame.__init__(self)
```

```
        self.pack(expand=YES, fill=BOTH)
        self.master.title('Simple Calculator')
        self.master.iconname("calc1")

        display = StringVar()                                          ❸
        Entry(self, relief=SUNKEN,
              textvariable=display).pack(side=TOP, expand=YES,
                                         fill=BOTH)

        for key in ("123", "456", "789", "-0."):                       ❹
            keyF = frame(self, TOP)
            for char in key:
                button(keyF, LEFT, char,
                       lambda w=display, s='%s'%char: w.set(w.get()+s))  ❺

        opsF = frame(self, TOP)
        for char in "+-*/=":
            if char == '=':
                btn = button(opsF, LEFT, char)
                btn.bind('<ButtonRelease-1>',
                         lambda e, s=self, w=display: s.calc(w), '+')   ❻
            else:
                btn = button(opsF, LEFT, char,
                         lambda w=display, c=char: w.set(w.get()+''+c+''))

        clearF = frame(self, BOTTOM)
        button(clearF, LEFT, 'Clr', lambda w=display: w.set(''))

    def calc(self, display):
        try:
            display.set(`eval(display.get())`)                         ❼
        except ValueError:
            display.set("ERROR")

if __name__ == '__main__':
    Calculator().mainloop()
```

代码注解

❶ 我们以定义适合的函数来创建框架和更简洁的按钮为开始，这些函数运用了压缩几何管理和常用且有效的行为。用一些只读函数（或合适的类）来代替一些常用代码是个好办法，因为这使得代码得到更好的保存且更为易读。

❷ 我们调用（框架）建构函数来创建顶层外壳和一个封装框架，然后，我们给窗口和图标加标题。

❸ 下一步，我们在计算器的顶部创建一个显示并定义一个可提供控件内容存取 Tkinter 变量。

```
display = StringVar()
Entry(self.master, relief=SUNKEN,
textvariable=variable).pack(side=TOP, expand=YES,
```

```
fill=BOTH)
```

❷ 记住，字符串是按 Python 中字符的顺序，所以每一个子序列实际上是我们可重复的字符数组。

```
For key in ("123", "456", "789", "-0. "):
KeyF = frame(self, TOP)
For char in key:
```

❸ 我们用结合的函数来创建按钮，传递框架，压缩选项，标签及回叫。

不要担心回叫的 lambda 表，我以后会更详细地叙述它，它的目的是清楚地定义一个直接（插入）函数。

```
button(keyF, LEFT, char,
lambda w=display, c=char: w.set(w.get()+c))
```

❹ “=”键对其他按钮有可换束缚，因为松开鼠标左键时，它调用了 Calc 方法。

```
btn.bind('<ButtonRelease-1',
lambda e, s=self, w=display:: s.calc(w))
```

❺ Calc 方法试图去计算显示中所包含的字符串，然后用计算过的变量或一个错误信息来代替原来的内容。

```
display.set(`eval(display.get()`)
```

就我个人而言，我并不喜欢这计算器，虽然它演示了压缩码，并且能十分轻易地进行功能扩展。对我而言，更是个艺术品，而不是计算器。

让我们来看看部分完成了的范例应用，它提供一个非常老练的计算器，它并没被完成。读者可以自己为范例加函数功能（你一读完本书就会准备建一个 Cray Calaulator!）虽然这个计算器未被完成，它仍有某些用途，正如下面我们将会发现的，这合理的短小的源代码中藏着一些令人吃惊的特点。

让我们开始看看这个计算器中的一些主要特点。

3.1 计算器范例：关键特点

计算器范例阐述了 Python 和 Tkinter 中的许多应用的特点，包括：

• GUI 应用结构。虽然这是一个简单的例子。但它包含了以后书中所描述的大型应用中的许多元素。

• 多重继承。范例中的多重继承很简单，但阐述了它如何简化 Python 代码。

• 第 1 章所提及的列表、字典、元组，这些语言因素给 Python 建造简明代码提供了可能性。特别是，这个例子阐述了运用字典来调度行为的方法，特别注意运用元组中的列表来定义每一个码的内容。复原该数据产生的每一个码、标签和相关的捆绑。

• Pmw (Python 大部件)。滚动的文本控件是以 Pmw 来实现的，这个例子阐述了设置属性和获得存取部件的方法。

• 基本 Tkinter 操作。创建控件，设置属性，运用文本标识符，捆绑事件以及用一个几何平面图形，管理者都被一一演示。

• eval 及 exec 函数。本例中运用了 eval 函数来显示多种数学函数；然而，在本章的

后面你将看到，exec 并不能用于实现任意的 Python，它是用于实现单行或多行码的（多行的包括控制流结构）。

3.2 计算器范例：源代码

Calc2.py

```
from Tkinter   import *
import Pmw                                          ❶Python MegaWidgets

class SLabel(Frame):
    """ SLabel defines a 2-sided label within a Frame. The
        left hand label has blue letters the right has white letters """
    def __init__(self, master, leftl, rightl):
        Frame.__init__(self, master, bg='gray40')
        self.pack(side=LEFT, expand=YES, fill=BOTH)
        Label(self, text=leftl, fg='steelblue1',
              font=("arial", 6, "bold"), width=5, bg='gray40').pack(
              side=LEFT, expand=YES, fill=BOTH)
        Label(self, text=rightl, fg='white',
              font=("arial", 6, "bold"), width=1, bg='gray40').pack(
              side=RIGHT, expand=YES, fill=BOTH)

class Key(Button):
  def __init__(self, master, font=('arial', 8, 'bold'),
               fg='white',width=5, borderwidth=5, **kw):
        kw['font'] = font
        kw['fg'] = fg
        kw['width'] = width
        kw['borderwidth'] = borderwidth
        apply(Button.__init__, (self, master), kw)
        self.pack(side=LEFT, expand=NO, fill=NONE)

class Calculator(Frame):
  def __init__(self, parent=None):
      Frame.__init__(self, bg='gray40')
      self.pack(expand=YES, fill=BOTH)
      self.master.title('Tkinter Toolkit TT-42')
      self.master.iconname('Tk-42')
      self.calc = Evaluator()          # This is our evaluator
      self.buildCalculator()           # Build the widgets
      # This is an incomplete dictionary - a good exercise!
      self.actionDict = {'second':self.doThis, 'mode':self.doThis,
                  'delete':self.doThis, 'alpha':  self.doThis,
                  'stat':  self.doThis, 'math':   self.doThis,
                  'matrix':self.doThis, 'program':self.doThis,
                  'vars':  self.doThis, 'clear':  self.clearall,
                  'sin':   self.doThis, 'cos':    self.doThis,
```

❷

❸

```
                    'tan':      self.doThis, 'up':     self.doThis,
                    'X1':     self.doThis, 'X2':     self.doThis,
                    'log':    self.doThis, 'ln':     self.doThis,
                    'store':  self.doThis, 'off':     self.turnoff,
                    'neg':    self.doThis, 'enter':   self.doEnter,
                    }
        self.current = ""

    def doThis(self,action):
        print '"%s" has not been implemented' % action

    def turnoff(self, *args):
        self.quit()

    def clearall(self, *args):
        self.current = ""
        self.display.component('text').delete(1.0, END)

    def doEnter(self, *args):
        self.display.insert(END, '\n')
        result = self.calc.runpython(self.current)
        if result:
            self.display.insert(END, '%s\n' % result, 'ans')
        self.current = ""

    def doKeypress(self, event):
        key = event.char
        if key == '\b':
            self.current = self.current + key
        else:
            self.current = self.current[:-1]

    def keyAction(self, key):
        self.display.insert(END, key)
        self.current = self.current + key

    def evalAction(self, action):
        try:
            self.actionDict[action](action)
        except KeyError:
            pass
```

代码注解

❶ 被应用的 Pmw 在本书中有显著特点，因为它们提供了一个卓越的机制来支持大范围的 GUI 需求，而且它们易被扩展来支持额外的需求。

❷ 在关键类结构中，我们（给关键字字典）添加了键码值对，然后在按钮结构上应用这些值。

```
def __init__(self, master, font=('arial', 8, 'bold'),
```

```
        fg='white',width=5, borderwidth=5, **kw):
    kw['font'] = font
    ...
    apply(Button.__init__, (self, master), kw)
```

这极大地方便了我们构建控件。

❸ 计算器类用字典为类内方法提供一个调度程序。记住字典可以处理比我们需要用这个计算器处理的相对简单的例子复杂得多的引用。

```
'matrix':self.doThis, 'program':self.doThis,
'vars':  self.doThis, 'clear':   self.clearall,
'sin':   self.doThis, 'cos':     self.doThis,
```

❹ 我们用的 Pmw ScrolledText（滚动文本）控件是一个复合的控件，用（控件）方法来获得内含控件的存取。

```
Self.display.component ('text').delete (1.0,END)
```

❺ 当按下 EnterR 键，收集字符串被传到计算器求值。

```
Result=Self .display.runpython (self.current)
```

❻ 文本插入函数的最后一个参数（自变量）是文本标签“ans”，它用于改变显示文本的前景色。

```
Self.display.insert (END,'%s\n'% result,'ans')
```

❼ dokeypress 是所有键的回叫范围。回叫中的事件参数为回叫提供了委托数据。比如按钮链或鼠标作状态的 x-y 坐标（看 6.2 节“Tkinter 事件”），在本例中我们输入了数据。

❽ 一个简单的在被选择的键上动作的例外机制被运用了。

```
def buildCalculator(self):
    FUN  = 1                  # A Function
    KEY  = 0                  # A Key
    KC1  = 'gray30'           # Dark Keys
    KC2  = 'gray50'           # Light Keys
    KC3  = 'steelblue1'       # Light Blue Key
    KC4  = 'steelblue'        # Dark Blue Key
    keys = [
        [('2nd',  '',   '',  KC3, FUN, 'second'),  # Row 1
         ('Mode', 'Quit', '', KC1, FUN, 'mode'),
         ('Del',  'Ins',  '', KC1, FUN, 'delete'),
         ('Alpha','Lock', '', KC2, FUN, 'alpha'),
         ('Stat', 'List', '', KC1, FUN, 'stat')],
        [('Math', 'Test', 'A', KC1, FUN, 'math'),   # Row 2
         ('Mtrx', 'Angle','B', KC1, FUN, 'matrix'),
         ('Prgm', 'Draw', 'C', KC1, FUN, 'program'),
         ('Vars', 'YVars','', KC1, FUN, 'vars'),
         ('Clr',  '',   '',  KC1, FUN, 'clear')],
        [('X-1',  'Abs',  'D', KC1, FUN, 'X1'),     # Row 3
         ('Sin',  'Sin-1','E', KC1, FUN, 'sin'),
         ('Cos',  'Cos-1','F', KC1, FUN, 'cos'),
         ('Tan',  'Tan-1','G', KC1, FUN, 'tan'),
         ('^',    'PI',   'H', KC1, FUN, 'up')],
```

```
        [('X2',   'Root', 'I', KC1, FUN, 'X2'),    # Row 4
         (',',    'EE',   'J', KC1, KEY, ','),
         ('(',    '{',    'K', KC1, KEY, '('),
         (')',    '}',    'L', KC1, KEY, ')'),
         ('/',    '',     'M', KC4, KEY, '/')],
        [('Log',  '10x',  'N', KC1, FUN, 'log'),    # Row 5
         ('7',    'Un-1', 'O', KC2, KEY, '7'),
         ('8',    'Vn-1', 'P', KC2, KEY, '8'),
         ('9',    'n',    'Q', KC2, KEY, '9'),
         ('X',    '[',    'R', KC4, KEY, '*']),
        [('Ln',   'ex',   'S', KC1, FUN, 'ln'),    # Row 6
         ('4',    'L4',   'T', KC2, KEY, '4'),
         ('5',    'L5',   'U', KC2, KEY, '5'),
         ('6',    'L6',   'V', KC2, KEY, '6'),
         ('-',    ')',    'W', KC4, KEY, '-')],
        [('STO',  'RCL',  'X', KC1, FUN, 'store'),  # Row 7
         ('1',    'L1',   'Y', KC2, KEY, '1'),
         ('2',    'L2',   'Z', KC2, KEY, '2'),
         ('3',    'L3',   '',  KC2, KEY, '3'),
         ('+',    'MEM',  '',  KC4, KEY, '+')],
        [('Off',  '',     '',  KC1, FUN, 'off'),  # Row 8
         ('0',    '',     '',  KC2, KEY, '0'),
         ('.',    ':',    '',  KC2, KEY, '.'),
         ('(-)',  'ANS',  '?', KC2, FUN, 'neg'),
         ('Enter','Entry','',  KC4, FUN, 'enter')]]

    self.display = Pmw.ScrolledText(self, hscrollmode='dynamic',
                        vscrollmode='dynamic', hull_relief='sunken',
                        hull_background='gray40', hull_borderwidth=10,
                        text_background='honeydew4', text_width=16,
                        text_foreground='black', text_height=6,
                        text_padx=10, text_pady=10, text_relief='groove',
                        text_font=('arial', 12, 'bold'))
    self.display.pack(side=TOP, expand=YES, fill=BOTH)

    self.display.tag_config('ans', foreground='white')
    self.display.component('text').bind('<Key>', self.doKeypress)
    self.display.component('text').bind('<Return>', self.doEnter)

    for row in keys:
        rowa = Frame(self, bg='gray40')
        rowb = Frame(self, bg='gray40')
        for p1, p2, p3, color, ktype, func in row:
            if ktype == FUN:
                a = lambda s=self, a=func: s.evalAction(a)
            else:
                a = lambda s=self, k=func: s.keyAction(k)
            SLabel(rowa, p2, p3)
            Key(rowb, text=p1, bg=color, command=a)
        rowa.pack(side=TOP, expand=YES, fill=BOTH)
```

```
            rowb.pack(side=TOP, expand=YES, fill=BOTH)

class Evaluator:
  def __init__(self):
      self.myNameSpace = {}
      self.runpython("from math import *")

  def runpython(self, code):
      try:
          return `eval(code, self.myNameSpace, self.myNameSpace)` ❽
      except SyntaxError:
          try:
              exec code in self.myNameSpace, self.myNameSpace      ❾
          except:
              return 'Error'

Calculator().mainloop()
```

代码注解（续）

❾ 这里定义了许多常量，以下的数据结构非常复杂，在这样复杂的结构中使用常量能轻易地改变值且使代码更容易被读懂，因此易于保持。

```
FUN  = 1              # A Function
KEY  = 0              # A Key
KC1  = 'gray30'       # Dark Keys
KC2  = 'gray50'       # Light Keys
```

这些是用来构成一个列表的嵌套列表，它包含了元组。元组包含三层：键颜色、函数或键标识符和绑定键的 cmd（活动的）回叫的方法。

❿ 我们创建了 Pmw 滚动文本控件和给出它大部分属性的值。

```
self.display = Pmw.ScrolledText(self, hscrollmode='dynamic',
                  vscrollmode='dynamic', hull_relief='sunken',
                  hull_background='gray40', hull_borderwidth=10,
                  text_background='honeydew4', text_width=16,
```

注意外层（hull，在 Pmw 控件中低级控件的容器）的属性和文本控件（在预置控件前提下）是如何存取的。

⓫ 我们定义一个（在计算器屏幕上）用来区分输入输出的文本标志。

```
self.display.tag_config('ans', foreground='white')
```

我们看到标签在早些的文本插入方法中使用过。

⓬ 再一次，我们必须运用一个 λ 表达式来连接回叫函数。

⓭ Python 的异常(exception)十分有弹性，它允许简单的纠错控制。在计算器评估器中，我们先运行一下 eval。

```
try:
    return `eval(code, self.myNameSpace, self.myNameSpace)`
```

这里要用来支持直接的数学运算，eval 不能处理代码顺序；然而，若我们试图评估一个码的顺序，将显示一个语法错误（SyntaxError）异常。

● 我们看一下这个异常休息：

```
except SyntaxError:
    try:
        exec code in self.myNameSpace, self.myNameSpace
    except:
        return 'Error'
```

代码在异常句中被 exec。注意到这个异常是如何被另一个 try 异常子句所封装。

图 3.2 给出了在计算器上按键计算简单的数学方程式的结果。与大部分计算器不同，这里的输入输出用不同颜色表示，也可以滚屏以察看以往的计算过程，相似于打印计算器（数据输出是打印在纸上的一种计算器。——译者注）你单击显示屏即可直接输出数据，这里有一个使人吃惊之处，你可以键入 Python，就可以得到 exec 来运行代码。



图 3.2 一个更好的计算器

图 3.3 给出了你如何才能输入 sys 模块并在 Python 内存取函数，从技术上来说（从这个窗口）（在这个非常小的显示窗口的限制之内）你几乎可以做任何事；然而，我认为这个计算器并不是 Python 大为缺乏的交互发展环境（IDE）。订阅了 Python 新闻组的读者会明白有这样一个不变的要求：给 Python 一个交互发展环境。

在 dir（）按 Enter 键，你可看见与图 3.4 相似的输出，内部符号的列表在屏幕上滚动了好几行（别忘了，控件只有 16 个字符宽）。

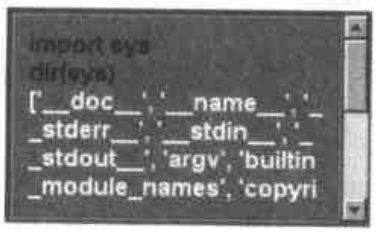


图 3.3 python 输入

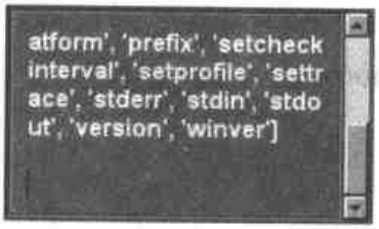


图 3.4 dir( )输出

因为我们保持了一些局部命名空间，使建立一个交互的可做一些有用工作的 Python 对话成为可能。图 3.5 说明了我们如何在命名空间中设置变量和用内部函数操作数据。

图 3.6 是我们能从交互层获得存取翻译程序的另一个例子，当这些例子被用来限制适合在计算器显示的限制空间中的操作，它们确实说明了更重大应用的潜力，注意 Python 是如何允许你在当前命名空间创建和运用变量的。

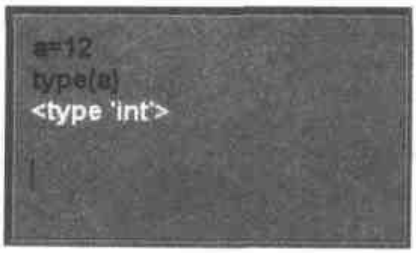


图 3.5　变量和内部函数

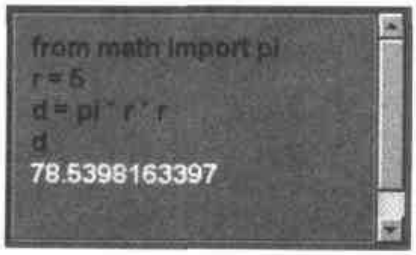


图 3.6　使用数学模块

注意　当发展应用时，我一般会隐藏一个"秘密"的键顺序来采用图形用户界面，这允许我执行任意的 Python，因此能在一个运行的系统中检查命名空间或修改对象。这真的是一个小型的调试程序，当发生什么不平常的事时，我经常在开发时存取它。有时重新启动调试话路应用并不能解决问题。在 15.4 节可找到这些工具中的一个的例子。

3.3　检查应用结构

计算器的范例的简明代码来源于这样一个事实：Tkinter 为应用提供了大量的结构，输入 Tkinter 建立了系统的基础对象，并且只需增加一点代码即可演示图形用户界面。实际上能写出的最少的 Tkinter 码只有 4 行：

```
from Tkinter import  *
aWidget = Label(None, text='How little code does it need?')
aWidget.pack()
aWidget.mainloop()
```

在这一段中，标签控件被用 pack 方法实现。启动 Tkinter 事件的循环则需靠 mainloop 来实现。在我们的这个计算器范例中，应用结构有一点复杂。

```
from Tkinter import *
...
define helper classes
...
class Calculator:
...
create widgets
...
```

```
Calculator.mainloop()
```

调用 Calculator.mainloop()创建了一个计算器实例并启动了 mainloop。

当我们扩大更多的应用，你将会反复地看到这种结构。这些对我们来说太抽象化，图 3.7 所示可以帮助我们理解。

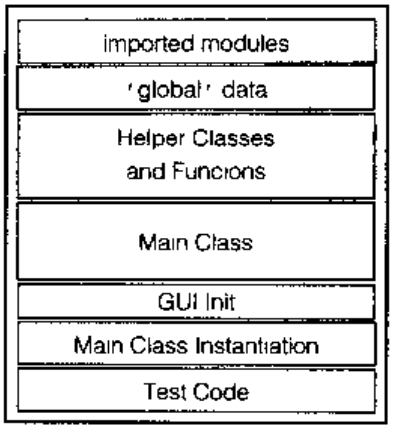


图 3.7 应用结构

我们所要做的全部只是填写这些分块，这就行了，基本上就行了。我认为这个结构中最重要的是最后一块："Test Code"。这部分的目的是允许你测试这样一个模块：它是不带有全部适合的应用结构的一组模块的一部分。用这种方法来写 Python 代码可以使综合应用成分这一过程大大地省力，当然，这种方法适用任何扩展。

3.4 扩展应用

我留给你一个扩展计算器和完成定义函数的练习，修改 keys 列表来移去不必要的键并生成一个更加集中的计算器是个简单的任务，修改键来生成一个商用的或十六进制的计算器也是可能的。

在后面的例子中，你将会看到更复杂的用范例来阐述的应用结构的表示。

第2部分

显 示

本部分我们将浏览创建应用的组件。我们从第 4 章 Tkinter 控件开始，并解释它们的主要特征以及与它们驱动的基本 Tkinter 控件的关系。记住 Tkinter 提供了一个到 GUI 的面向对象的途径。因此，尽管控件的行为与 Tcl/Tk 程序中创建的控件相同，但是创建和操纵的方法是大不一样的。

我们一旦看了控件和用来为控件提供有价值的库的 Pmw (Python MegaWidgets)，我们将讨论在第 5 章定义的使用几种几何管理器进行的屏幕显示。

第 6 章解释如何让应用程序响应外部事件。这章很重要，因为它涵盖了处理用户输入的多种方法。

第 7 章给出了类和继承在 Tkinter 中的应用。这对面向对象程序员新手很重要，对习惯于在 C++和 Java 中使用面向对象程序员也很有用。因为有些很值得注意的差异。然后，在第 8 章，我将介绍更多对话框和交互模型方面的高级技术。

第 9 章介绍面板和机器。这对读者可能是个新概念。它介绍如何创建与要控制或监控的设备相似的创新的用户界面。

第 10 章为创建允许用户在屏幕上画物体的界面提供信息。然后介绍改变属性的

方法。你将会看到这样的程序代码，它讲述了如何将随软件一块发布的应用程序Tcl/Tk程序快速地转变为Tkinter程序。第11章解释如何通过二维点以及三维图来方便地绘制图画。

第 4 章　Tkinter 控件

本章我将描述控件和 Tkinter 可用的功能，也会讨论 Pmw Python 大控件，因为它们可为 Tkinter 提供有价值的扩充。每个 Tkinter 和 Pmw 控件都将同显示产生的源代码片段一起被显示，范例短小简单，但其中的一些阐述了如何可轻易地用最少的代码来生成强大的图片。

本章不打算列出所有的对 Tkinter 程序员有用的选择，附录 B 中给出了完整可被每一个控件利用的选项与方法的文件管理，相似地，Pmw 选项和方法在附录 C 中被说明了。用这些附录来决定每个控件的全部选项。

4.1　Tkinter 控件漫游

以下控件演示显示了典型的 Tkinter 控件的外观与用途，代码被控制得非常短，它仅阐述了少许控件的有用选项，有时会用到一个或多个控件的方法，但并不深刻详细，如果你想查阅特殊的方法或选项，参考附录 B，每个控件在附录中都有相应的地方参考。

除第一个例子外，代码例删去了输入与初始化 Tkinter 的样板码，在第一个例子中常量代码是用粗体显示的，注意大部分范例更多的是作为函数而不是类来编码的，这有助于保持大量的低级码，在网上可查到所有显示的全部源代码。

4.1.1　顶层

Toplevel 控件为其他的控件提供了单独的容器，比如框架。对简单的单独视窗应用来说，初始化 Tk 时创建的根 Toplevel 可能是你所需要的唯一的外框。图 4.1 给出了 4 种类型的 Toplevel（顶层）。

1. 主顶层，作为根被引用。

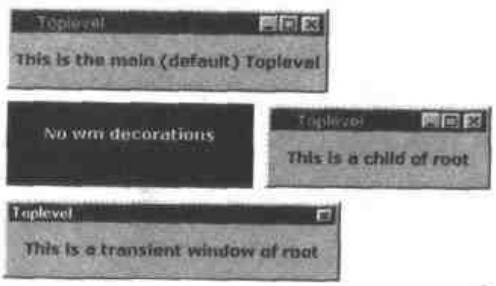


图 4.1　顶层控件

2. 子顶层，依赖于根，若根被破坏，则子顶层也被破坏。

3. 临时顶层，总是画于父顶层的顶部，如果父层被图标化或最小化之后，则它们被隐藏起来。

4. 未被视窗管理者创建过的顶层可以用通过设置一个 overrideredirect 标志为非零值来创建。该窗口不能被缩放或拖动。

```
from Tkinter import *
root = Tk()
root.option_readfile('optionDB')
root.title('Toplevel')

Label(root, text='This is the main (default) Toplevel').pack(pady=10)
t1 = Toplevel(root)
Label(t1, text='This is a child of root').pack(padx=10, pady=10)
t2 = Toplevel(root)
Label(t2, text='This is a transient window of root').pack(padx=10,
pady=10)
t2.transient(root)
t3 = Toplevel(root, borderwidth=5, bg='blue')
Label(t3, text='No wm decorations', bg='blue', fg='white').pack(padx=10,
pady=10)
t3.overrideredirect(1)
t3.geometry('200x70+150+150')
```

注意 option_readline 在每一个例子中的调用用来设置整个应用的默认的字体和颜色，在 4.2.4 小节中有说明。

Toplevel 控件的文档见附录 B。

4.1.2 框架

对于其他控件来说，框架控件是个容器（containers）。虽然你可以联系鼠标和键盘事件来同叫，除了标准控件选项外，框架有很有限的选项，没有方法。

框架的最普通的一种应用是作为几何管理器处理的一组控件的主体。这在图 4.2 中



图 4.2 框架控件

有标明，第二个框架的例子见图 4.3 所示，每行应用一个框架。

```
for relief in [RAISED, SUNKEN, FLAT, RIDGE, GROOVE, SOLID]:
  f = Frame(root, borderwidth=2, relief=relief)
  Label(f, text=relief, width=10).pack(side=LEFT)
  f.pack(side=LEFT, padx=5, pady=5)
```

与按钮和标签类似，框架的外观可选择一个浮雕类型和适当的边界宽度修改（见图 4.3）。事实上，很难讲出这些控件的区别。为此，给每个控件保留一个特殊的装饰及禁

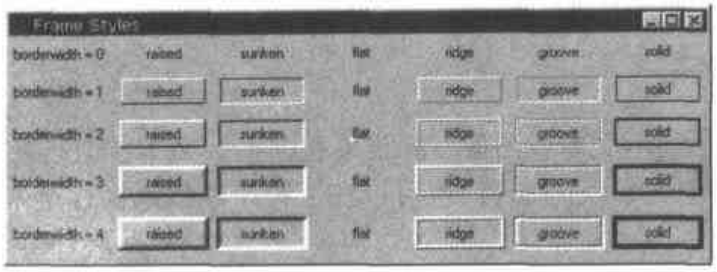


图 4.3 含有不同边界的浮雕式样的框架

止在按钮上应用标签是个好主意。例如：

```
class GUI:
  def__init__(self):
      of = [None] *5
      for bdw in range(5):
          of[bdw] = Frame(self.root. borderwidth=0)
          Label(of[bdw], text='borderwidth = %d ' % bdw).pack(side=LEFT)
          ifx = 0
          iff = []
          for relief in [RAISED, SUNKEN, FLAT, RIDGE, GROOVE, SOLID]:
          iff.append(Frame(of[bdw], borderwidth=bdw, relief=relief))
          label(iff[ifx], text=relief, width=10).pack(side=LEFT)
          iff[ifx].pack(side=LEFT, padx=7-bdw, pady=5+bdw)
          ifx = ifx+1
      of[bdw].pack()
```

GROOVE 浮雕类型的普通应用为一个或多个控件提供了一个作了标记的框架（有时叫面板）。有好几种做法，图 4.4 仅说明了一个例子：应用两个框架。注意外部的框架应用几何管理器来定位内部的框架及标签。内部框架里的控件用了压缩几何管理器（Packer geometry manager）。

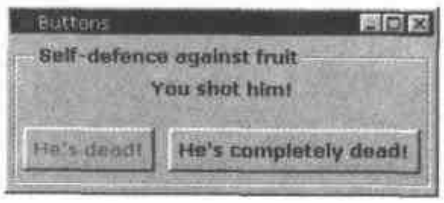


图 4.4 使用框架控件建构面板

```
f = Frame(root, width=300, height=110)
xf = Frame(f, relief=GROOVE, borderwidth=2)
Label(xf, text="You shot him! ").pack(pady=10)
Button(xf, text="He's dead! ", state=DISABLED).pack(side=LEFT, padx=5,
                                                    pady=8)
Button(xf,text="He's completely dead! ", command=root.quit).pack(side=RIGHT,
                                                    padx=5. pady=8)
xf.place(relx=0.01, rely=0.125,anchor=NW)
Label(f, text='Self-defence against fruit').place(relx=.06, rely=0.125,
                                                    anchor=W)
```

```
f.pack()
```

4.1.3 标签

标签控件被用于显示文本或图像。标签可包含多行文本，但只能用一种字体。你可以允许控件根据可用空间大小换行，也可以在字符串中加入换行符来控制换行。图 4.5 中列出了一些标签。

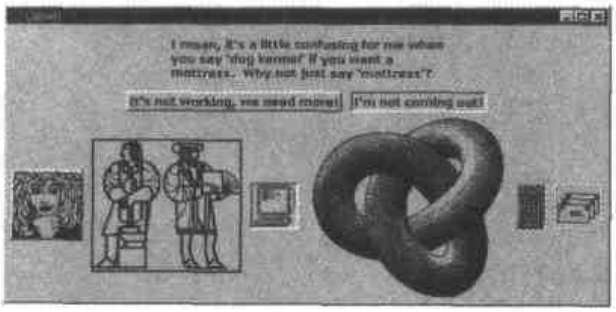


图 4.5 标签控件

虽然标签并不是与用户进行交互用的，但你可以结合鼠标和键盘事件来回叫。对一定的应用这可以作为一个“低级（cheap）”按钮用。

```
Label(root, text="I mean, it's a little confusing for me when you say"
  "'dog kennel' if you want a mattress. Why not just say 'mattress'? ",
  wraplength=300, justify=LEFT).pack(pady=10)

f1=Frame(root)
Label(f1, text="It's not working, we need more! ",
     relief=RAISED).pack(side=LEFT, padx=5)
Label(f1, text="I'm not coming out! ", relief=SUNKEN).pack(side=LEFT,
                                                          padx=5)
f1.pack()

f2=Frame(root)
for bitmap,rlf in [('woman',RAISED),('mensetmanus',SOLID),
                   ('terminal',SUNKEN),( 'escherknot',FLAT),
                   ('calculator',GROOVE),( 'letters',RIDGE)]:
  Label(f2, bitmap='@bitmaps/%s' % bitmap, relief=rlf).pack(side=LEFT,
                                                          padx=5)
f2.pack()
```

标签控件的文件管理参见附录 B 中的 Label 小节。

4.1.4 按钮

严格来说，按钮是对鼠标和键盘事件起反应的标签。当按钮被激活的时候，你就捆绑了一个程序调用或被调用的回叫。按钮可能被禁止，这样阻止用户激活按钮。按钮控件能包括文本(跨越了多重行的)或图像。按钮能存在于 Tab 群里，这意味着你能用 Tab 键给它们定位。图 4.6 给出了简单的按钮。

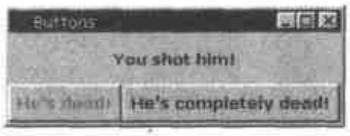


图 4.6　按钮控件

```
Label(root, text="You shot him! ").pack(pady=10)
Button(root, text="He's dead! ", state=DISABLED).pack(side=LEFT)
Button(root, text="He's completely dead! ",
       Command=root.quit).pack(side=RIGHT)
```

不是所有的图形用户界面程序员都知道浮雕选项可以用来制作不同外观的按钮。特别是平面和立体浮雕对创建工具栏非常有用，工具栏里的图标被用来传送功能信息。不过，当使用一些浮雕效果的时候，必须谨慎。例如，如果你用凹的浮雕定义按钮，当它被激活的时候，控件不具有不同的外观，因为默认行为显示了凹的浮雕按钮；必须设计可变行为，比如在按钮的范围内改变背景色、字体或用语。图 4.7 阐述了把可以利用的浮雕型和增长的边界宽度组合在一起的效果。注意，增加边界宽度对某些浮雕型来说有效（除非边界宽度≥2，否则脊（RIDGE）和沟（GROOVE）不起作用）。不过，如果边界宽度太大按钮就会变得很难看。

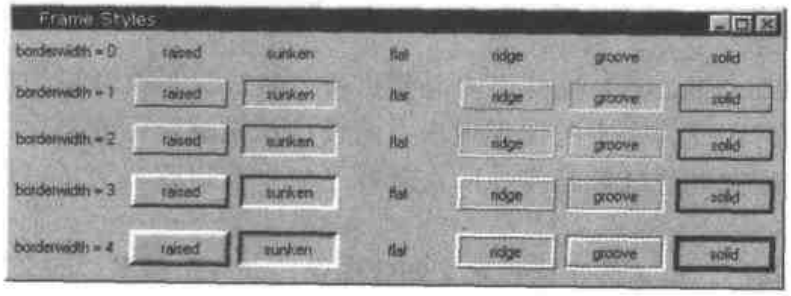


图 4.7　不同边界和不同浮雕类型

```
class GUI:
  def__init__(self):
      of = [None] *5
      for bdw in range(5):
          of[bdw] = Frame(self.root. borderwidth=0)
          Label(of[bdw], text='borderwidth = %d ' % bdw).pack(side=LEFT)
          For relief in [RAISED, SUNKEN, FLAT, RIDGE, GROOVE, SOLID]:
              Button(of[bdw], text=relief,
              borderwidth=bdw, relief=relief, width=10,
              command=lambda s=self, r=relief, b=bdw: s.prt(r,b))\
                .pack(side=LEFT, padx=7-bdw, pady=7-bdw)
          of[bdw].pack()
  def prt(self, relief, border):
       print '%s:%d' % (relief, border)
```

按钮控件更多的信息参见附录 B 中的 **Button** 小节。

4.1.5 输入

输入（Entry）控件是用来收集用户输入的基本控件。它们可能被用来显示信息，也可能被禁止，以阻止用户改变它们的值。

输入控件被限制在仅能容纳一种字体的单行文本范围内。图 4.8 给出了一个典型的输入控件。如果输入控件的正文比可用的显示范围长，控件内容可以滚动。你可以使用上下键改变可见位置。

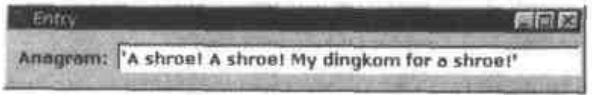


图 4.8 Entry 控件

另外，你可用控件的滚动方法把滚动行为与鼠标或你的应用程序连接起来。

输入控件的文档参见附录 B 中的 Entry 小节。

```
Label(root, text="Anagram: ").pack(side=LEFT, padx=5, pady=10)
e = StringVar()
Entry(root, width=40, textvariable=e).pack(side=LEFT)
e.set("'A shroe! A shroe! My dingkom for a shroe!'")
```

4.1.6 单选按钮

单选按钮（Radiobutton）控件可能需要重新命名！汽车收音机的机械的按钮选择器变得不寻常了，因此为未来的图形用户界面设计员做解释可能会变得困难。不过，想法是这样的：所有的选择都是排他的，选择了一个按钮就取消了已选定的其他任何按钮。

与按钮控件类似，单选按钮能显示文本或图像，并有跨越多行的文本，虽然仅为一种字体。图 4.9 显示了典型的单选按钮。

通常把一个组中的全部单选按钮和单一的变量联系起来。

```
Var = IntVar()
for text, value in [('Passion fruit', 1),( 'Loganberries', 2),
                    ('Mangoes in syrup', 3), ('Oranges', 4),
                    ('Apples', 5),( 'Grapefruit', 6)]:
  Radiobutton(root, text=text, value=value, variable=var).pack(anchor=W)
var.set(3)
```

如果指示标记被标为出错，单选按钮组表现为按钮框,如图 4.10 所示。被选择的按钮通常用凹的浮雕表示。

```
var = IntVar()
for text, value in [('Red Leicester', 1),('Tilsit', 2),('Caerphilly', 3),
                    ('Stilton', 4), ('Emental', 5),
                    ('Roquefort', 6), ('Brie', 7)]:
  Radiobutton(root, text=text, value=value, variable=var,
              indicatoron=0).ack(anchor=W, fill=X, ipadx=18)
var.set(3)
```

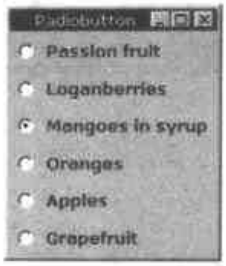


图 4.9　单选按钮控件



图 4.10　单选按钮：indication=0

4.1.7　复选按钮

复选按钮（Checkbutton）控件被用来为一个以上的项目提供开关选择。不同于单选按钮（参见第 4.1.6 节“单选按钮”），复选按钮之间并没有相互作用。你可以把任何一个文本或图像装到复选按钮里。复选按钮通常应该把变量（IntVar）分配给允许你决定复选按钮的状态变化的任意选择。另外（或同时），你可以给按钮捆绑一个回叫，当按钮被按时回叫将被调用。

注意复选按钮的外观在 Unix 与 Windows 上是很不同的；通常， Unix 通过使用填充颜色来表示选择，而 Windows 使用复选标记。图 4.11 显示了 Windows 表格。

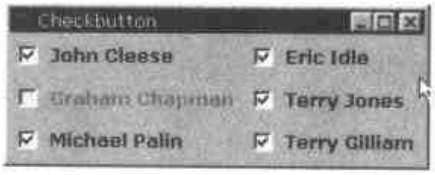


图 4.11　复选按钮控件

```
for castmember, row, col, status in [
  ('John Cleese', 0,0,NORMAL), ('Eric Idle', 0,1,NORMAL),
  ('Graham Chapman', 1,0,DISABLED), ('Terry Jones', 1,1,NORMAL),
  ('Michael Palin',2,0,NORMAL), ('Terry Gilliam', 2,1,NORMAL)]:
      setattr(var, castmember, IntVar())
      Checkbutton(root, text=castmember, state=status, anchor=W,
     Variable = getattr(var, castmember)).grid(row=row, col=col, sticky=W)
```

复选按钮控件的文档参见附录 B 中的 Checkbutton 小节。

4.1.8　主菜单

主菜单控件提供了个常见的方法，允许用户在应用的范围内选择操作。主菜单对结构来说相当麻烦，尤其是层叠超过几级（试着设计到达任何功能不超过二级的主菜单是最好的）。

Tkinter 使主菜单设计具有灵活性，允许多种的字体、图像和位图，及复选按钮和单选按钮。在几个方案中建造主菜单是可能的。图 4.12 中所示的例子是建造主菜单的一个方法；你将找到一个候补方案来建造和 altmenu.py 同样的在线主菜单。



图 4.12 主菜单控件

图 4.13 描述添加按钮命令到主菜单上。



图 4.13 主菜单：按钮命令

```
mBar = Frame(root, relief=RAISED, borderwidth=2)
mBar.pack(fill=x)
CmdBtn = makeCommandMenu()
CasBtn = makeCascadeMenu()
ChkBtn = makeCheckbuttonMenu()
RadBtn = makeRadiobuttonMenu()
NoMenu = makeDisabledMenu()
MBar.tk_menuBar(CmdBtn, CasBtn, ChkBtn, RadBtn, NoMenu)
Def makCommandMenu():
  CmdBtn = Menubutton(mBar, text='Button Commands', underline=0)
  CmdBtn.pack(side=LEFT, padx="2m")
  CmdBtn.menu = Menu(CmdBtn)

  CmdBtn.menu.add_command(label="Undo")
  CmdBtn.menu.entryconfig(0, state=DISABLED)

  CmdBtn.menu.add_command(label='New…',underline=0,command=new_file)
  CmdBtn.menu.add_command(label='Open…',underline=0,command=open_file)
  CmdBtn.menu.add_command(label='Wild Font',underline=0,
            font=('Tempus Sans ITC', 14), command=stub_action)
  CmdBtn.menu.add_command(bitmap="@bitmaps/RotateLeft")
  CmdBtn.menu.add('separator')
  CmdBtn.menu.add_command(label='Quit', underline=0,
            background='white', activebackground='green',
            command=CmdBtn.quit)

  CmdBtn['menu'] = CmdBtn.menu
  return CmdBtn
```

图 4.14 显示了层叠菜单输入的外观。

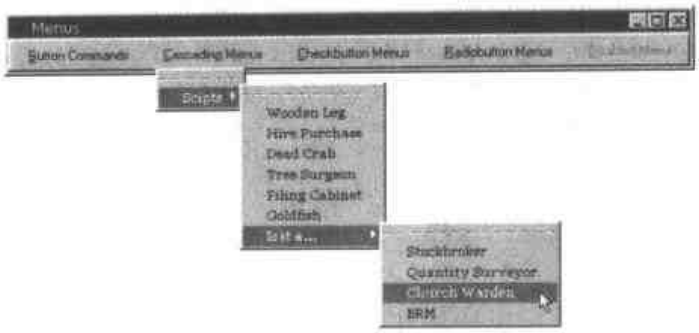


图 4.14　菜单：层叠菜单

```
def makeCascadeMenu():
  CasBtn = Menubutton(mBar, text='Cascading Menus', underline=0)
  CasBtn.pack(side=LEFT, padx="2m")
  CasBtn.menu = Menu(CasBtn)

  CasBtn.menu.choices = Menu(CasBtn.menu)
  CasBtn.menu.choices.wieerdones - Menu(CasBtn.menu.choices)

  CasBtn.menu.choices.wierdones.add_command(label='Stockbroker')
  CasBtn.menu.choices.wierdones.add_command(label='Quantity Surveyor')
  CasBtn.menu.choices.wierdones.add_command(label='Church Warden')
  CasBtn.menu.choices.wierdones.add_command(label='BRM')
  CasBtn.menu.choices.add_command(label='Wooden Leg')
  CasBtn.menu.choices.add_command(label='Hire Purchase')
  CasBtn.menu.choices.add_command(label='Dead Crab')
  CasBtn.menu.choices.add_command(label='Tree Surgeon')
  CasBtn.menu.choices.add_command(label'Filing Cabinet')
  CasBtn.menu.choices.add_command(label='Goldfish')
  CasBtn.menu.choices.add_cascade(label=Is it a…',
  Menu=CasBtn.menu.choices.wierdones)
  CasBtn.menu.add_cascade(label='Scipts', menu=CasBtn.menu.choices)
  CasBtn['menu'] = CasBtnn.menu
  Return CasBtn
```

如图 4.15 所示，复选按钮可能在主菜单的范围内被使用。



图 4.15　菜单：复选按钮

```
def makeCheckbuttonMenu():
  ChkBtn = Menubutton(mBar, text='Checkbutton Menus', underline=0)
  ChkBtn.pack(side=LEFT, padx='2m')
  ChkBtn.menu = Menu(ChkBtn)

  ChkBtn.menu.add_checkbutton(label='Doug')
```

```
  ChkBtn.menu.add_checkbutton(label='Dinsdale')
  ChkBtn.menu.add_checkbutton(label="Stig O' Tracy")
  ChkBtn.menu.add_checkbutton(label='Vince')
  ChkBtn.menu.add_checkbutton(label='Gloria Pules')

  ChkBtn.menu.invoke(ChkBtn.menu.index('Dinsdale'))
  ChkBtn['menu'] = ChkBtn.menu
  Return ChkBtn
```

另一个办法是在菜单中使用单选按钮，见图 4.16。

```
def makeRadiobuttonMenu():
  RadBtn = Menubutton(mBar, text='Radiobutton Menus', underline=0)
  RadBtn.pack(side=LEFT, padx='2m')
  RadBtn.menu = Menu(RadBtn)

  RadBtn.menu.add_radiobutton(label='metonymy')
  RadBtn.menu.add_radiobutton(label='zeugmatists')
  RadBtn.menu.add-radiobutton(label='synechdotists')
  RadBtn.menu.add-Radiobutton(label='axiomists')
  RadBtn.menu.add_radiobutton(label='anagogists')
  RadBtn.menu.add_radiobutton(label='catachresis')
  RadBtn.menu.add_radiobutton(label='periphrastic')
  RadBtn.menu.add_radiobutton(label='litotes')
  RadBtn.menu.add_radiobutton(label='circumlocutors')

  RadBtn['menu'] = RadBtn.menu
  return RadBtn

def makeDisabledMenu():
  Dummy_button = Menubutton(mBar, text='Disabled Menu', underline=0)
  Dummy_button.pack(side=LEFT, padx='2m')
  Return Dummy_button
```

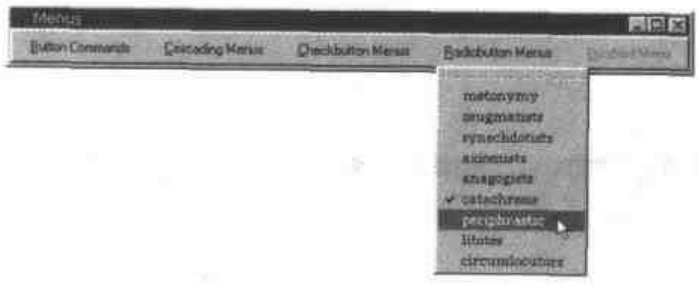


图 4.16　菜单：主菜单：单选按钮

主菜单控件的文档参见附录 B 的 Menu 小节。
主按钮控件的文档参见附录 B 的 Menubutton 小节。
选项菜单类的文档参见附录 B 的 OptionMenu class 小节。

4.1.9　消息

消息控件提供了显示多行文本的方便的方法。你能为整条消息使用一种字体和一种

前景/背景色。运用这个控件的例子见图 4.17。

该控件有标准的控件方法。

```
Message(root, text="Exactly. It's my belief that these sheep are laborin'"
   "under the misapprehension that they're birds. Observe their"
   "be 'avior. Take for a start the sheeps' tendency to 'op about"
   "the field on their 'ind legs. Now witness their attempts to"
   "fly from tree to tree. Notice that they do not so much fly"
   "as…plummet.", bg='royalblue', fg='ivory',
   relief=GROOVE).pack(padx=10, pady=10)
```

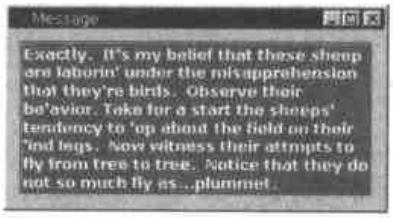


图 4.17 消息控件

消息控件的文档参见附录 B 的 Message 小节。

4.1.10 文本

文本控件是多用途的控件。其主要目的当然是显示文本，但是，它能作多种风格和字体，嵌入图像和窗体，使事件的捆绑只限于局部。

文本控件可能被用作简单的编辑器，在这种情况下，定义多重的标签和记号使实现简单。该控件很复杂，有很多的选项和方法，所以，详细细节请查阅完整的文档。图 4.18 给出了一些可能的风格和内嵌的对象。



图 4.18 含有多个内嵌对象的文本控件

```
text = Text(root, height=26, width=50)
scroll = Scrollbar(root, command=text.yview)
text.configure(yscrollcommand=scroll.set)

text.tag_configure('bold_italics',font=('Verdana',12,'bold','italic'))
text.tag_configure('big', font=('Verdana',24,'bold'))
text.tag_configure('color',foreground='blue',font=('Tempus Sans ITC',14))
```

```
text.tag_configure('groove', relief=GROOVE, borderwidth=2)

text.tag_bind('bite', '<1>',
        lambda e,t=text:t.insert(END,"I'll bite your legs off!"))

text.insert(END, 'Something up with my banter, chaps?\n')
text.insert(END, 'Four hours to bury a cat?\n', 'bold_italics')
text.insert(END, 'Can I call you "Frank"?\n', 'big')
text.insert(END, "What's happening Thursday then?\n", 'color')
text.insert(END, 'Did you write this symphony in the shed?\n', 'groove')

button = Button(text, text='I do live at 46 Horton terrace')
text.window_create(END, window=button)

photo=PhotoImage(file='lumber.gif')
text.image_create(END, image=photo)

text.insert(END, 'I dare you to click on this\n', 'bite')
text.pack(side=LEFT)
scroll.pack(side=RIGHT, fill=Y)
```

文本控件的文档参见附录 B 的 Text 小节。

4.1.11 画布

画布(canvas)是多用途控件。你不仅仅能用它们来画复杂的对象，如线、椭圆、多边形和长方形——而且你还能在画布上精确地放置图像和位图。除了这些特点，你能在画布的范围内放置任何控件（比如按钮、列表框和其他的控件），将它们与鼠标或键盘行为联系起来。

你在本书中可看到很多的例子，画布控件被用来为应用的多样性提供自由式样的容器。图 4.19 所示的例子是对阐述大部分可以利用的功能的粗糙尝试。

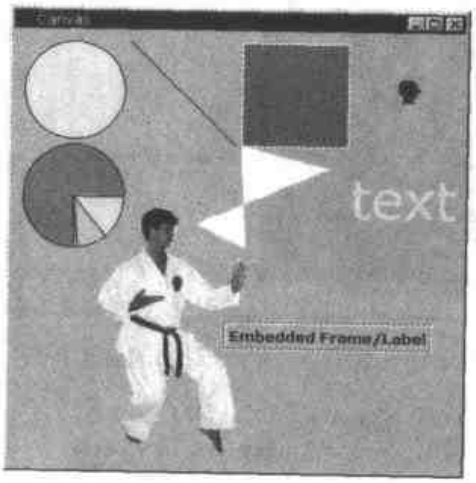


图 4.19 画布控件

```
canvas = Canvas(root, width=400, height=400)
canvas.create_oval(10,10,100,100, fill='gray90')
```

```
canvas.create_line(105,10,200,105, stipple='@bitmaps/gray3')
canvas.create_rectangle(205,10,300,105, outline='white', fill='gray50')
canvas.create_bitmap(355,53, bitmap='questhead')

xy = 10, 105, 100, 200
canvas.create_arc(xy, start=0, extent=270, fill='gray60')

canvas.create_arc(xy, start=270,extent=5, fill='gray70')
canvas.create_arc(xy, start=275, extent=35, fill='gray80')
canvas.create_arc(xy, start=310, extent=49, fill='gray90')

canvas.create_polygon(205,105,285,125,166,177,210,199,205,105,fill='white')
canvas.create_text(350,150,text='text',fill='yellow',font=('verdana',36))

img = PhotoImage(file='img52.gif')
canvas.create_image(145,280,image=img,anchor=CENTER)

frm = Frame(canvas, relief=GROOVE, borderwidth=2)
Label(frm, text="Embedded Frame/Label").pack()
Canvas.create_window(285, 280, window=frm, anchor=CENTER)
Canvas.pack()
```

画布控件的一个特性，它既能起作用，又会碍事，是对象被画到已经在画布上任何对象的顶端。如果必要，稍后你能改变画布项目的顺序。

画布控件、位图类的文档及照片图像（PhotoImage）类的文档参见附录 B。

4.1.12 滚动条

滚动条控件能被加到任何支持滚动，比如文本、画布和列表框组件的控件上。

把滚动条控件和另一个控件联系起来就如同给每个控件增加一个同叫、安排它们一起被显示一样简单。当然，没有必要使它们结合在一起，但是，如果你没有的话，你可以用一些特殊的图形用户界面来结束。图 4.20 显示了一个典型的应用程序。

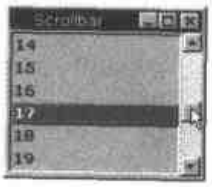


图 4.20 滚动条控件

```
list = Listbox(root, height=6, width=15)
scroll = Scrollbar(root, command=list.yview)
list.configure(yscrollcommand=scroll.set)
list.pack(side=LEFT)
scroll.pack(side=RIGHT, fill=Y)
for item in range(30):
  list.insert(END,item)
```

滚动条控件的文档参见附录 B 的 Scrollbar 小节。

4.1.13 列表框

列表框控件显示了一个也许会被用户选择的值的一览表。控件的默认行为允许用户在一览表中选择单一的项目。图 4.21 显示了一个简单的例子。你可以增加更多的捆绑，并用控件的挑选方法来允许多重的项目和其他的特性。

看上面的"滚动条"，列表框为有关消息增加了滚动能力。

```
list = Listbox(root, width=15)
list.pack()
for item in range(10):
  list.insert(END, item)
```

列表框控件的文档参见附录 B 的 Listbox 小节。

4.1.14 标尺

标尺控件允许你设置可在较低和较高的值之间选择的线性值，并且，它用图形方法来表示当前值。随意的数值可能会被显示。

标尺控件有一些控制其外观和行为的选项；另外，它是相当简单的控件。

下列例子——图 4.22 所示——是支持改编 Tcl/Tk 发送的一个示范。同样的，看一看怎样向 Tkinter 转换可能对 Tcl/Tk 程序员有用。

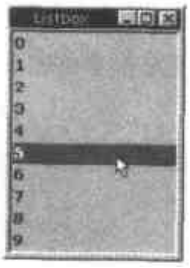


图 4.21 列表框控件

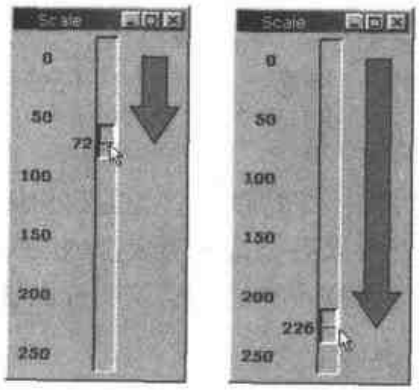


图 4.22 标尺控件：应用

```
def setHeight(canvas, heightStr):
  height = string.atoi(heightStr)
  height = height + 21
  y2 = height - 30
  if y2 < 21:
     y2 = 21
  canvas.cords('poly',
     15,20,35,20,35,y2,45,y2,25,height,5,y2,15,y2,15,20)
  canvas.cords('line',
     15,20,35,20,35,y2,45,y2,25,height,5,y2,15,y2,15,20)

  canvas = Canvas(root,width=50,height=50,bd=0,highlightthickness=0)
  canvas.create_polygon(0,0,1,1,2,2,fill='cadetblue',tags='poly')
  canvas.create_line(0,0,1,1,2,2,0,0,fill='black',tags='line')

  scale = Scale(root,orient=VERTICAL,length=284,from_=0,to=250,
       tickinterval=50,command=lambda h,c=canvas:setHeight(c,h))
  scale.grid(row=0,column=0,sticky='NE')
  canvas.grid(row=0,column=1,sticky='NWSE')
```

标尺控件文档参见附录 B 的 Scale 小节。

4.2 字体和颜色

如同它们应用于 Tkinter 那样，这一部分的目的是向读者显示对字体和颜色的概述。这将提供足够的上下文关系来理解贯穿全文的例子。

4.2.1 字体描述符

我们中的那些用 X 视窗工作的人已经习惯于 X 视窗字体描述符那笨拙但明确的格式了。幸运的是，有了 Tk 的 8.0 和 8.0 以上的版本，就有了这样的解决方案： Tk 定义字体描述符。字体描述符是独立于体系结构的。它们允许程序员创建一个包含族（family）、点长（pointsize）和包含可选择的风格字符串的元组来选择字体。以下是例子：

```
('Arial', 12, 'italic')
('Helvetica', 10)
('Verdana', 8, 'medium')
```

如果字体族不包括内嵌的空间，你可以像单个字符串那样传递描述符：

```
'Verdana 8 bold italic'
```

4.2.2 X 视窗系统字体描述符

当然，如果你真想使用它们，可以用旧版的字体描述符。大部分的 X 视窗字体在表格中有 14 字段的名字：

```
-foundry-family-weight-slant-setwidth-style-pixelSize-pointSize-
 Xresolution-Yresolution-spacing-averageWidth-registry-encoding
```

通常我们仅仅关注一小部分字段：

```
-*-family-weight-slant-*-*-*-pointSize-*-*-*-*-registry-encoding
```

字段定义如下：

- 族（family） 一个识别例如 arial 等基本排版风格的字符串。
- 粗度 根据清样的判断（例如：medium、粗体等）来识别黑色字体的字符串。
- 斜 用于字样设计的字体的全部形状的一个代码字符串— roman(R)，italic(I)或 oblique(0)。
- 大小（pointsize） 无符号的整数-字符串字体，文本的与设备无关的单位米制下的大小。
- 编码 识别代码特征的注册名，该特征为指定注册设置。

X 字体描述符的例子可能为：

```
'-*-verdana-medium-r-*-*-8-*-*-*-*-*-*-*'
```

这里描述了 8 点的 Verdana 字体，适中的粗度（weight）和罗马体（直立）。尽管描述符有点难看，大部分的程序员很快地习惯了这种格式。用 X-服务器，如果特定的点长在字体中不可用，则全部字体换算都不平滑；不幸的是，获得最佳屏幕显示字体和字号的正确结合是个反复试验摸索的过程。

4.2.3 颜色

Tkinter 允许你使用 X-服务器定义的色彩名字。这些名字非常华丽，但不总能充分地描述颜色，如 LavenderBlushl、LemonCiffon、LightSalmon、MediumOrchid3 和 OldLace 仅是一小部分。普通的名字类似红、黄、蓝和黑可能也被用到。这些名字和对应的三原色值被维持在 Tk 的包含文件内，所以这些名字可能被用在任何可移植的 Tkinter 平台上*。

通常使用彩色字符串精确地定义颜色更简单：

# 三原色	适用于 4 位（bit）的值(每种颜色 16 层)
#RRGGBB	适用于 8 位（bit）的值(每种颜色 256 层)
#RRRRGGGGBBBB	适用于 16 位（bit）的值(每种颜色 66526 层)

这里是如何为应用（不完全代码）设置部分色彩定义表的例子：

```
# These are the color schemes for xxx and yyy front panels
#        Panel   LED off  ON          Active    Warning
COLORS = [('#545454','#656565','LawnGreen','ForestGreen','DarkOrange',\
#          Alarm    Display   Inside   Chrome  Insidep   Chassis
           '#ff342f','#747474','#343434','#efefef','#444444','#a0a0a0',\
#       DkChassis LtChassis VdkChassis VltChassis Bronze
        '#767600','#848400','#6c6c00','#909000','#7e5b41'),
    etc.
```

4.2.4 设置宽应用默认字体和颜色

当设计应用的时候，你会发现系统提供的默认颜色、字体和字号并不适合你所想的特别的版面设计。这时你必须确切地设置它们的值。值能放入代码（你将在本书中看到几个这样做的例子）。不过，这妨碍了最终用户或系统管理员对应用进行裁剪，以适合其特别要求或商业标准。在这种情况下值应该用外部的选择数据库来安装，对 X 视窗程序员来说这和通常使用 X 默认文件设计加工的资源数据库相等。实际上，Tk 选项数据库的格式确切地说像 X 默认文件：

```
*font:                  Verdana 10
*Label*font:            Verdana 10 bold
*background:            Gray80
*Entry*background:      white
*foreground:            black
*Listbox*foreground:    RoyalBlue
```

这些输入的目的为了给除了 Verdana 10 标签（规则粗度）和 Verdana 10 粗体标签之外的所有的控件设置字体。同样地我们给背景和前景设置了默认色，更改了输入背景和前景列表框。如果我们将这些输入放到被叫做 optionDB 的文件中，我们就能应用 option_readline 调用的值：

---

* X 视窗颜色名在标准 X11 版里提供，但是不在 X11 协议或 Xlib 中指定。允许 X 服务器销售商改变名字，或者改变它们的解释程序。在较少情况下你会发现相同的颜色名，在 Tkinter 和 X 视窗应用中用于显示不同的颜色。

```
root = Tk()
root.option_readfile('optionDB')
```

这个调用应该早些在代码中使用，以确保全部的控件都如同原打算的那样被制作。

4.3 Pmw 大控件漫游

Python 大控件——Pmw——是合成的控件，以 Tkinter 控件为基类，是完全在 Python 内写的。它们可以很方便地增加功能性的应用，而不必写一堆代码。特别是，组合框和内部确认计划的输入字段放在一起是个很有用的控件。

在如上所示的类似的 Tkinter 漫游方式中，下面显示了典型的 Pmw 控件的外观和用法。代码尽可能得简短，并且它阐述了一些能用于控件的可用选项。如果你需要查阅特殊的方法或选项，可以参考附录 C。每个控件在附录中也都有相应的部分。

Pmw 有大量 HTML 格式文档，因而这一章将不在这里重复了。另外，有一个关于所有在 Pmw 分布演示程序库里的全部控件的代码例子。大部分被显示的例子是来自于那个代码的简化。

除第一个例子之外，代码例已经除去了必要的输入和初始化 Tkinter 的样板文件的代码。对于共享代码段，我们用粗体标记出来，以后我们就不再写了。所有显示的完整的源代码都是可以在线看到。

4.3.1 关于框

关于框(AboutDialog)控件提供了一个方便的对话模式来显示版本、著作权和开发者信息。使用少数的数据项，我们就能用最少的代码来显示出关于框。图 4.23 显示了典型的关于框。

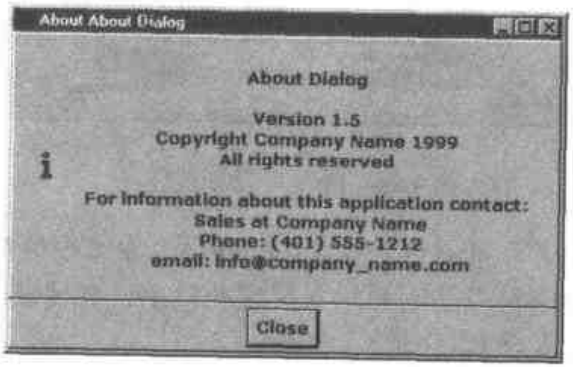


图 4.23 Pmw 关于框控件

```
from Tkinter import *
import Pmw
root = Tk()
root.option_readfile('optionDB')
Pmw.initialize()
```

```
Pmw.aboutversion('1.5')
Pmw.aboutcopyright('Copyright Company Name 1999\nAll rights reserved')
Pmw.aboutcontact(
    'For information about this application contact:\n' +
    '  Sales at Company Name\n' +
    '  Phone@401) 555-1212\n' +
    '  email:info@company_name.com'
    )
about = Pmw.AboutDialog(root, applicationname='About Dialog')

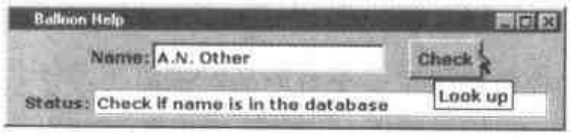
root.mainloop()
```

这个控件用在 AppShell 类中，该类在 8.2 节的“标准应用构架”中被使用，并且，稍后它将在本书的几个例子中被用到。

About Dialog 控件文档参见附录 C。

4.3.2 浮动图

浮动图控件实现了现在稍微常见的浮动帮助主题（有时被叫做工具技巧提示）。当光标在屏幕的控件之上时，通常短暂的停顿后控件的帮助信息就显现出来了。追加（或者替代）信息可能显示在屏幕上的状态区域。这个区域的信息在短暂的停留后就被移去。这在图 4.24 中有说明。



……几秒钟后

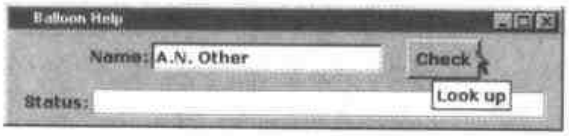


图 4.24 Pmw 浮动图控件

尽管浮动帮助对初学者非常有用，但是它也许会令专家很烦恼。如果提供浮动帮助，请确保提供了开关选项关闭对浮动图和状态区域的输出，并使其永久有效，以便用户不必每次使用该应用时都要关闭这一功能。

```
balloon = Pmw.Ballon(root)
frame = Frame(root)
frame.pack(padx = 10, pady = 5)
field = Pmw.EntryField(frame, labelpos=W, label_text='Name: ')
field.setentry('A.N. other')
field.pack(side=LEFT, padx = 10)

balloon.bind(field, 'Your name', 'Enter your name')
check = Button(frame, text='Check')
```

```
check.pack(side=LEFT, padx=10)
balloon.bind(check, 'Look up', 'Check if name is in the database')
frame.pack()

messageBar = Pmw.MessageBar(root, entry_width=40,
                            entry_relief=GROOVE,
                            labelpos=W, label_text='Status: ')
messageBar.pack(fill=X, expand=1, padx=10, pady=5)

balloon.configure(statuscommand = messageBar.helpmessage)
```

浮动图控件的文档参见附录 C 的 Balloon 小节。

4.3.3 按钮框

按钮框控件提供了实现大量按钮的方便方法，它通常被用来在应用的范围内提供指令区。框既可以水平也可以垂直地设置，并且可被定义为默认按钮。图 4.25 给出了个简单的按钮框。

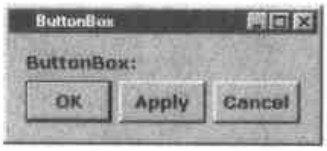


图 4.25 Pmw 按钮框控件

```
def buttonPress(btn):
   print 'The "%s" button was pressed' % btn
def defaultKey(event):
   buttonBox.invoke()

buttonBox = Pmw.ButtonBox(root, labelpos='nw', label_text='ButtonBox: ')
buttonBox.pack(fill=BOTH, expand=1, padx=10, pady=10)

buttonBox.add('OK',     command = lambda b='ok':     buttonPress(b))
buttonBox.add('Apply', command = lambda b='apply':  buttonPress(b))
butonBox.add('Cancel', command = lambda b='cancel': buttonPress(b))

buttonBox.setdefault('OK')
root.bind('<Return>', defaultKey)
root.focus_set()
buttonBox.alignbuttons()
```

按钮框控件的文档参见附录 C 的 ButtonBox 小节。

4.3.4 组合框

组合框控件是重要的控件，最初被用在 Macintosh 和 Windows 平台上，后来用在 Motif 上。和选项菜单不同，它可以滚动以提供大量的选项，允许用户从选项表中进行选择。如图 4.26 左所示的例子或图 4.26 右所示的下拉一览表，可以被永久显示。使用下

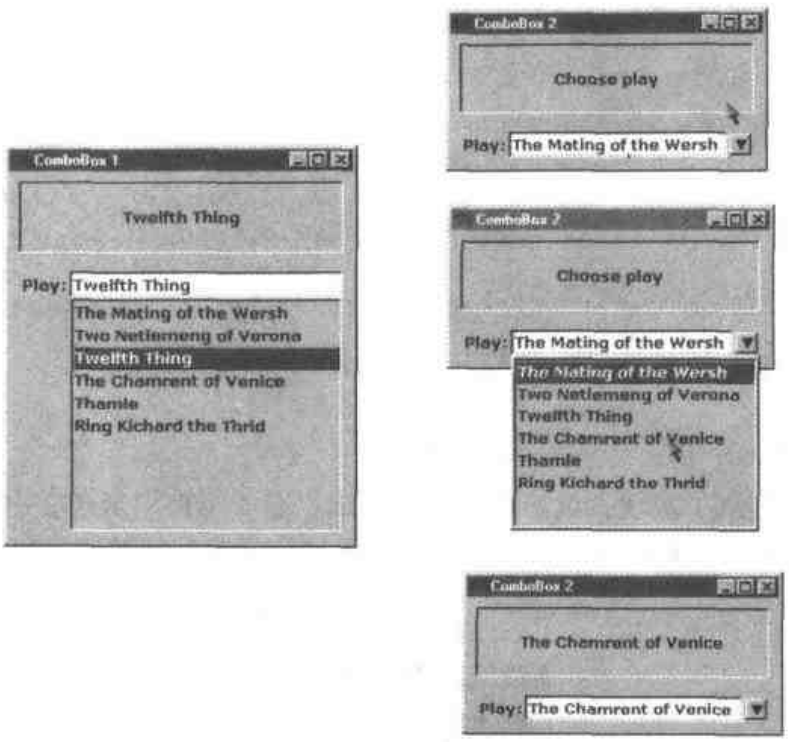


图 4.26　Pmw 组合框控件

拉表使得图形用户界面只需较少的空间就能实现复杂的界面。

```
choice = None
def choseEntry(entry):
   print 'You chose "%s"' % entry
   choice.configure(text=entry)

asply = ("The Mating of the Wersh","Two Netlemeng of Verona","Twelfth
Thing","The Chamrent of Venice","Thamle","Ring Kichard the Thrid")

choice = Label(root,text='Choose play',relief='sunken',padx=20,pady=20)
choice.pack(expand=1,fill='both',padx=8,pady=8)

combobox = Pmw.ComboBox(root,label_text='Play: 'labelpos='wn',
listbox_width=24,dropdown=0,
selectioncommand=choseEntry,
scrolledlist_items=asply)
combobox.pack(fill=BOTH,expand=1,padx=8,pady=8)
combobox.selectitem(asply[0])

# ===========
combobox = Pmw.ComboBox(root,label_text='Play: ',labelpos='wn',
listbox_width=24,dropdown=1,
...
```

组合框控件参见附录 C 的 **ComboBox** 小节。

4.3.5 组合对话框

组合对话框控件提供了方便的对话框，并且允许用户从组合框中选择一项来响应问题。除了允许用户在输入字段控件中，或在固定的显示列表中，或在下拉列表中输入一个值外，它与选项对话控件相似。图 4.27 显示了一个例子。

```
choice = None
def choseEntry(entry):
   print 'You chose "%s"' % entry
   choice.configure(text=entry)

plays = ("The Taming of the Shrew","Two Gentelmen of Verona","Twelfth
Night","The Merchant of Venice","Hamlet","King Richard the Third")

dialog = Pmw.ComboBoxDialog(root,title = 'ComboBoxDialog',
       buttons=('OK', 'Cancel'),defaultbutton='OK',
       combobox_labelpos=N,label_text='Which play? ',
       scrolledlist_items=plays,listbox_width=22)
dialog.tkraise()

result = dialog.activate()
print 'You clicked on',result,dialog.get()
```



图 4.27 Pmw 组合对话框控件

组合对话框控件的文档参见附录 C 的 ComboBoxDialog 小节。

4.3.6 计数器

计数器控件是个通用控件，它允许用户在可用值的顺序下循环。Pmw 提供整数、实数、时间和日期计数。并且，定义你自己的函数来增加或者减少显示的值是可能的。与增加计数的结果所显示的那样，没有对值加以限制，所以计数器没有理由不显示“eine，zwei，drei”或任何适合于应用的序列。图 4.28 显示了一些例子。

```
def execute(self):
   print 'Return pressed,value is',date.get()

   date = Pmw.Counter(root,labelpos=W,
```

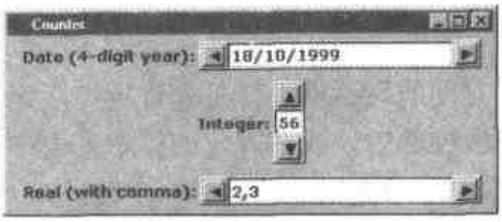


图 4.28 Pmw 计数器控件

```
        label_text='Date(4-digit year): ',
        entryfield_value=time.strftime('%d/%m/%Y',
        time.localtime(time.time())),
        entryfield_command=execute,
        entryfield_validate={'validator': 'date', 'format': 'dmy'},
        datatype = {'counter': 'date', 'format': 'dmy', 'yyyy':1})

real = Pmw.Counter(root,labelpos=W,
     label_text='Real (with comma): ',
     entryfield_value='1,5',
     datatype={'counter': 'real', 'separator': ', '},
     entryfield_validate={'validator': 'real',
         'min': '-2,0', 'max': '5,0',
         'separator': ', '},
     increment= .1)
int = Pmw.Counter(root, labelpos=W,
     label_text='Integer: ',
     orient=VERTICAL,
     entry_width=2,
     entryfield_value=50,
     entryfield_validate={'validator': 'integer',
         'min':0, 'max':99})

counters = (date, real)
Pmw.alignlabels(counters)
for counter in counters:
   counter.pack(fill=X,expand=1,padx=10,pady=5)
   int.pack(padx=10,pady=5)
```

计数器控件的文档参见附录 C 的 Counter 小节。

4.3.7 计数对话框控件

对话框控件提供了一个方便的对话框，要求用户从计数控件中挑选一个值。计数器可以包含控件能够做循环的任何数据类型，例如图 4.29 所示的这个不太像样的序列。

```
choice = None
dialog = Pmw.CounterDialog(root,
label_text='Enter the number of twits (2 to 8)\n',
counter_labelpos=N,entryfield_value=2,
counter_datatype='numeric',
entryfield_validate={'validator': 'numeric', 'min':2, 'max':8},
```

```
buttons=('OK', 'Cancel'),defaultbutton='OK',
title='Twit of the Year')
dialog.tkraise()

result = dialog.activate()
print 'You clicked on',result,dialog.get()
```

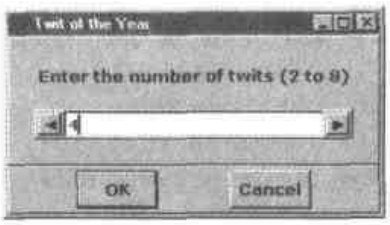


图 4.29 Pmw 计数对话框控件

计数对话框控件的文档参见附录 C 的 CounterDialog 小节。

4.3.8 对话框

对话框控件提供了制作包括按钮框和子站点区的顶层的简单方法。你可以用你的应用所需的任何东西来组装子站点。图 4.30 为一个对话框例子。



图 4.30 Pmw 对话框控件

```
dialog = Pmw.Dialog(root,buttons=('OK', 'Apply', 'Cancel', 'Help'),
       defaultbutton='OK',title='Simple dialog')
w = Label(dialog.interior(),text='Pmw Dialog\nBring out your dead! ',
      background='black',foreground='white',pady=20)
w.pack(expand=1,fill=BOTH,padx=4,pady=4)
dialog.activate()
```

关于对话框控件的文档参见附录 C 的 Dialog 小节。

4.3.9 输入域

输入域控件是带相关确认方法的输入控件。内置的检查过程提供了对整数、十六进制数、字母、字母数字、实数、日期与时间格式的检查。可以放在确认过程中的控制包括检查与所选数据格式是否相符，以及检查输入数据是否在最大和最小限制值之间。另外，你可以定义自己的检查器。如图 4.31 例子所示。

```
noval = Pmw.EntryField(root,labelpos=w,label_text='No validation',
     validate = None)
real  = Pmw.EntryField(root,labelpos=w,value = '98.4',
     label_text = 'Real (96.0 to 107.0): ',
```

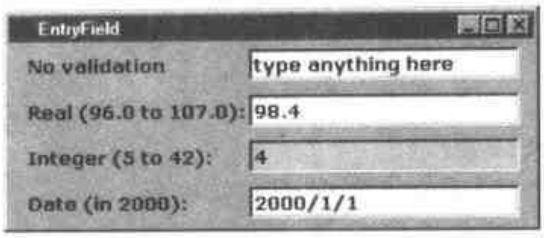


图 4.31 Pmw 输入域控件

```
        validate = {'validator': 'real',
            'min':96, 'max':107, 'minstrict':0})
int   = Pmw.EntryField(root,labelpos=w,label_text = 'Integer(5 to 42): ',
        validate = {'validator':42, 'minstrict':0),
        value = '12')
date = Pmw.EntryField(root,labelpos=w,label_text = 'Date (in 2000): ',
         value = '2000/1/1',validate = {'validator': 'date',
              'min': '2000/1/1', 'max': '2000/12/31',
              'minstrict':0, 'maxstrict':0,
              'format': 'ymd'})

widgets = (noval,real,int,date)
for widget in widgets:
  widget.pack(fill=X,expand=1,padx=10,pady=5)
Pmw.alignlabels(widgets)

real.component('entry').focus_set()
```

关于输入字段控件的文档参见附录 C 的 EntryField 小节。

4.3.10 组

组（group）控件提供了在一群控件周围放置贴了标签的框架的方便方法。标签可以是任何合理的控件，例如 Label，当然，它也可以是输入域、单选按钮或者复选按钮，这得根据应用程序的要求来决定。另外，也可以把控件当作一个没有标签的图形框来使用。图 4.32 所示的就是这样的一些例子。



图 4.32 Pmw 组控件

```
w = Pmw.Group(root,tag_text='place label here')
w.pack(fill=BOTH,expand=1,padx=6,pady=6)
cw = Label(w.interior(),text='A group with a\nsimple Label tag')
cw.pack(padx=2,pady=2,expand=1,fill=BOTH)
```

```
w = Pmw.Group(root,tag_pyclass=None)
w.pack(fill=BOTH,expand=1,padx=6,pady=6)
cw = Label(w.interior(),text='A group\nwithout a tag')
cw.pack(padx=2,pady=2,expand=1,fill=BOTH)

w = Pmw.Group(root,tag_pyclass=Checkbutton,
              tag_text='checkbutton',tag_foreground='blue')
w.pack(fill=BOTH,expand=1,padx=6,pady=6)
cw = Frame(w.interior(),width=150,height=20)
cw.pack(padx=2,pady=2,expand=1,fill=BOTH)
```

关于组控件的文档参见附录 C 的 Group 小节。

4.3.11 标签控件

标签控件是个方便的容器，它能给一个控件或一些控件贴标签。选项用来控制标签的安排和控制图形边界的外观。子站点能用任何控件的结合来组装。图 4.33 中所示的例子运用了作为框架的控件，比起运用单独的部分，它所需要的代码更少。

```
frame = Frame(root,background = 'gray80')
frame.pack(fill=BOTH,expand=1)

lw = Pmw.LabeledWidget(frame,labelpos='n',
       label_text='Sunset on Cat Island')
lw.component('hull').configure(relief=SUNKEN,borderwidth=3)
lw.pack(padx=10,pady=10)

img = PhotoImage(file='chairs.gif')
cw = Button(lw.interior(),background='yellow',image=img)
cw = Button(lw.interior(),background='yellow',image=img)
cw.pack(padx=10,pady=10,dxpand=1,fill=BOTH)
```



图 4.33 Pmw 标签控件

关于标签控件的文档参见附录 C 的 LabeledWidget 小节。

4.3.12 菜单条

菜单条控件是个管理控件。它提供了把菜单按钮和菜单加到菜单栏上，把菜单项目加到菜单上的方法。一个重要的便利之处是非常简单地把浮动帮助加到菜单和菜单项目上。几乎所有 Tkinter Menu 控件（参见 4.1.8 的“菜单”小节）的菜单项可通过 Pmw 菜单条来处理。图 4.34 给出一个与图 4.13 类似的 Tkinter 控件菜单。

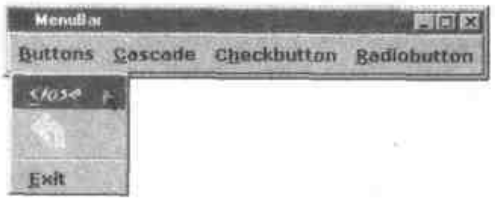


图 4.34 Pmw 菜单条控件

```
balloon = Pmw.Ballon(root)
menuBar = Pmw.MenuBar(root,hull_relief=RAISED,hull_borderwidth=1,
                      balloon=balloon)
menuBar.pack(fill=X)

menuBar.addmenu('Buttons', 'Simple Commands')
menuBar.addmenuitem('Buttons', 'command', 'Close this window',
                    font=('StingerLight',14),label='Close')
menuBar.addmenuitem('Buttons', 'command',
                    bitmap="@bitmaps/RotateLeft",foreground='yellow')
menuBar.addmenuitem('Buttons', 'separator')
menuBar.addmenuitem('Buttons', 'command',
                    'Exit the application',label='Exit')

menuBar.addmenu('Cascade', 'Cascading Menus')
menuBar.addmenu('Checkbutton', 'Checkbutton Menus')
menuBar.addmenu('Radiobutton', 'Radiobutton Menus')
```

关于菜单条控件的文档参见附录 C 的 MenuBar 小节。

4.3.13 消息栏

消息栏控件被用来为应用实现状态区。几个不同类里的信息被展示出来，每个信息都被展示一段时期，该时期是由其类型决定的。此外，每个类型都分配了一个优先级，优先级最高的消息首先被显示。

也可以设置收到各类消息的鸣铃时间，图 4.35 显示了系统错误是怎样出现的。

```
messagebar = box = None
def selectionCommand():
  sels = box.getcurselection()
  if len(sels) > 0:
  messagetype = sels[0]
```

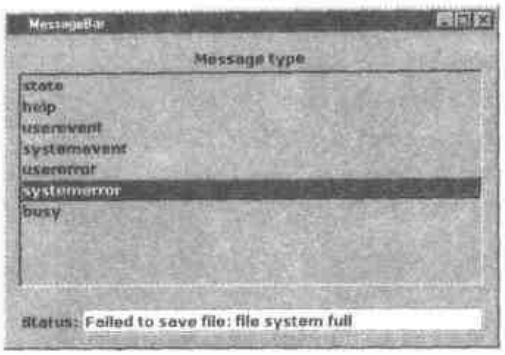


图 4.35 消息栏控件

```
    if messagetype == 'state':
        messagebar.message('state', 'Change of state message')
    else:
        text = messages[messagetype]
        messagebar.message(messagetype,text)

messages = {'help'      : 'Save current file',
            'userevent'  : 'Saving file "foo"',
            'busy'       : 'Busy deleting all files from file system...'
            'systemevent' : 'File "foo" saved',
            'usererror'  : 'Invalid file name "foo/bar"',
            'systemerror' : 'Failed to save file:file system full',
            }

messagebar = Pmw.MessageBar(root,entry_width=40,entry_relief=GROOVE,
                            labelpos=W,label_text='Status: ')
messagebar.pack(side=BOTTOM,fill=X,expand=1,padx=10,pady=10)

box = Pmw.ScrolledListBox(root,listbox_selectmode=SINGLE,
        items=('state', 'help', 'userevent', 'systemevent',
                'usererror', 'systemerror', 'busy',),
        label_text='Message type',labelpos=N,
        selectioncommand=selectionCommand)
box.pack(fill=BOTH,expand=1,padx=10,pady=10

Documentation for the MessageBar widget starts on page 574.
```

消息栏控件的文档参见附录 C 的 **MessageBar** 小节。

4.3.14 消息对话

消息对话控件是个显示单一信息的便利的对话框。它可以显示单行或多行文本，或一些按钮。对于制作简单“高速”的对话，它有用极了。图 4.36 显示了一个例子。

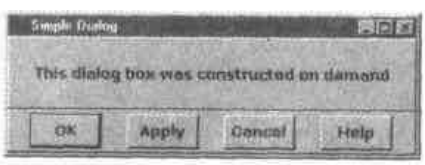


图 4.36 Pmw 消息对话控件

```
dialog = Pmw.MessageDialog(root,title = 'Simple Dialog',
                    defaultbutton = 0,
                    buttons = ('OK', 'Apply', 'Cancel', 'Help'),
                  message_text = 'This dialog box was constructed on demand')
dialog.iconname('Simple message dialog')

result = dialog.activate()
print 'You selected',result
```

消息对话控件的文档参见附录 C 的 MessageDialog 小节。

4.3.15 记事本 R

记事本 R 控件实现流行的 property sheet 主题。方法允许创建多页或面板。任何内容可以加到面板上。用户通过在其顶端单击 Tab 键选择面板。面板可以通过实例方法来升高或者降低。例子用图形 4.37 表示。



图 4.37 Pmw 记事本 R 控件

```
nb = Pmw.NoteBookR(root)

nb.add('p1', label='Page 1')
nb.add('p2', label='Page 2')
nb.add('p3', label='Page 3')

p1 = nb.page('p1').interior()
p2 = nb.page('p2').interior()
p3 = nb.page('p3').interior()

nb.pack(padx=5,pady=5,fill=BOTH,expand=1)
Button(p1,text='This is text on page 1',fg='blue').pack(pady=40)

c = Canvas(p2,bg='gray30')
w = c.winfo_reqwidth()
h = c.winfo_reqheight()
c.create_oval(10,10,w-10,h-10,fill='DeepSkyBluel')
c.create_text(w/2,h/2,text='This is text on a canvas',fill='white',
      font=('Verdana',14, 'bold'))
```

```
c.pack(fill=BOTH,expand=1)
```

记事本 R 控件的文档参见附录 C 的 NoteBookR 小节。

4.3.16　记事本 S

记事本 S 控件实现记事本的可选择的风格。记事本 S 提供了另外的选项来控制标签的颜色、尺寸和外观。另外，它和记事本 R 非常相似。图 4.38 为使用记事本 S 的类似的版面设计。

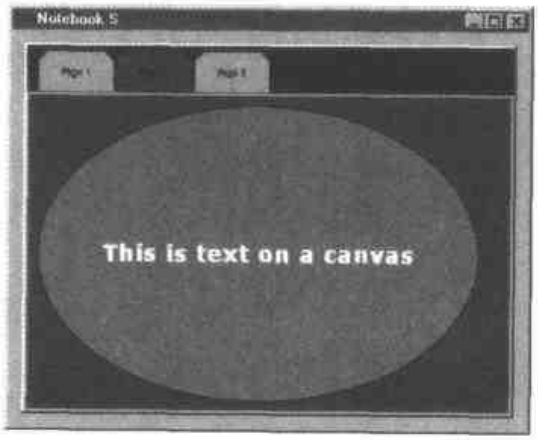


图 4.38　Pmw 记事本 S 控件

```
nb = Pmw.NoteBooksS(root)

nb.addPage('Page 1')
nb.addPage('Page 2')
nb.addPage('Page 3')

f1 = nb.getPage('Page 1')
f2 = nb.getPage('Page 2')
f3 = nb.getPage('Page 3')

nb.pack(pady=10,padx=10,fill=BOTH,expand=1)
Button(f1,text='This is text on page 1',fg='blue').pack(pady=40)

c = Canvas(f2,bg='gray30')
w = c.winfo_reqwidth()
h = c.winfo_reqheight()
c.create_oval(10,10,w-10,h-10,fill='DeepSkyBluel')
c.create_text(w/2,h/2,text='This is text on a canvas',fill='white',
      font=('Verdana',14, 'bold'))
c.pack(fill=BOTH,expand=1)
```

记事本 S 控件的文档参见附录 C 的 NoteBookR 小节。

4.3.17　记事本

Pmw 的 0.8.3 版用记事本(NoteBook)替换了记事本 R（NoteBookR）和记事本 S

（NoteBookS）。它和先前的记事本非常相似，但有一些小的变化。在面貌上，你必须改变你的代码用现有的代码来使用记事本。不过，变化是次要的。新形式可能更容易使用。图 4.39 阐述了该新控件。

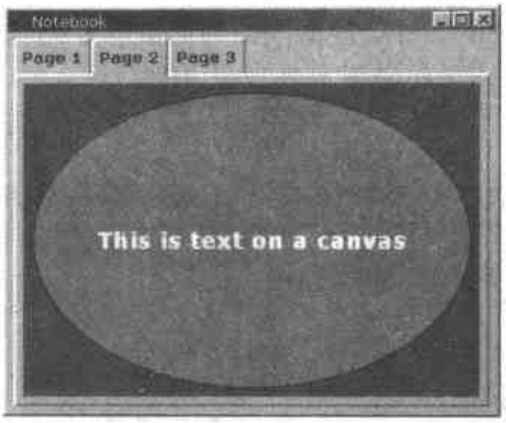


图 4.39 Pmw 记事本控件

```
from Tkinter import *
import Pmw

root = Tk()
root.option_readfile('optionDB')
root.title('Notebook')
Pmw.initialize()

nb = Pmw.NoteBook(root)
p1 = nb.add('Page 1')
p2 = nb.add('Page 2')
p3 = nb.add('Page 3')
nb.pack(padx=5,pady=5,fill=BOTH,expand=1)

Button(p1,text='This is text on page 1',fg='blue').pack(pady=40)
c = Canvas(p2,bg='gray30')
w = c.winfo_reqwidth()
h = c.winfo_reqhcight()
c.create_oval(10,10,w-10,h-10,fill='DeepSkyBluel')
       font=('Verdana',14, 'bold'))
c.pack(fill=BOTH,expand=1)

nb.setnaturalpagesize()
root.mainloop()
```

记事本控件的文档参见附录 C 的 NoteBook 小节。

4.3.18 选项菜单

选项菜单控件实现了程序员熟悉的经典主题：弹出式菜单主题。不过，如同图 4.40 所示的那样，相应的弹出式菜单的外观有点不同。选项菜单应该被用来选择受限制的数

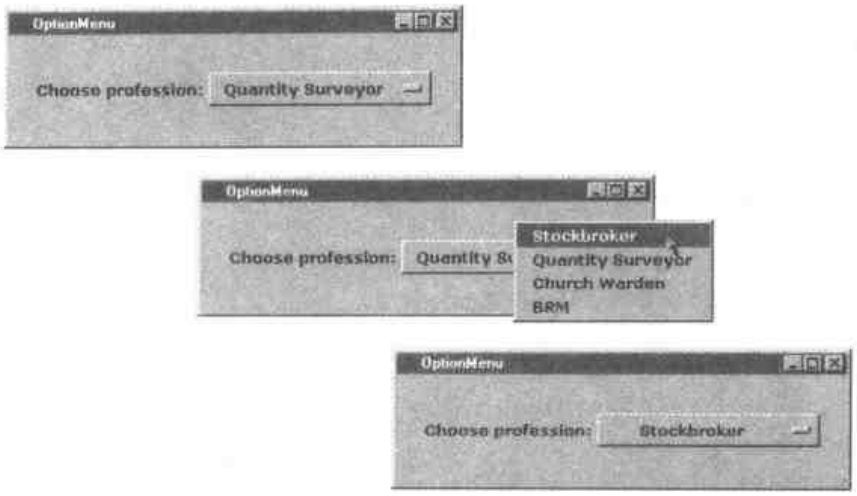


图 4.40　Pmw 选项菜单控件

据项。如果你用大量数据来组装控件，弹出式菜单可能会不适合屏幕大小，并且，控件会无法滚动。

```
var = StringVar()
var.set('Quantity Surveyor')
opt_menu = Pmw.OptionMenu(root,labelpos=W,
       label_text='Choose profession: ',menubutton_textvariable=var,
       items=('Stockbroker','Quantity Surveyor','Church Warden','BRM')
        menubutton_width=16)
opt_menu.pack(anchor=W,padx=20,pady=30)

Documentation for the OptionMenu widger starts on page 584.
```

选项菜单控件的文档参见附录 C 的 OptionMenu 小节。

4.3.19　窗格控件

窗格控件创建了一个包含多重框架的容器。每个框架都是其他控件的容器，也许是通过拖动其句柄或分割行来调整大小的。每个面板的范围内都是独立操作的，所以，单一的面板也许会变大或者缩小来修改其子面板的版面设计。图 4.41 显示了一个例子。

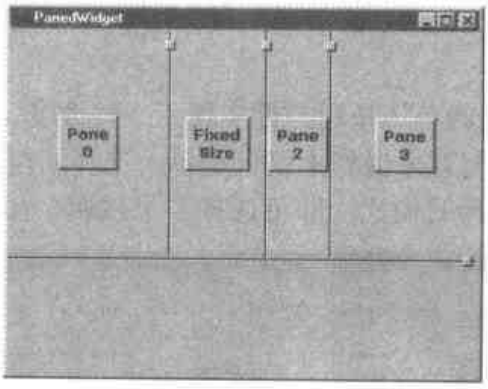


图 4.41　Pmw 面板控件

```
pane = Pmw.PanedWidget(root,hull_width=400,hull_height=300)
pane.add('top',min=100)
pane.add('bottom',min=100)

topPane = Pmw.PanedWidget(pane.pane('top'),orient=HORIZONTAL)
for num in range(4):
  if num == 1:
       name = 'Fixed\nSize'
       topPane.add(name,min=.2,max=.2)
  else:
       name = 'Pane\n' + str(num)
  button = Button(topPane.pane(name),text=name)
  button.pack(expand=1)
topPane.pack(expand=1,fill=BOTH)

pane.pack(expand=1,fill=BOTH)
```

窗格控件的文档参见附录 C 的 **PanedWidget** 小节。

4.3.20 提示对话框

提示对话框控件是一个方便的对话框，它在按钮框中显示了单一的输入字段和相当数量的按钮，对于制作简单的高速对话相当有用。图 4.42 所示的例子是从用户处收集密码。



图 4.42 Pmw 提示对话框控件

```
dialog = Pmw.PromptDialog(root,title='Password',label_text='Password:',
         entryfield_labelpos=N,entry_show='*',defaulatbutton=0,
         buttons=('OK', 'Cancel'))

result = dialog.activate()
```

提示对话框控件的文档参见附录 C 的 **PromptDialog** 小节。

4.3.21 单选选项

单选选项控件对 Tkinter 的单选按钮控件实现了一些改变。单选选项创建了包含大量按钮的管理器。控件可能被配置来操作不仅仅是单一选择的模式，即同一时间只能有一个按钮被激活；还有多重选择的模式，即可选择多个按钮。如图 4.43 所示。



图 4.43 Pmw 单选选项控件

```
horiz = Pmw.RadioSelect(root,labelpos=W,label_text=HORIZONTAL,
        frame_borderwidth=2,frame_relief=RIDGE)
horiz.pack(fill=X,padx=10,pady=10)

for text in ('Passion fruit', 'Loganberries', 'Mangoes in syrup',
             'Oranges', 'Apples', 'Grapefruit'):
    horiz.add(text)
horiz.invoke('Mangoes in syrup')

multiple = Pmw.RadioSelect(root,labelpos=W,label_text='Multiple\nselection',
       frame_borderwidth=2,frame_relief=RIDGE,selectmode=MULTIPLE)
multiple.pack(fill=X,padx=10)

for text in ('Doug', 'Dinsdale',"Stig O' Tracy",'Vince', 'Gloria Pules'):
   multiple.add(text)
multiple.invoke('Dinsdale')
```

单选选项控件的文档参见附录 C 的 RadioSelect 小节。

4.3.22　滚动画布

滚动画布控件是个很方便的控件。它是个能提供联合水平和垂直滚动条的画布控件。图 4.44 显示了一个例子。

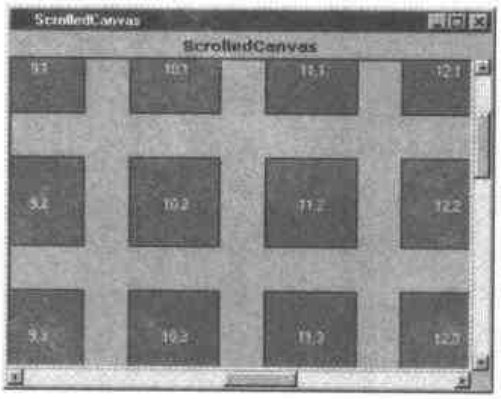


图 4.44　Pmw 滚动画布控件

```
sc = Pmw.ScrolledCanvas(root,borderframe=1,labelpos=N,
                        label_text='ScrolledCanvas',usehullsize=1,
                        hull_width=400,hull_height=300)

for i in range(20):
  x = -10 + 3*i
  y = -10
  for j in range(10):
     sc.create_rectangle('%dc'%x, '%dc'%y, '%dc'%(x+2), '%dc'%(y+2),
              fill='cadetblue',outline='black')
     sc.create_text('%dc'%(x+1), '%dc'%(y+1),text='%d,%d'%(i,j),
              anchor=CENTER,fill='white')
     y = y + 3
```

```
sc.pack()
sc.resizescrollregion()
```

滚动画布控件的文档参见附录 C 的 ScrolledCanvas 小节。

4.3.23 滚动区域

滚动区域控件提供了带有标签的输入字段控件,允许用户滚动看到空间所不能显示的多余数据。这个控件是被保留下来作特殊用途的，因为许多一般考虑的人为因素与图形用户界面要素是相抵触的。图 4.45 显示了使用键盘箭头键的字段滚动效果。



图 4.45 Pmw 滚动区域控件

```
lines = (
  "Mount Everset. Forbidding,aloof,terrifying. This year,this",
  "remote Himalayan mountain,this mystical temple,surrounded by the",
  "most difficult terrain in the world,repulsed yet another attempt to",
  "conquer it. (picture changes to wind-swept,snowy tents and people)",
  "This time,by the International Hairdresser's Expedition. In such",
  "freezing,adverse conditions,man comes very close to breaking",
  "point. What was the real cause of the disharmony which destryoed",
  "their chances at success?")

 global index
field = index = None
def execute():
    global index
    field.configure(text=lines[index % len(lines)])
field = Pmw.ScrolledField(root,entry_width=30,
            entry_relief=GROOVE,labelpos=N,
            label_text='Scroll the field using the\nmiddle mouse button')
field.pack(fill=X,expand=1,padx=10,pady=10)

button = Button(root,text='Change field',command=execute)
button.pack(padx=10,pady=10)

index = 0
execute()
```

滚动区域控件的文档参见附录 C 的 ScrolledField 小节。

4.3.24 滚动框架

滚动框架控件是个很方便的控件，它结合了水平和垂直的滚动。例子如图 4.46 所示。

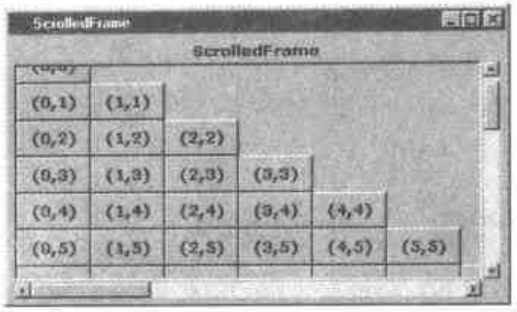


图 4.46　Pmw 滚动框架控件

```
global row,col
row = col = 0
sf = frame = None
def addButton():
    global row,col
    button = Button(frame,text = '(%d,%d) ' % (col,row))
    button.grid(row=row,col=col,sticky='nsew')
    frame.grid_rowconfigure(row,weight=1)
    frame.grid_columnconfigure(col,weight=1)
    sf.reposition()
    if col == row:
       col = 0
       row = row + 1
    else:
       col = col + 1
sf = Pmw.ScrolledFrame(root,labelpos=N,label_text='ScrolledFrame',
     usehullsize=1,hull_width=400,hull_height=220)
sf.pack(padx=5,pady=3,fill='both',expand=1)
frame = sf.interior()

for i in range(250):
  addButton()
```

滚动框架控件的文档参见附录 C 的 ScrolledFrame 小节。

4.3.25　滚动列表框

滚动列表框控件是个很方便的控件，它结合了水平和垂直的滚动。图 4.47 给出了典型的滚动列表框。



图 4.47　Pmw 滚动列表框控件

```
box = None
def selectionCommand():
    sels = box.getcurselection()
    if len(sels) == 0:
        print 'No selection'
    else:
        print 'Selection: ',sels[0]

box = Pmw.ScrolledListBox(root,listbox_selectmode=SINGLE,
            items=('John Cleese', 'Eric Idle', 'Graham Chapman',
                    'Terry Jones', 'Michael Palin', 'Terry Gilliam'),
            labelpos=NW,label_tet='Cast Members',
            listbox_height=5,vscrollmode='static',
            selectioncommand=selectionCommand,
            dblclickcommand=selectionCommand,
            usehullsize=1,hull_width=200,hull_height=200,)
box.pack(fill=BOTH,expand=1,padx=5,pady=5)
```

滚动列表框控件的文档参见附录 C 的 ScrolledListBox 小节。

4.3.26 滚动文本

如图 4.48 所示，滚动文本控件是个很方便的控件，它结合了水平和垂直的滚动。

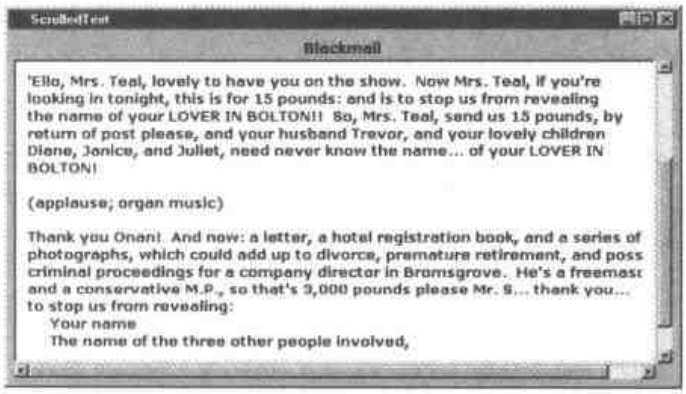


图 4.48 Pmw 滚动文本控件

```
st = Pmw.ScrolledText(root,borderframe=1,labelpos=N,
        label_text='Blackmail',usehullsize=1,
        hull_width=400,hull_height=300,
        text_padx=10,text_pady=10,
        text_wrap='none')
st.importfile('blackmail.txt')
st.pack(fill=BOTH,expand=1,padx=5,pady=5)
```

滚动文本控件的文档参见附录 C 的 ScrolledText 小节。

4.3.27 选项对话

选项对话控件为用户提供了方便的对话。它允许用户从滚动列表中选择项目来回答问题。除了没有规定让用户键入值之外，它与组合框非常相似。图 4.49 给出了一个例子。



图 4.49 Pmw 选择对话控件

```
dialog = None
def execute(result):
    sels = dialog.getcurselection()
    if len(sels) == 0:
        print 'You clicked on',result, '(no selection) '
    else:
        print 'You clicked on',result,sels[0]
    dialog.deactivate(result)

dialog = Pmw.SelectionDialog(root,title='String',
      buttons=('OK', 'Cancel'),defaultbutton='OK',
      scrolledlist_labelpos=N,label_text='Who sells string? ',
      scrolledlist_items=('Mousebat', 'Follicle', 'Goosecreature',
                    'Mr. Simpson', 'Ampersand', 'Spong', 'Wapcaplet',
                    'Looseliver', 'Vendetta', 'Prang'),
      command=execute)
dialog.activate()
```

选项对话控件的文档参见附录 C 的 SelectionDialog 小节。

4.3.28 文本对话

文本对话控件提供了方便的对话，它用来给用户显示多行文本。另外，它可能也用作简单的文本编辑程序。见图 4.50。

```
sketch = """Doctor: Mr. Bertenshaw?
Mr. B: Me, Doctor.
# ------Lines removed----------
Jane,you Trent,you Trillo…me doctor!"""

dialog = Pmw.TextDialog(root,scrolledtext_labelpos='n',
        title='Sketch',
        defaultbutton=0,
        label_text='The Hospital')
dialog.insert(END,sketch)
dialog.configure(text_state='disabled')
dialog.activate()
dialog.tkraise()
```

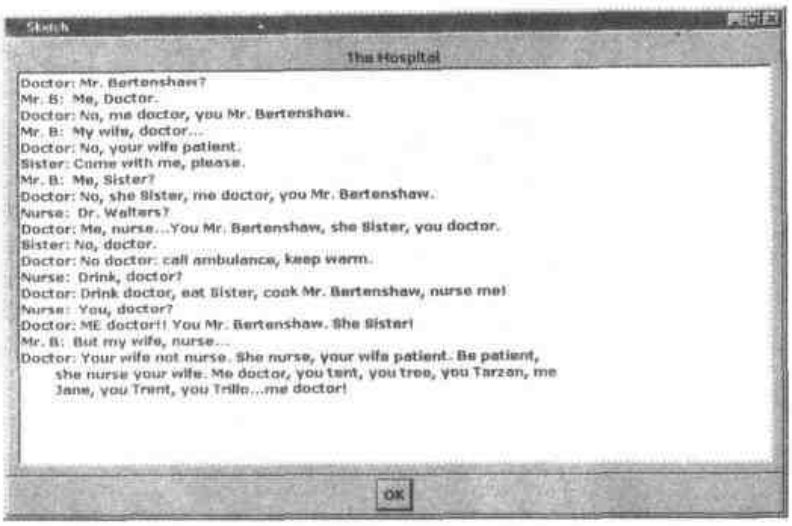


图 4.50　Pmw 文本对话控件

文本对话控件的文档参见附录 C 的 TextDialog 小节。

4.3.29　时间计数

时间计数控件实现了一个上下移动箭头来设置时、分、秒的装置。这个控件可设定自动重复，这样按下按钮就可以回转显示在控件中的值。图 4.51 给出了该控件的外观。



图 4.51　Pmw 时间计数控件

```
time = Pmw.TimeCounter(root,labelpos=W,label_text='HH:MM:SS',
                       min='00:00:00',max='23:59:59')
time.pack(padx=10,pady=5)
```

时间计数控件的文档参见附录 C 的 TimeCounter 小节。

4.4　制作新的大控件

Pmw 不仅提供了有用的控件，还提供简单的机制以允许你开发新型的大控件。Pmw 文档描述了把大控件编码的过程。这里的就是那些材料的改编。

4.4.1　大控件的描述

这个控件将实现一个简单的标尺，该标尺追踪比例控件提供的整数值，从一个范围内选择一个数。该标尺类似范围的百分比那样设定。完成的大控件类似图 4.52 中所示。



图 4.52　标尺控件

比例控件将成为大控件的一部分，因为范围可能由程序员来设置；适用于应用标尺的刻度和颜色可以被同样地改变，所以，

我们也使之成为一个元件。

4.4.2 选项

除了比例和标尺元件的选项，我们需要为大控件定义一些选项。首先，我们定义了最小值和最大值，来允许程序员用控件支撑的范围。其次，我们定义了填充和尺寸来控制标尺的颜色和大小。最后，我们定义了值以允许我们设置大控件的初始值。

4.4.3 制作大控件类

Pmw 大控件不仅继承了 Pmw.Megawidget、Pmw.MegaToplevel，还继承了 Pmw.Dialog。该标尺控件意在用于其他代码控件的范围内，所以它继承了 Pmw.MegaWidget。这里是大控件的代码。

pmw_megawindget.py

```
from Tkinter import *
import Pmw

class Gauge(Pmw.MegaWidget):
    def __init__(self, parent=None, **kw):
        # Define the options for the megawidget
        optiondefs = (
            ('min',         0,          Pmw.INITOPT),
            ('max',         100,        Pmw.INITOPT),
            ('fill',        'red',      None),
            ('size',        30,         Pmw.INITOPT),
            ('value',       0,          None),
            ('showvalue',   1,          None),
            )

        self.defineoptions(kw, optiondefs)

        # Initialize the base class
        Pmw.MegaWidget.__init__(self, parent)

        interior = self.interior()

        # Create the gauge component
        self.gauge = self.createcomponent('gauge',
                        (), None,
                        Frame, (interior,),
                        borderwidth=0)
        self.canvas = Canvas(self.gauge,
                    width=self['size'], height=self['size'],
                    background=interior.cget('background'))
        self.canvas.pack(side=TOP, expand=1, fill=BOTH)
        self.gauge.grid()
```

❶ ❹ ❹ ❹

```
        # Create the scale component
        self.scale = self.createcomponent('scale',
                        (), None,
                        Scale, (interior,),
                        command=self._setGauge,
                        length=200,
                        from_ = self['min'],
                        to    = self['max'],
                        showvalue=self['showvalue'])
        self.scale.grid()

        value=self['value']
        if value is not None:
            self.scale.set(value)

        # Check keywords and initialize options
        self.initialiseoptions(Gauge)

    def _setGauge(self, value):
        self.canvas.delete('gauge')
        ival = self.scale.get()
        ticks = self['max'] - self['min']
        arc = (360.0/ticks) * ival
        xy = 3,3,self['size'],self['size']
        start = 90-arc
        if start < 0:
            start = 360 + start
        self.canvas.create_arc(xy, start=start, extent=arc-.001,
                               fill=self['fill'], tags=('gauge',))

Pmw.forwardmethods(Gauge, Scale, 'scale')

root = Tk()
root.option_readfile('optionDB')
root.title('Gauge')
Pmw.initialise()

g1 = Gauge(root, fill='red', value=56, min=0, max=255)
g1.pack(side=LEFT, padx=1, pady=10)

g2 = Gauge(root, fill='green', value=60, min=0, max=255)
g2.pack(side=LEFT, padx=1, pady=10)

g3 = Gauge(root, fill='blue', value=36,  min=0, max=255)
g3.pack(side=LEFT, padx=1, pady=10)

root.mainloop()
```

代码注解

❶ 大控件的选项由选项名称、默认值和最终参数的三元素序列所给定。最终参数可以是回叫函数，Pmw.INITOPT，也可以为空。如果它是 Pmw.INITOPT，那么选项仅能作为设定了初值的选项，并且它不能被设置来调用配置。调用 self.defineoptions 包括在控件构造函数中传递关键参数。这些值可以屏蔽任何默认值。

❷ 为了设置选项，我们调用了基类的构造函数，将双亲控件作为单个参数传递。

❸ 依照惯例，Pmw 定义了一个作为控件容器的内部属性。

❹ 然后我们创建了标尺的指示器，它将被拖到包含在框架内的画布上。创建控件（createcomponent）方法有五个标准的参数（构造函数的名称、别名、组、类、参数），它是由任何关键字参数的数字决定的。

❺ 然后，我们以类似的方法构建了比例控件。

❻ 完成了构造函数之后，我们首先调用 initialiseoptions 来检查我们所支持的被应用的所有关键字参数。它调用了我们所定义的任何选项回叫。

❼ 一旦定义了大控件的类，我们就调用 Pmw.forwardmethods 方法来指导从其他控件到比例控件调用的任何方法。

图 4.53 阐述了标尺大控件可能作为色彩混频器的一个应用。为了显示或隐藏每个滑块的当前值，控件可能被会被重新装配。把更多的选项加到控件上是个简单的任务。

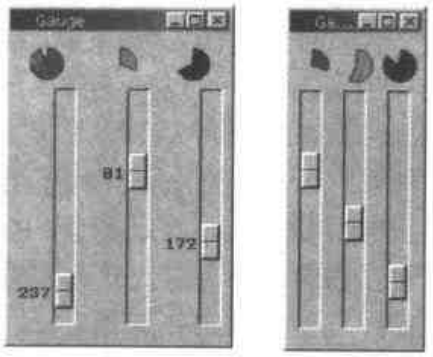


图 4.53　使用标尺大控件作为颜色混合器

第5章　屏幕版面设计

图形用户界面版面的设计是个未被充分理解的领域；可以想见程序员在上面浪费了大量时间。本章详细介绍了3个几何管理器：包（Pack）、网格（Grid）和位置（Place）。还将提出一些先进的课题，包括可变尺寸窗口的方法和附带的问题：维持界面的吸引力和有效性。

5.1　版面设计的介绍

几何管理器负责控制屏幕上控件的大小和位置。在 Motif 中，控件的放置通常是由几个管理器控件中的一个来处理的，包括 XmForm 控件在内的制约控件类就是个例子。这里，版面设计通过一个或多个靠近顶端、底部、左边或者右边的控件和容器的控件来控制。通过选择合适的附件的结合，程序员能控制相当数量的行为，这决定了当窗口放大或者缩小的时候，控件是怎样出现的。

Tk 提供了一个灵活的方法来在屏幕上显示控件。X 定义几个管理器类控件，但是，在 Tk 中，可能会用到三个几何管理器。实际上，互相使用管理器是可能的（虽然这里有一些关于一个人如何去做到它的重要规则）。Tk 通过开拓 X 行为完成了这种灵活性，也就是说控件几何是通过几何管理器而不是自身控件决定的。如同 X，如果你不处理控件，那么，虽然它存在于内存中，它将不被画在屏幕上。

可用于 Tkinter 的几何管理器有这些：打包器（Packer），它是用得最多的管理器；网格，它是 Tk 新近追加的；位，最少被提及，但是它在放置控件的过程起了极大的作用。在本书中你将看到全部三个几何管理器的例子。几何管理器可用于所有 Tkinter 支持的体系结构，所以，没有必要知道任何相关体系结构的工具包的实现。

几何管理是个非常复杂的话题，因为控件、它们的容器、窗口和支持窗口的管理器之间需要有很大的协调。目的是显示一个或多个从属控件，这些从属控件是作为某主控件的从属（一些程序员倾向于使用子控件和双亲控件）。主控件通常是框架或画布等之类的容器，但是，大部分的控件能作为主控件。例如，在框架的底部放置一个按钮。如同我们在主控件中简单定位从属一样，我们想在加入更多的控件或窗口放大和缩小时控制控件的行为。

协调过程从每个从属控件请求适当宽度和高度显示其内容开始。这依赖于大量的因素。例如，一个按钮根据显示文本的长度、选中字体的大小和粗度计算其所需的尺寸。

其次，主控件和其几何管理器一起决定可以用来满足从属请求的可用空间的大小。可以利用的空间可能多于或少于所要求的空间，导致了控件的挤压、伸展与重叠——这

依赖于所用的是哪一种几何管理器。

再次，根据窗口的设计，主文本的主空间必须在所有同等的容器之间分配。结果依赖于同等控件的几何管理器。

最后，在顶层控件（通常是顶层对象处理程序）和窗口管理器之间有一个协调过程。在其后可用空间用来确定控件的最终大小与位置。在一些例子中可能没有足够的空间来显示全部的控件，而且，它们根本没有被实现。甚至在窗口被初始化时，这个协调完成后，如果任何控件改变了其配置（例如，按钮上的文字改变了），或者用户调整了窗口大小，则协调过程又重新开始。幸运的是，使用几何管理器比讨论它们简单得多了！

当设计一个屏幕的时候，会应用大量的普通方案。打包器和较小范围的网格的特性之一是：有可能允许几何管理器来决定窗口的最终大小。这一点对灵活创建窗口很有用，但想预测控件的总数却很困难。运用这个方法，在显示时增加或移去控件，窗口就会改变其大小。同时，设计员可以在固定大小的窗口上使用放置器（Placer），这依赖于想要的效果。

让我们先来看打包器（Packer），它是最常用的管理器。

5.2　打包器

打包器通过逐个把从属控件从外边缘增加到窗口中心的办法将从属控件放置于主文本上。打包器用来管理行、列和二者兼之。不过，可能需要补充一些计划来得到预期的效果。

打包器用来维持一个从属列表或一个打包列表，这保持了从属空间原来在打包器中的顺序。见图 5.1（本图形是参考 John Ousterhout 关于打包器的描述而作的）。

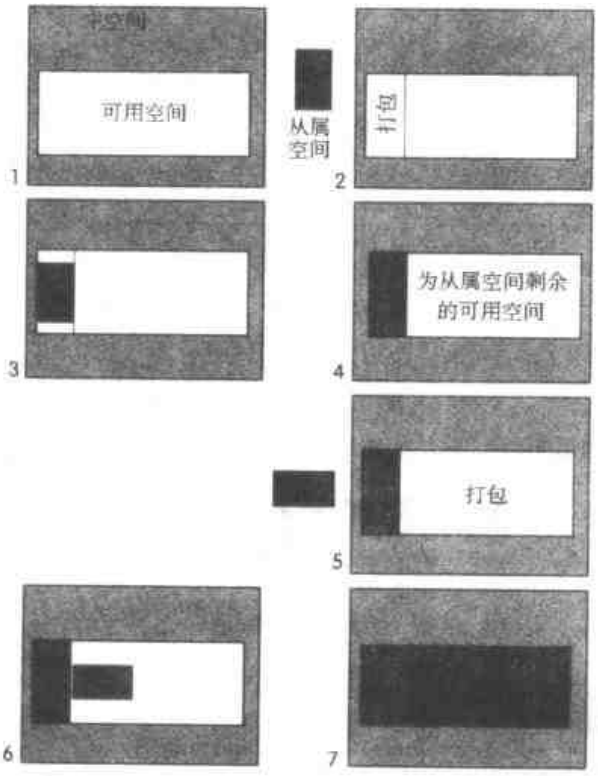


图 5.1　打包操作

图 5.1（1）给出了可用于放置控件的空间。这个空间可能在一框架内或放置了其他控件后剩下的空间内。打包器为下一个从属分配了一个包，这个从属是从可用空间截取了一部分来处理的。哪边将被分配是由包要求支持的选项决定的；就这个例子来说，side=LEFT 和 fill=Y 选项已经被指定了。打包器分配的实际大小是由大量因素决定的。当然，从属的大小是出发点，但是，可用空间和任何从属要求的任意填充法必须被考虑。图 5.1（2）表示了分配的包。

然后，从属在包内定位。如果可用的空间导致了比从属的规模还要小的包，它可能就要被压缩或裁剪，这要根据所需的选项来决定。就这个例子来说，从属比可用空间更小而且它的高度会增加以填充可用的包。图 5.1(4)给出了更多从属的可用空间。在图 5.1(5)中我们用 side=LEFT 和 fill=BOTH 选项来将另一个从属打包。此外，可用的包比从属的规模大（见图 5.1(6)），所以，控件就被发展来填充可用的空间。这个效果见图 5.1(7)。



图 5.2　打包几何管理器

这是一个使用包的方法的简单的范例，用图 5.2 表示。

Example 5 1 py

```
from Tkinter import *

class App:
    def __init__(self, master):
        Button(master, text='Left').pack(side=LEFT)
        Button(master, text='Center').pack(side=LEFT)
        Button(master, text='Right').pack(side=LEFT)

root = Tk()
root.option_add('*font', ('verdana', 12, 'bold'))
root.title("Pack - Example 1")
display = App(root)
root.mainloop()
```

❶

代码注解

❶ side=LEFT 参数让打包器开始定位在容器左边的包列表中的控件。在这种情况下，容器是 Tk 预置程序制作的默认顶层外壳。为了围住打包的控件，外壳会收缩或扩展。

把控件封装在一个框架内对打包器的收缩缠绕效果没有影响。在这个例子中（如图 5.3 所示），我们已经在中间的按钮中增加了正文的长度，并且框架伸长到所要求的大小。

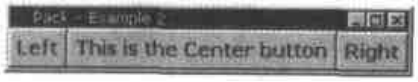


图 5.3　根据控件大小调整的打包

Example 5 2 py

```
fm = Frame(master)
Button(fm, text='Left').pack(side=LEFT)
```

```
Button(fm, text='This is the Center button').pack(side=LEFT)
Button(fm, text='Right').pack(side=LEFT)
fm.pack()
```

从框架的顶端打包所产生的结果显示在图 5.4 中。注意：因为没有提供更多的选项以及拉伸窗口以适合最大的控件，所以打包程序将控件集中在可用空间内。

Example_5_2a.py

```
Button(fm, text='Top').pack(side=TOP)
Button(fm, text='This is the Center button').pack(side=TOP)
Button(fm, text='Bottom').pack(side=TOP)
```

组合打包器列表中的边选项可以达到预期的效果（虽然更经常的是：你将不会以你没有预料到的结果结束！）。图 5.5 阐述了不寻常的规划是怎样引起的。

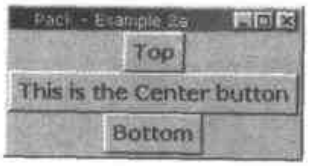


图 5.4　从框架的顶端打包



图 5.5　内部组合

在全部这些例子中我们看到，打包器为了适合所需的空间，协商处理了容器的全部规模。如果你想控制容器的大小，你必须使用几何选项，因为试图改变框架的大小（看 example_5_4.py）是没有效果的，如图 5.6 所示。

Example_5_4.py

```
fm = Frame(master, width=300, height=200)
Button(fm, text='Left').pack(side=LEFT)
```

当程序员开始用 Tkinter 和大部分其他的工具包工作时，测量窗口大小常常是个问题。并且，在控件说明上加上宽度和高度选项却没有响应时是非常令人沮丧的。

为了设置窗口的规模，我们需要用到 wm.geometry 选项。图 5.7 给出了在根窗口改变几何结构的效果。



图 5.6　改变帧尺寸的效果

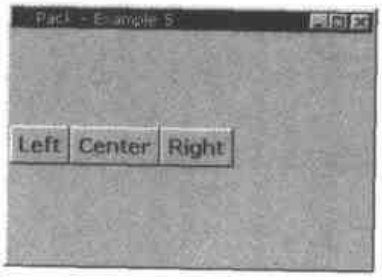


图 5.7　指定顶层外壳几何管理器

Example_5_5.py

```
Master.geometry("300x200")
```

5.2.1 使用展开选项

展开选项是控制当窗口被调整大小时打包器是否展开的控件。先前的所有例子都默认 expand=NO。事实上，如果展开是可用（正确）的，控件会在其包内扩展以填充可用的空间；它是否展开由填充选项决定（参见 5.2.2 的“使用填充选项”）。

Example 5 6.py

```
Button(fm, text='Left').pack(side=LEFT, expand=YES)
Button(fm, text='Center').pack(side=LEFT, expand=YES)
Button(fm, text='Right').pack(side=LEFT, expand=YES)
```

图 5.8 给出了不用填充选项而将展开设置为 true(YES)的效果（看 Example_5_6.py）。另一个屏幕里垂直的定向是与 side=TOP 相似的（见 Example_5_2a.py）。

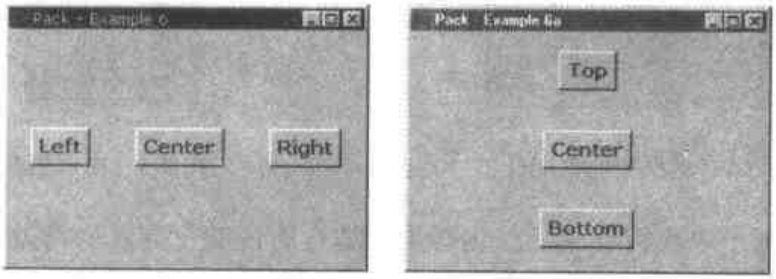


图 5.8 不使用填充选项扩展

5.2.2 使用填充选项

Example_5_7.py 阐述了联合填充和展开选项的效果；输出见图 5.9（1）所示。

如果填充选项被单独用于 Example_5_7.py，将得到一个类似于图 5.9（2）的显示结果。通过使用填充和扩展，我们看图 5.9（3）所示的效果。

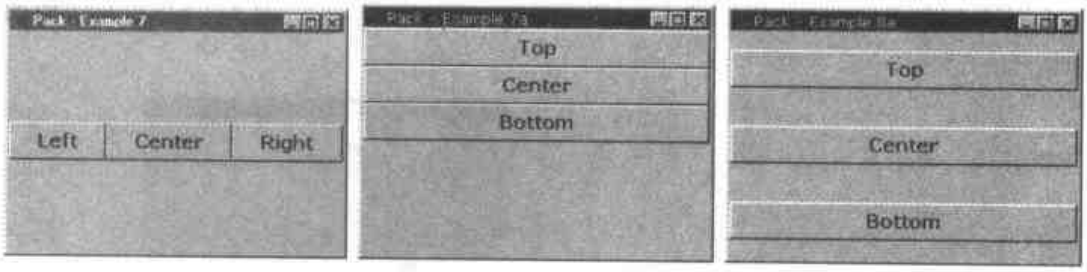


图 5.9 使用填充选项

Example 5 7.py

```
Button(fm, text='Left').pack(side=LEFT, fill=X, expand=YES)
Button(fm, text='Center').pack(side=LEFT, fill=X, expand=YES)
```

```
Button(fm, text='Right').pack(side=LEFT, fill=X, expand=YES)
```

改变填充和扩展选项的结合可用于在不同场合制造出不同效果。如果你混合扩展选项，例如类似 **example_5_8.py**，那么在其他窗口保持恒定大小时，你就能用一些控件来调整窗口大小。图 5.10 阐述了拉伸和压缩屏幕的效果。

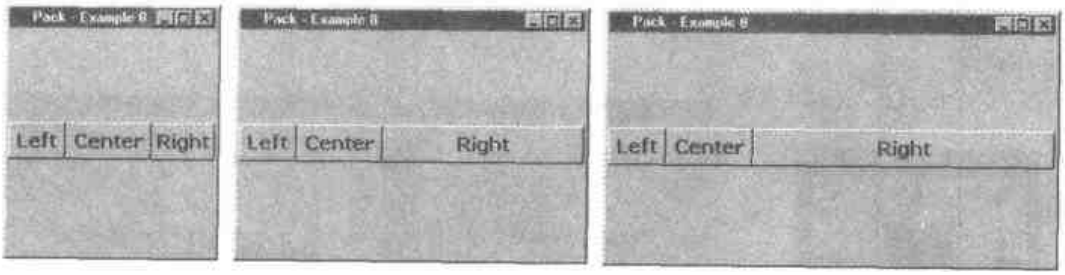


图 5.10　允许控件独立扩展和填充

Example 5_8.py

```
Button(fm, text='Left').pack(side=LEFT, fill=X, expand=NO)
Button(fm, text='Center').pack(side=LEFT, fill=X, expand=NO)
Button(fm, text='Right').pack(side=LEFT, fill=X, expand=NO)
```

使用 **fill=BOTH** 允许控件使用它所有的包。不过，如同图 5.11 所示的那样，它可能会制作一些相当难看的效果。另一方面，这个行为也许正是你的图形用户界面所需要的。

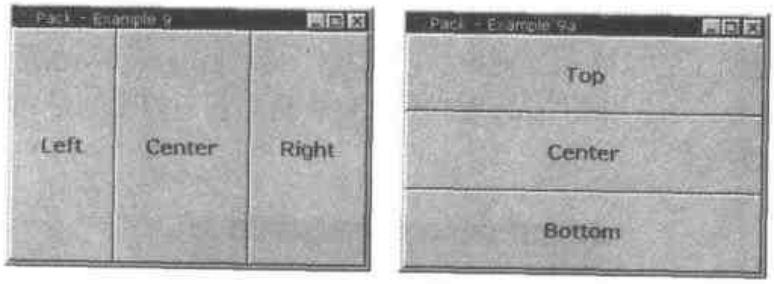


图 5.11　使用填充 **fill=BOTH**

5.2.3　使用 padx 和 pady 选项

padx 和 **pady** 选项允许控件用它周围的补充性空间来打包。图 5.12 显示把 **padx=10** 加到中心按钮的包需求上的效果。填充分别被用在 **padx** 和 **pady** 规定的左/右侧或顶端/底部，这可能还不能达到你所想要的效果，因为如果你并排放置两个控件，每个都是 **padx=10** 的话，那么，在两个控件与这一对控件的左右 10 个像素之间的将会是 20 个像素，这将导致一些不寻常的间隔。

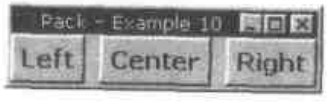


图 5.12　使用 **padx** 创造多余空间

5.2.4 使用锚选项

锚选项是用来决定当可用空间大于所要求的尺寸，并且没有或有一种填充法被规定时，一个控件在它的包中将被置于何处。图 5.13 阐述了如果提供了锚，控件将如何被打包。选项 anchor=CENTER 把控件放在包的中心位置。图 5.14 给出了这在实际中看起来的样子。

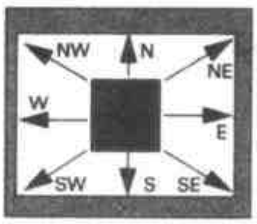


图 5.13 在控件内可及空间定锚

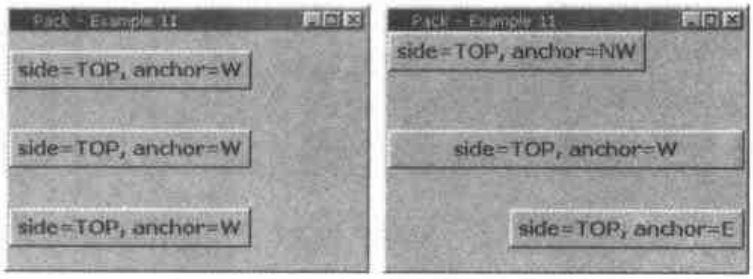


图 5.14 使用锚选项放置控件

5.2.5 使用分层性包装

使用打包器来布置简单的屏幕相对容易的时候，通常需要应用一个分层的方法并使用这样一个构思：在框架内将大量的控件打包。并且，不是并排的打包这些框图就是将它们放入另外的框图中。这允许了多于计划的控制，特别是如果需要填充或展开控件时的控制。

图 5.15 举例说明了尝试展示 2 列控件的结果。乍看起来，代码在运行，但是它并没有生成所想要的版面。如果你用 side=TOP 包装从属，剩下的空间是在从属下的，你不能将旁边现有的包打包。

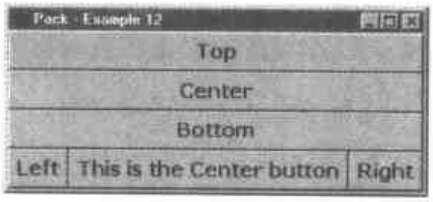


图 5.15 滥用 Packer

Example 5_12.py

```
fm = Frame(master)
Button(fm, text='Top').pack(side=TOP, anchor=W, fill=X, expand=YES)
Button(fm, text='Center').pack(side=TOP, anchor=W, fill=X, expand=YES)
Button(fm, text='Bottom').pack(side=TOP, anchor=W, fill=X, expand=YES)
Button(fm, text='Left').pack(side=LEFT)
Button(fm, text='This is the Center button').pack(side=LEFT)
Button(fm, text='Right').pack(side=LEFT)
fm.pack()
```

我们所要做的只是：在分开的框图中打包这两列控件，然后并排地打包这些框图。这里是修改过的代码：

Example 5_13.py

```
fm = Frame(master)
Button(fm, text='Top').pack(side=TOP, anchor=W, fill=X, expand=YES)
Button(fm, text='Center').pack(side=TOP, anchor=W, fill=X, expand=YES)
Button(fm, text='Bottom').pack(side=TOP, anchor=W, fill=X, expand=YES)
fm.pack(side=LEFT)
fm2 = Frame(master)
Button(fm2, text='Left').pack(side=LEFT)
Button(fm2, text='This is the Center button').pack(side=LEFT)
Button(fm2, text='Right').pack(side=LEFT)
fm2.pack(side=LEFT, padx=10)
```

图 5.16 给出了运行 Example_5_13.py 完成的效果。

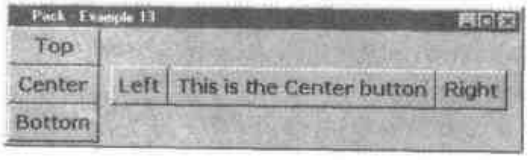


图 5.16　结构包

这是个重要技巧，能在本书的几个例子中见到。举一个使用一些内嵌框图的例子，看在线例子/17 章/Example_16_9.py。

5.3　网格

很多的程序员把网格几何管理器看成是最容易使用的管理器。就我个人而言，我并不完全同意这一点。而你可以自己作判断。看图 5.17。支持使用以“用例子”为主题的图像编辑器是个相当复杂的规划任务。展示使用打包器需要几个嵌套的框架来装入目标

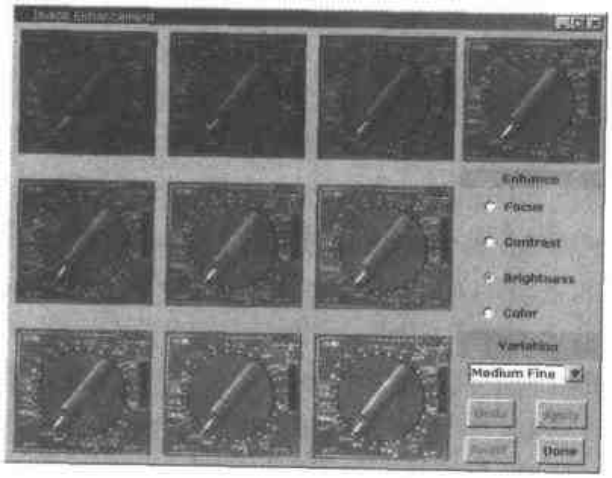


图 5.17　使用网格几何管理器的图像增强

控件的分层路径。它也要求小心计算填充及其他因素来完成最终的规划。使用网格更简单。

在我们固定配备（展示）图像编辑器之前，让我们来看一个更简单的例子。我们将制作一个和 OK 与 Cancel 按钮一起的、包括带有三个输入字段的三个标签的对话。字段需要整洁地排成一列（范例是个改变密码的对话）。图 5.18 显示出网格管理器为我们做了什么。代码非常简单，但是，我已经为透明度去掉了一些不重要的线。

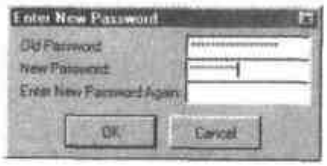


图 5.18　使用网格布置的对话框

Example_5_14.py

```
class GetPassword(Dialog):
   def body(self, master):
       self.title("Enter New Password")

       Label(master, text='Old Password:').grid(row=0, sticky=W)
       Label(master, text='New Password:').grid(row=1, sticky=W)
       Label(master, text='Enter New Password Again:').grid(row=2,
            sticky=W)

       self.oldpw   = Entry(master, width = 16, show='*')
       self.newpw1  = Entry(master, width = 16, show='*')
       self.newpw2  = Entry(master, width = 16, show='*')

       self.oldpw.grid(row=0, column=1, sticky=W)
       self.newpw1.grid(row=1, column=1, sticky=W)
       self.newpw2.grid(row=2, column=1, sticky=W)
```

❶ ❷ ❸

代码注解

❶ 首先，我们创建标签。因为我们不需要保存标签的索引(reference)，所以我们直接应用网格法。我们规定行号，但是列的值是默认的（在这种情况下列值为 0）。固定属性决定了控件在网格中其单元的范围内被附加的地点。固定属性和锚的结合及打包器的扩展选项相似，并且它使用 anchor=W 选项使控件看来象是打包过的控件。

❷ 我们确实需要输入字段的索引，所以我们分开创建它们。

❸ 最后，我们给网格加了输入字段，同时规定了行和列。

让我们回到图像编辑的例子来。如果你为网格中的字段设计了一个规划，看一看需要做什么来产生屏幕是很简单的。看一看图 5.19 来理解区域（area）是怎样被安装网格的。值得注意的重要特点是我们需要同时跨越行和列来给每个控件留出空间。你会发现在将它们提交给代码前，为复杂的网格勾画设计是很方便的。这里是图像编辑器的代码。

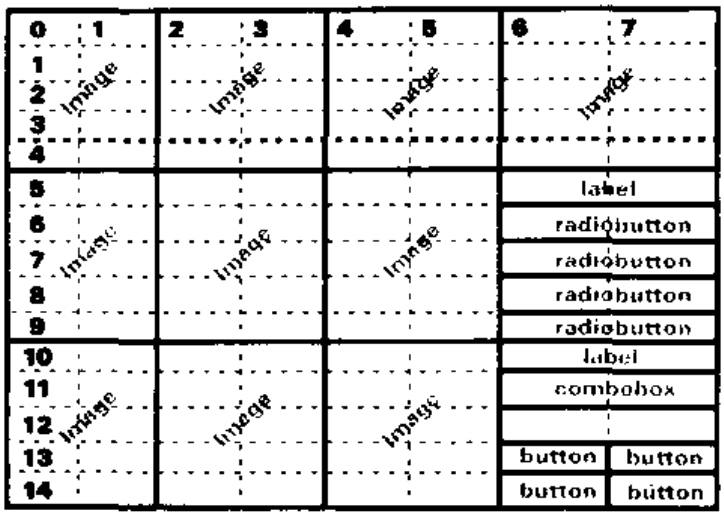


图 5.19　为网格显示设计版面

我已经移去了一些代码，因为我确实想集中于设计而不是应用的操作。这个例子的完整的源代码是可在线使用的。

imageEditor.py

```
from Tkinter import *
import sys, Pmw, Image, ImageTk, ImageEnhance                    ❶

class Enhancer:
    def __init__(self, master=None, imgfile=None):
        self.master = master
        self.masterImg = Image.open(imgfile)                     ❷
        self.masterImg.thumbnail((150, 150))

        self.images = [None]*9
        self.imag   = [None]*9
        for i in range(9):
            image = self.masterImg.copy()
            self.images[i] = image                               ❸
            self.imgs[i] = ImageTk.PhotoImage(self.images[i].mode,
                                              self.images[i].size)

        i = 0
        for r in rang(3):                                        ❹
            for c in range(3):
                lbl = Label(master, image=self.imgs[i])
                lbl.grid(row=r*5, column=c*2,
                    rowspan=5, columnspan=2,sticky=NSEW,
                    padx=5, pady=5)
            i = i + 1

        self.original = ImageTk.PhotoImage(self.masterImg)
```

```
        Label(master,image=self.original).grid(row=0, column=6),
                                rowspan=5, columnspan=2)

        Label(master, text='Enhance', bg='gray70').grid(row=5, column=6,
                                    columnspan=2, sticky=NSEW)

        self.radio = Pmw.RadioSelect(master,labelpos = None,
                          buttontype = 'radiobutton', orient = 'vertical',
                          command = self.selectFunc)

        self.radio.grid(row=6, column=6, rowspan=4, columnspan=2)

# --- Code Removed ----------------------------------------------------

        Label(master, text='Variation',
              bg='gray70').grid(row=10, column=6,
                        columnspan=2, sticky=NSWE)

        self.variation=Pmw.ComboBox(master, history=0, entry_width=11,
                          selectioncommand = self.setVariation,
                          scrolledlist_items=('Fine','Medium Fine','Medium',
                                    'Medium Course','Course'))

        self.variation.selectitem('Medium')

        self.variation.grid(row=11, column=6, columnspan=2)

        Button(master, text='Undo',
              state='disabled').grid(row=13, column=6)

        Button(master, text='Apply',
              state='disabled').grid(row=13, column=7)
        Button(master, text='Reset',
              state='disabled').grid(row=14, column=6)
        Button(master, text='Done',
              command=self.exit).grid(row=14, column=7)

# --- Code Removed ----------------------------------------------------

root = Tk()
root.option_add('*font', ('verdana', 10, 'bold'))
root.title('Image Enhancement')
imgEnh = Enhancer(root, sys.argv[1])
root.mainloop()
```

代码注解

❶ 这个例子用了 Python 图像库（PIL）来制作、显示、提高画像的质量。看附录 G 中的用作文件参考的“Python 图像库”，此文件是支持这个起作用的图像方法库的。

❷ 虽然它在说明网格管理器过程中不重要，我还是放置了一些适当的 PIL 代码来演

示如何使图像操作变得容易。这里，在构造函数中，我们打开主图像，在界限的规定的范围内制作缩略图。PIL 恰如其分的用缩尺制图。

```
self.masterImg = Image.open(imgfile)
self.masterImg.thumbnail((150, 150))
```

❸ 下一步我们制作图像地拷贝，在 3*3 的网格中为每个图像制作 Tkinter PhotoImage placeholder。

❹ 在循环的双精度内部，我们制作了一个标签并把它放到适合的网格单元中，再加上 rowspan 和 columnspan 选项。

```
lbl = Label(master, image=self.imgs[i])
lbl.grid(row=r*5, column=c*2,
     rowspan=5, columnspan=2,sticky=NSEW, padx=5,pady=5)
```

注意在这种情况下，固定选项将图像连接到网格的每一侧，这样，网格被设限来约束图像的大小。这意味着当全部窗口的尺寸被改变时，控件将会随之伸缩。

❺ 同样地，我们用不同的背景给标签安装网格，用固定选项来填充全部可以利用的单元。

```
Label(master, text='Enhance', bg='gray70').grid(row=5, column=6,
                                    Columnspan=2, sticky=NSEW)
```

❻ Pmw 单选选项控件以合适的跨度放置于合适的单元里：

```
self.radio = Pmw.RadioSelect(master, labelpos = None,
                    buttontype = 'radiobutton', orient = 'vertical',
                    command = self.selectFunc)

self.radio.grid(row=6, column=6, rowspan=4, columnspan=2)
```

❼ 最终，我们在它们分配的单元里放置按钮控件。

你已经看到了正在被使用的图像编辑器的一个例子（图 5.17）。当你以不同方法运行另一个图像应用的时候，网格几何管理器的真正优势变得明显了。图 5.20 很好地显示这

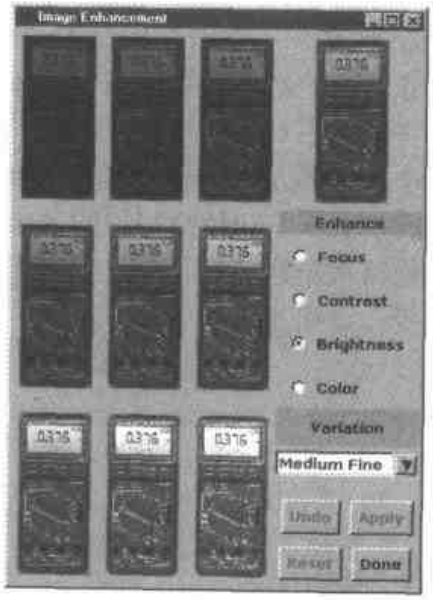


图 5.20　图像编辑器——图像大小缩放

一点：网格可以调整到完全适合图像。而用打包器来创建一个类似的效果需要花费更大的力气。

5.4 放置器

放置器（Placer）几何管理器是 Tkinter 里的可用的管理器中最简单的。一些程序员认为它很难是因为它允许在一个窗口或有关一个窗口控件的精确定位。在本书中你将发现很多这方面应用的例子，所以我能从这个精确度中获益。翻到第 9 章看图 9.5 中的一个有关图形用户界面的例子，这个例子要实现使用包或网格相当困难。因为我们将要看很多例子，所以在这里我仅仅举两个简单的。

让我们从创建如图 5.21 所示的简单的剪贴簿窗口开始。它的作用是显示一些按比例调整来适合于窗口的图像。这些图像是通过单击已编号的按钮来选择的。建造这样的小型应用是相当容易的；我们再一次使用 PIL 来支持图像。



图 5.21 简单剪贴簿

用包来规划窗口是可能的（当然，如果图像跨越大部分列，网格将起作用），但是，当窗口大小被调整的时候，place 会提供一些有用的行为。图 5.21 里的按钮附着于相关位置上，这意味着当窗口的尺寸改变的时候，它们会保持在同样的相关位置上。你用一个实数来表示相关位置，这个实数是用 0.0 来表示最小的 x 或 y、用 1.0 来表示最大的 x 或 y 的。座标轴的最小值习惯于用屏幕左面的 x0 窗口坐标和屏幕上边的 y0 窗口坐标。如果你运行 scrapbook.py，测试一下压缩和拉伸窗口的效果。你会注意到按钮是如何复位的。如果你压缩得过多了会使得按钮碰撞在一起，但是不知为什么用 place 的效果比打包而发生的裁剪更容易被接受。这里是剪贴簿的代码。

scrapbook.py

```
from Tkinter import *
import Image, ImageTk, os

class Scrapbook:
    def __init__(self, master=None):
        self.master = master
```

```
        self.frame = Frame(master, width=400, height=420, bg='gray50',
                     relief=RAISED, bd=4)

        self.lbl = Label(self.frame)
        self.lbl.place(relx=0.5, rely=0.48, anchor=CENTER)

        self.images = []
        images = os.listdir("images")

        xpos = 0.05
        for i in range(10):
            Button(self.frame, text='%d'%(i+1), bg='gray10',
             fg='white', command=lambda s=self, img=i: \
             s.getImg(img)).place(relx=xpos, rely=0.99, anchor=S)
            xpos = xpos + 0.08
            self.images.append(images[i])

        Button(self.frame, text='Done', command=self.exit,
         bg='red',fg='yellow').place(relx=0.99,rely=0.99,anchor=SE)
        self.frame.pack()
        self.getImg(0)

    def getImg(self, img):
        self.masterImg = Image.open(os.path.join("images",
                                    self.images[img]))
        self.masterImg.thumbnail((400, 400))
        self.img = ImageTk.PhotoImage(self.masterImg)
        self.lbl['image'] = self.img

    def exit(self):
        self.master.destroy()

root = Tk()
root.title('Scrapbook')
scrapbook = Scrapbook(root)
root.mainloop()
```

❶ ❷ ❸ ❹ ❺

代码注解

❶ 我们创建包含图像的标签，大致地将它放在窗口的中间并把它固定在中心位置。注意相关位是用容器的长和宽的百分比来表示的。

```
self.lbl.place(relx=0.5, rely=0.48, anchor=CENTER)
```

❷ 我们从图像文件夹得到文件的列表(目录)。

❸ place 确实将自己提供给计算出来的定位应用。在循环中我们创建了一个按钮，给激活的回叫捆绑按钮的索引并将按钮放在下一个可用的位置。

❹ 我们将一个按钮放在屏幕右下方使我们能够退出剪贴簿。注意我们将它固定在 SE 角落。另外要注意这一点：我们要打包外部的框架。打包放置在容器中的一组控件是非常普通的。

打包器做了用外部的容器和窗口管理器之间协调空间的全部工作。

❶ getImg 是个 PIL 代码，它可以载入图像、制作缩略图，并将它载入标签。

除了提供精确的窗口位置，place 也提供了允许程序员根据主窗口的尺寸来规定从属窗口的大小和定位的胶皮。更可能的是我们使用一个并不是从属窗口双亲的主窗口。如果你想跟踪任意的窗口的尺寸，这就非常有用了。与包和网格不同，place 允许你在主（或者同属）窗口之外放置一个窗口。图 5.22 举例说明了这样一个窗口：在一个窗口内每个图像的上方显示图像的一些属性。当图像的尺寸改变时，信息窗口会缩放来适合图像的宽度。



图 5.22　增加跟踪粘帖窗口变化的同属窗口

放置器有另一个重要的属性：与其他的 Tkinter 管理器不同，它并不试图设置主窗口的几何结构。如果你想控制容器控件的尺寸，你必须使用类似有允许你控制其大小的配置选项的框架或画布的控件。让我们来看看可以实现信息窗口的代码。

scrapbook2.py

```
from

Tkinter import *
import Image, ImageTk, os, string

class Scrapbook:
  def __init__(self, master=None):

# --- Code Removed ----------------------------------------------------

        Button(self.fram, text='Info', command=self.info,
         bg='blue',fg='yellow').place(relx=0.99),rely=0.90,anchor=SE)
        self.infoDisplayed = FALSE

  def getImg(self, img):

# --- Code Removed ----------------------------------------------------
```

❶

```
        if self.infoDisplayed:                                          ❷
            self.info();self.info()

    def info(self):
        if self.infoDisplayed:
            self.fm.destroy()                                           ❷
            self.infoDisplayed = FALSE
        else:                                                           ❸
            self.fm = Frame(self.master, bg='gray10')
              self.fm.place(in_=self.lbl, relx=0.5,                     ❺
                    relwidth=1.0, height=50, anchor=S,
                    rely=0.0, y=-4, bordermode='outside')
            ypos = 0.15
            for lattr in ['Format', 'Size', 'Mode']:
                Label(self.fm, text='%s:\t%s' % (lattr,
                    getattr(self.masterImg,
                        '%s' % string.lower(lattr))),
                    bg='gray10', fg='white',
                    font=('verdana', 8)).place(relx=0.3,                ❻
                    rely= ypos, anchor=W)
                ypos = ypos + 0.35
            self.infoDisplayed = TRUE

# --- Code Removed ----------------------------------------------
```

代码注解

❶ 我们加了个按钮来显示图像信息。

❷ 为了使图像信息更新，我们切换了信息显示。

```
Self.info();self.info()
```

❸ 信息方法替换了信息显示。

❹ 如果窗口目前正在被显示，我们就破坏了它。

❺ 另外，我们创建一个新的窗口，将之放在图像上方并设置了它的宽度使之与该图像相匹配。

我们也给 y 位置增加一个负增量来提供很小的空白空间。

```
self.fm.place(in_=self.lbl, relx=0.5,
     relwidth=1.0, height=50, anchor=S,
     rely=0.0, y=-4, bordermode='outside')
```

❻ 信息窗口的输入被渐渐地显示。

5.5 小结

掌握几何管理器是开发产生有吸引力且有效的图形用户界面的能力的重要步骤。当开始用 Tkinter 的时候，大部分的读者将发现网格和包非常易于使用，而且当窗口大小被调整的时候，能够产生最好的结果。要想给控件以非常精确的定位的话，位是更好的选择。不过，这确实需要付出相当多的努力。

在本书中你将看到许多关于使用这三个管理器的例子。记住通常在单个窗口中将几何管理器结合起来是个适当的举措。如果你这样做，你必须注意遵守一些规则，如果事情不好解决，你很可能已经打破了那些规则中的一条！

第6章　事件，捆绑和回叫

图形用户界面应用极其依赖于事件，这样可以给控件附加上事件和这些事件的捆绑回叫功能性。

我预计很多读者可能对这个话题会有点熟悉。

不过，这对于你们中的一些人来说是个新领域，所以我将讲述一些细节来确保这个主题能充分地被覆盖在内。高级的话题将被讨论,包括动态回叫操作者，数据确认技术和“聪明的”控件。

6.1　事件驱动系统：评论

不管应用是否在 Unix、Windows 或者 Macintosh 环境中运行，不知道任何潜在的事件机制而创建一个复杂的图形用户界面应用是很有可能的。不过，如果在你的应用范围内你知道如何去要求和处理事件，发展一个你所希望的那种方式的应用通常是会更容易些。

熟悉X视窗下的事件和事件操作器或者视窗信息的读者可能会希望略过这些而直接阅读 6.2 节的“Tkinter 事件”，因为这是具体讲述 Tkinter 信息的。

6.1.1　事件是什么

事件是视窗系统（例如 X 视窗的 X 服务器）发给客户代码的通知（消息是 Windows 的说法）。它们指出一些事情发生了或者一些受控对象的状态改变了，不仅是因为用户输入也是因为你的代码发出了导致服务器改变的请求。

一般来说，应用不会自动接收事件。不过，你也许不会知道由你的程序间接要求的或控件要求的事件。例如，你可以规定这样一个命令：当按下一个按钮的时候，就调用一个回叫；控件捆绑了一个激活的回叫事件。要求通常在别处处理事件的通知也是可能的。这样做一般允许你的应用改变控件和窗口的行为；这是件好事，但是它破坏了复杂系统的行为，所以需要小心使用它。

所有事件都被置于事件行列中。事件通常从应用的 mainloop 中被一个函数调用所移去。一般来说，你将使用 Tkinter 的 mainloop。但是，如果你有特殊需求（比如需要以使用标准不可能的方法处理内部锁定的线程应用），那么你支持一个专门的 mainloop 就成为可能了。

Tkinter 对事件提供了实现-独立的存取，这样，你不需要过多地了解事件管理者和过滤器。例如，为了检测什么时候光标进入一个框架，试看以下的短例：

Example 6_1.py

```
from Tkinter import *
root = Tk()

def enter(event):
   print 'Entered Frame: x=%d, y=%d' % (event.x, event.y)

frame = Frame(root, width=150, height=150)
frame.bind('<Any-Enter>', enter)           # Bind event
frame.pack()

root.mainloop()
```

框架的捆绑方式被用来连接进入回叫和 Any-Enter 事件。只要光标穿过从框架外部到内部的界线，信息就会被打印出来。

注意 这个例子介绍了一个有趣的问题。根据光标进入框架的速度，你会观察到 x 和 y 坐标显示出一些可变性。这是因为 x 和 y 的值是由事件循环中所加工处理的时间决定而不是由实际事件发生的时间决定的。

6.1.2 事件的传播

事件的发生与窗口有关，该窗口通常被描述为事件的源窗口。如果没有客户为源窗口的特别事件注册，则事件将上传到上一层的窗口。除非它找到客户注册的一个窗口，或者找到一个禁止事件传递的窗口，或者到达了根窗口，否则事件将被忽视。

只有发生作为键、指针运动或鼠标单击结果的装置事件才能被传播。其他的事件，例如暴露(exposure)事件和配置事件，必须清楚地登记下来。

6.1.3 事件类型

事件根据 X 事件掩码被分成几个范畴。当运行视窗体系结构的时候，Tk 把视窗事件映射为相同的掩码。用 Tk 确认的事件掩码（因而是 Tkinter）显示在表 6.1 中。

表 6.1 用来规类 X 事件的事件掩码

NoEventMask	StructureNotifyMask	Button3MotionMask
KeyReleaseMask	SubstructureNotifyMask	Button5MotionMask
ButtonReleaseMask	FocusChangeMask	KeymapStateMask
LeaveWindowMask	ColormapChangeMask	VisibilityChangeMask
PointerMotionHintMask	KeyPressMask	ResizeRedirectMask
Button2MotionMask	ButtonPressMask	SubstructureRedirectMask
Button4MotionMask	EnterWindowMask	PropertyChangeMask
ButtonMotionMask	PointerMotionMask	OwnerGrabButtonMask
ExposureMask	Button1MotionMask	

键盘事件

只要一个键被按下，按键(KeyPress)事件就产生了；只要键被放开，松键(KeyRelease)事件就产生了。类似 Shift 和 Ctrl 的修改键产生了键盘事件。

指针事件

如果鼠标按钮被按下或者鼠标被移动，ButtonPress、ButtonRelease 和 MotionNotify 事件就产生了。

除非一个指针抓（grab）存在，与事件有关联的窗口是层次中最低的窗口，在那种情况下，由指针抓锁定的窗口会被识别出来。如同键盘事件一样，修改键可能会和指针事件结合起来。

交叉事件

只要指针进入或离开窗口边界，EnterNotify 或 LeaveNotify 事件就产生了。交叉是移动指针的结果或窗口的堆栈顺序的变化是无关紧要的。例如，若一个包括指针的窗口在另一个窗口后面的下面，而且现在指针在顶层窗口内，低层窗口接收到一个 LeaveNotify 事件，同时顶层窗口收到一个 EnterNotify 事件。

焦点事件

收到键盘事件的窗口被称作焦点窗口。只要焦点窗口改变了，就产生 FocusIn 和 Focusout 事件。处理焦点事件与处理指针事件相比有点棘手，因为指针不是必需在收到焦点事件的窗口里。通常你并不需要亲自处理焦点事件，因为在控件中将 takefocus 设置为“真”的话，就允许你通过按 Tab 键移动控件之间的焦点。

暴露事件

只要窗口或部分窗口变为可见，暴露事件就产生了。你在 Tkinter 图形用户界面中不是典型地管理暴露事件，但是如果你有一些非常专门的绘图来支持的话，你就有接收到这些事件的能力。

配置事件

当窗口大小、位置或边界改变的时候，ConfigureNotify 事件就产生了。只要窗口的堆栈顺序改变了，一个 ConfigureNotify 事件就会被创建。其他类型的配置事件包括 gravity、map、unmap、Reparent 和 visibility。

颜色映像（colormap）事件

如果新的颜色映像被安装的话，ColormapNotify 事件就产生了。这可能在你的应用中被用来防止当另一个应用安装颜色映像时颜色映像的闪烁。不过，大部分的应用不能直接控制它们的彩绘。

6.2　Tkinter 事件

一般来说，与在 X/Motif、Win32 或 QuickDraw 中做同样的事情相比，在 Tkinter 应用中处理事件被认为容易些。Tkinter 给特殊事件捆绑回叫提供了便利的方法。

我们使用下列格式将事件表达为字符串：

```
<modifier-type-qualifier>
```

- *modifier* is optional and may be repeated, separated by spaces or a dash.
- *type* is optional if there is a qualifier.
- *qualifier* is either a *button-option* or a keysym and is optional if type is present.

许多事件恰好可用类型来描述，所以，可能修饰语和限定词就被省略了。类型定义受约束（在 X 的术语中它定义事件的掩码）事件的类。很多事件可能被输入了速记表。举个例子来说，<Key-a>，<Key-press-a>和 a 都是可接受的用来按小写字母的 a 标识符。

这里是一些更常用的事件。在附录 E 中你会发现一个完整的事件和键盘系统表。

事件	Alt.1	Alt2	Mod	类型	限值	产生事件的行为
<Any-Enter>			Any	Enter		Enter event regardless of mode.
<Button-1>	ButtonPress-1	1		Button	1	Left mouse button click.
<Button-2>	ButtonPress-2	2		ButtonPress	1	Middle mouse button click.
<B2-Motion>			B1	Motion		Mouse movement with middle mouse button down.
<ButtonRelease-3>				ButtonRelease	3	Release third mouse button 3.
<Configure>				Configure		Size stacking or position has changed.
<Control-Insert>			Control		Insert	Press INSERT key with CONTROL key down.
<Control-Shift-F3>			Control-Shift		F3	Press CONTROL-SHIFT and F3 keys simultaneously.
<Destroy>				Destroy		Window is being destroyed.
<Double-Button-1>			Double	Button	1	Double-click first mouse button 1.
<Enter>				Enter		Cursor enters widow.
<Expose>				Expose		Window fully or partially exposed.
<FocusIn>				FocusIn		Widget gains focus.
<FocusOut>				FocusOut		Widget loses focus.
<KeyPress>	Key			KeyPress		Any key has been pressed.
<KeyRelease-back-slash>				KeyRelease	Backslash	Backslash key has been released.
<Leave>				Leave		Cursor leaves window.
<Map>				Map		Window has been mapped.
<Print>					Print	PRINT key has been pressed.
z					z	Captal Z has been pressed.

让我们看一些代码例，当它用 Tkinter 支持时，它允许我们研究事件机制。

Example 6.2.py

```
from Tkinter import *
import Pmw

eventDict = {
             '2': 'KeyPress', '3': 'KeyRelease', '4': 'ButtonPress',
            '5': 'ButtonRelease', '6': 'Motion', '7': 'Enter',
            '8': 'Leave', '9': 'FocusIn', '10': 'FocusOut',
            '12': 'Expose', '15': 'Visibility', '17': 'Destroy',
            '18': 'Unmap', '19': 'Map', '21': 'Reparent',
            '22': 'Configure', '24': 'Gravity', '26': 'Circulate',
            '28': 'Property',  '32': 'Colormap','36': 'Activate',
            '37': 'Deactivate'
}

root = Tk()

def reportEvent(event):
    rpt = '\n\n%s' % (80*'=')
    rpt = '%s\nEvent: type=%s (%s)' %  (rpt, event.type,
                            eventDict.get(event.type, 'Unknown'))
    rpt = '%s\ntime=%s'   %           (rpt, event.time)
    rpt = '%s  widget=%s' %           (rpt, event.widget)
    rpt = '%s  x=%d, y=%d'%           (rpt, event.x, event.y)
    rpt = '%s  x_root=%d, y_root=%d' % (rpt, event.x_root, event.y_root)
    rpt = '%s  y_root=%d' %           (rpt, event.y_root)
    rpt = '%s\nserial=%s' %           (rpt, event.serial)
    rpt = '%s  num=%s' %              (rpt, event.num)
    rpt = '%s  height=%s' %           (rpt, event.height)
    rpt = '%s  width=%s' %            (rpt, event.width)
    rpt = '%s  keysym=%s' %           (rpt, event.keysym)
    rpt = '%s  ksNum=%s' %            (rpt, event.keysym_num)

    #### some event types don't have these attributes
    try:
        rpt = '%s  focus=%s' %   (rpt, event.focus)
    except:
        try:
            rpt = '%s  send=%s' %    (rpt, event.send_event)
        except:
            pass

    text2.yview(END)
    text2.insert(END, rpt)

frame = Frame(root, takefocus=1, highlightthickness=2)
text  = Entry(frame, width=10, takefocus=1, highlightthickness=2)
text2 = Pmw.ScrolledText(frame)
```

```
for event in eventDict.values():
    frame.bind('<%s>' % event, reportEvent)
    text.bind('<%s>' % event, reportEvent)

text.pack()
text2.pack(fill=BOTH, expand=YES)
frame.pack()
text.focus_set()
root.mainloop()
```

代码注解

❶ eventDict 定义 Tkinter（严格地 Tk）所承认的所有的事件类型。不是所有由 X 定义的事件掩码都对可以直接用于 Tkinter 应用。所以，你将会看到可数事件类型的值非常少。

```
'12': 'Expose', '15': 'Visibility', '17': 'Destroy',
```

当事件被发现的时候，字典也被用于循环查找事件-类型的名称。

❷ reportEvent 是事件的管理者。

它有责任把数据事件格式化，然后将事件类型从 eventDict 中取出；如果发生未被识别的事件，我们将它标为未知。

```
def reportEvent(event):
    rpt = '\n\n%s' % (80*'=')
    rpt = '%s\nEvent: type=%s (%s)' % (rpt, event.type,
                                       eventDict.get(event.type,'Unknown'))
```

❸ 不是所有的事件都提供焦点和 send_event 属性，所以我们应恰当地处理 AttributeErrors。

❹ 最后，我们把每个事件和框架与进入控件的 reportEvent 回叫捆绑在一起：

```
for event in eventDict.values():
    frame.bind('<%s>' % event, reportEvent)
    text.bind('<%s>' % event, reportEvent)
```

图 6.1 显示了运行 Example_6_2.py 的结果。显示的事件说明了键入 Shift-M 的结果。你可以看到紧接着相应的松键事件的 Shift 键和 M 键的按键。

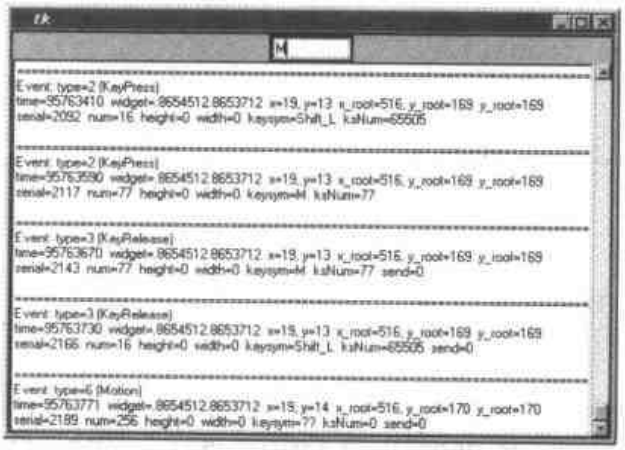


图 6.1 事件监视器

注意　如果你对处理事件还不熟悉，你也许会发现，运行 Example_6_2.py 在窗口内完成一些简单的任务来研究系统的行为很有用。例如，按下 Shift 键就创建了一个事件流；移动鼠标在更频繁的频率上创建了一个运动事件流。

最初，这可能会令人惊奇，因为用户（还有程序员）通常是看不见这些事件的。知道这个行为和当你编程来考虑事件事实上是如何被产生的都是重要的。确认回叫没有做任何密集的处理是特别重要的，否则，会很容易引起严重的执行问题。

6.3　回叫

回叫只是作为事件产生的结果那样被调用的函数。不过，刚开始对 Tkinter 程序员来说处理参数可能是个问题，它们可能会成为 bug(故障)的来源，甚至对经验丰富的程序员来说也是这样。

参数的数目是由被处理的事件类型及你是否直接或间接的将回叫与事件捆绑在一起决定的。这里是一个直接捆绑的例子。

```
btn = Button(frame, text='OK', command=buttonAction)
```

指令真是按钮控件提供的方便功能，当控件被激活的时候，它调用 buttonAction 回叫。这通常是<Button-press-1>事件的结果，但是，如果控件有焦点的话，<KeyPress-space>也能起同样的作用。不过，要知道很多事件是作为在按钮被激活之前移动和定位鼠标的结果发生的。

我们能通过直接捆绑得到相同的效果：

```
btn.bind('Button-1>', buttonAction)
btn.bind('<KeyPress-space>', buttonAction)
```

那么，差别在哪儿呢？

唔，除了直接捆绑事件的代码的附加线以外，真正的差异是在回叫的调用中。如果回叫从事件中被调用，事件对象将作为回叫的第一个（仅在本例中）参数被传递。

注意　如果你不彻底测试你的应用，事件管理器将会是潜在的 bug 的根源（即事件管理器将会导致 bug 的产生）。如果事件与一个回叫捆绑（故意地或错误地），并且这个回叫不期望事件对象被作为参数传递，应用就可能被破坏了。如果这种事件很少出现，或者难以在测试的过程中模拟，这种情况更有可能发生。

如果你想再使用 buttonAction 并且调用它来响应直接或间接的事件，你必须写回叫。这样，它才能接受可变参数：

```
def buttonAction(event=None):
```

```
    if event:
        print 'event in: %s' % event.type
    else:
        print 'command in: '
```

当然，这确实增加复杂性。尤其是如果函数已经有了参数，因为你将必须决定第一个参数是事件对象还是规则参数。

6.4 λ表达

哦不！不要又是那令人恐惧的λ！虽然早前在书中提起过λ，并广泛地用于例子中，但在我们进行下一节之前我们必须先看看λ的用法。

术语λ最初来源于 Alonzo Church 的λ微积分学。现在你会发现，λ被用于一些泛函编程规程中特殊的上下文。Python 里的λ被用来定义一个看来是解释程序声明的匿名函数。以这种方式你能设置执行码的简单线路，但通常这种方式并不成立。

看看这个代码段：

```
var  = IntVar()
value = 10
...
btn.bind('Button-1',(btn.flash(),var.set(value)))
```

粗看一下粗体行并不会引起任何注意，但是，在运行时该行将会失效。这个目的是使被单击的按钮闪烁，把一些事先确定的值放入变量。事实上发生的是：当实行捆绑方式时，两个调用都将被调用。稍后，当按钮被单击的时候，我们将得不到预期效果，因为回叫字典正好包含了这两种方法的返回值，这个例子中是(None,None)。此外，我们本可以错过事件对象——它总是作为回叫里的第一个参数——我们有可能收到一个运行时间错误。这里是捆绑这个回叫的正确方法：

```
btn.bind('Button-1', lambda event, b=btn, v=var, I=value:
     (b.flash(), v.set(i)))
```

注意事件参数（在这个代码段中被忽略了）。

如果你不喜欢λ表达，要完全避开λ，有其他方法可以延迟你的函数调用。

Timothy R. Evans 给定义包起函数的指令种类的 Python 新闻组提供了一个建议。

```
class Command:
    def __init__(self, func, *args, **kw):
        self.func = func
        self.args = args
        self.kw = kw
    def __call__(self, *args, **kw):
        args = self.args + args
        kw.update(self.kw)
        apply(self.func,args, kw)
```

然后，像这样定义回叫：

```
Button(text='label', command=Command(function,arg [, moreargs…])))
```

被传递给指令种类的函数和参数（包括关键字）的参照是由其构造器存储的，而当

回叫被激活的时候是对函数进行传递的。对读和维持来说，这个用来定义回叫的格式可能比λ表达容易些。最少它有选择性！

6.5 捆绑事件和回叫

迄今为止的例子已经演示了如何捆绑一个事件管理器和控件的实例，这样，其接受事件的行为将不会继承控件的其他实例。Tkinter给在几个层内捆绑提供了灵活性：

1. 在应用层：这样，只要一个应用中的窗口有焦点，同样的捆绑就可以用于应用中的所有的窗口和控件。
2. 在类层：这样，最少在开始时全部控件的实例都有相同的行为。
3. 在外层（顶层或根）。
4. 在实例层，如已记录的那样。

应用层和类层的捆绑事件必须被慎重对待，因为它很容易在你的应用中创建出乎意料的行为。

特别是，类层中的不加选择的捆绑可以解决直接的问题，但是在应用中加入一个新功能性的时候就会引起新问题。

注意 避免引起控件或界面中用户都熟悉的高非规范行为一般是个好的实践。例如，创建一个进入字段进行反向填充（输入123显示的是321）的捆绑是很容易的，但是这不是典型的进入行为，会使用户感到迷惑。

6.5.1 捆绑方法

在附录B中的“普通选项”中，你会发现更多有关捆绑和非捆绑的信息。所以，在这一节中，我仅结合上下文的4种捆绑层来阐述一下捆绑方法。

应用层

应用经常用F1来给予帮助。在应用层捆绑这个keysym意味着：当某一个应用窗口有焦点，按F1就会显示弹出一个帮助屏幕。

类层

在类层捆绑让你确定这一点：类是一起穿越一个应用的。实际上，Tkinter用这种方法捆绑来给控件提供标准捆绑方法。例如，如果你要实现新控件的话，可能你会用到类捆绑，或者你可以给跨越应用的进入字段提供音频反馈。

顶层窗口层

如果焦点在外层的任何一部分，那么在根层捆绑一个函数就允许产生一个事件。例如，这可以被用来捆绑一个屏幕打印功能（函数）。

实例层

我们已经看到了这几个例子，所以现阶段我们将不再说更多的了。

下列假设的例子举例说明了所有这4个捆绑模式。

Example 6_3.py

```
from Tkinter import *
def displayHelp(event):

def displayHelp(event):                          ❶
    print 'hlp', event.keysym

def sayKey(event):
    print 'say',event.keysym, event.char

def printWindow(event):
    print 'prt', event.keysym

def cursor(*args):
    print 'cursor'

def unbindThem(*args):
    root.unbind_all('<F1>')                      ❷ unbind all bindings
    root.unbind_class('Entry', '<KeyPress>')
    root.unbind('<Alt_L>')
    root.unbind('<Control-Shift-Down>')
    print 'Gone...'

root = Tk()
frame = Frame(root, takefocus=1, highlightthickness=2)
text  = Entry(frame, width=10, takefocus=1, highlightthickness=2)

root.bind_all('<F1>', displayHelp)

text.bind_class('Entry', '<KeyPress>', lambda e, x=101: sayKey(e))   ❸ ❹
root.bind('<Alt_L>', printWindow)                ❺

frame.bind('<Control-Shift-Down>' , cursor)

text.bind('<Control-Shift-Up>', unbindThem)      ❻

text.pack()
frame.pack()
text.focus_set()
root.mainloop()
```

代码注解

❶ 首先定义回叫这些全是简单的例子，除了最后作为回叫参数被传递的考虑事件对象的那个，我们从中可以提取键产生事件的 keysym。

```
def displayHelp(event):
    print 'hlp', event.keysym
```

❷ 尽管类-层捆绑对进入控件使用了方法调用，bind_class 是一个继承关系，所以任

何实例都能起作用。并且，root.umbind_class 是可以接受的。这对于实例捆绑来说是不正确的，那仅局部于实例。

❸ 我们做了个应用层的捆绑：

```
root.bind_all('<F1>', displayHelp)
```

❹ 在这个类层捆绑中我们用 λ 函数来给回叫构建一个参数表：

```
text.bind_class('Entry', '<KeyPress>', Lambda e, x=101: sayKey(e,x))
```

❺ 在这里我们为打印屏回叫做了个顶层捆绑：

```
root.bind('<Alt_L>', printWindow)
```

❻ 最后，我们用双倍的修饰语进行实例捆绑：

```
frame.bind('<Control-Shift-Down>', cursor)
text.bind('<Control-Shift-Up>', unbindThem)
```

注意 如果你将四个重复捆绑的捆绑层结合使用，那么你要为处理事件的多重回叫做好准备。

Tkinter 在每一层选择最佳的捆绑。从任何实例捆绑开始，然后是跟在任何类捆绑后面的顶层捆绑，最后选择应用层捆绑。这允许你在任何层都不必考虑捆绑。

6.5.2 处理多重捆绑

如我在上面的注意中所提到的，你能在四个捆绑事件的任何一个中捆绑事件。不过因为事件是被传递的，所以可能不一定能得出你所希望的行为。

对一个简单的例子来说，假定你不考虑控件的行为，你想在控件中插入\h，而不是用退格键移去之前的字符。

```
Text.bind('<BackSpace>', lambda e: dobackspace(e))
```

你可以像这样设置捆绑：

```
def dobackspace(event):
    event.widget.insert(END, '\\h')
```

不巧的是，这不起作用，因为事件是与应用层捆绑在一起的。控件仍与退格键有个捆绑，所以在应用层被调用和\h 被插入控件之后，事件就被传递给类层并且 h 被移去。

有个简单的解决方案：从最后一个你希望从中传递事件的事件管理者那里返回“break”比较高一层并不得到事件。回叫看起来就是这样：

```
def dobackspace(event):
    event.widget.insert(END, '\\h')
    return "break"
```

6.6 定时器和背景程序

mainloop 支持不从事件产生的回叫。这一做法的最重要的结果是：这使得在事先预

定的延迟之后或者只要图形用户界面闲置时很容易设置调用回叫的定时器。这里是在本书后面地方一个例子的代码片段：

```
if self.blink:
    self.frame.after(self.blinkrate * 1000, self.update)

def update(self):
  # Code removed
  self.canvas.update_idletasks()
  if self.blink:
      self.frame.after(self.blinkrate * 100, self.update)
```

这个代码将在 self.blinkrate * 1000ms 之后调用 self.update 设置。回叫做它该做的并设置为再次调用它本身（这些定时器仅被调用一次，如果你想使它们再次计时，你必须再次设置它们）。

想知道更多有关定时器的信息，看附录 B 中的“普通选项”。

6.7 动态回叫管理者

简单的回叫经常被捆绑成一个应用时间的事件。不过，我们需要改变控件的捆绑来支持应用需求有很多种情况。一个例子是：当一个字符输入时，可以依附一个回叫来消去字段上的反向视频（被用来作一个确认错误的结果）。

使动态回叫运行仅仅是个捆绑和拆分事件的问题。我们可以 6.5.1 节中的 Example_6_3.py 里看到有关的例子，而源代码里有些其他例子。

注意 如果你发现你在代码中经常捆绑和拆分事件，那么回过头来检查一下你为什么要这样做可能是个好办法。记住，事件能够快速连续的产生——比如，鼠标移动会产生旋转运动。在事件风暴里的改变捆绑能产生不可预测的结果并且难以调试。当然，我们也会烧坏中央处理器(CPU)，所以它能在应用特性中产生相当大的影响。

6.8 使事件运作

在前些章中，我们看到了一些在应用中用数据来设置控件、得出和使用那个数据的例子。在第 8 章的“对话框和窗体”中，我们将看到一些出示和得到数据的计划。这是个重要话题，它可以在你那部分上要求一些独创性（灵活性）来设计正确的行为。在以下几段中，我将给出一些办法来帮助你们解决你们自己的需求。

6.8.1 捆绑动态数据与控件

Tkinter 提供了简单的机制来捆绑变量必须从变量类中再细分类;其中一些是预先定义了的,如有必要,你可以自己定义它们。只要变量改变了，孔径值就会更新为新的值。看这个简单的例子。

Example 6 4.py

```
from Tkinter import *
root = Tk()

class Indicator:
    def __init__(self, master=None, label='', value=0):
        self.var = BooleanVar()
        self.i = Checkbutton(master, text=label, variable = self.var,
                             command=self.valueChanged)
        self.var.set(value)
        self.i.pack()

    def valueChanged(self):
        print 'Current value = %s' % ['Off','On'][self.var.get()]

ind = Indicator(root, label='Furnace On', value=1)
root.mainloop()
```

这个例子定义了 self.var 并将它与控件变量捆绑在一起；它也定义了一个回叫，只要控件的值改变即可被调用。在这个例子中控件的值是由单击复选按钮而改变的——它等价于计划性地设置。

作为一个外部变化的结果，设置值是合理的方案，但是如果数据改变太快的话，它会引入性能问题。如果我们的图形用户界面包括了很多显示系统状态和组件值的控件，并且如果这些值被异步地改变(例如，每个值都作为系统网络分析程序中断到达系统)，过度经常地更新控件会引起应用性能上相反的效果。这里是一个简单图形用户界面的可能的实现，它通过十台传感器报告来监视温度。

Example 6 5.py

```
from Tkinter import *
import random
root = Tk()

class Indicator:
    def __init__(self, master=None, label='', value=0.0):
        self.var = DoubleVar()                                    ❶
        self.s = Scale(master, label=label, variable=self.var,   ❷
                       from_=0.0, to=300.0, orient=HORIZONTAL,
                       length=300)
        self.var.set(value)                                       ❸
        self.s.pack()

def setTemp():
    slider = random.choice(range(10))                             ❹
    value  = random.choice(range(0, 300))
    slist[slider].var.set(value)                                  ❺
    root.after(5, setTemp)                                        ❻
```

```
slist = []
for i in range(10):
    slist.append(Indicator(root, label='Probe %d' % (i+1)))

setTemp()                                                    ❼
root.mainloop()
```

代码注解

❶ 首先我们创建一个 Tkinter 变量。我们为这个例子存储了一个实数值：

```
self.var = DoubleVar()
```

❷ 然后我们将它与 Tk 变量捆绑：

```
self.s = Scale(master, label=label, variable=self.var,
                                        self.var.set(value)
```

❸ 然后我们就设置其值。这能立即更新这个控件使之显示新值。

❹ setTemp 函数的目的是在 5ms 间隔内为“传感器”之中的 1 台随机地创建一个值。

❺ 每变化一次，变量就被更新一次：

```
slist[slider].var.set(value)
```

❻ 因为 after 是只限一次的计时器，所以我们必须设置下一个超时：

```
root.after(5, setTemp),
```

❼ 调用 setTemp 启动了传感器信息的模拟流。

这个例子不再在这里显示（当然代码在线可用）。不过，显示的行为类似控件经常显示新值的 Brownian 运动。在“真正的”应用中的，更新率将会使用户厌烦，而且它需要减速来创造一个合理的更新率。另外，频繁地刷新控件会消耗格外高的 CPU cycles。比较 Example_6_6.py 与 Example_6_5.py 的代码。

Example_6_6.py

```
from Tkinter import *
import random
root = Tk()

class Indicator:
    def __init__(self, master=None, label='', value=0.0):
        self.var = DoubleVar()
        self.s = Scale(master, label=label, variable=self.var,
                       from_=0.0, to=300.0, orient=HORIZONTAL,
                       length=300)
        self.value = value                                   ❶
        self.var.set(value)
        self.s.pack()
        self.s.after(1000, self.update)                     ❻

    def set(self, value):
        self.value = value                                   ❸
```

```
    def update(self):
        self.var.set(self.value)
            self.s.update_idletasks()          ❹
            self.s.after(1000, self.update)

def setTemp():
    slider = random.choice(range(10))
    value  = random.choice(range(0, 300))
    slist[slider].set(value)                   ❺
    root.after(5, setTemp)

slist = []
for i in range(10):
  slist.append(Indicator(root, label='Probe %d' % (i+1)))

setTemp()
root.mainloop()
```

代码注解

❶ 不仅是 Tkinter 变量，我们也为控件的当前值创建了实例变量：

```
self.value=value
```

❷ 安排了 after 超时，在 1 秒内调用 update 方法：

```
self.s.after(1000,self.update)
```

❸ 类定义了一套方法来设置当前值。

❹ 更新方法用当前值来设置 Tkinter 变量，更新控件显示。为了刷新控件，我们调用了在事件队列中等待处理事件的 update_idletasks。

```
self.s.update_idletasks()
```

❺ 现在，当值改变的时候，我们设定了实例变量：

```
slist[alider].set(value)
```

现在显示器每秒更新一次控件，这导致了更不严格的显示和显著降低了 CPU 的总开销。如果你希望进一步降低总开销的话，你可以进一步优化代码。举例来说，从 one-per-widget 调用来更新控件还不如从简单的更新超时更新。

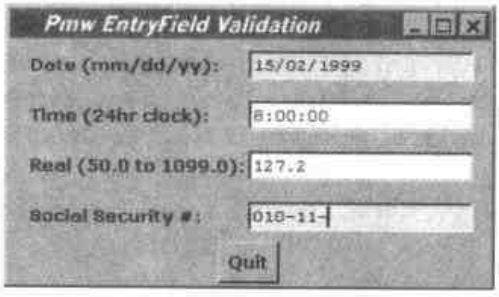


图 6.2　输入域确认（例_6_7.py）

6.8.2 数据确认

实行数据输入的图形用户界面的一个重要部分被适当的输入值所检验。这一块会消耗程序员相当多的时间和精力。有许多方法来确认输入，但是我们在这里不能涵盖全部。

Pmw 进入字段控件为类似日期、时间和数字字段的共同进入字段类型提供了一个内置确认程序。使用这些工具可以节省你相当多的时间。这里是一个使用 Pmw 确认的简单例子。

Example 6_7.py

```
import time, string
from Tkinter import *
import Pmw

class EntryValidation:
    def __init__(self, master):
        now = time.localtime(time.time())
        self._date = Pmw.EntryField(master,
                    labelpos = 'w', label_text = 'Date (mm/dd/yy):',
                    value = '%d/%d/%d' % (now[1], now[2], now[0]),
                        validate = {'validator':'date',
                            'format':'mdy', 'separator':'/'})

        self._time = Pmw.EntryField(master,
                    labelpos = 'w', label_text = 'Time (24hr clock):',
                    value = '8:00:00',
                        validate = {'validator':'time',
                            'min':'00:00:00', 'max':'23:59:59',
                            'minstrict':0, 'maxstrict':0})

        self._real = Pmw.EntryField(master,
                    labelpos = 'w', value = '127.2',
                        label_text = 'Real (50.0 to 1099.0):',
                    validate = {'validator':'real',
                            'min':50, 'max':1099,
                            'minstrict':0},
                    modifiedcommand = self.valueChanged)

        self._ssn = Pmw.EntryField(master,
                    labelpos = 'w', label_text = 'Social Security #:',
                    validate = self.validateSSN, value = '')

        fields = (self._date, self._time, self._real, self._ssn)

        for field in fields:
            field.pack(fill='x', expand=1, padx=12, pady=8)
```

❶

❶

● Setup real field

●

```
        Pmw.alignlabels(fields)                                    ❹
        self._date.component('entry').focus_set()

    def valueChanged(self):
        print 'Value changed, value is', self._real.get()
    def validateSSN(self, contents):
        result = -1
        if '-' in contents:
            ssnf = string.split (contents, '-')
            try:
                if len(ssnf[0]) == 3 and \
                    len(ssnf[1]) == 2 and \
                    len(ssnf[2]) == 4:
                      result = 1
            except IndexError:
                result = -1
        elif len(contents) == 9:
                result = 1
        return result

if __name__ == '__main__':
    root = Tk()
    root.option_add('*Font', 'Verdana 10 bold')
    root.option_add('*EntryField.Entry.Font', 'Courier 10')
    root.option_add('*EntryField.errorbackground', 'yellow')
    Pmw.initialise(root, useTkOptionDb=1)

    root.title('Pmw EntryField Validation')
    quit = Button(root, text='Quit', command=root.destroy)
    quit.pack(side = 'bottom')
    top = EntryValidation(root)
    root.mainloop()
```

❺

❻

❼

代码注解

❶ 日期字段使用内置的日期确定器，指定数据和分离器的格式：

```
validate = {'validator': 'date',
        'format': 'mdy', 'separator': '/'})
```

❷ 时间字段和 minstrict 和 maxstrict 一起设置最大值和最小值选项：

```
validate = {'validator': 'time',
        'min': '00:00:00', 'max': '23:59:59',
        'minstrict':0, 'maxstrict':})
```

设置 minstrict 和 maxstrict 为“假（False）”（0）允许设置最小和最大范围外部的值。背景将变色来指出错误。如果它们被设置为“真”，范围外部的值将不能被输入。

❸ 社会安全字段使用用户支持的确定器：

```
validate = self.validateSSN, value = '')
```

❹ Pmw 提供了便利的方法来排列标签。这减少了在几何管理器中设置补充性格式化的必要。

```
Pmw.alignlabels(fields)
self._date.component('entry').focus_set()
```

在数据输入屏中设置一个第一个可编辑字段的输入焦点始终是个好主意。

❺ validateSSN 方法很简单：它寻找由破折号分隔的三个组群或特性。

❻ 因为进入是累积的，在第 3 个组群被输入之前，string.split 调用将会失败。

❼ 我们设置了 Tk 选项数据库来覆盖 Pmw 控件中应用全部控件的字体和颜色。

```
root.option_add('*Font', 'Verdana 10 bold')
root.option_add('*EntryField.Entry.Font', 'Courier 10')
root.option_add('*EntryField.errorbackground', 'yellow')
Pmw.initialise(root, useTkOptionDb=1)
```

这种结构在很多例子中都能见到。不过，这对 Pmw.initialise 来说是个很少应用到的选项来强制应用 Tk 选项数据库。

运行 Example_6_7 diplays 出现了一个与图 6.2 相似的屏幕。注意有阴影背景数据和社会安全字段是如何显示它们包括无效的格式。

虽然这种确认是由 Pmw 进入字段控件自动提供的，它仍有一些缺点。

（1）没有实际确认错误的迹象。用户被要求自行决定错误的原因。

（2）可用的有效数据完整时，它如同照原样在错误中被放入的那样显示（图 6.2 里的社会安全字段就是个好的例子）。

（3）在确认需要复杂的计算和对服务器和数据库等进行存取处，处理负担很高。这是在某些环境中性能问题的根源。

为了克服这些和其他的问题你可以使用替代的方法。当然，你的应用不可以使用 Pmw 控件，这样，另一个方法可能就会被需要了。

注意 就我个人而言，我倒是希望不要在 Pmw 控件中使用内置的确认控件。如果格式化内容的行为需要重写，你会发现些恼人的显示故障，特别是如果系统负荷过重。这些都可能会烦扰用户。下列方法可以避开这些问题。

为了避免确认每一个击键（这是 Pmw 进入字段管理数据输入的方法），我们安排了在下列情况下做确认：

1 当用户从当前字段中（域）将鼠标指针移开的时候。

2 当焦点是移自使用 Tab 键的字段（域）的时候。

3 当按下 Enter 键的时候。

用这种方法确认意味着当一个输入字符串被建立时，你不会得到假的错误。例如在图 6.3 中，当字段被留下或 Return 被按下的时候，输入 192.311.40.10 将引起一个确认错误，因此减少了操作混乱和中央处理器总开销。

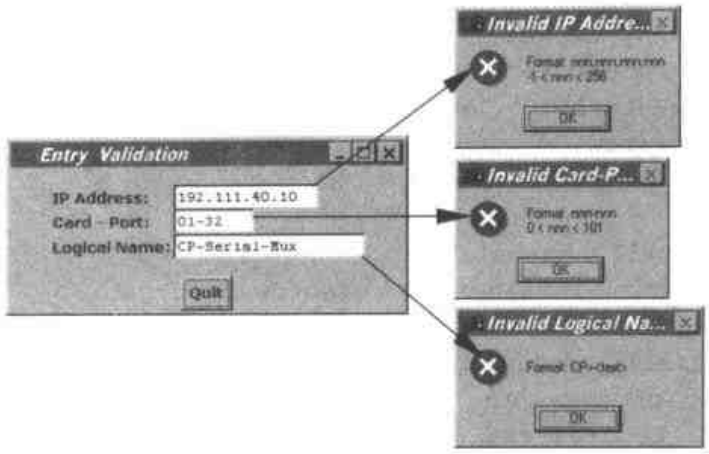


图 6.3　数据确认：错误对话框

Example 6 8.py

```
import string
from Tkinter import *
from validation import *

class EntryValidation:
    def __init__(self, master):
        self._ignoreEvent = 0
        self._ipAddrV = self._crdprtV = self._lnameV = ''

        frame = Frame(master)
        Label(frame, text='   ').grid(row=0, column=0,sticky=W)
        Label(frame, text='   ').grid(row=0, column=3,sticky=W)

        self._ipaddr = self.createField(frame,width=15,row=0,col=2,
                       label='IP Address:',valid=self.validate,
                       enter=self.activate)
        self._crdprt = self.createField(frame,width=8,row=1,col=2,
                       label='Card - Port:',valid=self.validate,
                       enter=self.activate)
        self._lname = self.createField(frame,width=20,row=2,col=2,
                      label='Logical Name:',valid=self.validate,
                      enter=self.activate)

        self._wDict = {self._ipaddr: ('_ipAddrV', validIP),
                       self._crdprt: ('_crdprtV', validCP),
                       self._lname:  ('_lnameV',  validLName) }

        frame.pack(side=TOP, padx=15, pady=15)

    def createField(self, master, label='', text='', width=1,
                    valid=None, enter=None, row=0, col=0):
        Label(master, text=label).grid(row=row, column=col-1, sticky=W)
```

```
        id = Entry(master, text=text, width=width, takefocus=1)
        id.bind('<Any-Leave>', valid)
        id.bind('<FocusOut>', valid)
        id.bind('<Return>',   enter)
        id.grid(row=row, column=col, sticky=W)
        return id

    def activate(self, event):
        print '<Return>: value is', event.widget.get()

    def validate(self, event):
        if self._ignoreEvent:
            self._ignoreEvent = 0
        else:
            currentValue = event.widget.get()
            if currentValue:
                var, validator = self._wDict[event.widget]
                nValue, replace, valid = validator(currentValue)

                if replace:
                    self._ignoreEvent = 1
                    setattr(self, var, nValue)
                    event.widget.delete(0, END)
                    event.widget.insert(0, nValue)
                if not valid:
                    self._ignoreEvent = 1
                    event.widget.focus_set()

root = Tk()
root.option_add('*Font', 'Verdana 10 bold')
root.option_add('*Entry.Font', 'Courier 10')
root.title('Entry  Validation')

top = EntryValidation(root)
quit = Button(root, text='Quit', command=root.destroy)
quit.pack(side = 'bottom')

root.mainloop()
```

代码注解

❶ 有时表格几何管理器需要点帮助来布置屏幕。在这个例子中，我们用了个空的 first 和 last 列：

```
Label(frame, text='   ').grid(row=0, column=0,sticky=W)
Label(frame, text='   ').grid(row=0, column=3,sticky=W)
```

如果列（或者行）为空，你就不能用表格管理器的 minsize 选项；你必须用这里显示的技术。作为选择，你可以把安装表格的控件打包入框架并在旁边用填料(padding)添加空间。

❷ 因为我们使用本国的 Tkinter 控件，所以我们必须给表格的每一行创建标签和进

入控件并将它们放置到合适的列中。我们用创建域(createField)方法来做到这一点。

❸ 我们制作了字典来定义被用来存储每个控件内容的变量。

使用字典能使我们对有多重效验简单的事件-管理者进行捆绑，这样就简化了代码。

```
self._wDict = {self._ipaddr: ('_ipAddrV', validIP),
               self._crdprt: ('_crdprtV', validCP),
               self._lname:  ('_lnameV',  validLName) }
```

❹ 对确认的捆绑是光标离开控件和焦点消失的时间。（在字段外设标记）当 Enter 键被按下时，我们也捆绑被激活的函数调用。

```
id.bind('<Any-Leave>', valid)
id.bind('<FocusOut>', valid)
id.bind('<Return>',   enter)
```

❺ 使用这种确认计划类型的因素之一是：只要字段失去焦点，其确定器就被调用——包括我们什么时候返回字段来允许用户改正错误。我们提供了个机制来忽略事件：

```
if self._ignoreEvent:
    self._ignoreEvent = 0
```

❻ 我们得到了制作事件控件的变量和确定器：

调用确定器来检查控件内容——适当的时候可能会编辑内容。

```
var, validator = self._wDict[event.widget]
nValue, replace, valid = validator(currentValue)
```

❼ 最后，我们要对确认的结果、设置的控件内容起反应。在确认错误的情况下，我们给控件重置了焦点。这里我们设置了标记来忽略作为结果的事件。

```
Self._ignoreEvent=1
```

6.8.3　格式化（smart）控件

几个数据输入格式从照原样格式数据的控件中获益。一些例子包括日期、时间、电话号码、社会安全数字和因特网（IP）地址。做这项工作可以重新引出一些被早前的例子解决过的问题，因为控件的理想行为不断地更新不同的版本来更换输入之后重新格式化字段的计划。这引进了更多的问题。例如，就输入电话号码来说，几个号码分组是典型的：

1. 1-（401） 111-2222 带有区号的完整号码
2. 1-401-111-2222 带有虚线的完整号码
3. 401 111-2222 不带 1 的地区号码
4. 111-2222 区号号码
5. 017596-4752222 国际（美国）
6. 3-1111-2222 国际（日本）

加上如此多的组合，显示在用户面前的控件标签里电话号码或其他数据的格式很重要。如果你的应用需要提供相矛盾的格式范围，最好在字符串完全进入或离开对用户的格式化之后进行格式化。对日期和时间字段来说，你可能想用 Pmw 控件，它能帮助用户在正确的格式下获得输入。

对其他格式来说，你将必须书写代码。这个例子演示了如何把电话号码和社会安全数字格式化的方法。

Example 6_9.py

```
import string
from Tkinter import *

class EntryFormatting:
   def __init__(self, master):
       frame = Frame(master)
       Label(frame, text='   ').grid(row=0, column=0,sticky=W)
       Label(frame, text='   ').grid(row=0, column=3,sticky=W)

       self._ipaddr = self.createField(frame, width=16, row=0, col=2,
                        label='Phone Number:\n(nnn)-nnn-nnn',
                        format=self.fmtPhone, enter=self.activate)
       self._crdprt = self.createField(frame, width=11, row=1, col=2,
                        label='SSN#:', format=self.fmtSSN,
                        enter=self.activate)
       frame.pack(side=TOP, padx=15, pady=15)

   def createField(self, master, label='', text='', width=1,
           format=None, enter=None, row=0, col=0):
       Label(master, text=label).grid(row=row, column=col-1,
                          padx=15, sticky=W)
       id = Entry(master, text=text, width=width, takefocus=1)
       id.bind('<KeyRelease>', format)
       id.bind('<Return>',   enter)
       id.grid(row=row, column=col, pady=10, sticky=W)
       return id

   def activate(self, event):
       print '<Return>: value is', event.widget.get()

   def fmtPhone(self, event):
       current = event.widget.get()
       if len(current) == 1:
           current = '1-(%s' % current
       elif len(current) == 6:
           current = '%s)-' % current
       elif len(current) == 11:
           current = '%s-' % current
       event.widget.delete(0, END)
       event.widget.insert(0, current)

   def fmtSSN(self, event):
       current = event.widget.get()
       if len(current) in [3, 6]:
           current = '%s-' % current
```

```
        event.widget.delete(0, END)
        event.widget.insert(0, current)

root = Tk()
root.title('Entry  Formatting')

top = EntryFormatting(root)
quit = Button(root, text='Quit', command=root.destroy)
quit.pack(side = 'bottom')

root.mainloop()
```

代码注解

❶ createField 方法为当用户按键时运行的格式化函数提供了一个捆绑的封装。

❷ 初始化格式化的捆绑。

❸ fmtphone 方法必须计数输入该域，以提供额外的分离符。

❹ 相同地，fmtSSN 在适当地位置插入连字号。

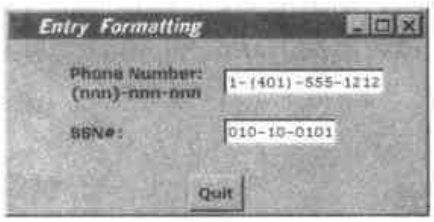


图 6.4　简单格式化控件

6.9　小结

本章包含了对 GUI 用户来说很重要的材料，几乎所有的 GUIs 是事件驱动的，对操作敏感型应用来说，对那些可能是洪水般的事件的适当响应是很重要的。

本章第 2 部分介绍了数字输入有效性检查。这也是个很重要的论题，因为错误识别不适合的数值对用户是很恼火的。尤其当用户重新输入信息到数据输入屏。

第 7 章　使用类、控件和特殊控件

Python 面向对象程序（OOP）开发语言的性能使它成为一个理想的原型开发平台，而且在大多数情况下完全适用于原型开发。面向对象语言存在的一个问题在于面向对象本身引发的争论，所以许多程序员只是简单地回避面向对象的程序开发而停留在结构式程序设计（或其他非结构方法）上。其实面向对象程序开发并没有什么神奇之处，对于简单的问题，它可能不值得采用，但一般来说，Python 面向对象语言在实际应用中是很有效的。

在这一章里我们假定读者已经熟悉了 C++、Java 或者 Python 程序设计，同时理解了一些基本概念。更加深入的内容请参阅 Harms 和 Mcdonald's 的 *Quick Python* 与 Lutz 和 Ascher's 的 *Learning Python*。

7.1　创建发光二极管类

下面的例子介绍了一种“发光二极管类（LED）”来定义发光二极管对象。这些对象有开、关、预警和报警几种状态。这些状态由二极管的闪烁来显示。LED 还定义了设定这些状态及闪烁时间的方法。图 7.1 给出了可能得到的若干种形式。

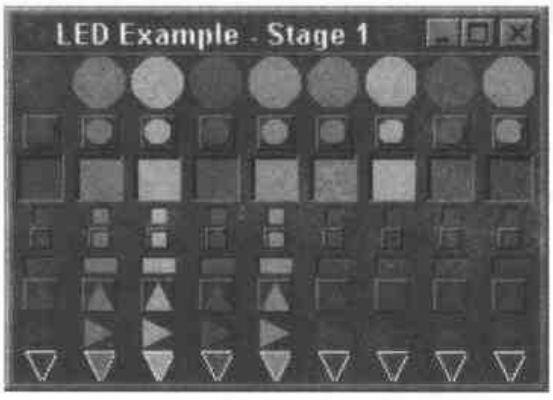


图 7.1　发光二极管例子

Example 7 1.py

```
from Tkinter   import *
```

```
SQUARE          = 1
ROUND           = 2
ARROW           = 3

POINT_DOWN      = 0
POINT_UP        = 1
POINT_RIGHT     = 2
POINT_LEFT      = 3

STATUS_OFF      = 1
STATUS_ON       = 2
STATUS_WARN     = 3
STATUS_ALARM    = 4
STATUS_SET      = 5
```

Define constants

```
class StructClass:
  pass

Color  = StructClass()

Color.PANEL       =     '#545454'
Color.OFF         =     '#656565'
Color.ON          =     '#00FF33'
Color.WARN        =     '#ffcc00'
Color.ALARM       =     '#ff4422'

class LED:
   def __init__(self, master=None, width=25, height=25,
                      appearance=FLAT,
                      status=STATUS_ON, bd=1,
                      bg=None,
                      shape=SQUARE, outline="",
                      blink=0, blinkrate=1,
                      orient=POINT_UP,
                      takefocus=0):
   # Peserve attributes
       self.master  = master
       self.shape  = shape
       self.onColor  = Color.ON
       self.offColor  = Color.OFF
       self.alarmColor  = Color.ALARM
       self.warningColor = Color.WARN
       self.specialColor = '#00ffdd'
       self.status  = status
       self.blink  = blink
       self.blinkrate  = int(blinkrate)
       self.on     = 0
       self.onState  = None
       if not bg:
            bg = Color.PANEL
```

```
        ## Base frame to contain light
            self.frame=Frame(master, relief=appearance, bg=bg, bd=bd,
                        takefocus=takefocus)
            basesize = width
            d = center = int(basesize/2)
            if self.shape == SQUARE:
                self.canvas=Canvas(self.frame,height=height,width=width,
                                bg=bg, bd=0, highlightthickness=0)
                self.light=self.canvas.create_rectangle(0,0, width,height,
                                            fill=Color.ON)
            elif self.shape == ROUND:
                r = int((basesize-2)/2)
                self.canvas=Canvas(self.frame, width=width, height=width,
                                highlightthickness=0, bg=bg, bd=0)
                if bd > 0:
                    self.border=self.canvas.create_oval(center-r,center-r,
                                            center+r, center+r)
                    r = r - bd
                self.light=self.canvas.create_oval(center-r-1,center-r-1,
                                center+r, center+r, fill=Color.ON,
                                outline=outline)
            else:  # Default is an ARROW
                self.canvas=Canvas(self.frame, width=width, height=width,
                                highlightthickness=0, bg=bg, bd=0)
                x = d
                y = d

                if orient == POINT_DOWN:      ❶
                    self.light=self.canvas.create_polygon(x-d,y-d, x,y+d,
                                            x+d,y-d,x-d,y-d,outline=outline)
                elif orient == POINT_UP:
                    self.light=self.canvas.create_polygon(x,y-d, x-d,y+d,
                                            x+d,y+d,x,y-d,outline=outline)
                elif orient == POINT_RIGHT:
                    self.light=self.canvas.create_polygon(x-d,y-d, x+d,y,
                                            x-d,y+d,x-d,y-d,outline=outline)
                elif orient == POINT_LEFT:
                    self.light=self.canvas.create_polygon(x-d,y, x+d,y+d,
                                            x+d,y-d,x-d,y,outline=outline)

            self.canvas.pack(side=TOP, fill=X, expand=NO)
            self.update()

        def turnon(self):      ❷
            self.status = STATUS_ON
            if not self.blink: self.update()
        def turnoff(self):
            self.status = STATUS_OFF
            if not self.blink: self.update()
```

```
def alarm(self):
    self.status = STATUS_ALARM
    if not self.blink: self.update()
def warn(self):
    self.status = STATUS_WARN
    if not self.blink: self.update()
def set(self, color):
    self.status  = STATUS_SET
    self.specialColor = color
    self.update()
def blinkon(self):
    if not self.blink:
        self.blink   = 1
        self.onState = self.status
        self.update()
def blinkoff(self):
    if self.blink:
         self.blink   = 0
         self.status  = self.onState
         self.onState = None
         self.on      = 0
         self.update()

def blinkstate(self, blinkstate):
    if blinkstate:
        self.blinkon()
    else:
        self.blinkoff()

def update(self):
    # First do the blink, if set to blink
    if self.blink:
         if self.on:
              if not self.onState:
                   self.onState = self.status
              self.status  = STATUS_OFF
              self.on      = 0
        else:
            if self.onState:
               self.status = self.onState   # Current ON color
            self.on = 1

    if self.status == STATUS_ON:   ❸
         self.canvas.itemconfig(self.light, fill=self.onColor)
    elif self.status == STATUS_OFF:
         self.canvas.itemconfig(self.light, fill=self.offColor)
    elif self.status == STATUS_WARN:
         self.canvas.itemconfig(self.light, fill=self.warningColor)
    elif self.status == STATUS_SET:
         self.canvas.itemconfig(self.light, fill=self.specialColor)
```

```
            else:
                self.canvas.itemconfig(self.light, fill=self.alarmColor)
            self.canvas.update_idletasks() ❹
            if self.blink:
                self.frame.after(self.blinkrate * 1000, self.update)

    if __name__ == '__main__':
        class TestLEDs(Frame):
            def __init__(self, parent=None):
                # List of Colors and Blink On/Off
                states = [(STATUS_OFF, 0),
                          (STATUS_ON, 0),
                          (STATUS_WARN, 0),
                          (STATUS_ALARM, 0),
                          (STATUS_SET, 0),
                          (STATUS_ON, 1),
                          (STATUS_WARN, 1),
                          (STATUS_ALARM, 1),
                          (STATUS_SET,  1)]
                # List of LED types to display,
                # with sizes and other attributes
                leds = [(ROUND, 25, 25, FLAT, 0, None, ""),
                        (ROUND, 15, 15, RAISED, 1, None, ""),
                        (SQUARE, 20, 20, SUNKEN, 1, None, ""),
                        (SQUARE, 8, 8, FLAT, 0, None, ""),
                        (SQUARE, 8, 8, RAISED, 1, None, ""),
                        (SQUARE, 16, 8, FLAT, 1, None, ""),
                        (ARROW, 14, 14, RIDGE, 1, POINT_UP, ""),
                        (ARROW, 14, 14, RIDGE, 0, POINT_POINT_UP, ""),
                        (ARROW, 14, 14, FLAT, 0, POINT_DOWN, "white")]

                Frame.__init__(self)  # Do superclass init
                self.pack()
                self.master.title('LED Example - Stage 1')

                # Iterate for each type of led
                for shape, w, h, app, bd, orient, outline in leds:
                    frame = Frame(self, bg=Color.PANEL)
                    frame.pack(anchor=N, expand=YES, fill=X)
                    # Iterate for selected states
                    for state, blink in states:
                        LED(frame, shape=shape, status=state,
                            width=w, height=h, appearance=app,
                            orient=orient, blink=blink, bd=bd,
                            outline=outline).frame.pack(side=LEFT,
                                        expand=YES, padx=1, pady=1)

    TestLEDs().mainloop()
```

Example_7_1.py 注解

❶ 已经有一些简单的绘图结构可以用来绘制三角形。

❷ LED 控件有若干种方法改变对象的显示，可以改变多种颜色和闪烁来对应开或者关。

❸ 改变所选状态。

```
if self.status == STATUS_ON:
    self.canvas.itemconfig(self.light, fill=self.onColor)
```

❹ 通常刷新事务队列以确定绘出的控件是否是当前的状态。

注意　在这本书里作者想鼓励大家设法精简自己的程序，但这并不意味着鼓励你写一些含糊不清的程序。评价结构优良的 Python 程序自有一套标准，Example_7_1.py 中的 LED 类就是一个精炼的例子。这里作者创建了一大堆的 LED，所以构造了两个表：一个放入打算显示的各种状态；另一个放入 LED 的形状和定义的作用。然后用两个嵌套循环来方便地创建 LED。

这种用循环来创建若干对象的例子将在以后多次出现。大家也可以从中感受到循环的妙处。

Example_7_1.py 创建的屏幕显示在图 7.1。虽然此时看来并不是很有用处，但它指明了 Tkinter 在具体应用中产生输出的功能。可惜不能在纸张上反映出 LED 的闪烁，但可以运行这个程序看一看。

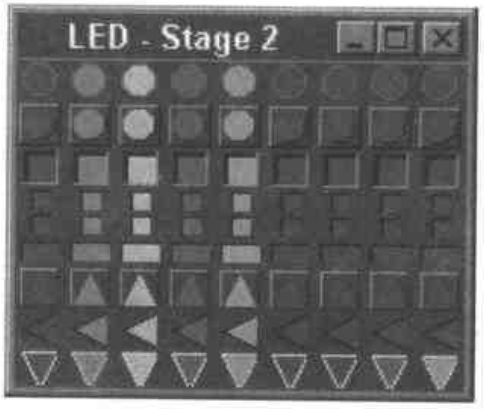


图 7.2　发光二极管例子（较短代码）

7.1.1　再试试

Python 程序员会迅速地发现一个现象：任何时候重新阅读自己从前编写的程序，都能发现可用更简练的代码重新编写。另外，已经写好的程序段落很可能以后再被利用。

为了清楚地表明精炼的代码在程序设计中的作用，让我们来看看如何改进代码。首先，把在程序开头定义的常量存为 Common_7_1.py，以后还会用到。

Common_7_1.py

```
SQUARE              = 1
ROUND               = 2
...
...
Color.WARN          = '#ffcc00'
Color.ALARM         = '#ff4422'
```

现在是将产生 LED 的方法综合起来的时机了！因为我们已经有了构建其他窗体控件的基础。

GUICommon_7_1.py

```
from Common_7_1 import *

class GUICommon:
    def turnon(self):
        self.status = STATUS_ON
        if not self.blink: self.update()

    def turnoff(self):
        self.status = STATUS_OFF
        if not self.blink: self.update()

    def alarm(self):
        self.status = STATUS_ALARM
        if not self.blink: self.update()

    def warn(self):
        self.status = STATUS_WARN
        if not self.blink: self.update()

    def set(self, color):
        self.status       = STATUS_SET
        self.specialColor = color
        self.update()

    def blinkon(self):
        if not self.blink:
            self.blink   = 1
            self.onState = self.status
            self.update()

    def blinkoff(self):
        if self.blink:
            self.blink   = 0
            self.status  = self.onState
            self.onState = None
            self.on=0
```

```
            self.update()

    def blinkstate(self, blinkstate):
        if blinkstate:
            self.blinkon()
        else:
            self.blinkoff()
    def update(self):
        raise NotImplementedError

# The following define drawing vertices for various
# graphical elements
ARROW_HEAD_VERTICES = [
    ['x-d', 'y-d', 'x',   'y+d', 'x+d', 'y-d', 'x-d', 'y-d'],
    ['x',   'y-d', 'x-d', 'y+d', 'x+d', 'y+d', 'x',   'y-d'],
    ['x-d', 'y-d', 'x+d', 'y',   'x-d', 'y+d', 'x-d', 'y-d'],
    ['x-d', 'y',   'x+d', 'y+d', 'x+d', 'y-d', 'x-d', 'y'  ]]
```

❶

❷

GUICommon_7_1.py 注解

❶ 注意尽管加入了 **turnon** 和 **blinkoff** 这些方法，程序中还是定义了 **update** 方法来支持 **NotImplementedError**。由于每一个控件都有不同的显示方法，这一方法就提醒开发人员明白可以使用的基本类之外的别的方法。

❷ 前面的程序用了 4 个 **if-else-else** 语句来处理箭头方向。我们不妨改变一下，用不同的方法来实现，可以把它们存入另一个列表以备后用。

Example 7_2.py

```
from Tkinter          import *
from Common_7_1       import *
from GUICommon_7_1    import *

class LED(GUICommon):
    def __init__(self, master=None, width=25, height=25,
                 appearance=FLAT,
                 status=STATUS_ON, bd=1,
                 bg=None,
                 shape=SQUARE, outline="",
                 blink=0, blinkrate=1,
                 orient=POINT_UP,
                 takefocus=0):
        # preserve attributes
        self.master  = master
        self.shape  = shape
        self.Colors  = [None, Color.OFF, Color.ON,
                        Color.WARN, Color.ALARM, '#00ffdd']
        self.status  = status
        self.blink  = blink
        self.blinkrate  = int(blinkrate)
        self.on  = 0
```

❶

❷

❸

```
        self.onState  = None

        if not bg:
            bg = Color.PANEL

        ## Base frame to contain light
        self.frame=Frame(master, relief=appearance, bg=bg, bd=bd,
                         takefocus=takefocus)

        basesize = width
        d = center = int(basesize/2)

        if self.shape == SQUARE:
            self.canvas=Canvas(self.frame, height=height, width=width,
                               bg=bg, bd=0, highlightthickness=0)

            self.light=self.canvas.create_rectangle(0, 0, width, height,
                                                    fill=Color.ON)
        elif self.shape == ROUND:
            r = int((basesize-2)/2)
            self.canvas=Canvas(self.frame, width=width, height=width,
                               highlightthickness=0, bg=bg, bd=0)
            if bd > 0:
                self.border=self.canvas.create_oval(center-r,center-r,
                                                    center+r,center+r)
              r = r - bd
            self.light=self.canvas.create_oval(center-r-1, center-r-1,
                                               center+r, center+r,
                                               fill=Color.ON,
                                               outline=outline)
        else:  # Default is an ARROW
            self.canvas=Canvas(self.frame, width=width, height=width,
                               highlightthickness=0, bg=bg, bd=0)
            x = d
            y = d
            VL = ARROW_HEAD_VERTICES[orient] # Get the vertices for the arrow
            self.light=self.canvas.create_polygon(eval(VL[0]),
                           eval(VL[1]), eval(VL[2]), eval(VL[3]),
                           eval(VL[4]), eval(VL[5]), eval(VL[6]),
                           eval(VL[7]), outline = outline)

        self.canvas.pack(side=TOP, fill=X, expand=NO)
        self.update()

    def update(self):
        # First do the blink, if set to blink
        if self.blink:
            if self.on:
            if not self.onState:
                self.onState = self.status
```

```
            self.status  = STATUS_OFF
            self.on      = 0
        else:
            if self.onState:
                self.status = self.onState   # Current ON color
            self.on = 1

    # Set color for current status
    self.canvas.itemconfig(self.light, fill=self.Colors[self.status])

    self.canvas.update_idletasks()

    if self.blink:
        self.frame.after(self.blinkrate * 1000, self.update)
```

Example_7_2.py 注解

❶ 首先，导入（Import）新建的常量文件和 GUI mixins。

❷ 由 GUICommon mixins 继承而来，它没有构造函数所以不需要调用。

❸ 创建颜色表，起作用是调整当前状态的目录。

❹ 选取适当的 x、y 数组，然后将每一组值取等。

7.1.2　什么改变了

事实上并没有实质的变化。只是删除了一些普通的代码，创建了一个 mixins 以便创建超类来封装可重用的代码。将颜色属性封装在列表中精简了 if-else-else 结构，而那些繁琐的绘制箭头的代码已经被替换成一个有关顶点的列表。相似的，有关形态属性也被转换成一个列表。最终，可以通过改变尺寸来改变对象的外观，你会发现我们并没有生搬硬套图 7.1。

运行 Example_7_2.py 可以看到与图 7.2 相似的结果。不要过多地去注意结果的改变，而应该注意代码的精炼。

7.2　构建类库

在清楚地了解了 mixin 类的概念之后，我们可以开始着手为有用的 GUI 创建类库了。实际中常常有必要为显示效果创建一系列的颜色，所以我们先来制定一套由基本颜色产生所有颜色的“规则”。

首先必须扩展 GUICommon 类以增加一些色彩变化的方法。下面是其中的一种，将 GUICommon_7_1.py 转成 GUICommon_7_2.py。

GUICommon_7_2.py(modifications only)

```
# This routine modifies an RGB color (returned by winfo_rgb),
# applys a factor, maps -1 < Color < 255 and returns new RGB string
  def transform(self, rgb, factor):
      retval = "#"
      for v in [rgb[0], rgb[1], rgb[2]]:
```

```
            v = (v*factor)/256
            if v > 255: v = 255
            if v < 0:   v = 0
            retval = "%s%02x" % (retval, v)
        return retval

# This routine factors dark, very dark, light and very light colors
# from the base color using transform

    def set_colors(self):
        rgb = self.winfo_rgb(self.base)                    ❶
        self.dbase  = self.transform(rgb, 0.8)
        self.vdbase = self.transform(rgb, 0.7)             ❷
        self.lbase  = self.transform(rgb, 1.1)
        self.vlbase = self.transform(rgb, 1.3)
```

GUICommon_7_2.py 注解

❶ 计算由基本色衍生出来的颜色种类。Winfo_rgb 返回一个元组记录 RGB 值。

❷ 将每一个颜色转换设置为任意值。

下面看例子：

Example_7_3.py

```
from Tkinter            import *
from GUICommon_7_2      import *

import string

class TestColors(Frame, GUICommon):
    def __init__(self, parent=None):
        Frame.__init__(self)          ● Lnit base class
        self.base = "#848484"         ● Set base color
        self.pack()
        self.set_colors()             ● Spread colors
        self.make_widgets()

    def make_widgets(self):
        for tag in ['VDBase', 'DBase', 'Base', 'LBase', 'VLBase']:
            Button(self,text=tag,bg=eval(self,'%s'% string.lower(tag)),
                   fg='white', command=self.quit).pack(side=LEFT)

if __name__ == '__main__':
    TestColors().mainloop()
```

运行 Example_7_3.py 显示如图 7.3 所示。



图 7.3 颜色转换

7.2.1　将六边形螺帽加入类库

现在来实际应用一下色彩转换来创建一些可视的对象。在下面的例子中，我们将创建六边形螺帽。不久你会看到这些简单对象很有用。

我们从扩展 Common_7_1.py 中的一些定义开始，将其存为 Common_7_2.py。

Common_7_2.py

```
NUT_FLAT    = 0
NUT_POINT   = 1

Color.BRONZE    = '#7e5b41'
Color.CHROME    = '#c5c5b8'
Color.BRASS     = '#cdb800'
```

这就是六边形螺帽的代码，本例比较详细地给出了多种形状的螺帽。运行结果见图 7.4。

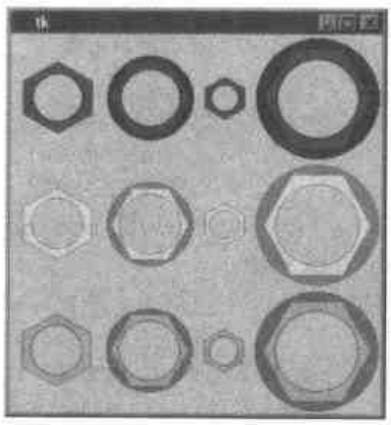


图 7.4　基本螺帽

Example_7_4.py

```
from Tkinter      import *
from GUICommon_7_2 import *
from Common_7_2    import *

class HEXNUT(GUICommon):
   def __init__(self, master, frame=1, mount=1, outside=70, inset=8,
                bg=Color.PANEL, nutbase=Color.BRONZE,
                top=NUT_FLAT, takefocus=0, x=-1, y=-1):
      points = [ '%d-r2,%d+r,%d+r2,%d+r,%d+r+2,%d,%d+r2,%d-r, \
                  %d-r2,%d-r,%d-r-2,%d,%d-r2,%d+r',
                 '%d,%d-r-2,%d+r,%d-r2,%d+r,%d+r2,%d,%d+r+2, \
                  %d-r,%d+r2,%d-r,%d-r2,%d,%d-r-2' ]
      self.base   = nutbase
      self.status = STATUS_OFF
      self.blink  = 0
```

```
        self.set_colors()
        basesize = outside+4
        if frame:
            self.frame = Frame(master, relief="flat", bg=bg, bd=0,
                               highlightthickness=0,
                               takefocus=takefocus)
            self.frame.pack(expand=0)
            self.canv=Canvas(self.frame, width=basesize,  bg=bg,
                             bd=0, height=basesize,
                               highlightthickness=0)
        else:
            self.canv = master   # it was passed in...
        center = basesize/2
        if x >= 0:
             centerx = x
             centery = y
        else:
            centerx = centery = center
        r = outside/2
        ## First, draw the mount, if needed
        if mount:
            self.mount=self.canv.create_oval(centerx-r, centery-r,
                                             centerx+r, centery+r,
                                             fill=self.dbase,
                                             outline=self.vdbase)
        ## Next, draw the hex nut
        r  = r - (inset/2)
        r2 = r/2
        pointlist = points[top] % (centerx,centery,centerx,centery,
                                   centerx,centery,centerx,centery,
                                   centerx,centery,centerx,centery,
                                   centerx,centery)

        setattr(self, 'hexnut', self.canv.create_polygon(pointlist,
              outline=self.dbase, fill=self.lbase))

        ## Now, the inside edge of the threads
        r = r - (inset/2)
        self.canv.create_oval(centerx-r, centery-r,
                              centerx+r, centery+r,
                              fill=self.lbase, outline=self.vdbase)
        ## Finally, the background showing through the hole
        r = r - 2
        self.canv.create_oval(centerx-r, centery-r,
                              centerx+r, centery+r,
                              fill=bg, outline="")

        self.canv.pack(side="top", fill='x', expand='no')

class NUT(Frame, HEXNUT):
```

```
    def __init__(self, master, outside=70, inset=8, frame=1, mount=1,
                 bg="gray50", nutbase=Color.CHROME, top=NUT_FLAT):
        Frame.__init__(self)
        HEXNUT.__init__(self, master=master, outside=outside,
                        inset=inset, frame=frame, mount=mount,
                        bg=bg, nutbase=nutbase, top=top)

class TestNuts(Frame, GUICommon):
    def __init__(self, parent=None):
        Frame.__init__(self)
        self.pack()
        self.make_widgets()

    def make_widgets(self):
        # List of Metals to create
        metals = [(Color.BRONZE), (Color.CHROME), (Color.BRASS)]
        # List of nut types to display,
        # with sizes and other attributes
        nuts  = [(70, 14, NUT_POINT, 0), (70, 10, NUT_FLAT,  1),
                 (40,  8, NUT_POINT, 0), (100,16, NUT_FLAT,  1)]
        # Iterate for each metal type
        for metal in metals:
            mframe = Frame(self, bg="slategray2")
            mframe.pack(anchor=N, expand=YES, fill=X)
            # Iterate for each of the nuts
            for outside, inset, top, mount in nuts:
                NUT(mframe, outside=outside, inset=inset,
                    mount=mount, nutbase=metal,
                    bg="slategray2",
                    top=top).frame.pack(side=LEFT,
                                        expand=YES,
                                        padx=1, pady=1)

if __name__ == '__main__':
    TestNuts().mainloop()
```

注意　在 Example_7_2.py 中使用了一种与绘制箭头时相同的方法来绘制多边形的顶点。在这个例子中出现了另一种技术，它会在后文中多次被使用。

由于用多边形来绘制螺帽的复杂性和用 point 及 flat 形式计算多边形顶点数的必要性，我们使用 setattr 函数。于是就可以用对象的参数和属性表达来设置数值。

7.2.2　创建开关类

还有一些东西比 LED 和螺帽更有意思，比如说开关。可能总是创建新的类使人生厌，但我们还是会在描绘一些开关控制控件上花点功夫。下面介绍一些虽然简单但是比较生

动有趣的例子。

注意 在前面的例子里，GUICommon.py 和 Common.py 是直接被调用的，而不需要每次都重写。

我们需要在 Common.py 多定义两个常数。因为在美国开关向上是开，而在英国则是向下为开（虽然这是一个容易被忽略的现象，但在这里非常重要）。

Common.py

```
MODE_UK       = 0
MODE_US       = 1
```

下面来绘制一个栓状开关。

Example 7 5.py

```
from Tkinter     import *
from GUICommon   import *
from Common      import *
from Example_7_4 import HexNut

class ToggleSwitch(Frame, HexNut):
   def __init__(self, master, outside=70, inset=8, bg=Color.PANEL,
                nutbase=Color.CHROME, mount=1, frame=1,
                top=NUT_POINT, mode=MODE_US, status=STATUS_ON):
       Frame.__init__(self)
       HexNut.__init__(self, master=master, outside=outside+40,
                       inset=35, frame=frame, mount=mount,
                       bg=bg, nutbase=nutbase, top=top)
       self.status = status
       self.mode   = mode
       self.center = (outside+44)/2
       self.r      = (outside/2)-4
       ## First Fill in the center
       self.r1=self.canv.create_oval(self.center-self.r,
            self.center-self.r, self.center+self.r,
            self.center+self.r, fill=self.vdbase,
            outline=self.dbase, width=1)
       self.update()  ## The rest is dependent on the on/off state

   def update(self):
       self.canv.delete('lever')  ## Remove any previous toggle lever
       direction = POINT_UP
       if (self.mode == MODE_UK and self.status == STATUS_ON) or \
          (self.mode == MODE_US and self.status == STATUS_OFF):
           direction = POINT_DOWN
       # now update the status
       if direction == POINT_UP:    ❶
```

call constructors for base classes

```
            ## Draw the toggle lever
            self.p1=self.canv.create_polygon(self.center-self.r,
                 self.center, self.center-self.r-3,
                 self.center-(4*self.r), self.center+self.r+3,
                 self.center-(4*self.r), self.center+self.r,
                 elf.center, fill=self.dbase,
                 outline=self.vdbase, tags="lever")
            centerx = self.center
            centery = self.center - (4*self.r)
            r = self.r + 2
            ## Draw the end of the lever
            self.r2=self.canv.create_oval(centerx-r, centery-r,
                 centerx+r, centery+r, fill=self.base,
                 outline=self.vdbase, width=1, tags="lever")
            centerx = centerx - 1
            centery = centery - 3
            r = r / 3
            ## Draw the highlight
            self.r2=self.canv.create_oval(centerx-r, centery-r,
                 centerx+r, centery+r, fill=self.vlbase,
                 outline=self.lbase, width=2, tags="lever")

        else:
            ## Draw the toggle lever
            self.p1=self.canv.create_polygon(self.center-self.r,
                 self.center, self.center-self.r-3,
                 self.center+(4*self.r), self.center+self.r+3,
                 self.center+(4*self.r), self.center+self.r,
                 self.center, fill=self.dbase,
                 outline=self.vdbase, tags="lever")
            centerx = self.center
            centery = self.center + (4*self.r)
            r = self.r + 2
            ## Draw the end of the lever
            self.r2=self.canv.create_oval(centerx-r, centery-r,
                 centerx+r, centery+r, fill=self.base,
                 outline=self.vdbase, width=1, tags="lever")
            centerx = centerx - 1
            centery = centery - 3
            r = r / 3
            ## Draw the highlight
            self.r2=self.canv.create_oval(centerx-r, centery-r,
                 centerx+r, centery+r, fill=self.vlbase,
                 outline=self.lbase, width=2, tags="lever")
        self.canv.update_idletasks()

class TestSwitches(Frame, GUICommon):
    def __init__(self, parent=None):
        Frame.__init__(self)
        self.pack()
```

```
        self.make_widgets()

    def make_widgets(self):
        # List of metals to create
        metals = [(Color.BRONZE), (Color.CHROME), (Color.BRASS)]

        # List of switchesdisplay, with sizes and other attributes
        switches = [(NUT_POINT, 0, STATUS_OFF, MODE_US),
                    (NUT_FLAT,  1, STATUS_ON,  MODE_US),
                    (NUT_FLAT,  0, STATUS_ON,  MODE_UK),
                    (NUT_POINT, 0, STATUS_OFF, MODE_UK)]
        # Iterate for each metal type
        for metal in metals:
            mframe = Frame(self, bg="slategray2")
            mframe.pack(anchor=N, expand=YES, fill=X)
            # Iterate for each of the switches
            for top, mount, state, mode in switches:
              ToggleSwitch(mframe,
                           mount=mount, outside=20,
                           nutbase=metal, mode=mode,
                           bg="slategray2", top=top,
                           status=state).frame.pack(side=LEFT,
                                                    expand=YES,
                                                    padx=2, pady=6)
if __name__ == '__main__':
  TestSwitches().mainloop()
```

Example_7_5.py 注解

❶ Direction（方向）决定了开关的上下。由于它可在程序中随意改变，所以 GUI 之提供简单的状态绘制。

运行程序结果见图 7.5。

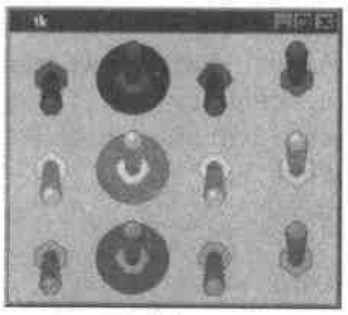


图 7.5 切换开关

7.2.3 创建大控件

现在大家已经掌握了创建对象的方法，现在可以开始做点复杂的东西了。其实这些复杂的东西更加有用，因为生成这些控件的代码都相当简洁。首先把先前的类如发光二极管、六边形螺帽和开关统一放在一个类库中，取名为 Components.py。

接着，创建一个新类 SwitchIndicator，其作用是画一个上方有发光二极管指示的开关。把它们放在一个框架之中以便于收入类库中。下面是程序：

Example_7_6.py

```
from Tkinter     import *
from Common      import *
from Components  import *

class SwitchIndicator:
  def __init__(self, master, outside=70, bg=Color.PANEL,
               metal=Color.CHROME, mount=1, frame=1,
               shape=ROUND, top=NUT_POINT, mode=MODE_US, status=1):
      self.frame = Frame(master, bg=bg)
      self.frame.pack(anchor=N, expand=YES, fill=X)

      self.led = LED(self.frame, width=outside, height=outside,
                     status=status, bg=bg, shape=shape,
                     outline=metal)
      self.led.frame.pack(side=TOP)

      self.switch = ToggleSwith(self.frame, mount=mount,
                          outside=outside, nutbase=metal,
                          mode=mode, bg=bg, top=top,
                          status=status)
      self.switch.frame.pack(side=TOP)
      self.update()

   def update(self):
      self.led.update()
      self.switch.update()

class TestComposite(Frame):
  def __init__(self, parent=None):
      Frame.__init__(self)
      self.pack()
      self.make_widgets()

  def make_widgets(self):
       # List of switches to display,
       # with sizes and other attributes
       switches = [(NUT_POINT, 0, STATUS_OFF, MODE_US),
                   (NUT_FLAT,  1, STATUS_ON,  MODE_US),
                   (NUT_FLAT,  0, STATUS_ON,  MODE_UK),
                   (NUT_POINT, 0, STATUS_OFF, MODE_UK)]

         frame = Frame(self, bg="gray80")
         frame.pack(anchor=N, expand=YES, fill=X)

         for top, mount, state, mode in switches:
```

```
                SwitchIndicator(frame,
                                mount=mount,
                                outside=20,
                                metal=Color.CHROME,
                                mode=mode,
                                bg="gray80",
                                top=top,
                                status=state).frame.pack(side=LEFT,
                                                         expand=YES,
                                                         padx=2,
                                                         pady=6)

if __name__ == '__main__':
    TestComposite().mainloop()
```

可以看出实际的代码已经开始多于创建控件所用的代码，这个现象在 Python 程序设计中并不多见。运行 Example_7_6.py 结果见图 7.6。

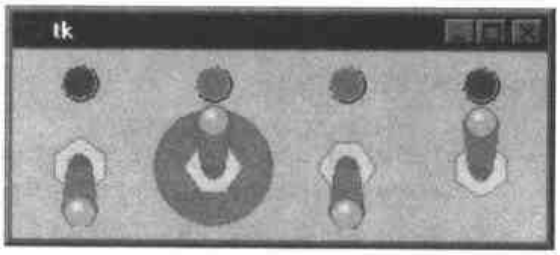


图 7.6　切换/指示控件组合

注意　左边两个开关是美式的，右边的是英式。英美两国读者如果没有注意到的话会觉得混淆不清。

在前面的例子中，我们通过直接使用已存在的对象简化了程序，这在现实应用中非常常见。以后注意把程序存储为可以直接调用的，必须先将例子里的具体数值和函数调用部分分离，例如可把下面的代码：

```
for top, mount, state, mode in switches:
    SwitchIndicator(frame, mount=mount, outside=20, metal=Color.CHROME,
                    mode=mode, bg="gray80",top=top,
                    status=state).frame.pack(side=LEFT,
                                             expand=YES, padx=2, pady=6)
```

变成

```
idx = 0
for top, mount, state, mode in switches:
    setattr(self, 'swin%d' % idx, None)
    var = getattr(self, 'swin%d'  % idx)
    var = SwitchIndicator(frame,
                          mount=mount,
                          outside=20,
```

```
                    metal=Color.CHROME,
                    mode=mode,
                    bg="gray80",
                    top=top,
                    status=state)
var.frame.pack(side=LEFT, expand=YES,
           padx=2, pady=6)
idx = idx + 1
```

虽然这些代码不是很简洁，但它依然可以完成例子里的功能。

```
self.swin0.turnon()
self.swin3.blinkon()
```

以后的例子中还会常出现相对的控件和继承的方法。

7.3　小结

在这一章里，大家应掌握通过创建类来定义比较复杂的对象，而且了解不同的复制可以实现使相似的对象呈现不同外观。另外，我们也举了用 mixin 压缩程序的方法以及使用多重继承。

第8章　对话框和窗体

本章包含了较多的对话框和窗体的设计。如果程序不需要数据输入的话可以节省大量的时间和精力。这倒不是因为设计窗体困难，而是很小的设计失误就会导致严重的问题，如第 16 章所述的“设计有效的图形应用程序”真可谓“差之毫厘，谬以千里”。

对话框可能比较容易理解，窗体则可以用很多种方式来描述。在本章中，对话框的作用是人和机器的沟通桥梁，它的外形可以随意而定。其中数据的处理取决于对话框中的信息的处理。对话窗口可以理解为一种简单的窗体。本章列举了各种各样的例子，可能会让读者感到混淆不清，但是实际应用的时候只要选择一个最合适的就可以了。

下面从标准对话框和典型的填空式窗体开始。且将会有更多的例子介绍如何用最有效的窗体而非冗余的代码。这些例子中含有可以直接在实际中使用的模，以后的章节也会直接使用。

Pmw 控件在例子中使用广泛，因为这些控件将众多的函数封装在内，使得程序员可以用很少的程序实现复杂的功能。更多应用详见附录 C。

8.1　对话框

对话框是最特殊的窗体。一般来说，对话框向用户显示错误或警告信息、提问或要求用户输入位数较小的数值。读者可能会问：难道不是所有的窗体都有对话功能吗？以正常的对话框为例就可以说明了：它一直显示直至被关闭。对话框的外形可以自定也可以用系统值，但要注意你设定的尺寸不能影响其他应用窗口的显示。

注意　仔细考虑什么时候需要一个让用户输入数据的对话框。输入也可以用别的方法来实现，但要注意：对话框太多会使用户感到厌恶。典型的错误是在每一步操作后都问“确实要吗？”初级的用户可能不会有什么反感，但是专业人士则会感到很不舒服。所以尽量设计出适合专业人士使用的软件是很重要的。

Tkinter 本身提供了对话框控件，但用它生成有关错误、警告和其他图标的 X 位图不是很好，它们在 Windows 和 MacOS 中可能不能正常显示。TkSimpleDialog 定义了 askstving、askinteger 和 aslfloat 三种对话框来输入整数、浮点数和字符串。TkmessageBox

定义了一些很方便的函数，比如：showinfo、showwarning、showerror 和 askyesno。其中出现的图标都是经过具体特殊设计的，可以支持所用的平台。

8.1.1　标准对话框

标准对话框使用起来是最简单的。TkMessageBox 提供了若干个方便的函数，包括 showerror、showwarning 和 askyetrycancel。下面的例子只举出了一个最简单的形式（askquestion），但图 8.1 包括了 Unix 和 Windows 系统中都能出现的各种对话框。

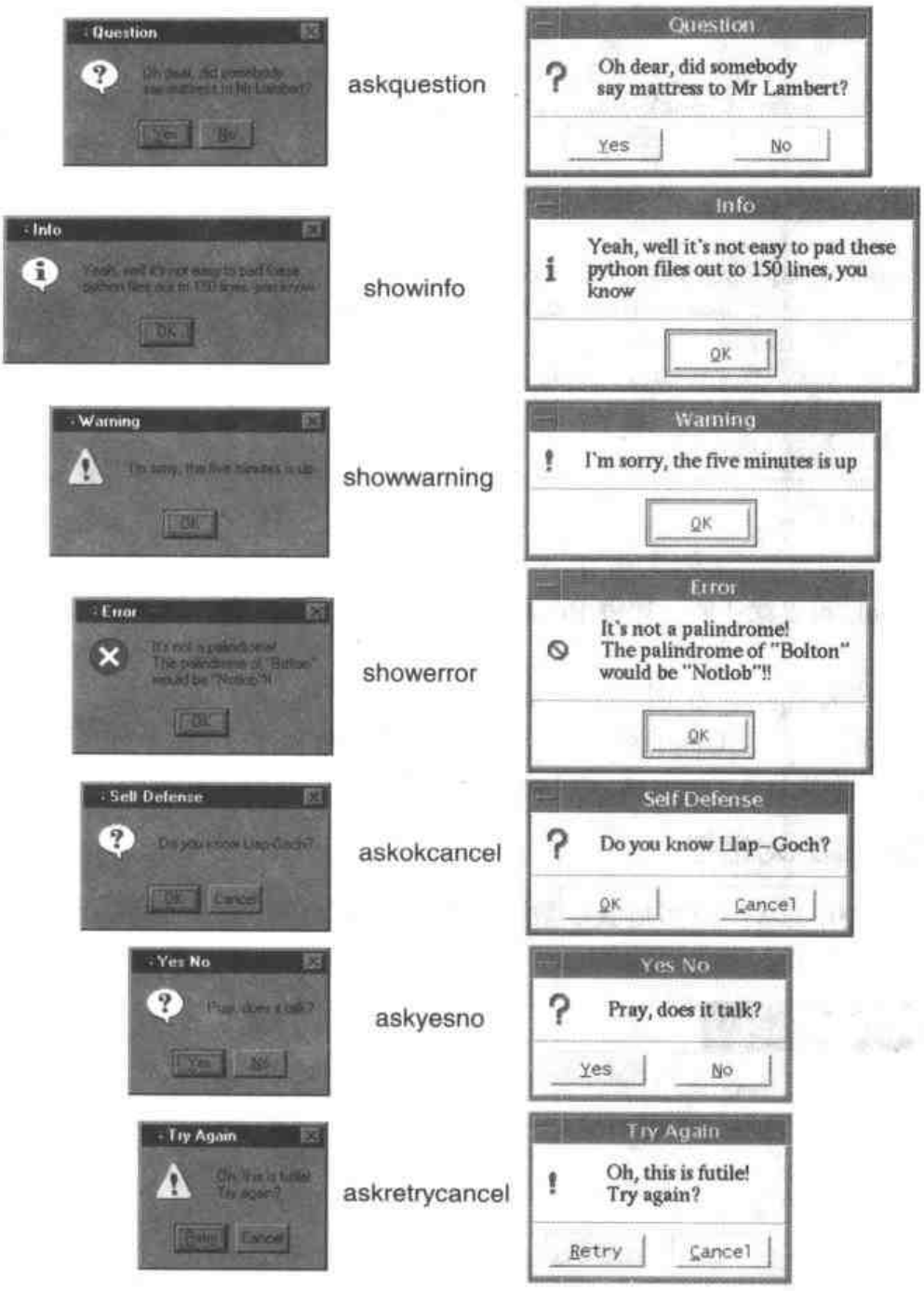


图 8.1　标准对话框

Example_8_1.py

```
from Tkinter import *
from tkMessageBox import *
import Pmw

class App:
  def __init__(self, master):
      self.result = Pmw.EntryField(master, entry_width=8,
                          value='',
                          label_text='Returned value: ',
                          labelpos=W, labelmargin=1)
      self.result.pack(padx=15, pady=15)

root = Tk()
question = App(root)

button = askquestion("Question:",
            "Oh Dear, did somebody\nsay mattress to Mr Lambert?",
            default=NO)
question.result.setentry(button)

root.mainloop()
```

❶
❷
❸

Example_8_1.py 注解

❶ 前两个参数表示标题和信息。

❷ Default 设置默认值（如将 Return 设为默认值，以直接回车发生作用的是 Return 键）。

❸ 返回值为键名对应的字符串。如 OK 键返回“OK”。

本例中列出了所有的标准对话框，同时支持 Windows 和 Unix 两种操作系统（Unix 环境下的背景为浅色）。Example_8_1.py 的结果见图 8.1 的第一个图形。

8.1.2 数据输入对话框

对话框可用于从用户那里获取数据。大家很快地看一看下面这个例子中使用的 tkSimpleDialog 控件。与前面的相比，本例的代码比较简短。

Example_8_2.py

```
from Tkinter import *
from tkSimpleDialog import askinteger
import Pmw

class App:
  def __init__(self, master):
      self.result = Pmw.EntryField(master, entry_width=8,
                          value='',
```

```
                                label_text='Returned value: ',
                                labelpos=W, labelmargin=1)
        self.result.pack(padx=15, pady=15)

root = Tk()
display = App(root)

retVal = askinteger("The Larch",                    ❶
           "What is the number of The Larch?",
           minvalue=0, maxvalue=50)                 ❷
display.result.setentry(retVal)

root.mainloop()
```

Example_8_2.py 注解

❶ Askinteger 可以只带两个参数：标题和要求。

❷ 规定最大、最小值，一旦用户的输入超出范围，会弹出对话框指出输入错误。见图 8.1。

注意　凡是用户输入的时候都弹出对话框的现象一定要避免。如果发现一个输入框出现得太频繁，那就要怀疑是不是前面的输入不充分。尽量使对话框出现得少，并且接近窗口边界出现。

Example_8_2.py 运行结果见图 8.2。

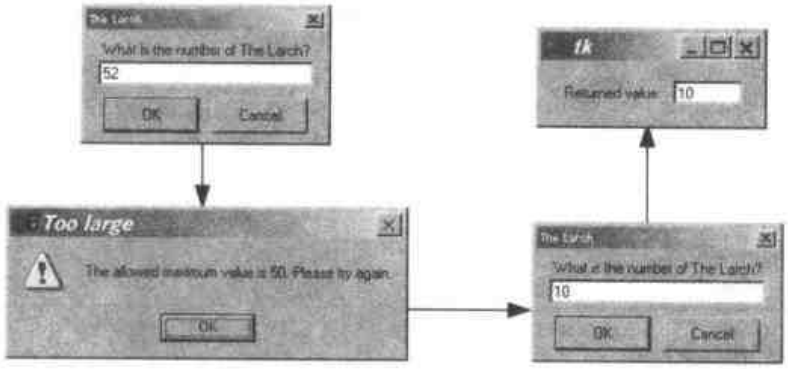


图 8.2　tk 简单对话：askinteger

当然上面的警告并不绝对，假如用以输入的区域大小有限，也可以用对话窗口，尤其在不是每次运行都要求输入的情况下更是如此。因为这种情况下若是把输入信息直接加在屏幕上的话，会使屏幕杂乱。使用对话框可以减少大量工作，虽然这样容易让人反感（尤其是输入数据的控件种类不唯一的时候）。见 Example_8_3.py。

Example_8_3.py

```
from   Tkinter import *
from   tkSimpleDialog import Dialog
```

```
import tkMessageBox
import Pmw

class GetPassword(Dialog):
  def body(self, master):
        self.title("Enter New Password")

        Label(master,text='Old Password:').grid(row=0,sticky=W)
        Label(master,text='New Password:').grid(row=1,sticky=W)
        Label(master,text='Enter New Password Again:').grid(row=2,sticky=W)

        self.oldpw  = Entry(master, width = 16, show='*')
        self.newpw1  = Entry(master, width = 16, show='*')
        self.newpw2  = Entry(master, width = 16, show='*')

        self.oldpw.grid(row=0, column=1, sticky=W)
        self.newpw1.grid(row=1, column=1, sticky=W)
        self.newpw2.grid(row=2, column=1, sticky=W)
        return self.oldpw

  def apply(self):
        opw  = self.oldpw.get()
        npw1 = self.newpw1.get()
        npw2 = self.newpw2.get()

        if not npw1 == npw2:
           tkMessageBox.showerror('Bad Password',
                  'New Passwords do not match')
        else:
            # This is where we would set the new password...
            pass

root = Tk()
dialog = GetPassword(root)
```

❶ ❷ ❸ Validate

Example_8_3.py 注解

❶ 本例使用了 grid 制表控件，相关属性用来定义项目名称显示在表格最左一列（有关 grid 的相关细节见 5.3 节）。

```
label(master,text='Old Password,').grid(row=0,sticky=w)
```

❷ 由于需要用户输入口令，所以不显示具体字符，而以 show 后面的字符代替以保密。

```
self.oldpw=Entry(master,width=16,show='*')
```

❸ 单击 OK。程序先获取输入的信息，然后核对口令是否正确。本例中，用户输入两次口令，如果两次不符，弹出错误对话框。

```
TkMessageBox.showerror('Bad Password','New Passwords do not match')
```

图 8.3 为 Example_8_3.py 的运行结果。

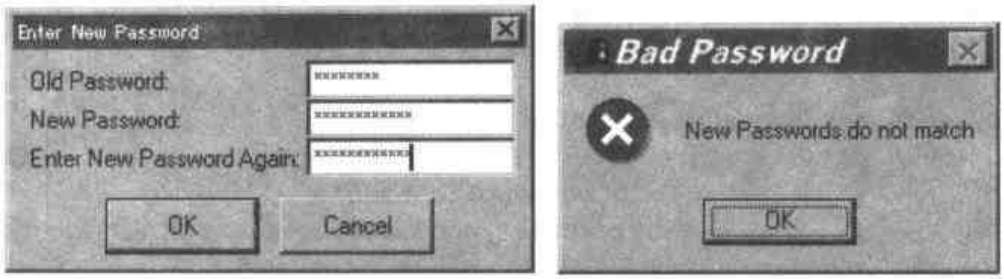


图 8.3　一个用来获取密码的 tk 简单对话框。错误输入产生错误对话框

8.1.3　简单表格

如果程序需要的输入很简单，就用简单窗体。许多用户界面都有以下特征：

1. 有编辑区，可包含默认值。
2. 允许用户输入和修改编辑区。
3. 从窗体中选值。
4. 处理数据。
5. 显示处理结果。

你是否觉得上述特征太熟悉了？那正是因为它常用。事实上，创建窗体常被看作很乏味的工序，但我们可以自己把它搞的丰富多彩一些。

在创建窗体的过程中有一个很大的问题，设计员往往写了很多的冗余代码。事实上，很多时候并不需要。在后面，读者可以体会到一些精简代码的技巧，但现在还是多写一点好。

本例就是要输入一些有关用户的基本信息并显示出来。程序中使用了 Pmw 控件，代码显得长了一些。所以还可以再简化，见后面的例子。

Example 8 4.py

```
from Tkinter import *
import Pmw
import string

class Shell:
   def __init__(self, title=''):
       self.root = Tk()
       Pmw.initialise(self.root)
       self.root.title(title)

   def doBaseForm(self, master):
           # Create the Balloon.
       self.balloon = Pmw.Balloon(master)
```

❶

❷

```
        self.menuBar = Pmw.MenuBar(master, hull_borderwidth=1,
                                   hull_relief = 'raised',
                                   hotkeys=1, balloon = self.balloon)
        self.menuBar.pack(fill='x')

        self.menuBar.addmenu('File', 'Exit')
        self.menuBar.addmenuitem('File', 'command',
                                 'Exit the application',
                                 label='Exit', command=self.exit)
        self.menuBar.addmenu('View', 'View status')
        self.menuBar.addmenuitem('View', 'command',
                                 'Get user information',
                                 label='Get info',
                                 command=self.getStatus)
        self.menuBar.addmenu('Help', 'About Example 8-4', side=RIGHT)
        self.menuBar.addmenuitem('Help', 'command',
                                 'Get information on application',
                                 label='About...', command=self.help)

        self.dataFrame = Frame(master)
        self.dataFrame.pack(fill=BOTH, expand=1, padx = 10)

        self.infoFrame = Frame(self.root, bd=1, relief='groove')
        self.infoFrame.pack(fill='both', expand=1, padx = 10)

        self.statusBar = Pmw.MessageBar(master, entry_width = 40,
                                        entry_relief='groove',
                                        labelpos = w,
                                        label_text = '')
        self.statusBar.pack(fill = 'x', padx = 10, pady = 10)

        # Add balloon text to statusBar
        self.balloon.configure(statuscommand = self.statusBar.helpmessage)

        # Create about dialog.
        Pmw.aboutversion('8.4')
        Pmw.aboutcopyright('Copyright My Company 1999'
                           '\nAll rights reserved')
        Pmw.aboutcontact(
            'For information about this application contact:\n' +
            '  My Help Desk\n'
            '  Phone: 800 555-1212\n'
            '  email: help@my.company.com'
            )
        self.about = Pmw.AboutDialog(master,
                             applicationname = 'Example 8-4')
        self.about.withdraw()

    def exit(self):
        import sys
```

```
    sys.exit(0)
```

Example_8_4.py 注解

❶ 初始化函数 Tk 和 Pmw。

```
Self.root = Tk()
Pmw.initialize(self.root)
```

注意　Pmw.initialise 不是笔误,它来自澳大利亚。

❷ 创建 Pmw.Balloon 连接帮助信息。显然这一段可以不要,但它运行起来很容易,不妨保留。

```
    self.ballon = Pmw.Balloon(master)
Actions are bound later.
```

❸ 下面几行说明如何用 Pmw 控件创建简单的菜单。先定义 MenuBar（菜单条）。菜单项为 Balloon，再定义 hotkey（热键）为 ture 。

```
self.menuBar = Pmw.MenuBar(master, hull_borderwidth=1,
                  hull_relief = 'raised',
                  hotkeys=1, balloon = self.balloon)
self.menuBar.pack(fill='x')
```

注意　用适当的顺序排列菜单项。另外，最好不要把菜单放在窗体底部。

❹ 文件菜单按钮用调用 addmenu 实现。

```
self.menuBar.addmenu('File', 'Exit')
```

第二个参数为显示帮助信息，可以在 addmenuitem 中加一些项目。

```
self.menuBar.addmenuitem('File', 'command',
               'Exit the application',
               label='Exit', command=self.exit)
```

❺ 用一个框架显示日期的输入，另一个框架显示其他。

```
self.dataFrame = Frame(master)
self.dataFrame.pack(fill=BOTH, expand=1)
```

❻ 窗体底部设置状态栏显示帮助信息和其他。

```
self.statusBar = Pmw.MessageBar(master, entry_width = 40,
                    entry_relief='groove',
                    labelpos = w,
                    label_text = '')
self.statusBar.pack(fill = 'x', padx = 10, pady = 10)
```

❼ 将帮助绑定在 MessageBar 控件上。

```
self.balloon.configure(statuscommand = self.statusBar.help)
```

❽ 创建 about（关于）对话框。它显示相关的一些资料。

```
Pmw.aboutversion('8.4')
Pmw.aboutcopyright('Copyright My Company 1999'
```

```
                        '\nAll rights reserved')
Pmw.aboutcontact(
     'For information about this application contact:\n' +
     '  My Help Desk\n'
     '  Phone: 800 555-1212\n'
     '  email: help@my.company.com'
```

❶ 对话框建好了，退出。这时对话框不可见，直到再次被调用。

```
self.about = Pmw.AboutDialog(master,applicationname = 'Example 8-1')
self.about.withdraw()
```

Example 8 4 py(续)

```
def getStatus(self):
     username   = self.userName.get()
     cardnumber = self.cardNumber.get()

     self.img = PhotoImage(file='%s.gif' % username)
     self.pictureID['image'] = self.img

     self.userInfo.importfile('%s.txt' % username)
     self.userInfo.configure(label_text = username)

def help(self):
     self.about.show()

def doDataForm(self):
     self.userName = Pmw.EntryField(self.dataFrame, entry_width=8,
                              value='',
                              modifiedcommand=self.upd_username,
                              label_text='User name:',
                              labelpos=W, labelmargin=1)
     self.userName.place(relx=.20, rely=.325, anchor=W)

     self.cardNumber = Pmw.EntryField(self.dataFrame, entry_width=8,
                           value='',
                           modifiedcommand=self.upd_cardnumber,
                           label_text='Card number: ',
                           labelpos=W, labelmargin=1)
     self.cardNumber.place(relx=.20, rely=.70, anchor=W)

def doInfoForm(self):
     self.pictureID=Label(self.infoFrame, bd=0)
     self.pictureID.pack(side='left', expand=1)

     self.userInfo = Pmw.ScrolledText(self.infoFrame,
                           borderframe = 1,
                           labelpos = N,
                           usehullsize = 1,
                           hull_width = 270,
                           hull_height = 100,
```

```
                                text_padx = 10,
                                text_pady = 10,
                                text_wrap=NONE)
        self.userInfo.configure(text_font = ('verdana', 8))
        self.userInfo.pack(fill = BOTH, expand = 1)

    def upd_username(self):
        upname = string.upper(self.userName.get())
        if upname:
            self.userName.setentry(upname)

    def upd_cardnumber(self):
        valid = self.cardNumber.get()
        if valid:
            self.cardNumber.setentry(valid)

if __name__ == '__main__':
    shell=Shell(title='Example 8-4')
    shell.root.geometry("%dx%d" % (400,350))
    shell.doBaseForm(shell.root)
    shell.doDataForm()
    shell.doInfoForm()
    shell.root.mainloop()
```

代码注解（续）

❿ 首先使用 get 方法获取输入区的数据。

```
username   = self.userName.get()
cardnumber = self.cardNumber.get()
```

● 用 Username 载入一张图片放在先前创建的标签框中。

```
self.img = PhotoImage(file='%s.gif' % username)
self.pictureID['image'] = self.img
```

● 载入文本文件到 ScrolledText 控件中,并修改标题。

```
self.userInfo.importfile('%s.txt' % username)
self.userInfo.configure(label_text = username)
```

●“关于”对话框的显示用 show 的方法实现。

```
def help(self):
    self.about.show()
```

● 窗体内用两个 Pmw EntryField 控件输入数据。

```
self.userName = Pmw.EntryField(self.dataFrame, entry_width=8,
                               value='',
                               modifiedcommand=self.upd_username,
                               label_text='User name:',
                               labelpos=W, labelmargin=1)
self.userName.place(relx=.20, rely=.325, anchor=W)
```

● Modifiedcommand 绑定了当控件发生变化时调用的函数。这时我们可以使用确认

框或者可以将字母变换成大写。

```
upname = string.upper(self.userName.get())
if upname:
    self.userName.setentry(upname)
```

● 最后，我们创建命令行解释器的位置并为它加上窗体的一些细节。

```
shell=Shell(title='Example 8-4')
shell.root.geometry("%dx%d" % (400,350))
shell.doBaseForm(shell.root)
shell.doDataForm()
shell.doInfoForm()
shell.root.mainloop()
```

注意 已经调用了 doBaseForm、doDataForm 和 doInfoForm 方法使我们能清楚怎样由基本的类生成窗体的。

运行 Example_8_4.py 可见形如图 8.4 的结果。注意 ScrolledText 自动在窗体上添加了滚动条，实际上，界面尺寸出现了细微的变化。例如，框架向上扩展了一些但并没有影响输入区。这也就是为什么用户界面设计一定要进行认真调试的原因。

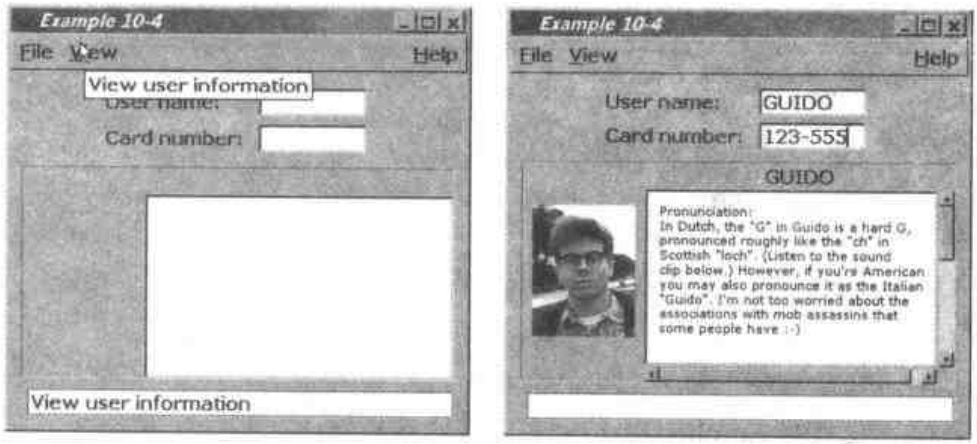


图 8.4 简单表格

注意 使用滚动条会带来一些令人讨厌的变化。在图 8.4 中，当文本长度超出了现实区域的大小时，该滚动条自动出现。水平滚动条则是由于垂直滚动条占用了一定空间而文本太长的缘故，假如将窗体定义得宽几个像素，水平滚动条就不会出现。

8.1.4 Tkinter 变量

在前面的例子中，用 Pmw 控件设定了 setentry 和 get 过程。Tkinter 可以把当前的值和实际的变量结合起来。但 Tkinter 本身并不支持这种模式，而是提供一个变量类，

它可以为 variable、textvariable、value 和其他的控件修改提供接口。通常，Tkinter 支持 StringVar、IntVar、DoubleVar 和 BooleanVar。这些对象都有 get 和 set 方法。

Example_8_5.py

```
from Tkinter import *

class Var(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()

        self.field = Entry()
        self.field.pack()

        self.value = StringVar()                          ❶
        self.value.set("Jean-Paul Sartre")                ❷
        self.field["textvariable"] = self.value           ❸

        self.field.bind('<Key-Return>', self.print_value)

    def print_value(self, event):
        print 'Value is "%s"' % self.value.get()          ❹

test = Var()
test.mainloop()
```

Example_8_5.py 注解

❶ 切记在创建 Tkinter 变量之前不能直接调用。此处不妨设为 StringVar。

❷ 设置初始值。

❸ 将变量绑定在 textvariable 上。

❹ 用字符串变量的 get 方法获取当前值。

运行该程序可以看到如图 8.5 的对话框，这是最简单的一种对话框。另一方面，这样的对话框并不易于使用。首先，不按“Return”键就不能输入新的信息；再者，对话框本身怎么使用也是一个问题。但不管怎么样，它只是一个使用 Tk 变量的例子。



图 8.5　使用 tk 变量

Pmw 提供了设置和读取数据的方法，所以不直接使用 Tk 变量是可以的。另外，validation、valuechanged 和 selection 只是在适当的时候定义。

Example_8_6.py

```
from Tkinter import *
```

```
from tkSimpleDialog import Dialog
import Pmw

class MixedWidgets(Dialog):
   def body(self, master):
       Label(master, text='Select Case:').grid(row=0, sticky=W)
       Label(master, text='Select Type:').grid(row=1, sticky=W)
       Label(master, text='Enter Value:').grid(row=2, sticky=W)

       self.combo1 = Pmw.ComboBox(master,
                 scrolledlist_items=("Upper","Lower","Mixed"),
                 entry_width=12, entry_state="disabled",
                 selectioncommand = self.ripple)
       self.combo1.selectitem("Upper")
       self.combo1.component('entry').config(bg='gray80')

       self.combo2 = Pmw.ComboBox(master, scrolledlist_items=(),
                        entry_width=12, entry_state="disabled")
       self.combo2.component('entry').config(background='gray80')

       self.entry1  = Entry(master, width = 12)

       self.combo1.grid(row=0, column=1, sticky=W)
       self.combo2.grid(row=1, column=1, sticky=W)
       self.entry1.grid(row=2, column=1, sticky=W)

       return self.combo1

   def apply(self):
       c1 = self.combo1.get()
       c2 = self.combo2.get()
       e1 = self.entry1.get()
       print c1, c2, e1

   def ripple(self, value):
       lookup = {'Upper': ("ANIMAL", "VEGETABLE", "MINERAL"),
                 'Lower': ("animal", "vegetable", "mineral"),
                 'Mixed': ("Animal", "Vegetable", "Mineral")}
       items = lookup[value]
       self.combo2.setlist(items)
       self.combo2.selectitem(items[0])

root = Tk()
dialog = MixedWidgets(root)
```

❶ ❷ ❸ ❹

❺ ❻

Example_8_6.py 注解

❶ ComboBoxes（组合框）在数据输入和选取时很有用处。它最大的好处在于占用空间小，却能给用户选择的范围。

```
self.combo1 = Pmw.ComboBox(master,
          scrolledlist_items=("Upper","Lower","Mixed"),
```

在这个例子中，在选择框中放入了三个可选值。这些数值可以直接从数据库中调用或计算出来。

❷ 这里不需要改变组合框里的值，因此将相应的属性设为“disabled”（不可用），且用与背景相同的灰色来直观地告诉用户它不可用。

```
entry_width=12, entry_state="disabled",
selectioncommand = self.ripple)
```

❸ 这是一个不太常见的现象。通常情况下，同一个屏幕上显示的各区域中的数据是相互独立的。所以，一旦一个区域中的数据改变了，就有必要将其他的相应数据进行刷新。

```
selectioncommand=self.ripple
```

注意　粗心大意地使用刷新是非常危险的！请务必小心使用。因为万一在具体的例子中各个区域中的数据是相互独立的话，就会出错了。另外有一些控制标志可以用来避免 selectioncommand 的循环调用而过多占用 CPU。

具体请见 17.2 节“Tkinter 性能”。

❹ 选择默认值或显示空格要求输入，都不适合于不可编辑的选项框。

```
self.combo1.selectitem("Upper")
```

❺ 这是 ripple 回调函数。Selectioncommand 调用返回了选做变量的那项的值。用这个过程来查一个要被第二个组合框应用的变量。

```
def ripple(self, value):
    lookup = {'Upper': ("ANIMAL", "VEGETABLE", "MINERAL"),
              'Lower': ("animal", "vegetable", "mineral"),
              'Mixed': ("Animal", "Vegetable", "Mineral")}
    items = lookup[value]
```

❻ 新值取代了当前的值。

```
self.combo2.setlist(items)
self.combo2.selectitem(items[0])
```

如前所述，必须在组合框中做出选择。

运行 Example_8_6.py 可以看到这个简单的例子。部分结果见图 8.6。

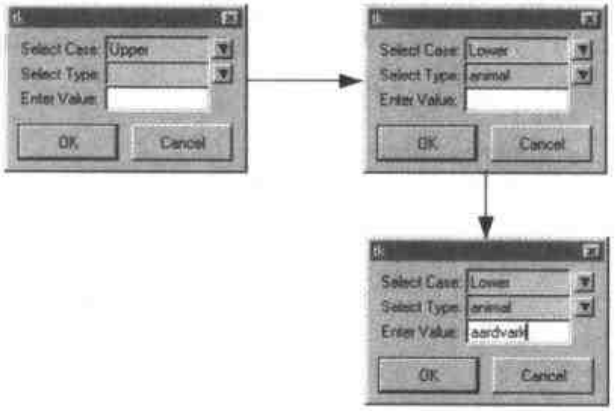


图 8.6　处理控件间的独立

8.2 标准窗体框架

窗体设计中有一个很大的问题，那就是许多窗体都有相类似的外形。于是我们就需要一种标准窗体，可以在实际应用中很方便地构造窗体。这会造成一定的代码重用。许多实际例子都可以使用形如图 8.7 的窗体。另外，我们还要制作出忙状态的光标*、帮助、关于……并加上适当的按钮。由此目的出发，下面介绍 Appshell.py，它提供了最普遍使用的框价格式。自然而然地，这个窗体也并非万能，但它毕竟可以大大减少窗体设计的工作量。

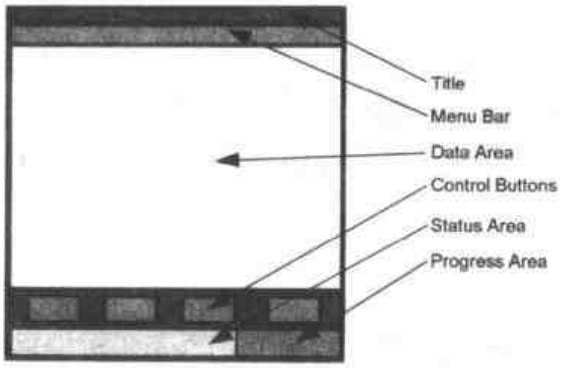


图 8.7 标准应用框架

由于 Appshell.py 在本书中也非常常用，因此下面给出其详细的代码。要想在程序中使用它，请务必理解其中的每一个工具和每一步操作。

AppShell.py

```
from Tkinter import *
import Pmw
import sys, string
import ProgressBar                                          ❶

class AppShell(Pmw.MegaWidget):
    appversion= '1.0'
    appname    = 'Generic Application Frame'
    copyright= 'Copyright YYYY Your Company. All Rights Reserved'
    contactname= 'Your Name'
    contactphone= '(999) 555-1212'
    contactemail= 'youremail@host.com'

    frameWidth= 450
    frameHeight= 320
    padx= 5
    pady= 5
    usecommandarea= 0
    balloonhelp= 1
```

* 忙状态光标常出现在系统任务没有完成的时候，它形如一只表或是一个沙漏。有时也会使窗体内的按钮不可操纵。

```
busyCursor = 'watch'

def __init__(self, **kw):
    optiondefs = (
        ('padx',    1,                  Pmw.INITOPT),
        ('pady',    1,                  Pmw.INITOPT),
        ('framewidth',    1,            Pmw.INITOPT),
        ('frameheight',   1,            Pmw.INITOPT),
        ('usecommandarea', self.usecommandarea, Pmw.INITOPT))
    self.defineoptions(kw, optiondefs)

    self.root = Tk()
    self.initializeTk(self.root)
    Pmw.initialise(self.root)
    self.root.title(self.appname)
    self.root.geometry('%dx%d' % (self.frameWidth,
                                  self.frameHeight))

    # Initialize the base class
    Pmw.MegaWidget.__init__(self, parent=self.root)

    # Initialize the application
    self.appInit()

    # Create the interface
    self.__createInterface()

    # Create a table to hold the cursors for
    # widgets which get changed when we go busy
    self.preBusyCursors = None

    # Pack the container and set focus
    # to ourselves
    self._hull.pack(side=TOP, fill=BOTH, expand=YES)
    self.focus_set()
    # Initialize our options
    self.initialiseoptions(AppShell)

def appInit(self):
    # Called before interface is created (should be overridden).
    pass

def initializeTk(self, root):
    # Initialize platform-specific options
    if sys.platform == 'mac':
        self.__initializeTk_mac(root)
    elif sys.platform == 'win32':
        self.__initializeTk_win32(root)
    else:
```

```
            self.__initializeTk_unix(root)

    def __initializeTk_colors_common(self, root):
        root.option_add('*background', 'grey')
        root.option_add('*foreground', 'black')
        root.option_add('*EntryField.Entry.background', 'white')
        root.option_add('*MessageBar.Entry.background', 'gray85')
        root.option_add('*Listbox*background', 'white')
        root.option_add('*Listbox*selectBackground', 'dark slate blue')
        root.option_add('*Listbox*selectForeground', 'white')

    def __initializeTk_win32(self, root):
        self.__initializeTk_colors_common(root)
        root.option_add('*Font', 'Verdana 10 bold')
        root.option_add('*EntryField.Entry.Font', 'Courier 10')
        root.option_add('*Listbox*Font', 'Courier 10')

    def __initializeTk_mac(self, root):
        self.__initializeTk_colors_common(root)

    def __initializeTk_unix(self, root):
        self.__initializeTk_colors_common(root)
```

Appshell.py 注解

❶ Appshell 导入 ProgressBar。程序没有写出来，但可以从网上获取。

```
Import ProgressBar
```

❷ Appshell 继承自 Pmw MegaWidget。

```
class AppShell(Pmw.MegaWidget):
    appversion= '1.0'
    appname   = 'Generic Application Frame'
    copyright= 'Copyright YYYY Your Company. All Rights Reserved'
    contactname= 'Your Name'
    contactphone= '(999) 555-1212'
    contactemail= 'youremail@host.com'
```

接下来定义几个具有各自缺省值及标题和关于信息的窗体。假设这些值都被覆盖。

❷ 默认的尺寸和要填的数据已经给出。同样我们假设这些值都被覆盖。

```
  frameWidth= 450
  frameHeight= 320
  padx= 5
  pady= 5
  usecommandarea= 0
  balloonhelp= 1
usecommandarea is used to inhibit or display the command(button)area.
```

❸ Appshell 的_init_部分建立由 MegaWidge 提供的选项。

```
def __init__(self, **kw):
    optiondefs = (
        ('padx',    1,                  Pmw.INITOPT),
```

```
            ('pady',    1,                  Pmw.INITOPT),
            ('framewidth',    1,            Pmw.INITOPT),
            ('frameheight',   1,            Pmw.INITOPT),
            ('usecommandarea', self.usecommandarea, Pmw.INITOPT))
        self.defineoptions(kw, optiondefs)
```

Pmw.INITOPT 定义了只能在初始化状态显示的选项，它不能用 configure 调用（详见附录 C）。

❶ 现在我们可以初始化 Tk 和 Pmw，并设置窗体的标题和几何形状了。

```
self.root = Tk()
self.initializeTk(self.root)
Pmw.initialise(self.root)
self.root.title(self.appname)
self.root.geometry('%dx%d' % (self.frameWidth,
                              self.frameHeight))
```

❷ 定义了选项和 Tk 初始化之后，调用基本的类结构。

```
Pmw.MegaWidget.__init__(self, parent=self.root)
```

❸ Appshell 可以支持主要的 Tkinter 样式。后面的几种方法定义了颜色和字体。

AppShell.py(续)

```
    def busyStart(self, newcursor=None):          ❷
    if not newcursor:
        newcursor = self.busyCursor LLLLLLLLL
    newPreBusyCursors = {}

    for component in self.busyWidgets:
         newPreBusyCursors[component] = component['cursor']
         component.configure(cursor=newcursor)
         component.update_idletasks()
    self.preBusyCursors = (newPreBusyCursors, self.preBusyCursors)

def busyEnd(self):
     if not self.preBusyCursors:
          return
     oldPreBusyCursors = self.preBusyCursors[0]
     self.preBusyCursors = self.preBusyCursors[1]

     for component in self.busyWidgets:
          try:
              component.configure(cursor=oldPreBusyCursors[component])
          except KeyError:
              pass
          component.update_idletasks()

def __createAboutBox(self):                   ❸
    Pmw.aboutversion(self.appversion)
    Pmw.aboutcopyright(self.copyright)
    Pmw.aboutcontact(
```

```
            'For more information, contact:\n %s\n Phone: %s\n Email: %s' %\
            (self.contactname, self.contactphone,
             self.contactemail))
        self.about = Pmw.AboutDialog(self._hull,
                             applicationname=self.appname)
        self.about.withdraw()
        return None

    def showAbout(self):
        # Create the dialog to display about and contact information.
        self.about.show()
        self.about.focus_set()

    def toggleBalloon(self):                    ⓜ
        if self.toggleBalloonVar.get():
            self.__balloon.configure(state = 'both')
        else:
            self.__balloon.configure(state = 'status')

    def __createMenuBar(self):                  ●
        self.menuBar = self.createcomponent('menubar', (), None,
                                Pmw.MenuBar,
                                (self._hull,),
                                hull_relief=RAISED,
                                hull_borderwidth=1,
                                balloon=self.balloon())
        self.menuBar.pack(fill=X)
        self.menuBar.addmenu('Help','About %s' % self.appname, side='right')
        self.menuBar.addmenu('File', 'File commands and Quit')

    def createMenuBar(self):
        self.menuBar.addmenuitem('Help', 'command',
                         'Get information on application',
                         label='About...', command=self.showAbout)
        self.toggleBalloonVar = IntVar()
        self.toggleBalloonVar.set(1)
        self.menuBar.addmenuitem('Help', 'checkbutton',
                         'Toggle balloon help',
                         label='Balloon help',
                         variable = self.toggleBalloonVar,
                         command=self.toggleBalloon)

        self.menuBar.addmenuitem('File', 'command', 'Quit this application',
                         label='Quit',
                         command=self.quit)
```

代码注解（续）

● 下面几种方法设置是否使用忙状态光标。

```
def busyStart(self, newcursor=None):
```

```
...
```

❾ 定义支持关于信息的使用，这些信息是提前定义好的，以便使用时能快速弹出。

```
def __createAboutBox(self):
...
```

❿ 帮助信息可以帮助对界面不是很熟悉的用户，但对专业用户是多余的。Appshell 提供的菜单项选择关闭帮助，只留下正常的信息，以免干扰用户使用。

● 菜单条的创建分别由两个方法完成：createMenuBar 创建 Pmw 类的菜单条元素；createMenuBar 则定义具有标准选项的部分，这部分可根据需要增删。

```
def toggleBalloon(self):
    if self.toggleBalloonVar.get():
        self.__balloon.configure(state = 'both')
    else:
        self.__balloon.configure(state = 'status')
```

AppShell.py(续)

```
def __createBalloon(self):                    ●
    # Create the balloon help manager for the frame.
    # Create the manager for the balloon help
    self.__balloon = self.createcomponent('balloon', (), None,
                                          Pmw.Balloon, (self._hull,))

def balloon(self):
    return self.__balloon

def __createDataArea(self):                   ●
    # Create data area where data entry widgets are placed.
    self.dataArea = self.createcomponent('dataarea',
                                         (), None,
                                         Frame, (self._hull,),
                                         relief=GROOVE,
                                         bd=1)
    self.dataArea.pack(side=TOP, fill=BOTH, expand=YES,
                       padx=self['padx'], pady=self['pady'])

def __createCommandArea(self):                ●
    # Create a command area for application-wide buttons.
    self.__commandFrame=self.createcomponent('commandframe',(),None,
                                             Frame,
                                             (self._hull,),
                                             relief=SUNKEN,
                                             bd=1)
    self.__buttonBox=self.createcomponent('buttonbox',(),None,
                                          Pmw.ButtonBox,
                                          (self.__commandFrame,),
                                          padx=0, pady=0)
    self.__buttonBox.pack(side=TOP, expand=NO, fill=X)
    if self['usecommandarea']:
```

```
            self.__commandFrame.pack(side=TOP,
                                     expand=NO,
                                     fill=X,
                                     padx=self['padx'],
                                     pady=self['pady'])

    def __createMessageBar(self):
        # Create the message bar area for help and status messages.
        frame = self.createcomponent('bottomtray', (), None,
                                     Frame,(self._hull,), relief=SUNKEN)
        self.__messageBar = self.createcomponent('messagebar',
                                     (), None,
                                     Pmw.MessageBar,
                                     (frame,),
                                     #entry_width = 40,
                                     entry_relief=SUNKEN,
                                     entry_bd=1,
                                     labelpos=None)
        self.__messageBar.pack(side=LEFT, expand=YES, fill=X)

        self.__progressBar = ProgressBar.ProgressBar(frame,
                                     fillColor='slateblue',
                                     doLabel=1,
                                     width=150)
        self.__progressBar.frame.pack(side=LEFT, expand=NO, fill=NONE)

        self.updateProgress(0)
        frame.pack(side=BOTTOM, expand=NO, fill=X)

        self.__balloon.configure(statuscommand = \
                        self.__messageBar.helpmessage)

    def messageBar(self):
        return self.__messageBar

    def updateProgress(self, newValue=0, newLimit=0):
        self.__progressBar.updateProgress(newValue, newLimit)

    def bind(self, child, balloonHelpMsg, statusHelpMsg=None):
        # Bind a help message and/or status message to a widget.
        self.__balloon.bind(child, balloonHelpMsg, statusHelpMsg)

    def interior(self):
        # Retrieve the interior site where widgets should go.
        return self.dataArea

    def buttonBox(self):
        # Retrieve the button box.
        return self.__buttonBox
```

```
    def buttonAdd(self, buttonName, helpMessage=None,
                  statusMessage=None, **kw):
        # Add a button to the button box.
        newBtn = self.__buttonBox.add(buttonName)
        newBtn.configure(kw)
        if helpMessage:
            self.bind(newBtn, helpMessage, statusMessage)
        return newBtn
```

●

代码注解（续）

● 创建帮助部分。

```
def __createBalloon(self):
    self.__balloon = self.createcomponent('balloon', (), None,
                                          Pmw.Balloon, (self._hull,))
```

● dataarea 区域仅用来放所有其他的控件。

```
def __createDataArea(self):
    self.dataArea = self.createcomponent('dataarea',
                                         (), None,
                                         Frame, (self._hull,),
                                         relief=GROOVE,
                                         bd=1)
```

● commandarea 框架封装了 Pmw ButtonBox。

```
def __createCommandArea(self):
    self.__commandFrame=self.createcomponent('commandframe',(),None,
                                             Frame,
                                             (self._hull,),
                                             relief=SUNKEN,
                                             bd=1)
    self.__buttonBox=self.createcomponent('buttonbox',(),None,
                                          Pmw.ButtonBox,
                                          (self.__commandFrame,),
                                          padx=0, pady=0)
```

● 相似地 messagebar 框架封装 MessageBox。

```
def __createMessageBar(self):
    ...
```

● 为了完成主要的部分在 progressbar 元素旁创建 messagebar。

```
self.__progressBar = ProressBar.ProgressBar(frame,
...
```

● Pmw 规定返回了参数的方法，不妨称其为 interior。

```
def interior(self):
    return self.dataArea
```

● 在 commandarea 中创建按钮，并绑定信息条和帮助信息。

```
def buttonAdd(self, buttonName, helpMessage=None,
              statusMessage=None, **kw):
    newBtn = self.__buttonBox.add(buttonName)
    newBtn.configure(kw)
    if helpMessage:
        self.bind(newBtn, helpMessage, statusMessage)
```

```
        return newBtn
```

AppShell.py(续)

```
    def __createInterface(self):
        self.__createBalloon()
        self.__createMenuBar()
        self.__createDataArea()
        self.__createCommandArea()
        self.__createMessageBar()
        self.__createAboutBox()
        #
        # Create the parts of the interface
        # which can be modified by subclasses
        #
        self.busyWidgets = ( self.root, )
        self.createMenuBar()
        self.createInterface()

    def createInterface(self):
        # Override this method to create the interface for the app.
        pass

    def main(self):
        self.pack()
        self.mainloop()

    def run(self):
        self.main()

class TestAppShell(AppShell):
    usecommandarea=1

    def createButtons(self):
        self.buttonAdd('Ok',
                       helpMessage='Exit',
                       statusMessage='Exit',
                       command=self.quit)

    def createMain(self):
        self.label = self.createcomponent('label', (), None,
                                Label,
                                (self.interior(),),
                                text='Data Area')
        self.label.pack()
        self.bind(self.label, 'Space taker')

    def createInterface(self):
        AppShell.createInterface(self)
        self.createButtons()
        self.createMain()
```

```
if __name__ == '__main__':
   test = TestAppShell(balloon_state='both')
   test.run()
```

代码注解（续）

● CreateInterface 创建了每一个标准区域并且调用 CreateInterface 方法来完成各种区域的细化。

```
def __createInterface(self):
     self.__createBalloon()
     self.__createMenuBar()
     self.__createDataArea()
     self.__createCommandArea()
     self.__createMessageBar()
     self.__createAboutBox()
```

● 在这个例子中仅用一个按钮来选择退出，这在程序的其他地方也可使用。

```
def createButtons(self):
     self.buttonAdd('Ok',
                    helpMessage='Exit',
                    statusMessage='Exit',
                    command=self.quit)
```

● 同样为了举例 dataarea 被设置得比较具体。注意这里的帮助是怎样定义的。

```
def createMain(self):
     self.label = self.createcomponent('label', (), None,
                              Label,
                              (self.interior(),),
                              text='Data Area')
     self.label.pack()
     self.bind(self.label, 'Space taker')
```

● 最后的 createInterface 是对 Appshell 的扩展。

```
def createInterface(self):
      AppShell.createInterface(self)
      self.createButtons()
      self.createMain()
```

运行程序，结果类似于图 8.8，注意看看菜单中的帮助部分。

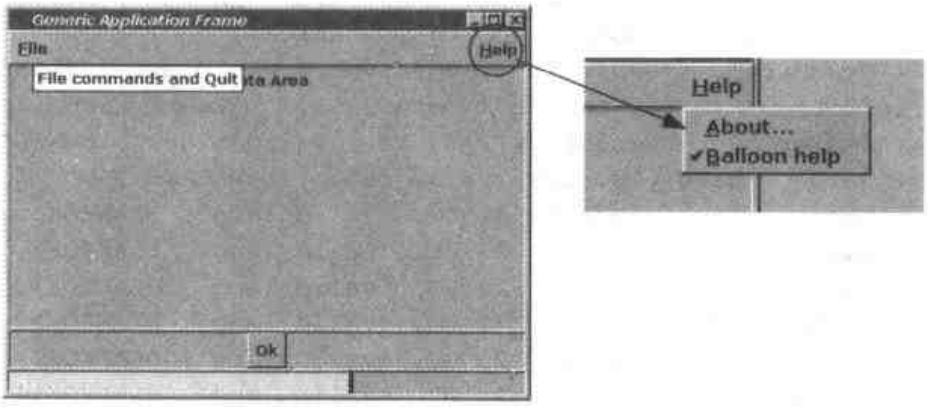


图 8.8　AppShell-标准应用框架

8.3 数据字典

前面向大家介绍的窗体的代码都已经很清楚地写出来了，但是当一个程序中有很多个窗体的时候就显得过于重复。解决这个问题的办法是：使用数据字典。顾名思义数据字典就是用来定义各种控件的标签、控件类型等信息的。此外它还可以用来联系后台数据库和前台的显示内容，而且定义确定的条件、可编辑区域和其他的事件。在“组装起来”中可以看到更多的例子，而这里的例子会让你对数据字典在简化窗体设计上的重要作用有深刻的认识。

首先来看一个简单的数据字典，它可谓是一个正宗的 Python 数据字典，但显然表达方式不是唯一的。

datadictionary.py

```
LC    = 1      # Lowercase Key      ❶
UC    = 2      # Uppercase Key
XX    = 3      # As Is
DT    = 4      # Date Insert
ND    = 5      # No Duplicate Keys
ZP    = 6      # Pad Zeroes
ZZ    = 7      # Do Not Display
ZS    = 8      # Do Not display, but fill in with key if blank

BLANKOK  = 0   # Blank is valid in this field
NONBLANK = 1   # Field cannot be blank

dataDict = {
   'crewmembers': ('crewmembers',    0.11, 0.45, 0.05, [                    ❶
       ('Employee #', 'employee_no', 9, XX, 'valid_blank', NONBLANK),       ❷
       ('PIN', 'pin', 4, XX, '', BLANKOK),
       ('Category', 'type', 1, UC, 'valid_category', NONBLANK),
       ('SSN #', 'ssn', 9, XX, 'valid_ssn', BLANKOK),
       ('First Name', 'firstname', 12, XX, 'valid_blank', NONBLANK),
       ('Middle Name', 'middlename', 10, XX, '', BLANKOK),
       ('Last Name', 'lastname', 20, XX, 'valid_blank', NONBLANK),
       ('Status', 'status', 1, UC, '', BLANKOK),
       ('New Hire', 'newhire', 1, UC, 'valid_y_n_blank', BLANKOK),
       ('Seniority Date', 'senioritydate', 8, XX, 'valid_blank', NONBLANK),
       ('Seniority', 'seniority', 5, XX, 'valid_blank', NONBLANK),
       ('Base', 'base', 3, UC, 'valid_base', NONBLANK),
       ('Language 1', 'lang1', 2, UC, 'valid_lang', BLANKOK),
       ('Language 2', 'lang2', 2, UC, 'valid_lang', BLANKOK),
       ('Language 3', 'lang3', 2, UC, 'valid_lang', BLANKOK),
       ('Language 4', 'lang4', 2, UC, 'valid_lang', BLANKOK),
       ('Language 5', 'lang5', 2, UC, 'valid_lang', BLANKOK),
       ('Language 6', 'lang6', 2, UC, 'valid_lang', BLANKOK)],
            'Crew Members', [0]),                                           ❶
```

```
    'crewqualifications': ('crewqualification',0.25,0.45,0.075, [
        ('Employee #', 'employee_no', 9, XX, '', BLANKOK),
        ('Equipment', 'equipment', 3, UC, '', BLANKOK),
        ('Eqpt. Code', 'equipmentcode', 1, UC, '', BLANKOK),
        ('Position', 'position', 2, UC, '', BLANKOK),
        ('Pos. Code', 'positioncode', 2, UC, '', BLANKOK),
        ('Reserve', 'reserve', 1, UC, 'valid_r_blank', BLANKOK),
        ('Date of Hire', 'hiredate', 8, UC, '', BLANKOK),
        ('End Date', 'enddate', 8, UC, '', BLANKOK),
        ('Base Code', 'basecode', 1, UC, '', BLANKOK),
        ('Manager', 'manager', 1, UC, 'valid_y_n_blank', BLANKOK)],
                    'Crew Qualifications', [0]) }
```

Datadictionary.py 注解

❶ 定义几个常量来个性化输入区和控制方式。

```
LC    = 1      # Lowercase Key
UC    = 2      # Uppercase Key
XX    = 3      # As Is
...
```

❷ 字典输入取得第一个部分定义了关键字、数据表格和结构数据，一次定义表格第一行的位置和行距等。

```
'crewmembers': ('crewmembers',  0.11, 0.45, 0.05, [
```

❸ 每一项数据值定义了标签、数据库关键字、区域长度、输入处理的确认和区域是否为空。

```
('Employee #', 'employee_no', 9, XX, 'valid_blank', NONBLANK),
('PIN', 'pin', 4, XX, '', BLANKOK),
('Category', 'type', 1, UC, 'valid_category', NONBLANK),
```

❹ 最后的输入值定义了标题和次级关键字的列表（本例只用单一关键字）。

```
'Crew Members',[0]),
```

现在我们一起来用数据字典创建一个用户界面，同时用 Appshell 来构造框架。

Example 8_7.py

```
from  Tkinter import *
import Pmw
import os
import AppShell
from  datadictionary import *

class DDForm(AppShell.AppShell):          ❶
  usecommandarea = 1
  appname        = 'Update Crew Information'
  dictionary     = 'crewmembers'
  frameWidth     = 600
  frameHeight    = 590
```

```
    def createButtons(self):                                    ❷
        self.buttonAdd('Save',
                       helpMessage='Save current data',
                       statusMessage='Write current information to database',
                       command=self.unimplemented)
        self.buttonAdd('Undo',
                       helpMessage='Ignore changes',
                       statusMessage='Do not save changes to database',
                       command=self.unimplemented)
        self.buttonAdd('New',
                       helpMessage='Create a New record',
                       statusMessage='Create New record',
                       command=self.unimplemented)
        self.buttonAdd('Delete',
                       helpMessage='Delete current record',
                       statusMessage='Delete this record',
                       command=self.unimplemented)
        self.buttonAdd('Print',
                       helpMessage='Print this screen',
                       statusMessage='Print data in this screen',
                       command=self.unimplemented)
        self.buttonAdd('Prev',
                       helpMessage='Previous record',
                       statusMessage='Display previous record',
                       command=self.unimplemented)
        self.buttonAdd('Next',
                       helpMessage='Next record',
                       statusMessage='Display next record',
                       command=self.unimplemented)
        self.buttonAdd('Close',
                       helpMessage='Close Screen',
                       statusMessage='Exit',
                       command=self. unimplemented)

    def createForm(self):                                       ❸
        self.form = self.createcomponent('form', (), None,
                              Frame, (self.interior(),),)
        self.form.pack(side=TOP, expand=YES, fill=BOTH)
        self.formwidth = self.root.winfo_width()

    def createFields(self):
        self.table,self.top,self.anchor,self.incr,self.fields,\
                self.title,self.keylist = dataDict[self.dictionary]   ❹
        self.records= []
        self.dirty= FALSE
        self.changed= []
        self.newrecs= []
        self.deleted= []
        self.checkDupes = FALSE
        self.delkeys= []
```

```
        self.ypos = self.top                                    ❹
        self.recrows = len(self.records)
        if self.recrows < 1:  # Create one!
            self.recrows = 1
            trec = []
            for i in range(len(self.fields)):
                trec.append(None)
                self.records.append((trec))

        Label(self.form, text=self.title, width=self.formwidth-4,    ❺
             bd=0).place(relx=0.5, rely=0.025, anchor=CENTER)
        self.lmarker = Label(self.form, text="", bd=0, width=10)
        self.lmarker.place(relx=0.02, rely=0.99, anchor=SW)
        self.rmarker = Label(self.form, text="", bd=0, width=10)
        self.rmarker.place(relx=0.99, rely=0.99, anchor=SE)

        self.current = 0
        idx = 0
        for label, field, width, proc, valid, nonblank in self.fields: ❻
            pstr = 'Label(self.form,text="%s").place(relx=%f,rely=%f,'\
            'anchor=E)\n' % (label, (self.anchor-0.02), self.ypos)
            if idx == self.keylist[0]:
                 pstr = '%sself.%s=Entry(self.form,text="",'\
                 'insertbackground="yellow", width=%d+1,'\
                 'highlightthickness=1)\n' % (pstr,field,width)
            else:
                 pstr = '%sself.%s=Entry(self.form,text="",'\
                 'insertbackground="yellow",'\
                 'width=%d+1)\n' % (pstr,field,width)
                 pstr = '%sself.%s.place(relx=%f, rely=%f,'
                 'anchor=W)\n'%(pstr,field,(self.anchor+0.02),self.ypos)
            exec '%sself.%sV=StringVar()\n'\
            'self.%s["textvariable"] = self.%sV' % \
                            (pstr,field,field,field)
            self.ypos = self.ypos + self.incr
            idx = idx + 1
        self.update_display()

    def update_display(self):                                   ❼
        idx = 0
        for label, field, width, proc, valid, nonblank in self.fields:
             v=self.records[self.current][idx]
             if not v:v=""
             exec 'self.%sV.set(v)' % field
             idx = idx + 1
        if self.current in self.deleted:
             self.rmarker['text'] = 'Deleted'
        elif self.current in self.newrecs:
             self.rmarker['text'] = 'New'
```

```
        else:
             self.rmarker['text'] = ''
        if self.dirty:
            self.lmarker['text']       = "Modified"
            self.lmarker['foreground'] = "#FF3333"
        else:
            self.lmarker['text']       = ""
            self.lmarker['foreground'] = "#00FF44"
        # We'll set focus on the first widget
        label, field, width, proc, valid, nonblank = self.fields[0]
        exec 'self.%s.focus_set()' % field

    def update_display(self):          ❶
         pass

    def createInterface(self):
         AppShell.AppShell.createInterface(self)
         self.createButtons()
         self.createForm()
         self.createFields()

if __name__ == '__main__':
    ddform = DDForm()
    ddform.run()
```

Example_8_7.py 注解

❶ 先定义应用类，由 Appshell 继承而来。同样可设置标题、长度、宽度等值。

```
class DDForm(AppShell.AppShell):
    usecommandarea = 1
    appname        = 'Update Crew Information'
    dictionary     = 'crewmembers'
    frameWidth     = 600
    frameHeight    = 590
```

❷ 本例中会定义更多的属性让控制按钮更加逼真。

```
def createButtons(self):
     self.buttonAdd('Save',
                    helpMessage='Save current data',
                    statusMessage='Write current information to database',
                    command=self.save)
```

❸ 此处直接定义窗体元素而非使用默认的大部件 MegaWidget。

```
def createForm(self):
     self.form = self.createcomponent('form', (), None,'
                           Frame, (self.interior(),),)
     self.form.pack(side=TOP, expand=YES, fill=BOTH)
     self.formwidth = self.root.winfo_width()
```

❹ 从已选好的数据字典中取出数据并初始化数据结构。

```
def createFields(self):
```

```
        self.table,self.top,self.anchor,self.incr,self.fields,\
                self.title,self.keylist = dataDict[self.dictionary]
        self.records= []
        self.dirty= FALSE
```

❻ 此例不需要数据库，但仍然有必要创建一个空白的纪录。即为每个区域创建空白的输入。

```
self.ypos = self.top
self.recrows = len(self.records)
if self.recrows < 1:  # Create one!
        self.recrows = 1
        trec = []
        for i in range(len(self.fields)):
            trec.append(None)
            self.records.append((trec))
```

❼ 尽管不能保存输入的信息，但我们还是在屏幕的左下和右下角定义一些符号来表明纪录的修改或添加。

```
Label(self.form, text=self.title, width=self.formwidth-4,
        bd=0).place(relx=0.5, rely=0.025, anchor=CENTER)
self.lmarker = Label(self.form, text="", bd=0, width=10)
self.lmarker.place(relx=0.02, rely=0.99, anchor=SW)
self.rmarker = Label(self.form, text="", bd=0, width=10)
self.rmarker.place(relx=0.99, rely=0.99, anchor=SE)
```

❽ 这里我们创建了标签栏来组成界面，以此告知用户高亮显示的部分是关键词。

```
for label, field, width, proc, valid, nonblank in self.fields:
        pstr = 'Label(self.form,text="%s").place(relx=%f,rely=%f,'\
        'anchor=E)\n' % (label, (self.anchor-0.02), self.ypos)
        if idx == self.keylist[0]:
             pstr = '%sself.%s=Entry(self.form,text="",'\
             'insertbackground="yellow", width=%d+1,'\
             'highlightthickness=1)\n' % (pstr,field,width)
else:
    ...
```

注意　在这个例子中，我们选择用 highlightthickness 来向用户提供直观的指示——这些内容是数据的关键字。你也可用别的方式，比如改变背景色或改变边框的宽度。

❾ Update_display 方法用于设置“新记录、删除记录、修改记录”等标志。

```
def update_display(self):
        idx = 0
        for label, field, width, proc, valid, nonblank in self.fields:
             v=self.records[self.current][idx]
             if not v:v=""
             exec 'self.%sV.set(v)' % field
             idx = idx + 1
        if self.current in self.deleted:
```

...

❶ 在这个例子中绑定在控制钮上的方法并不能实现什么功能，但 Python 规定必须要在按钮上绑定相应的方法。

```
def unimplemented(self):
    pass
```

运行 Example_8_7.py 显示如图 8.9。可见当各个区域分开或同一行有不止一个区域时就会自动扩展，但也给使用数据字典带来一些麻烦。

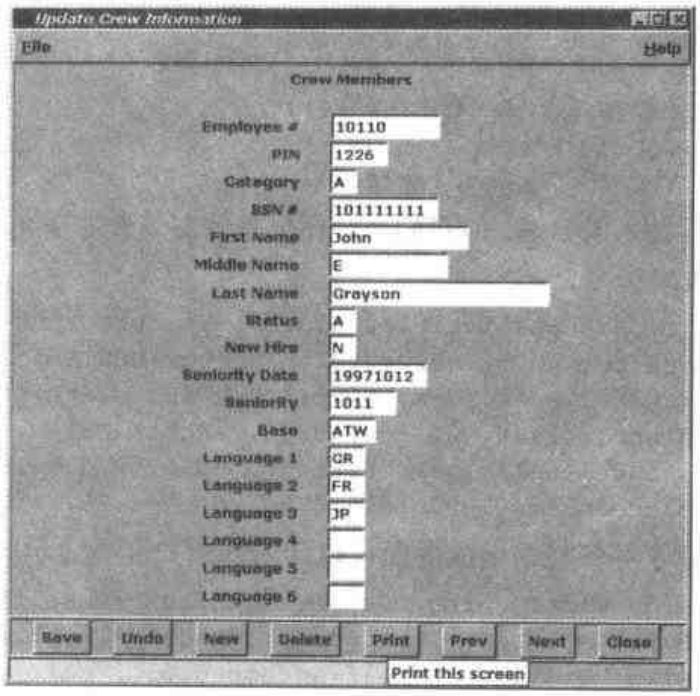


图 8.9　从数据字典产生的屏幕

8.4　活页夹

活页夹已经成为用户界面的一种潮流。其一大优点就是可以让设计者在同一区域中添加更多的东西而不会让用户如坠云雾中。另外，相关的部分可以组合在一起，同样也可以把经常要变换的部分分成若干个小区域。

下面的例子示范了活页夹、数据字典和 Appshell 的使用。将 Example_8_7.py 的内容分开放在三个活页夹中。Datadictionary.py 另存为 Datadictionary2.py 保存而将前者分开使用。

Example_8_9.py

```
from   Tkinter import *
import Pmw
import os
import AppShell
```

```
from  datadictionary2 import *

class DDNotebook(AppShell.AppShell):
   usecommandarea = 1
   appname        = 'Update Crew Information'
   dictionary     = 'crewmembers'
   frameWidth     = 435
   frameHeight    = 520

   def createButtons(self):
       self.buttonAdd('Save',
                  helpMessage='Save current data',
                  statusMessage='Write current information to database',
                  command=self.save)
       self.buttonAdd('Close',
                  helpMessage='Close Screen',
                  statusMessage='Exit',
                  command=self.close)

   def createNotebook(self):
       self.notebook = self.createcomponent('notebook', (), None,
                                 Pmw.NoteBookR, (self.interior(),),)
       self.notebook.pack(side=TOP, expand=YES, fill=BOTH, padx=5, pady=5)
       self.formwidth = self.root.winfo_width()

   def addPage(self, dictionary):
       table, top, anchor, incr, fields, \
              title, keylist = dataDict[dictionary]
       self.notebook.add(table, label=title)
       self.current = 0
       ypos = top
       idx = 0

       for label, field, width, proc, valid, nonblank in fields:
           pstr = 'Label(self.notebook.page(table).interior(),'\
                  'text="%s").place(relx=%f,rely=%f, anchor=E)\n' % \
                  (label, (anchor-0.02), ypos)
               if idx == keylist[0]:
               pstr='%sself.%s=Entry(self.notebook.page(table). \
                    'interior(),'text="",insertbackground="yellow",'
                    'width=%d+1,'highlightthickness=1)\n' % \
                             (pstr,field,width)
               else:
                   pstr = '%sself.%s=Entry(self.notebook.page(table).\
                          'interior(),text="", insertbackground="yellow"',
                     'width=%d+1)\n' % (pstr,field,width)
       pstr = '%sself.%s.place(relx=%f, rely=%f,'\
              'anchor=W)\n' % (pstr,field,(anchor+0.02),ypos)
       exec '%sself.%sV=StringVar()\n'\
            'self.%s["textvariable"] = self.%sV' % (pstr,field,field,field)
```

```
            ypos = ypos + incr
            idx = idx + 1

        def createPages(self):                    ❶
            self.addPage('general')
            self.addPage('language')
            self.addPage('crewqualifications')
            self.update_display()

        def update_display(self):
            pass

        def save(self):
            pass
        def close(self):
            self.quit()

        def createInterface(self):
            AppShell.AppShell.createInterface(self)
            self.createButtons()
            self.createNotebook()
            self.createPages()

    if __name__ == '__main__':
        ddnotebook = DDNotebook()
        ddnotebook.run()
```

Example_8_9.py 注解

❶ 在 Appshell 中创建活页夹只要使用 Pmw NoteBookR 元素就可以了。

```
def createNotebook(self):
    self.notebook = self.createcomponent('notebook', (), None,
                                Pmw.NoteBookR, (self.interior(),),)
    self.notebook.pack(side=TOP, expand=YES, fill=BOTH, padx=5, pady=5)
```

Pmw 提供了可更新的活页夹控件 NotebookS（见图 8.10）。如果已经存在下层的窗体，则不推荐使用这一控件。

❶ 显示在活页夹标签上的名字直接由数据字典中而来。

```
def addPage(self, dictionary):
    table, top, anchor, incr, fields, \
          title, keylist = dataDict[dictionary]
    self.notebook.add(table, label=title)
```

❶ 调用数据字典中的数据同前例一样。

```
for label, field, width, proc, valid, nonblank in fields:
    pstr = 'Label(self.notebook.page(table).interior(),'\
          'text="%s").place(relx=%f,rely=%f, anchor=E)\n' % \
          (label, (anchor-0.02), ypos)
    ...
```

❶ 分页按数据字典中的关键字命名。

图 8.10 给出了运行结果，可以看到现在的页面是何等清爽，逻辑是何等清晰！

```
def createPages(self):
    self.addPage('general')
    self.addPage('language')
    self.addPage('crewqualifications')
    self.update_display()
```

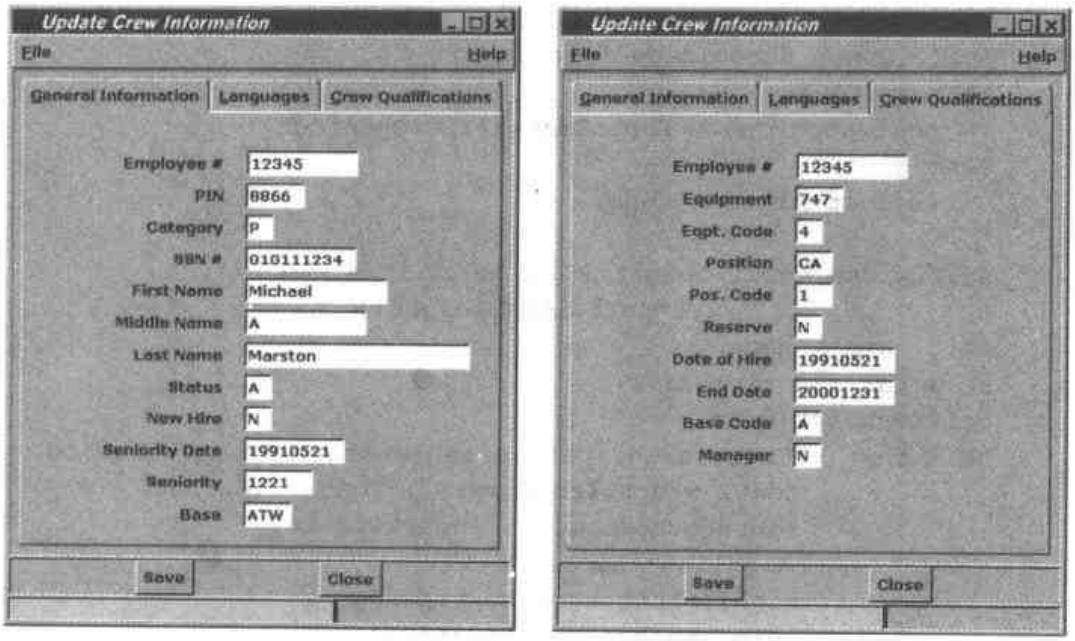


图 8.10　记事本

8.5　浏览器

由于可以分层次显示文件结构，浏览器已经成为导航器的主流形式。比较好的例子就是 Netscape 和 Windows Explorer。它的好处在于可以选择是否显示下级的树形结构，以避免冗长的显示。

作为举例，我们来看一个简单的图片浏览器，它将显示指定目录中的所有图片。Tk 支持三种图片格式：gif、ppm（真彩色）和 XBM。为了扩展例子，我们用 Secret LabsA.B 的 PIL 来创建图片。这并不会使例子复杂很多。

浏览器使用不同的图标来指示不同的文件格式。为简单起见，我们在例子中混用一种图标。

树形浏览器类很具有一般性，可以在别的程序中使用。

Example 8_10.py

```
from  Tkinter import *
import Pmw
import os
import AppShell
import Image, ImageTk
```

```
path = "./icons/"
imgs = "./images/"

class Node:                                          ❶
   def __init__(self, master, tree, icon=None,
                openicon=None, name=None, action=None):
       self.master, self.tree = master, tree
       self.icon = PhotoImage(file=icon)
       if openicon:
          self.openicon = PhotoImage(file=openicon)
       else:
          self.openicon = None

       self.width, self.height = 1.5*self.icon.width(), \
                                 1.5*self.icon.height()
       self.name = name
       self.var = StringVar()                        ❷
       self.var.set(name)
       self.text = Entry(tree, textvariable=self.var, bg=tree.bg,
                   bd=0, width=len(name)+2, font=tree.font,
                   fg=tree.textcolor, insertwidth=1,
                   highlightthickness=1,                    ❸
                   highlightbackground=tree.bg,
                   selectbackground="#044484",
                   selectborderwidth=0,
                   selectforeground='white')
       self.action = action
       self.x = self.y = 0  #drawing location
       self.child = []
       self.state = 'colapsed'
       self.selected = 0

   def addChild(self, tree, icon=None, openicon=None, name=None,    ❹
                action=None):
       child = Node(self, tree, icon, openicon, name, action)
       self.child.append(child)
       self.tree.display()
       return child

   def deleteChild(self, child):
       self.child.remove(child)
       self.tree.display()

   def textForget(self):
       self.text.place_forget()
       for child in self.child:
           child.textForget()

   def deselect(self):
```

```
        self.selected = 0
        for child in self.child:
            child.deselect()

    def boxpress(self, event=None):                    ❹
        if self.state == 'expanded':
            self.state = 'colapsed'
        elif self.state == 'colapsed':
            self.state = 'expanded'
        self.tree.display()

    def invoke(self, event=None):                      ❺
        if not self.selected:
            self.tree.deselectall()
            self.selected = 1
            self.tree.display()
            if self.action:
                self.action(self.name)
        self.name = self.text.get()
        self.text.config(width=len(self.name)+2)
```

Example_8_10.py 注解

❶ 从导入 PIL 模块开始。

```
Import image,imagetk
```

❷ 节点类定义子级树形，且有开、关的树形与之对应。

```
class Node:
    def __init__(self, master, tree, icon=None,
                 openicon=None, name=None, action=None):
        ...
```

❸ 每一个节点都可以重命名（但本例中不可）。

```
self.name = name
self.var = StringVar()
self.var.set(name)
self.text = Entry(tree, textvariable=self.var, bg=tree.bg,
```

❸ 输入控件在默认状态下不高亮显示。添加高亮部分显示目标被选中状态。

❸ 当创建节点的子级节点时又适用节点类的 addchild 方法。

```
def addChild(self, tree, icon=None, openicon=None, name=None,
             action=None):
    child = Node(self, tree, icon, openicon, name, action)
    self.child.append(child)
    self.tree.display()
    return child
```

程序创建字节点并加入子级列表。

❹ Boxpress 方法设置节点的显示方式：点击-打开子层、点击-关闭子层。

```
def boxpress(self, event=None):
    if self.state == 'expanded':
```

```
            self.state = 'colapsed'
        elif self.state == 'colapsed':
            self.state = 'expanded'
        self.tree.display()
```

❻ 若目录没选中，invoke 方法支持单击、双击两种动作选择所需节点。

```
def invoke(self, event=None):
    if not self.selected:
        self.tree.deselectall()
        self.selected = 1
        self.tree.display()
        if self.action:
            self.action(self.name)
    self.name = self.text.get()
    self.text.config(width=len(self.name)+2)
```

Example 8_10.py 续 1

```
def displayIconText(self):                          ❺
    tree, text = self.tree, self.text
    if self.selected and self.openicon:
        self.pic = tree.create_image(self.x, self.y,
                                     image=self.openicon)
    else:
        self.pic = tree.create_image(self.x, self.y,
                                     image=self.icon)

    text.place(x=self.x+self.width/2, y=self.y, anchor=W)
    text.bind("<ButtonPress-1>", self.invoke)
    tree.tag_bind(self.pic, "<ButtonPress-1>", self.invoke, "+")
    text.bind("<Double-Button-1>", self.boxpress)
    tree.tag_bind(self.pic, "<Double-Button-1>",
                  self.boxpress, "+")

def displayRoot(self):
    if self.state == 'expanded':
        for child in self.child:
            child.display()
    self.displayIconText()

def displayLeaf(self):                                  ❻
    self.tree.hline(self.y, self.master.x+1, self.x)
    self.tree.vline(self.master.x, self.master.y, self.y)
    self.displayIconText()

def displayBranch(self):                                ❼
    master, tree = self.master, self.tree
    x, y = self.x, self.y
    tree.hline(y, master.x, x)
    tree.vline(master.x, master.y, y)
    if self.state == 'expanded' and self.child != []:
```

```
            for child in self.child:
                child.display()
            box = tree.create_image(master.x, y,
                                    image=tree.minusnode)
        elif self.state == 'colapsed' and self.child != []:
            box = tree.create_image(master.x, y,
                                    image=tree.plusnode)
        tree.tag_bind(box, "<ButtonPress-1>", self.boxpress, "+")
        self.displayIconText()

    def findLowestChild(self, node):
        if node.state == 'expanded' and node.child != []:
            return self.findLowestChild(node.child[-1])
        else:
            return node

    def display(self):
        master, tree = self.master, self.tree
        n = master.child.index(self)
        self.x = master.x + self.width
        if n == 0:
            self.y = master.y + (n+1)*self.height
        else:
            previous = master.child[n-1]
            self.y = self.findLowestChild(previous).y + self.height

        if master == tree:
            self.displayRoot()
        elif master.state == 'expanded':
            if self.child == []:
                self.displayLeaf()
            else:
                self.displayBranch()
           tree.lower('line')

class Tree(Canvas):
    def __init__(self, master, icon, openicon, treename, action,
                 bg='white', relief='sunken', bd=2,
                 linecolor='#808080', textcolor='black',
                 font=('MS Sans Serif', 8)):
        Canvas.__init__(self, master, bg=bg, relief=relief, bd=bd,
                        highlightthickness=0)
        self.pack(side='left', anchor=NW, fill='both', expand=1)

        self.bg, self.font= bg, font
        self.linecolor, self.textcolor= linecolor, textcolor
        self.master = master
        self.plusnode = PhotoImage(file=os.path.join(path,'plusnode.gif'))
        self.minusnode = PhotoImage(file=os.path.join(path,'minusnode.gif'))
        self.inhibitDraw = 1
```

```
        self.imageLabel = None
        self.imageData = None
        self.child = []
        self.x = self.y  = -10

        self.child.append( Node( self, self, action=action,
                        icon=icon, openicon=openicon, name=treename))

    def display(self):
        if self.inhibitDraw: return
        self.delete(ALL)
        for child in self.child:
            child.textForget()
            child.display()

    def deselectall(self):
        for child in self.child:
            child.deselect()

    def vline(self, x, y, y1):                          ●
        for i in range(0, abs(y-y1), 2):
            self.create_line(x, y+i, x, y+i+1, fill=self.linecolor,
                        tags='line')

    def hline(self, y, x, x1):
        for i in range(0, abs(x-x1), 2):
            self.create_line(x+i, y, x+i+1, y, fill=self.linecolor,
                        tags='line')
```

代码注解（续）

● DisplayIconText 显示目录的开关状态以及“单击或双击”打开方式的选择。

```
def displayIconText(self):
    tree, text = self.tree, self.text
    if self.selected and self.openicon:
        self.pic = tree.create_image(self.x, self.y,
                            image=self.openicon)
    ...
    ...
    text.bind("<ButtonPress-1>", self.invoke)
    tree.tag_bind(self.pic, "<ButtonPress-1>", self.invoke, "+")
    text.bind("<Double-Button-1>", self.boxpress)
    tree.tag_bind(self.pic, "<Double-Button-1>",
                self.boxpress, "+")
```

● DisplayLeaf 用短竖线条将图标连成树形。

```
def displayLeaf(self):
    self.tree.hline(self.y, self.master.x+1, self.x)
    self.tree.vline(self.master.x, self.master.y, self.y)
    self.displayIconText()
```

● 同样 displayBranch 描绘连线和目录。

```
def displayBranch(self):
     master, tree = self.master, self.tree
     x, y = self.x, self.y
     tree.hline(y, master.x, x)
     tree.vline(master.x, master.y, y)
     if self.state == 'expanded' and self.child != []:
          for child in self.child:
               child.display()
          box = tree.create_image(master.x, y,
                         image=tree.minusnode)
     elif self.state == 'colapsed' and self.child != []:
          box = tree.create_image(master.x, y,
                         image=tree.plusnode)
     tree.tag_bind(box, "<ButtonPress-1>", self.boxpress, "+")
     self.displayIconText()
```

● findLowestChild 方法可以直接找到指定目录的最低层。

```
def findLowestChild(self, node):
     if node.state == 'expanded' and node.child != []:
          return self.findLowestChild(node.child[-1])
     else:
         return node
```

● 定义标志 inhibitDraw 使得添加节点的时候无需重绘树形图。这一点为创建复杂的树形节省了时间，也减轻了中央处理器的负担。

```
Self.inhibitDraw=1
```

● vline 和 hline 是特定的绘制短竖线的规则。

```
def vline(self, x, y, y1):
     for i in range(0, abs(y-y1), 2):
         self.create_line(x, y+i, x, y+i+1, fill=self.linecolor,
                   tags='line')
```

Example 8_10.py

```
class ImageBrowser(AppShell.AppShell):
  usecommandarea=1
  appname = 'Image Browser'
  def createButtons(self):
      self.buttonAdd('Ok',
                  helpMessage='Exit',
                  statusMessage='Exit',
                  command=self.quit)

  def createMain(self):
      self.panes = self.createcomponent('panes', (), None,
                           Pmw.PanedWidget,
                           (self.interior(),),
                           orient='horizontal')
```

```
        self.panes.add('browserpane', min=150, size=160)
        self.panes.add('displaypane', min=.1)

        f = os.path.join(path, 'folder.gif')
        of = os.path.join(path, 'openfolder.gif')
        self.browser = self.createcomponent('browse', (), None,
                                Tree,
                                (self.panes.pane('browserpane'),),
                                icon=f,
                                openicon=of,
                                treename='Multimedia',
                                action=None)
        self.browser.pack(side=TOP, expand=YES, fill=Y)

        self.datasite = self.createcomponent('datasite', (), None,
                                Frame,
                                (self.panes.pane('displaypane'),))

        self.datasite.pack(side=TOP, expand=YES, fill=BOTH)

        f  = os.path.join(path, 'folder.gif')
        of = os.path.join(path, 'openfolder.gif')
        gf = os.path.join(path, 'gif.gif')
        jf = os.path.join(path, 'jpg.gif')
        xf = os.path.join(path, 'other.gif')

        self.browser.inhibitDraw = 1

        top=self.browser.child[0]
        top.state='expanded'
        jpeg=top.addChild(self.browser, icon=f, openicon=of,
                    name='Jpeg', action=None)
        gif=top.addChild(self.browser, icon=f, openicon=of,
                    name='GIF', action=None)
        other=top.addChild(self.browser, icon=f, openicon=of,
                    name='Other', action=None)

        imageDir = {'.jpg': (jpeg, jf), '.jpeg': (jpeg, jf),
                    '.gif': (gif, gf),  '.bmp':  (other, xf),
                    '.ppm': (other, xf)}

        files = os.listdir(imgs)
        for file in files:
            r, ext = os.path.splitext(file)
            cont, icon = imageDir.get(ext, (None, None))
            if cont:
                cont.addChild(self.browser, icon=icon,
                              name=file, action=self.showMe)
        self.browser.inhibitDraw = 0
        self.browser.display()
```

```
        self.panes.pack(side=TOP, expand=YES, fill=BOTH)

    def createImageDisplay(self):
        self.imageDisplay = self.createcomponent('image', (), None,
                                  Label,
                                  (self.datasite,))
        self.browser.imageLabel = self.imageDisplay
        self.browser.imageData  = None
        self.imageDisplay.place(relx=0.5, rely=0.5, anchor=CENTER)

    def createInterface(self):
        AppShell.AppShell.createInterface(self)
        self.createButtons()
        self.createMain()
        self.createImageDisplay()

    def showMe(self, dofile):
        if self.browser.imageData: del self.browser.imageData
        self.browser.imageData = ImageTk.PhotoImage(\
                        Image.open('%s%s' % \
                             (imgs, dofile)))
        self.browser.imageLabel['image'] = self.browser.imageData

if __name__ == '__main__':
    imageBrowser = ImageBrowser()
    imageBrowser.run()
```

代码注解（续）

● 定义所有要用到的图标。

```
f  = os.path.join(path, 'folder.gif')
of = os.path.join(path, 'openfolder.gif')
gf = os.path.join(path, 'gif.gif')
jf = os.path.join(path, 'jpg.gif')
xf = os.path.join(path, 'other.gif')
```

● 根节点已创立，加上相应的图形文件。

```
top=self.browser.child[0]
top.state='expanded'
jpeg=top.addChild(self.browser, icon=f, openicon=of,
           name='Jpeg', action=None)
...
```

● 创建一个数据字典保存文件和对应图标及文件格式的转换（数据字典是一种定义有多个处理过程的对象的有效形式）。

```
imageDir = {'.jpg': (jpeg, jf), '.jpeg': (jpeg, jf),
          '.gif': (gif, gf),  '.bmp':  (other, xf),
          '.ppm': (other, xf)}
```

● 搜索整个磁盘，找出符合要求的目录加入树图。

```
files = os.listdir(imgs)
```

```
for file in files:
    r, ext = os.path.splitext(file)
    cont, icon = imageDir.get(ext, (None, None))
    if cont:
        cont.addChild(self.browser, icon=icon,
                      name=file, action=self.showMe)
```

这些代码有一点复杂，且有两个潜在性的问题，请读者自己找出。

● 树已建好，重置 inhibitDraw 以显示树图。

```
self.browser.inhibitDraw = 0
self.browser.display()
```

看起来好像代码很多，但结果却很简洁。另外，用户也容易理解，所以这些程序用途十分广泛。

运行 Example_8_10.py 结果见图 8.11。

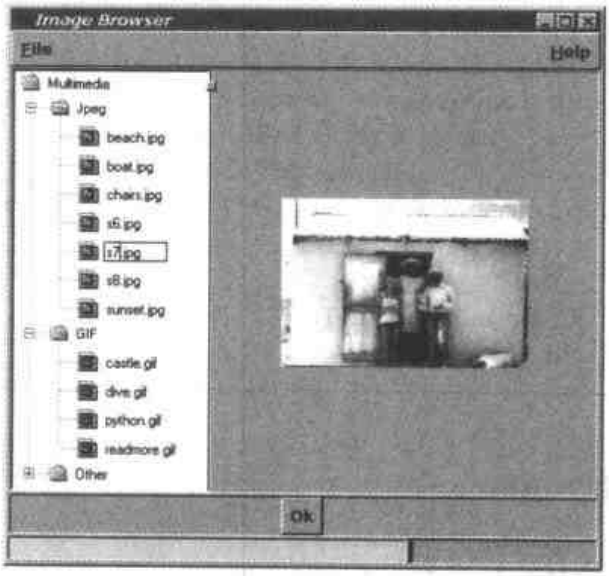


图 8.11 图像浏览

8.6 压缩程序

Windows 95/98/NT 的用户想必对压缩程序非常熟悉，因为压缩程序常常伴随着安装程序和结构程序出现。压缩程序指导用户一步步地进行操作，可以前进也可以后退。在很多情况下它的结构类似于活页夹，只是随机存取的顺序相反。

下面的例子是一个软件安装的压缩程序。WizardShell.py 从 Appshell.py 中生成，但它并没有继承 Appshell.py 的所有属性，其绝大部分的代码与之相似，在此不再赘述，完整的程序在线提供。

WizardShell.py

```
from Tkinter import *
import Pmw
```

```
import sys, string

class WizardShell(Pmw.MegaWidget):
    wizversion = '1.0'
    wizname    = 'Generic Wizard Frame'
    wizimage = 'wizard.gif'

    panes          = 4                                    ❶

    def __init__(self, **kw):
        optiondefs = (
            ('framewidth',     1,          Pmw.INITOPT),
            ('frameheight',    1,          Pmw.INITOPT))
        self.defineoptions(kw, optiondefs)

        # setup panes                                     ❷
        self.pCurrent = 0
        self.pFrame = [None] * self.panes

    def wizardInit(self):
        # Called before interface is created (should be overridden).
        pass

    def __createWizardArea(self):                         ❸
        self.__wizardArea = self.createcomponent('wizard',(), None,
                                Frame,(self._hull,),
                                relief=FLAT, bd=1)
        self.__illustration = self.createcomponent('illust',(), None,
                                 Label,(self.__wizardArea,))
        self.__illustration.pack(side=LEFT, expand=NO, padx=20)
        self.__wizimage = PhotoImage(file=self.wizimage)
        self.__illustration['image'] = self.__wizimage

        self.__dataArea = self.createcomponent('dataarea',(), None,
                                Frame,(self.__wizardArea,),
                                relief=FLAT, bd=1)

        self.__dataArea.pack(side=LEFT, fill = 'both', expand = YES)
        self.__wizardArea.pack(side=TOP, fill=BOTH, expand=YES)

    def __createSeparator(self):
        self.__separator = self.createcomponent('separator',(), None,
                               Frame,(self._hull,),
                               relief=SUNKEN,
                               bd=2, height=2)
        self.__separator.pack(fill=X, expand=YES)

    def __createCommandArea(self):
        self.__commandFrame = self.createcomponent('commandframe',(), None,
                               Frame,(self._hull,),
```

```
                                    relief=FLAT, bd=1)
        self.__commandFrame.pack(side=TOP, expand=NO, fill=X)

    def interior(self):
        return self.__dataArea

    def changePicture(self, gif):
        self.__wizimage = PhotoImage(file=gif)
        self.__illustration['image'] = self.__wizimage

    def buttonAdd(self, buttonName, command=None, state=1):
        frame = Frame(self.__commandFrame)
        newBtn = Button(frame, text=buttonName, command=command)
        newBtn.pack()
        newBtn['state'] = [DISABLED,NORMAL][state]
        frame.pack(side=RIGHT, ipadx=5, ipady=5)
        return newBtn

    def __createPanes(self):
        for i in range(self.panes):
            self.pFrame[i] = self.createcomponent('pframe',(), None,
                                    Frame,(self.interior(),),
                                    relief=FLAT, bd=1)
            if not i == self.pCurrent:
                self.pFrame[i].forget()
            else:
                self.pFrame[i].pack(fill=BOTH, expand=YES)

    def pInterior(self, idx):
        return self.pFrame[idx]

    def next(self):
        cpane = self.pCurrent
        self.pCurrent = self.pCurrent + 1
        self.prevB['state'] = NORMAL
        if self.pCurrent == self.panes - 1:
            self.nextB['text']    = 'Finish'
            self.nextB['command'] = self.done
        self.pFrame[cpane].forget()
        self.pFrame[self.pCurrent].pack(fill=BOTH, expand=YES)

    def prev(self):
        cpane = self.pCurrent
        self.pCurrent = self.pCurrent - 1
        if self.pCurrent <= 0:
            self.pCurrent = 0
            self.prevB['state'] = DISABLED
        if cpane == self.panes - 1:
```

```
            self.nextB['text']    = 'Next'
            self.nextB['command'] = self.next
        self.pFrame[cpane].forget()
        self.pFrame[self.pCurrent].pack(fill=BOTH, expand=YES)

    def done(self):
        #to be Overridden
        pass

    def __createInterface(self):
        self.__createWizardArea()
        self.__createSeparator()
        self.__createCommandArea()
        self.__createPanes()
        self.busyWidgets = ( self.root, )
        self.createInterface()

class TestWizardShell(WizardShell):
    def createButtons(self):
        self.buttonAdd('Cancel',    command=self.quit)          ❶
        self.nextB = self.buttonAdd('Next', command=self.next)
        self.prevB = self.buttonAdd('Prev', command=self.prev, state=0)

    def createMain(self):
        self.w1 = self.createcomponent('w1', (), None,
                              Label,(self.pInterior(0),),
                              text='Wizard Area 1')
        self.w1.pack()
        self.w2 = self.createcomponent('w2', (), None,
                              Label,(self.pInterior(1),),
                              text='Wizard Area 2')
        self.w2.pack()

    def createInterface(self):
        WizardShell.createInterface(self)
        self.createButtons()
        self.createMain()

    def done(self):              ❷
        print 'All Done'

if __name__ == '__main__':
    test = TestWizardShell()
    test.run()
```

WizardShell.py 注解

❶ WizardShell 使用了 Appshell 类，并加上了 panes（窗格）来定义分离的操作步骤

数。

```
Class WizardShell(Pmw.MegaWidget):
     panes     = 4
```

❹ 为每个步骤初始化一个空的 pane。

```
self.pCurrent = 0
self.pFrame = [None] * self.panes
```

❺ 创建主要的 WizardArea。加入 Separator 和 CommandArea 方法。

```
def __createWizardArea(self):
    ...
def __createSeparator(self):
    ...
def __createCommandArea(self):
    ...
```

❻ Buttonadd 比 Appshell 更全面，因为必须在每一个步骤的操作中定义“向前”、“向后”可用或者不可用。

```
def buttonAdd(self, buttonName, command=None, state=1):
    frame = Frame(self.__commandFrame)
    newBtn = Button(frame, text=buttonName, command=command)
    newBtn.pack()
    newBtn['state'] = [DISABLED,NORMAL][state]
    frame.pack(side=RIGHT, ipadx=5, ipady=5)
    return newBtn
```

❼ 现在创建每一步骤的 pane，封装当前窗口，隐藏其他。

```
def __createPanes(self):
    for i in range(self.panes):
        self.pFrame[i] = self.createcomponent('pframe',(), None,
                                  Frame,(self.interior(),),
                                  relief=FLAT, bd=1)
        if not i == self.pCurrent:
            self.pFrame[i].forget()
        else:
            self.pFrame[i].pack(fill=BOTH, expand=YES)
```

❽ 与定义 interior 相似，定义一个 pInterior 来为每一个单独的 pane 提供入口。

```
def pInterior(self, idx):
     return self.pFrame[idx]
```

❾ Next（向后）和 prev（向前）隐藏当前 pane 进入下一个，同时在必要和适当的地方改变按钮和标签。

```
def next(self):
    cpane = self.pCurrent
    self.pCurrent = self.pCurrent + 1
```

```
        self.prevB['state'] = NORMAL
        if self.pCurrent == self.panes - 1:
            self.nextB['text']    = 'Finish'
            self.nextB['command'] = self.done
        self.pFrame[cpane].forget()
        self.pFrame[self.pCurrent].pack(fill=BOTH, expand=YES)
```

❻ 与 AppShell 不同，我们必须将参照标绑定在控制键上，以便操作。

```
class TestWizardShell(WizardShell):
    def createButtons(self):
        self.buttonAdd('Cancel',    command=self.quit)
        self.nextB = self.buttonAdd('Next', command=self.next)
        self.prevB = self.buttonAdd('Prev', command=self.prev, state=0)
```

❼ 定义“done”方法显示“完成”。

```
def done(self):
    print 'All Done'
```

运行结果见图 8.12。下面介绍一个具体的例子。

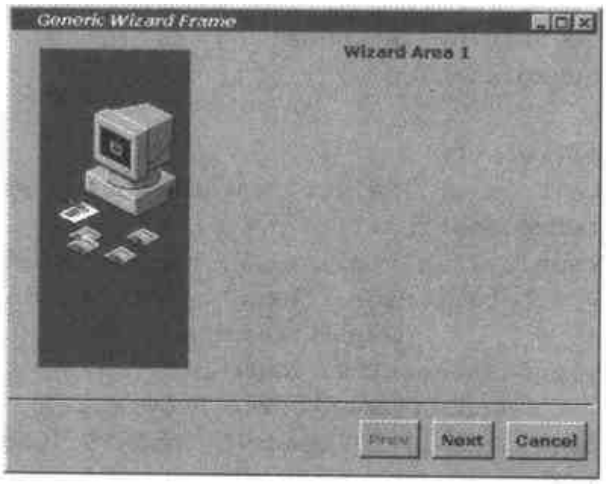


图 8.12　向导

Example 8_11.py

```
from Tkinter import *
import Pmw
import sys, string
import WizardShell

class Installer(WizardShell.WizardShell):
    wizname = 'Install Widgets'
    panes = 4

    def createButtons(self):
        self.buttonAdd('Cancel',    command=self.quit, state=1)
```

```
        self.nextB = self.buttonAdd('Next', command=self.next, state=1)
        self.prevB = self.buttonAdd('Prev', command=self.prev, state=0)

    def createTitle(self, idx, title):                                    ❶
        label = self.createcomponent('l%d' % idx, (), None,
                             Label,(self.pInterior(idx),),
                             text=title,
                             font=('verdana', 18, 'bold', 'italic'))
        label.pack()
        return label

    def createExplanation(self, idx):                                     ❷
        text = self.createcomponent('t%d' % idx, (), None,
                            Text,(self.pInterior(idx),),
                            bd=0, wrap=WORD, height=6)
        fd = open('install%d.txt' % (idx+1))
        text.insert(END, fd.read())
        fd.close()
        text.pack(pady=15)

    def createPanelOne(self):
        self.createTitle(0, 'Welcome!')
        self.createExplanation(0)

    def createPanelTwo(self):                                             ❸
        self.createTitle(1, 'Select Destination\nDirectory')
        self.createExplanation(1)
        frame = Frame(self.pInterior(1), bd=2, relief=GROOVE)
        self.entry = Label(frame, text='C:\\Widgets\\WidgetStorage',
                           font=('Verdana', 10))
        self.entry.pack(side=LEFT, padx=10)
        self.btn   = Button(frame, text='Browse...')
        self.btn.pack(side=LEFT, ipadx=5, padx=5, pady=5)
        frame.pack()

    def createPanelThree(self):
        self.createTitle(2, 'Select Components')
        self.createExplanation(2)
        frame = Frame(self.pInterior(2), bd=0)
        idx = 0
        for label, size in [('Monkey','526k'),('Aardvark','356k'),
                    ('Warthog','625k'),
                    ('Reticulated Python','432k')]:
            ck  = Checkbutton(frame).grid(row=idx, column=0)
            lbl = Label(frame, text=label).grid(row=idx, column=1,
                                    columnspan=4, sticky=W)
            siz = Label(frame, text=size).grid(row=idx, column=5)
            idx = idx + 1
        frame.pack()
```

```
    def createPanelFour(self):
        self.createTitle(3, 'Finish Installation')
        self.createExplanation(3)

    def createInterface(self):
        WizardShell.WizardShell.createInterface(self)
        self.createButtons()
        self.createPanelOne()
        self.createPanelTwo()
        self.createPanelThree()
        self.createPanelFour()

    def done(self):                    ❶
        print 'This is where the work starts!'

if __name__ == '__main__':
    install = Installer()
    install.run()
```

Example_8_11.py 注解

❶ 从定义一些完成普通功能的规则开始，每一个 pane 都有标题。

```
def createTitle(self, idx, title):
    label = self.createcomponent('l%d' % idx, (), None,
                        Label,(self.pInterior(idx),),
                        text=title,
                        font=('verdana', 18, 'bold', 'italic'))
    label.pack()
    return label
```

❷ 压缩程序必须向用户提供简洁清晰的说明，下面的规程规定使用 Tkinter Text 控件，其中文本从文件中读取。

```
def createExplanation(self, idx):
    text = self.createcomponent('t%d' % idx, (), None,
                        Text,(self.pInterior(idx),),
                        bd=0, wrap=WORD, height=6)
    fd = open('install%d.txt' % (idx+1))
    text.insert(END, fd.read())
    fd.close()
    text.pack(pady=15)
```

❸ 每一个 pane 互相独立。

```
def createPanelTwo(self):
    self.createTitle(1, 'Select Destination\nDirectory')
    self.createExplanation(1)
    frame = Frame(self.pInterior(1), bd=2, relief=GROOVE)
```

```
        self.entry = Label(frame, text='C:\\Widgets\\WidgetStorage',
                           font=('Verdana', 10))
        self.entry.pack(side=LEFT, padx=10)
        self.btn   = Button(frame, text='Browse...')
        self.btn.pack(side=LEFT, ipadx=5, padx=5, pady=5)
        frame.pack()
```

❶ 本例依然十分简练，但读者应该有一些体会了吧！

图 8.13 为本例结果——一个安装程序。

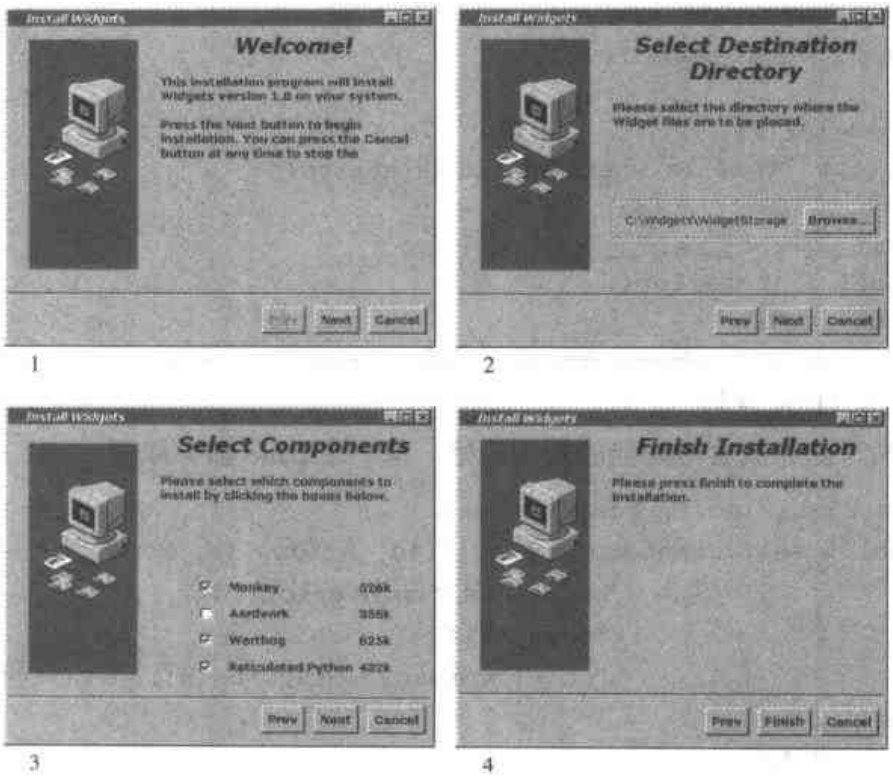


图 8.13　安装向导

8.7　图像映射

本章最后讨论一种在网页中常出现的输入技术——图像映射。它可以将图形变为可以单击的区域，这一点还将在第 9 章的“面板和机器”中再讨论，但这里依然要提出来，因为它是一种用户输入的有效方式。

回顾第 3 章的“创建实例”，你想起怎样用按钮控件绑定用户的输入信息和计算函数来创建一个计算器，这个可以用图像映射来实现。主要的偏重点是提高计算器的可使用性。

创建图片的问题之一是没有定义对象位置的工具，而度量的办法又太费时。看看图 8.14，每一个按钮的周围都有了一道灰色的边框。放大的图形显示得更清楚，对于每一个对象，必须定义其左上角和右下角的坐标，二者共同决定了按钮的位置。

下面的例子给出了怎样用简单的工具创建一系列的矩形图片映射，以及用一个小程

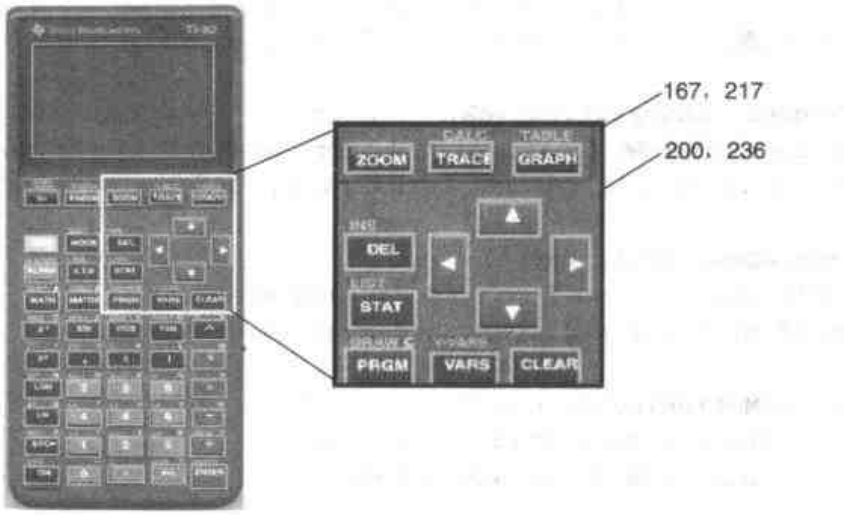


图 8.14　图像的坐标系统

序来测试映射效果。这个例子本身只支持矩形图标，读者可以自己改写支持其他图标的程序。

Example 8_12.py

```
from Tkinter import *
import sys, string
class MakeImageMap:

  def __init__(self, master, file=None):
      self.root = master
      self.root.title("Create Image Map")
      self.rubberbandBox = None
      self.coordinatedata = []
      self.file = file

      self.img = PhotoImage(file=file)
      self.width = self.img.width()
      self.height = self.img.height()

      self.canvas = Canvas(self.root, width=self.width,
                           height=self.height)
      self.canvas.pack(side=TOP, fill=BOTH, expand=no)
      self.canvas.create_image(0,0,anchor=NW, image=self.img)
      self.frame1 = Frame(self.root, bd=2, relief=RAISED)
      self.frame1.pack(fill=X)
      self.reference = Entry(self.frame1, width=12)
      self.reference.pack(side=LEFT, fill=X, expand=1)
      self.add = Button(self.frame1, text='Add', command=self.addMap)
      self.add.pack(side=RIGHT, fill=NONE, expand=0)
      self.frame2 = Frame(self.root, bd=2, relief=RAISED)
      self.frame2.pack(fill=X)
      self.done = Button(self.frame2, text='Build ImageMap',
```

```
                    command=self.buildMap)
        self.done.pack(side=TOP, fill=NONE, expand=0)

        Widget.bind(self.canvas, "<Button-1>", self.mouseDown)
        Widget.bind(self.canvas, "<Button1-Motion>", self.mouseMotion)
        Widget.bind(self.canvas, "<Button1-ButtonRelease>", self.mouseUp)

    def mouseDown(self, event):
        self.startx = self.canvas.canvasx(event.x)
        self.starty = self.canvas.canvasy(event.y)

    def mouseMotion(self, event):
        x = self.canvas.canvasx(event.x)
        y = self.canvas.canvasy(event.y)

        if (self.startx != event.x) and (self.starty != event.y):
            self.canvas.delete(self.rubberbandBox)
            self.rubberbandBox = self.canvas.create_rectangle(
                 self.startx, self.starty, x, y, outline='white',width=2)
            self.root.update_idletasks()

    def mouseUp(self, event):
        self.endx = self.canvas.canvasx(event.x)
        self.endy = self.canvas.canvasy(event.y)
        self.reference.focus_set()
        self.reference.selection_range(0, END)

    def addMap(self):
        self.coordinatedata.append(self.reference.get(),
                           self.startx, self.starty,
                           self.endx, self.endy)

    def buildMap(self):
        filename = os.path.splitext(self.file)[0]
        ofd = open('%s.py' % filename, 'w')
        ifd = open('image1.inp')
        lines = ifd.read()
        ifd.close()
        ofd.write(lines)

        for ref, sx,sy, ex,ey in self.coordinatedata:
             ofd.write("    self.iMap.addRegion(((%5.1f,%5.1f),"
                   "(%5.1f,%5.1f)),'%s')\n" % (sx,sy, ex,ey, ref))

        ofd.write('\n%s\n' % ('#'*70))
        ofd.write('if __name__ == "__main__":\n')
        ofd.write('    root = Tk()\n')
        ofd.write('    root.title("%s")\n' % self.file)
        ofd.write('    imageTest = ImageTest(root, width=%d, height=%d,'
               'file="%s")\n' % (self.width, self.height, self.file))
```

```
        ofd.write('    imageTest.root.mainloop()\n')
        ofd.close()
        self.root.quit()

if __name__ == '__main__':
    file = sys.argv[1]
    root = Tk()
    makeImageMap = MakeImageMap(root, file=file)
    makeImageMap.root.mainloop()
```

Example_8_12.py 注解

❶ 第一步是确定图标的大小。由于我们是把图标显示在面板上，而面板的大小并不能随着图标而改变，所以要先确定图标的大小，再根据它确定面板的大小。

```
self.img = PhotoImage(file=file)
self.width = self.img.width()
self.height = self.img.height()
```

❷ 使用工具将选中的矩形图片放在要放它的地方。将相应的函数绑定在鼠标的 press（点击）和 releace（松开）以及 motion（移动）方法上。

```
Widget.bind(self.canvas, "<Button-1>", self.mouseDown)
Widget.bind(self.canvas, "<Button1-Motion>", self.mouseMotion)
Widget.bind(self.canvas, "<Button1-ButtonRelease>", self.mouseUp)
```

❸ mouseDown 将鼠标点击处的 x 和 y 坐标转换成与面板上的图片相对应的值。

```
def mouseDown(self, event):
    self.startx = self.canvas.canvasx(event.x)
    self.starty = self.canvas.canvasy(event.y)
```

❹ mouseMotion 使矩形图标的尺寸不断变化到当前的值。

```
def mouseMotion(self, event):
    x = self.canvas.canvasx(event.x)
    y = self.canvas.canvasy(event.y)

    if (self.startx != event.x) and (self.starty != event.y):
        self.canvas.delete(self.rubberbandBox)
        self.rubberbandBox = self.canvas.create_rectangle(
            self.startx, self.starty, x, y, outline='white',width=2)
```

❺ 每一次变换矩形时都要调用 update_idletasks 以显示变化。每做一次拖动鼠标都会给系统带来如洪水般的处理流，所以要适当安排事件的处理。

```
self.root.update_idletasks()
```

❻ 当鼠标被松开时，将最终的位置转换成相应的值，然后将它与输入控件联系在一起。

```
def mouseUp(self, event):
    self.endx = self.canvas.canvasx(event.x)
    self.endy = self.canvas.canvasy(event.y)
    self.reference.focus_set()
    self.reference.selection_range(0, END)
```

❺ 一旦图片映射的 ID 设定，就单击 Add 按钮将 ID 加入映射相应的列表。

```
def addMap(self):
    self.coordinatedata.append(self.reference.get(),
                       self.startx, self.starty,
                       self.endx, self.endy)
```

❻ 当单击 Build 键时，让 Python 文件来监测图像映射。

```
def buildMap(self):
    filename = os.path.splitext(self.file)[0]
    ofd = open('%s.py' % filename, 'w')
```

❼ 程序的第一部分相当于一个样板文件，可以直接从文件中读取。

```
    ifd = open('image1.inp')
    lines = ifd.read()
    ifd.close()
    ofd.write(lines)
```

❽ 为选中的图片映射设置输入。

```
for ref, sx,sy, ex,ey in self.coordinatedata:
    ofd.write("    self.iMap.addRegion(((%5.1f,%5.1f),"
          "(%5.1f,%5.1f)),'%s')\n" % (sx,sy, ex,ey, ref))
```

❾ 最后，用代码实现图形映射。

```
    ofd.write('\n%s\n' % ('#'*70))
    ofd.write('if __name__ == "__main__":\n')
    ofd.write('    root = Tk()\n')
    ofd.write('    root.title("%s")\n' % self.file)
    ofd.write('    imageTest = ImageTest(root, width=%d, height=%d,'
          'file="%s")\n' % (self.width, self.height, self.file))
    ofd.write('    imageTest.root.mainloop()\n')
    ofd.close()
```

你所要做的一切就是调用矩形的 GIF 文件然后将它放到先前选择的位置上，为它设置名称单击 Add 键，完成命名之后单击 Build。

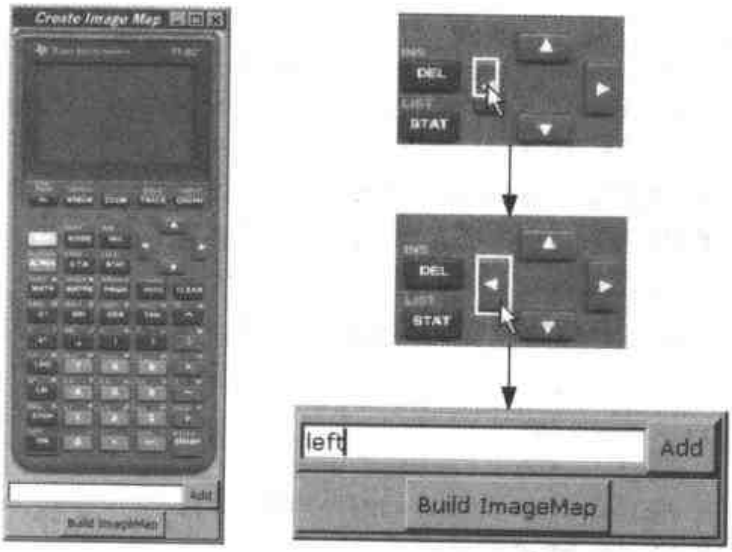


图 8.15 创建图像映射

注意　这个例子告诉我们怎样用 Python 文件来处理输入的数据。Python 用起来很简单而且可以生成很多复杂的系统。只要真正领会了代码的真谛，你也可以编出自己的程序。当然这只是用于很多重复工作的时候，它可以节省很多时间和精力。

快速浏览一下这些代码。

Calculator.py

```
from Tkinter import *
from imagemap import *
class ImageTest:

   def hit(self, event):
       self.infoVar.set(self.iMap.getRegion(event.x, event.y))

   def __init__(self, master, width=0, height=0, file=None):
       self.root = master
       self.root.option_add('*font', ('verdana', 12, 'bold'))
       self.iMap = ImageMap()

       self.canvas = Canvas(self.root, width=width, height=height)
       self.canvas.pack(side="top", fill=BOTH, expand='no')

       self.img = PhotoImage(file=file)
       self.canvas.create_image(0,0,anchor=NW, image=self.img)
       self.canvas.bind('<Button-1>', self.hit)
       self.infoVar = StringVar()
       self.info = Entry(self.root, textvariable=self.infoVar)
       self.info.pack(fill=X)

       self.iMap.addRegion((( 61.0,234.0),( 96.0,253.0)),'mode')
       self.iMap.addRegion(((104.0,234.0),(135.0,250.0)),'del')
       self.iMap.addRegion((( 19.0,263.0),( 55.0,281.0)),'alpha')
       self.iMap.addRegion((( 63.0,263.0),( 96.0,281.0)),'x-t-phi')
# ----- Some lines removed for brevity---------------
       self.iMap.addRegion((( 24.0,467.0),( 54.0,488.0)),'on')
       self.iMap.addRegion((( 64.0,468.0),( 97.0,486.0)),'0')
       self.iMap.addRegion(((104.0,469.0),(138.0,486.0)),'.')
       self.iMap.addRegion(((185.0,469.0),(220.0,491.0)),'enter')

if __name__ == "__main__":
   root = Tk()
   root.title("calculator.gif")
   imageTest = ImageTest(root, width=237, height=513,file="calculator.gif")
   imageTest.root.mainloop()
```

的确非常简单。图像映射用的 ImageMap 类可以扩展得适用于其他图形。

imagemap.py

```
class Region:                                          ●
   def __init__(self, coords, ref):
       self.coords = coords
       self.ref    = ref

   def inside(self, x, y):                             ●
       isSide = 0
       if self.coords[0][0] <= x <= self.coords[1][0] and \
          self.coords[0][1] <= y <= self.coords[1][1]:
            isSide = 1
       return isSide

class ImageMap:
   def __init__(self):
       self.regions = []
       self.cache   = {}

   def addRegion(self, coords, ref):
        self.regions.append(Region(coords, ref))

   def getRegion(self, x, y):                          ●
       try:
            return self.cache[(x,y)]
       except KeyError:
            for region in self.regions:                ●
                 if region.inside(x, y) == 1:
                      self.cache[(x,y)] = region
                      return region.ref
            return None
```

ImageMap.py 注解

● Region 类可容纳目标区域。

```
class Region:
   def __init__(self, coords, ref):
       self.coords = coords
       self.ref    = ref
```

● 简单测试点击动作在何时响应。

```
def inside(self, x, y):
    isSide = 0
    if self.coords[0][0] <= x <= self.coords[1][0] and \
       self.coords[0][1] <= y <= self.coords[1][1]:
         isSide = 1
    return isSide
```

● 当要查找一个区域时，先从前面查找得到的缓存中的堆栈中寻找。

```
def getRegion(self, x, y):
```

```
    try:
        return self.cache[(x,y)]
```

❽ 如果它不在堆栈中，再从头搜寻。找到后将它放入缓存。

```
except KeyError:
    for region in self.regions:
        if region.inside(x, y) == 1:
            self.cache[(x,y)] = region
            return region.ref
```

结果见图 8.16。



图 8.16　运行 calculator.py

8.8　小结

本章介绍了多种窗体和对话框，有简单的填空式对话框也有浏览器和压缩程序及图像映射技术。希望读者能够学以致用。

第9章　面板与机器

在这章中，Tkinter 将变得有趣起来（也许我会找到自己的兴趣所在）。网络管理应用已经为图像格式建立了一系列的标准。许多的硬件生产者都为硬件提供了相应的用于前台显示的软件。这些软件能显示发光二极管的当前状态：连接插头、电源电压值、以及所有可测量的指标。总的来说，这些器件遵守 SNMP，虽然其他的系统也存在着。

这种模型可以被推广到非机械形式的领域，甚至是数据库管理，也可以用这种模型建立生动的界面。如果你是一位想为类似的用户界面建立一个基准体系的应用开发人员，那么，本章中所举的例子对你将十分有用。

9.1　创建前端面板

让我们介绍一个假想的设备，其目的是要向管理者说明一个交换系统（也许是一个 ATM 交换机或路由器）的前端展示。该前端展示显示出界面，线路卡，处理机及其他组件的当前状态。为了达到预期的目的，我们先假设此设备服从简单网络管理协议，而用于检测机器代理商并接受或产生中断的代码与 GUI（图形用户界面）相对独立。

如果这不是一个假想的设备，那么你既要有设备又要有它的说明书。比如在这个例子中，我们几乎能想象任何器件。图 9.1 为这个设备的简图。设备上有 2 个电源，每个电源都有一个插座，一个开关和指示电源状态（开、关或故障）的发光二极管。有 9 个空的卡式插槽，用来插各种卡。还有一些没有多大意义的装饰物，比如通气屏（air-intake screens）、底盘螺丝等。卡槽里将装上交换卡、处理器卡、8 端口 10-BASE-T 以太网卡*、一个四端口光纤分布式数据接口+、一个双通道的 T3 访问卡†、4 个高速卡。我并不关心这些设备是用来做什么的，或谁会出钱来买它，但是把它想象出来是很有趣的。

每个卡都有发光二极管、插头和无源控件。听起来好像很多，但经过初步分析，你会感到事情并非像它看上去那么难。而且一旦基本的控件被创建出来，你会发现代码会被多次重复使用。

---

* 安装最为广泛的以太局域网使用的是普通的电话双绞线。当双绞线被用来组建以太网时，它就被称为 10BASE-T。10BASE-T 能支持以太网 10Mb/s 的传输速率。

+ FDDI 是一种局域网中数据在光纤上传输的标准。这种局域网可扩展至 200km(124mile)。

† 能支持 44.736Mb/s 的传输速率。而被网络服务提供商广泛使用。

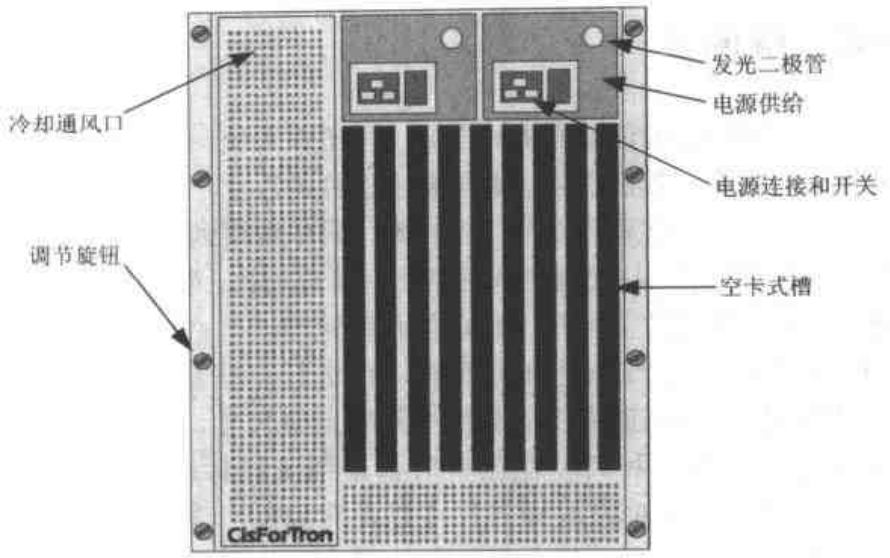


图 9.1　理想路由/开关框架

9.2　模块性

在 7.2 节中，我们开始建立了控件的类库，比如发光二极管、开关等。在这章中，我们要使用包括指示器、连接插头、面板等的扩展类库。我们还会用到复合小控件的内置状态的方法，在前面的例子中我们只给出了它的简单的注解，我们还将谈到在 GUI 中漫游的内容。因为我们的前端面板必须提供使管理者能够使用面板内的每个绘图元素功能特性的功能。一个好的例子就是，把用户与报警表单封装在一起，而报警表单是与一个可以显示的发光二极管或一个允许用户在选定点设置操作参数的配置屏结合在一起的。

如果你再看看图 9.1，可能会定义许多绘图控件，这些控件必须经进一步改进才能用于建立前端面板。虽然此时每个卡的外形没有被显示出来，但是，控件的一些潜在的要求已被显示在卡上构成了下列的列表：

1. 一个底座，它包括固定件，基本前端面板和被动的控件，如用于安装的螺钉等。

2. 卡槽，可用来安装各种卡。

3. 大量的卡，包括发光二极管、插头和其他的主动器件以及用来计算器件数量的卡的正面,还有一些被动器件如拔卡器和标签。

4. 电源模组，用来安装连接插头、开关和发光二极管。

5. 被动控件，如通气孔屏和商标。

6. 发光二极管，连接插头（J-45*,BNC+,FDDI†,J-25,J-50 和电源）以及电源开关。

---

* J 插头被典型地用于各种电路连接，插头标识的后缀表明有多少针可用于连接，常用的有 J-9,J-25 和 J-50。

+ BNC(Bayonet Neil-Concdlman)是一种用于连接同轴电缆的插头。

† FDDI 插头用于连接光纤或常用于连接一对电缆，一根用于接收，另一根用于传输。

9.3 实现前端面板

为了把一个纯理论上的前端面板转变为一个工作系统，还需要一些准备工作。尤其是，基于某种比例因素，我们必须计算屏幕上控件的尺寸，因为大多数面板都比典型的计算机屏幕大得多。正如读者可以从以下样本代码中观察到的那样，作者更倾向于在画布上确定控件的相对位置。与使用包或网格几何管理相比，确定小控件的位置要更困难。然而，精确的图形对象定位需要精确地定位几何管理。

作者用于实现面板的方法是画出面板，再进行一些基本的测量。在图 9.2 中，已画出了一些线来标记重建图形描述时所需的主要尺寸。在图上进行测量要比测量真正的设备简单。总宽度和高度在测量时都采用某种标准单位（如英寸、厘米），然后，计算出每个长方形对象的相对尺寸以及对象的一个角的相对偏移量。被选出的角应作为访问的锚点。这看起来很复杂，但实际上只消花几分钟就可以得到所需的信息。

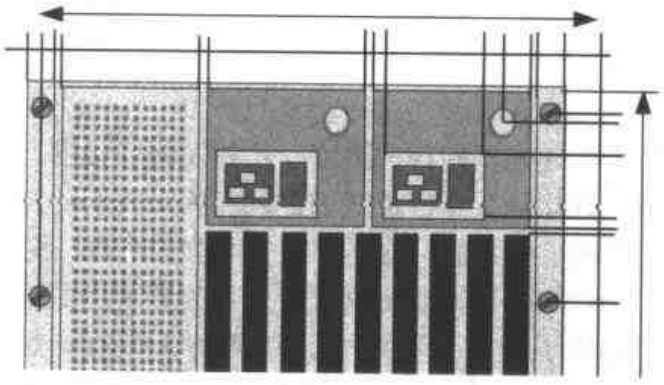


图 9.2 进行路由/开关框架测量

这个例子扩展了类库，使之可以提供许多新的绘图元素。在下面的列表中，已经描述过的元素就不再赘述了。

Components_1.py

```
from Tkinter   import *
from GUICommon import *
from Common    import *

class LED(GUICommon):
  def __init__(self, master, bg=Color.PANEL, height=1, width=1):
      self.screen_frame = Frame(master, width=width, height=height,
                bg=bg, bd=0)
      self.base = bg
      self.set_colors(self.screen_frame)
      radius = 4   # radius of an air hole
      ssize  = radius*3  # spacing between holes
```

❶ 建立一个实例

```
        rows = int(height/ssize)
        cols = int(width/ssize)

        self.canvas = Canvas(self.screen_frame, height=height, width=width,
                     bg=bg, bd=0, highlightthickness=0)

        self.canvas.pack(side=TOP, fill=BOTH, expand=NO)

        y = ssize - radius#                                  ❷ 优化性能
        for r in range(rows):
           x0 = ssize -radius
           for c in range(cols):
               x = x0 + (ssize*c)
               self.canvas.create_oval(x-radius, y-radius,
                              x+radius, y+radius,
                              fill=self.dbase,
                              outline=self.lbase)
           y = y + ssize

class PowerConnector:
   def __init__(self, master, bg=Color.PANEL):
        self.socket_frame = Frame(master, relief="raised", width=60,
                         height=40,  bg=bg, bd=4)

        inside=Frame(self.socket_frame, relief="sunken", width=56,
                   height=36, bg=Color.INSIDE, bd=2)
        inside.place(relx=.5, rely=.5, anchor=CENTER)

        ground=Frame(inside, relief="raised", width=6, height=10,
                  bg=Color.CHROME, bd=2)
        ground.place(relx=.5, rely=.3, anchor=CENTER)

        p1=Frame(inside, relief="raised", width=6, height=10,
              bg=Color.CHROME, bd=2)
        p1.place(relx=.25, rely=.7, anchor=CENTER)

        p2=Frame(inside, relief="raised", width=6, height=10,
              bg=Color.CHROME, bd=2)
        p2.place(relx=.75, rely=.7, anchor=CENTER)

class PowerSwitch(GUICommon):
   def __init__(self, master, label='I  0', base=Color.PANEL):
        self.base = base
        self.set_colors(master)                              ❶ 计算颜色

        self.switch_frame = Frame(master, relief="raised", width=45,
                         height=28, bg=self.vlbase, bd=4)
        switch = Frame(self.switch_frame, relief="sunken", width=32,
                   height=22, bg=self.base, bd=2)
        switch.place(relx=0.5, rely=0.5, anchor=CENTER)
```

```
        lbl=Label(switch, text=label, font=("Verdana", 10, "bold"),
                fg='white', bd=0, bg=self.dbase)
        lbl.place(relx=0.5, rely=0.5, anchor=CENTER)

class PowerSupply(GUICommon):
    def __init__(self, master, width=160, height=130, bg=Color.PANEL,
            status=STATUS_ON):
        self.base = bg
        self.set_colors(master)

        self.psu_frame = Frame(master, relief=SUNKEN, bg=self.dbase, bd=2,
                        width=width, height=height)

        Label(self.psu_frame, text='DC OK',fg='white',
             bg=self.dbase,font=('Verdana',10,'bold'),bd=0).place(relx=.8,
                        rely=.15, anchor=CENTER)

        self.led = LED(self.psu_frame,height=12,width=12,shape=ROUND,
                    bg=self.dbase)
        self.led.led_frame.place(relx=0.8, rely=0.31, anchor=CENTER)

        lsub = Frame(self.psu_frame, width=width/1.2, height=height/2,
                bg=self.dbase, bd=1, relief=GROOVE)
        lsub.place(relx=0.5, rely=0.68, anchor=CENTER)

        pwr=PowerConnector(lsub)
        pwr.socket_frame.place(relx=0.30, rely=0.5, anchor=CENTER)
        sw=PowerSwitch(lsub)
        sw.switch_frame.place(relx=0.75, rely=0.5, anchor=CENTER)

class Screw(GUICommon):
    def __init__(self, master, diameter=18, base="gray40", bg=Color.PANEL):
        self.base = base

        basesize = diameter+6
        self.screw_frame = Frame(master, relief="flat", bg=bg, bd=0,
                        highlightthickness=0)
        self.set_colors(self.screw_frame)

        canvas=Canvas(self.screw_frame, width=basesize, height=basesize,
                highlightthickness=0, bg=bg, bd=0)
        center = basesize/2
        r = diameter/2
        r2 = r - 4.0

        canvas.create_oval(center-r, center-r, center+r, center+r,
                    fill=self.base, outline=self.lbase)
        canvas.create_rectangle(center-r2, center-0.2,
                        center+r2, center+0.2,
```

```
                    fill=self.dbase, width=0)
        canvas.create_rectangle(center-0.2, center-r2,
                    center+0.2, center+r2,
                    fill=self.dbase, width=0)
        canvas.pack(side="top", fill='x', expand='no')

class CardBlank(GUICommon):
   def __init__(self, master=None, width=20, height=396,
            appearance="raised", bd=2, base=Color.CARD):
         self.base = base
         self.set_colors(master)
         self.card_frame=Frame(master, relief=appearance, height=height,
                     width=width, bg=base, bd=bd)

        top_pull = CardPuller(self.card_frame, CARD_TOP, width=width)
        top_pull.puller_frame.place(relx=.5, rely=0, anchor=N)

        bottom_pull = CardPuller(self.card_frame, CARD_BOTTOM, width=width)
        bottom_pull.puller_frame.place(relx=.5, rely=1.0, anchor=S)
```

代码注解

❶ 在先前的一些例子中，我们用过 Tkinter 的小控件实例的内部附注。所以，下列内容是可以实现的。

```
Button(parent, text='OK').pack(side=LEFT)
```

这个例子中的代码结构要求我们弄清楚对象的事例是唯一的。每一个控件必须参照它的子控件。

```
self.screen_frame = Frame(master, width=width, height=height,
                bg=bg, bd=0)
```

上面的代码在内部创建了 screen_frame 的特殊实例。

❷ 进气孔屏描述了使用者的轻松。通过它，重复的图画对象可被创建。它还强调了严谨的代码结构的重要性——它很容易使人忘记 Python 是一种交互式的语言。能明确地意识到结构化的代码能在某种程度上优化实施也是很重要的。

```
y = ssize - radius#
for r in range(rows):
     x0 = ssize -radius
     for c in range(cols):
         x = x0 + (ssize*c)
         self.canvas.create_oval(x-radius, y-radius,
                     x+radius, y+radius,
                     fill=self.dbase,
                     outline=self.lbase)
y = y + ssize
```

一些附加代码在这是很适宜的，因为第一个进气孔为高窄型，但矮的那个则相反。有着相反的一面，用外部循环来重复大量操作以减少一些内部循环的数学操作可以改善循环，当然这会增加代码的复杂程度，而且许多的操作是不必要的，但是，这种方

法还是值得考虑的。记住，一个好的 C 或 C++程序会为你优化循环，你是 Python 的优化者。

❸ GUICommon.set_colors 已经发展到可以通过一个器件在初始化的时候就能提供 access to winfo 的功能。

```
def __init__(self, master, label='I  0', base=Color.PANEL):
    self.base = base
    self.set_colors(master)
```

这样，主要的盛载器件和基本的颜色就被送入创建函数而被用于设置对象的颜色变化。

Components 1.py（续）

```
class CardPuller(GUICommon):
  def __init__(self, master, torb, width=20):
      self.base = master['background']                                    ❷
      self.set_colors(master)
      self.puller_frame=Frame(master, width=width, height=32,
                    bg=self.lbase, relief='flat')

      Frame(self.puller_frame, width=width/8, height=8,
            bg=self.dbase).place(relx=1.0, rely=[1.0,0][torb],            ❶
                                 anchor=[SE,NE][torb])

      Frame(self.puller_frame, width=width/3, height=24,
            bg=self.vdbase).place(relx=1.0, rely=[0,1.0][torb],
                                  anchor=[NE,SE][torb])

      Screw(self.puller_frame, diameter=10, base=self.base,
           bg=self.lbase).screw_frame.place(relx=0.3,rely=[0.2,0.8][torb],
                                 anchor=CENTER)

class Chassis:
  def __init__(self, master):
      self.outer=Frame(master, width=540, height=650,
                 borderwidth=2, bg=Color.PANEL)
      self.outer.forget()                                                 ❻

      self.inner=Frame(self.outer, width=490, height=650,
                 borderwidth=2, relief=RAISED, bg=Color.PANEL)
      self.inner.place(relx=0.5, rely=0.5, anchor=CENTER)

      self.rack = Frame(self.inner, bd=2, width=325, height=416,
                  bg=Color.CHASSIS)
      self.rack.place(relx=0.985, rely=0.853, anchor=SE)

      incr = 325/9
      x = 0.0
      for i in range(9):                                                  ❹ Creating blank cards
```

```
            card =CardBlank(self.rack, width=incr-1, height=414)
            card.card_frame.place(x=x, y=0, anchor=NW)
            x = x + incr

        self.img = PhotoImage(file='images/logo.gif')
        self.logo=Label(self.outer, image=self.img, bd=0)
        self.logo.place(relx=0.055, rely=0.992, anchor=SW)

        for x in [0.02, 0.98]:
            for y in [0.0444, 0.3111, 0.6555, 0.9711]:
                screw = Screw(self.outer, base="gray50")
                screw.screw_frame.place(relx=x, rely=y, anchor=CENTER)

        self.psu1 = PowerSupply(self.inner)
        self.psu1.psu_frame.place(relx=0.99, rely=0.004, anchor=NE)
        self.psu2 = PowerSupply(self.inner)
        self.psu2.psu_frame.place(relx=0.65, rely=0.004, anchor=NE)

        self.psu2.led.turnoff()                    ❽ Deactivating LED

        screen1 = Screen(self.inner, width=150, height=600, bg=Color.PANEL)
        screen1.screen_frame.place(relx=0.16, rely=0.475, anchor=CENTER)
        screen2 = Screen(self.inner, width=330, height=80, bg=Color.PANEL)
        screen2.screen_frame.place(relx=0.988, rely=0.989, anchor=SE)
```

代码注解

❹ 在 CardPuller 类中我们从父控件中获取了基本颜色，而非把它送入构建函数。

```
def __init__(self, master, torb, width=20):
    self.base = master['background']
    self.set_colors(master)
```

❺ 我们通过索引一个表来获得 y 坐标和锚点用于地点访问。这种很有价值的技术被用于全书的许多例子中。

```
bg=self.dbase).place(relx=1.0, rely=[1.0,0][torb],
                     anchor=[SE,NE][torb])
```

❻ 如果器件被创建在一个复杂的 GUI 中，那么，当窗口实现后，对其外观显示会有不好的影响。一种影响为在群集和网格的图形管理的作用下，器件被作为已创建的附加部分而被重调好几次；另一个影响是，由于控件结构的改变，系统多次重画控件，而使画出控件需要更长的时间。

```
self.outer.forget()
```

❼ 这个循环使得卡架装上一些空卡。

```
incr = 325/9
x = 0.0
for i in range(9):
    card =CardBlank(self.rack, width=incr-1, height=414)
    card.card_frame.place(x=x, y=0, anchor=NW)
```

```
        x = x + incr
```

❻ 最后，我们改变发光二极管的显示状态，以后在这个方面你会了解得更多。

```
self.psu2.led.turnoff()
```

因为前端面板会增加，为了便于描述，一个单独的模块——FrontPanel_py 被用来创建设备。

FrontPanel_1.py

```
#! /bin/env/python

from Tkinter  import *
from Components import *
from GUICommon import *
from Common    import *

class Router(Frame):
   def __init__(self, master=None):
       Frame.__init__(self, master)
       Pack.config(self)
       self.createChassis()

   def createChassis(self):
       self.chassis = Chassis(self)
       # Realize the outer frame(which
       # was forgotten when created)
       self.chassis.outer.pack(expand=0)          ❶ Realize the frame

if __name__ == '__main__':
   root = Router()
   root.master.title("CisForTron")
   root.master.iconname("CisForTron")
   root.mainloop()
```

代码注解

❶ 如果你检查了 Components_1.py 中各种类的每一个框架的_unit_方法，你会注意到并没有出现绘图管理访问。通过传递位置来放置对象或简单地将对象在构建函数封装是可能的，但是这儿使用的这种代码方式允许使用者对器件的几何图形有更强的控制能力。对底盘框架来说尤其是这样，这个器件被彻底遗忘了，这样在底盘实现之前屏幕就被刷新了。当有大量的图形对象需要画时，这种改进操作的优越性就更显著了。

```
self.chassis.outer.pack (expand=0)
```

这样，当实现控件和画出承载器件时，底盘框架就被封装了。这的确不同凡响！

当 FrontPanel.py 运行时，屏幕显示见图 9.3。图形绘出的速度很快，即使我们没有个别地创建每一个进气孔。对于那些用于描画纯被动控件的需高速计算、密集存储的图

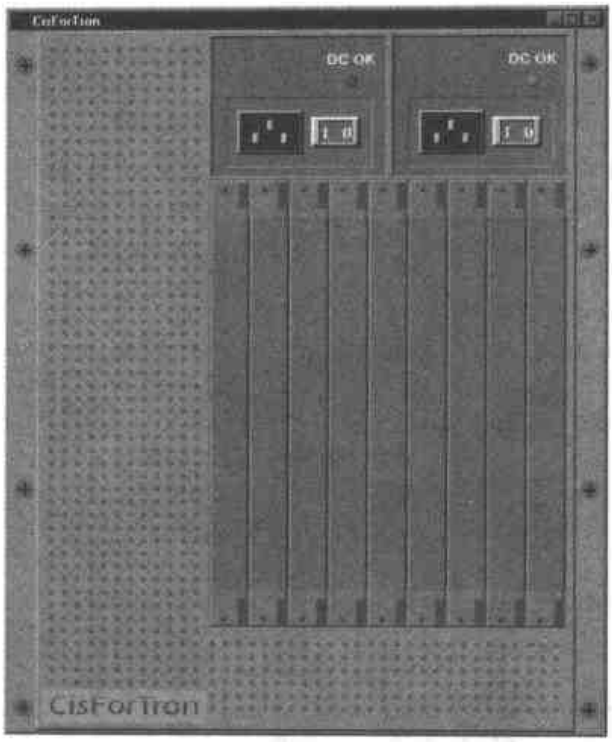


图 9.3　基本布线面板

形来说，很可能使用 GIF 或位图更好。有些内容将在 9.4 节的“GIF,BMP 和层叠技术”中进行讨论。注意我们是如何利用小控件的三维特性在显示中表现深度的。总的来说，最后避免总是试图模仿真实的设备，而要进行一定程度的抽象。

让我们创建一些卡用来填充底盘。T3 Access 卡有 4 个 BNC 插头（两对 RX/TX 插头），每对 BNC 插头都有一个电源（PWR）和指示出错的发光二极管（FLT）。

下面的代码是用于创建 BNC 插头的。

Components.py(fragment)

```
class BNC(GUICommon):                                              ❶
    def __init__(self, master, status=0, diameter=18,
                 port=-1, fid=''):
        self.base = master['background']
        self.hitID = fid
        self.status=status
        self.blink       = 0
        self.blinkrate   = 1
        self.on          = 0
        self.onState     = None
        self.Colors      = [None, Color.CHROME, Color.ON,
                            Color.WARN, Color.ALARM, '#00ffdd']

        basesize = diameter+6
        self.bnc_frame = Frame(master, relief="flat", bg=self.base,
```

```
                        bd=0, highlightthickness=0, takefocus=1)
        self.bnc_frame.pack(expand=0)
        self.bnc_frame.bind('<FocusIn>', self.focus_in)
        self.bnc_frame.bind('<FocusOut>', self.focus_out)

        self.canvas=Canvas(self.bnc_frame, width=basesize,
                    height=basesize, highlightthickness=0,
                    bg=self.base, bd=0)
        center = basesize/2
        r = diameter/2
        self.pins=self.canvas.create_rectangle(0, center+2, basesize-1,
                                    10, fill=Color.CHROME)
        self.bnc=self.canvas.create_oval(center-r, center-r,
                            center+r, center+r,
                            fill=Color.CHROME,
                            outline="black")
        r = r-3
        self.canvas.create_oval(center-r, center-r, center+r, center+r,
                    fill=Color.INSIDE, outline='black')
        r = r-2
        self.canvas.create_oval(center-r, center-r, center+r, center+r,
                    fill=Color.CHROME)
        r = r-3
        self.canvas.create_oval(center-r, center-r, center+r, center+r,
                    fill=Color.INSIDE, outline='black')

        self.canvas.pack(side=TOP, fill=X, expand=0)
        if self.hitID:
            self.hitID = '%s.%d' % (self.hitID, port)
            for widget in [self.bnc_frame]:
                widget.bind('<KeyPress-space>', self.panelMenu)
                widget.bind('<Button-1>', self.panelMenu)
            for widget in [self.canvas]:
                widget.bind('<1>', self.panelMenu)

    def focus_in(self, event):
        self.last_bg = self.canvas.itemcget(self.bnc, 'fill')
        self.canvas.itemconfig(self.bnc, fill=Color.HIGHLIGHT)
        self.update()

    def focus_out(self, event):
        self.canvas.itemconfig(self.bnc, fill=self.last_bg)
        self.update()

    def update(self):
        # First do the blink, if set to blink
        if self.blink:
            if self.on:
                if not self.onState:
```

```
                self.onState = self.status
                self.status  = STATUS_OFF
                self.on      = 0
            else:
                if self.onState:
                    self.status = self.onState   # Current ON color
                    self.on = 1
        # now update the status
        self.canvas.itemconfig(self.bnc, fill=self.Colors[self.status])
        self.canvas.itemconfig(self.pins,fill=self.Colors[self.status])
        self.bnc_frame.update_idletasks()
        if self.blink:
            self.bnc_frame.after(self.blinkrate * 1000, self.update)
```

代码注解

❶ 这个例子用 GUICommon 混合类来定义控件状态操作的基本方法。

```
Class BNC(GUICommon);
```

❷ 这里我们把 callbacks 与 FocusIn 及 Focusout 事件捆绑在一起。

```
self.bnc_frame.bind('<FocusIn>', self.focus_in)
self.bnc_frame.bind('<FocusOut>', self.focus_out)
```

上面的代码给控件加上了 focus_in 和 focus_out 的功能。这样，如果我们给控件加上标签或点击控件，那么控件将以高亮显示而且功能被激活。

❸ 所有的绘图对象（发光二极管、BNC、J、FDDI 插头）定义了特定的刷新方法，基于当前状态来改变控件的外观。我们需要用特殊化的方法来允许我们在合成控件中刷新特定区域的颜色。这种方法也能使控件以 1s 为间隔不停地闪烁。

```
def update(self):
    # First do the blink, if set to blink
    if self.blink:
        if self.on:
            if not self.onState:
                self.onState = self.status
                self.status  = STATUS_OFF
                self.on      = 0
            else:
                if self.onState:
                self.status = self.onState   # Current ON color
                    self.on = 1
    # now update the status
    self.canvas.itemconfig(self.bnc, fill=self.Colors[self.status])
    self.canvas.itemconfig(self.pins,fill=self.Colors[self.status])
    self.bnc_frame.update_idletasks()
    if self.blink:
        self.bnc_frame.after(self.blinkrate * 1000, self.update)
```

现在，我们通过定义 T3 Access 卡的布局来完成这个例子。

```
class StandardLEDs(GUICommon):                                    ❶
   def __init__(self, master=None, bg=Color.CARD):
       for led, label, xpos, ypos, state in [('flt', 'Flt', 0.3, 0.88, 1),
                                             ('pwr', 'Pwr', 0.7, 0.88, 2)]:
           setattr(self, led, LED(self.card_frame,shape=ROUND,width=8,
                status=state, bg=bg))
           getattr(self, led).led_frame.place(relx=xpos, rely=ypos,
                                              anchor=CENTER)
           Label(self.card_frame, text=label, font=("verdana", 4),
                 fg="white",bg=bg).place(relx=xpos,rely=(ypos+0.028),
                                         anchor=CENTER)

class T3AccessCard(CardBlank,StandardLEDs):                       ❷
   def __init__(self, master, width=1, height=1):
       CardBlank.__init__(self, master=master, width=width, height=height)
       bg=master['background']
       StandardLEDs.__init__(self, master=master, bg=bg)
       for port, lbl, tag, ypos in [(1,'RX1','T3AccessRX', 0.30),
                                    (2,'TX1','T3AccessTX', 0.40),
                                    (3,'RX2','T3AccessRX', 0.65),
                                    (4,'TX2','T3AccessRX', 0.75)]:

           setattr(self, 'bnc%d' % port, BNC(self.card_frame,
                                             fid=tag,port=port))
           getattr(self, 'bnc%d' % port).bnc_frame.place(relx=0.5,
                                             rely=ypos,anchor=CENTER)
           Label(self.card_frame,text=lbl,
                 font=("verdana", 6), fg="white",
                 bg=bg).place(relx=0.5,rely=(ypos+0.045),anchor=CENTER)

       for led, lbl, xpos, ypos, state in [('rxc','RXC',0.3,0.18,2),
                                           ('oos','OOS',0.7,0.18,1),
                                           ('flt','FLT',0.3,0.23,1),
                                           ('syn','SYN',0.7,0.23,2),
                                           ('rxc','RXC',0.3,0.53,2),
                                           ('oos','OOS',0.7,0.53,1),
                                           ('flt','FLT',0.3,0.58,1),
                                           ('syn','SYN',0.7,0.58,2)]:
           setattr(self, led, LED(self.card_frame,shape=ROUND,width=8,
                                  status=state, bg=bg))
           getattr(self, led).led_frame.place(relx=xpos,rely=ypos,
                                              anchor=CENTER)
           Label(self.card_frame,text=lbl,
                 font=("verdana", 4), fg="white",
                 bg=bg).place(relx=xpos,rely=(ypos+0.028),anchor=CENTER)
```

❸

❹

代码注解

❶ 我们增加一个类来画每个卡上的发光二极管。

```
Class standardLEDs(GUICommon)
```

❻ T3 Access 卡是从已经明确构建的 CardBlank 和 StandardLEDs 类中继承的。

```
class T3AccessCard(CardBlank,StandardLEDs):
  def __init__(self, master, width=1, height=1):
      CardBlank.__init__(self, master=master, width=width,
                  height=height) bg=master['background']
      StandardLEDs.__init__(self, master=master, bg=bg)
```

❼ 要是你仔细观察过作者的编程风格，那么你一定会注意到，作者喜欢从元组表重创建对象。这个例子也不例外：

```
for port, lbl, tag, ypos in [(1,'RX1','T3AccessRX', 0.30),
                             (2,'TX1','T3AccessTX', 0.40),
                             (3,'RX2','T3AccessRX', 0.65),
                             (4,'TX2','T3AccessRX', 0.75)]:
```

Python 从元组表中解封装一个元组的功能提供了一种压缩代码的机制，这些代码是用于实现意想中的效果的。

❽ 从元组中获得数据放入 setattr 和 getattr 调用中。

```
setattr(self, 'bnc%d' % port, BNC(self.card_frame,
fid=tag,port=port))
getattr(self, 'bnc%d' % port).bnc_frame.place(relx=0.5,
rely=ypos,anchor=CENTER)
Label(self.card_frame,text=lbl,
font=("verdana", 6), fg="white",
bg=bg).place(relx=0.5,rely=(ypos+0.045),anchor=CENTER)
```

这种编程风格导致代码比较紧凑。最初读起来可能会有点难，但这依然不失为在循环中创建绘图元素的一个有效途径。

```
for i in range(9):
    if i == 4:
        card =T3AccessCard(self.rack, width=incr-1, height=414)
    else:
        card =CardBlank(self.rack, width=incr-1, height=414)
    card.card_frame.place(x=x, y=0, anchor=NW)
    x = x + incr
```

当最后一步要加入 T3 卡时，我必须修改产生空白卡的循环来加入一个 T3 Access 卡。

运行 FrontPanel2，屏幕显示如图 9.4。下一步可能会有点惊人。创建附加绘图元素并把它们置入卡内并不需要很多代码。这些代码我们不在这里列出了，它们可以在网上获得。如果你运行 FrontPanel_3.py，你会看见如图 9.5 的屏幕。

再解释一下前面给出的代码：我们给小控件加上菜单，进入菜单要通过键盘，按空格键或用鼠标点击。

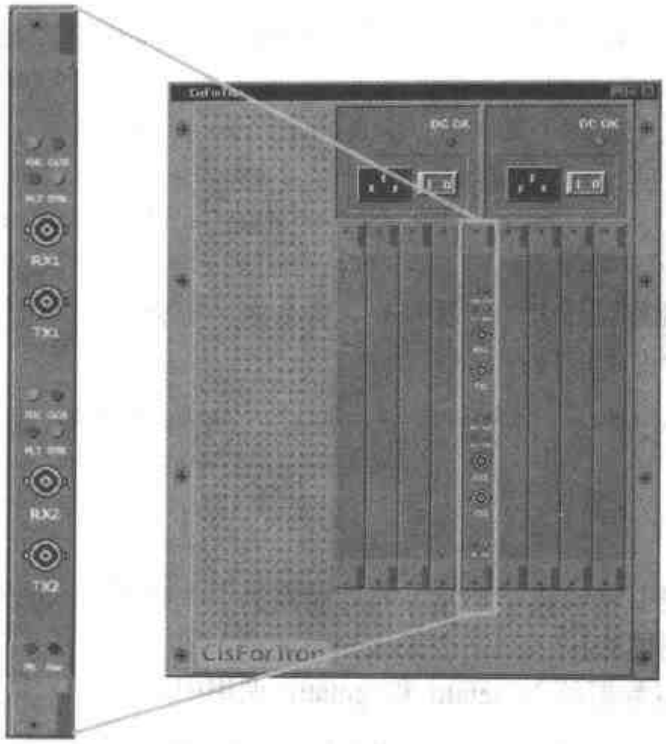


图 9.4　T3 access 卡

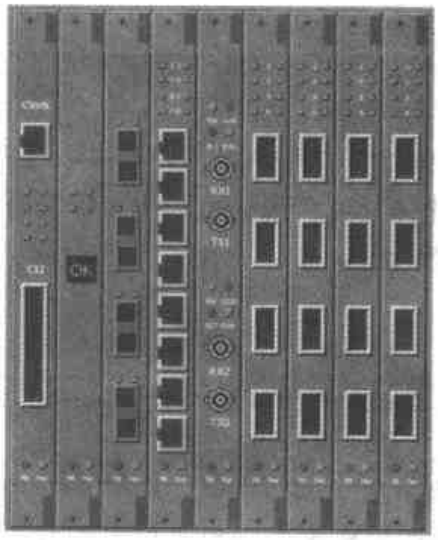


图 9.5　组装好面板

```
if self.hitID:
    self.hitID = '%s.%d' % (self.hitID, port)
    for widget in [self.hnc_frame]:
        widget.bind('<KeyPress-space>', self.panelMenu)
        widget.bind('<Button-1>', self.panelMenu)
    for widget in [self.canvas]:
        widget.bind('<1>', self.panelMenu)
```

```
We define the focus_in and focus_out methods.

def focus_in(self, event):
     self.last_bg= self.canvas.itemcget(self.bnc, 'fill')
     self.canvas.itemconfig(self.bnc, fill=Color.HIGHLIGHT)
     self.update()

def focus_out(self, event):
     self.canvas.itemconfig(self.bnc, fill=self.last_bg)
     self.update()
```

我们定义了 Focus_in 和 Focus_out 方法。

这些方法的目的是要当我们用鼠标点击或用快捷键选择控件时改变控件的高亮色彩。当我们从一个控件切换到另一个控件，我们通过高亮显示来告诉使用者焦点的所在。图 9.6 显示出了轻击某区域的效果。虽然颜色不太明显，但实际上被选中的插头为蓝色；相反，另外的插头为灰色。

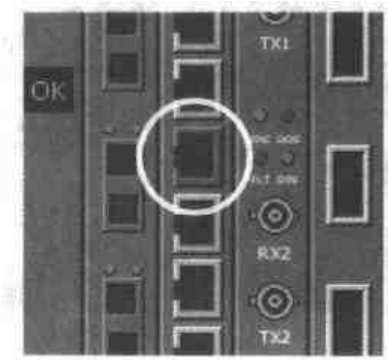


图 9.6 控件焦点

对于有条件使用鼠标来切换 GUI 的使用者来说，这种切换的方法有些陌生。但是，这种能取微小图形对象的功能是很有用的，而且它能明显改变使用者的看法。没有命名，我就看见了需要使用者在 2mm^2 的图形元素上点击的网络管理系统。

Component_3.py 包括了能更加生动地显示状态改变效果的附加代码。从根本上说，每个定义了能显示状态的对象的类，都在控件表单上增加了实例。

```
St_wid.append(self)      #为动画注册
```

然后，我们将使效果生动的功能与标志捆绑在一起。

```
self.img = PhotoImage(file='logo.gif')
self.logo=Label(self.outer, image=self.img, bd=0)
self.logo.place(relx=0.055, rely=0.992, anchor=SW)
self.logo.bind('<Button-1>', self.animate)
```

使效果生动起来其实很简单。

```
def animate(self, event):
  import random
  choice = random.choice(range(0, len(st_wid)-1))
  op     = random.choice(range(0, len(ops)-1))

  pstr = 'st_wid[%d].%s()' % (choice, ops[op])
```

```
        self.cobj = compile(pstr, 'inline', 'exec')
        self.rack.after(50, self.doit)

    def doit(self):
        exec(self.cobj)
        self.rack.after(50, self.animate(None))
```

如果你运行 FrontPanel_3.py 并点击标志语，你会击活该功能。当然，在黑白图中很难描绘出具体的效果，但是你可以从面板中的控制元件的阴影里辨认出差别。尤其是图 9.7 中左起第 4 个面板上的 J45 插头。

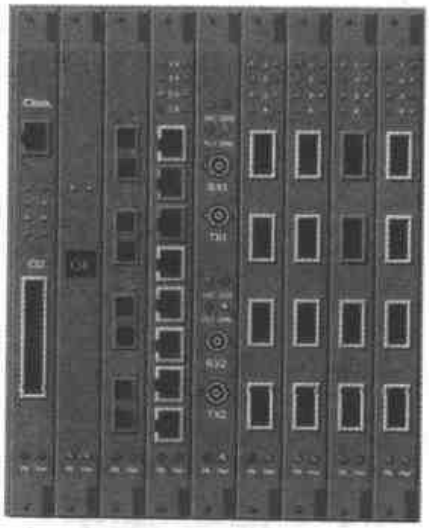


图 9.7　动画控件

当然，要把这样的一个面板变为功能系统还有很多工作要做。你很可能会使用设备的周期性 SNMP 测验来取得每个组件的状态，并把发光二极管设置为正确的状态。而且，如果设备支持热插卡你还可以监听卡架的内容以探知硬件的变化。最后，菜单将被加入窗口，用于给出结构功能的调用。

9.4　GIF、BMP 和图层

在前面章节中介绍的面板和机器使用了画图界面。通过努力，创建一个与真实设备很相似的控件并非难事。有时候，需要具有一些艺术领悟力才能达到令人满意的效果，所以必须采用交替的方法。有时，使用设备的照片来产生它精确的描绘是要更容易些的，特别是当这些设备很大时。在这一节中，我将告诉你许多用 GUIs 制作逼真图片的技巧。

让我们来看看图 9.8 中的 Cabletron SmartSwitch 6500 的前端面板。如果你把这幅图中放大的选区与图 9.6 中的组件相对比，你能发现，画出的面板显出了更为清楚的细节，尤其是文本标签。然而，如果你认为需要创建代码以精确设定组件在面板中的位置，那么照片图像会使工作变得容易。而且，照片图像再现了所有细节，不管它有多少或多复杂，而且它有很强的三维效果，而这种效果要通过绘图面板来创建是非常耗时的。

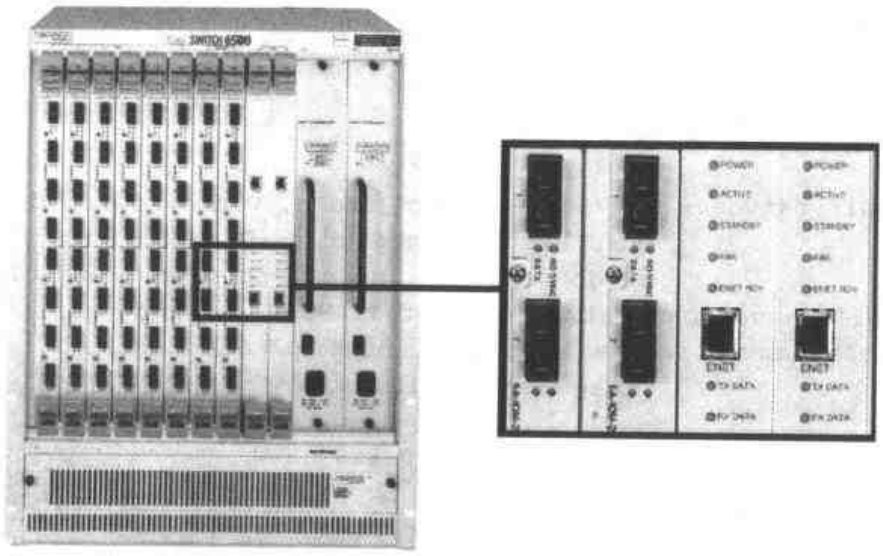


图 9.8 EC6110 开关，加亮区域被放大

创建面板模板的工作比创建相似的面板要更加容易。建立一个有图片的系统需要以下几步：

1. 如果可能的话，用空卡架给设备拍照；
2. 以同样的尺度用插入的卡给设备拍照（如果可能，最好个别进行）；
3. 把卡的图像剪切下来以使能精确地定义卡的外形；
4. 为每个卡的类型创建一个类，为动态组件（发光二极管，信号器等）或可用于导航的组件（插头，按钮等）；
5. 根据轮廓创建框架群；
6. 补全实现各种功能的代码。

下列的代码仅仅是一些样本，完整的源代码可在网上获得。

Components_4.py

```
class C6C110_CardBlank(GUICommon):
  def __init__(self, master=None, width=10, height=10,
               appearance=FLAT, bd=0):
      self.card_frame=Frame(master, relief=appearance, height=height,
                            width=width, bd=bd, highlightthickness=0)

class C6C110_FDDI(C6C110_CardBlank):
  def __init__(self, master, slot=0):
      self.img = PhotoImage(file='images/6c110_enet.gif')
      setattr(glb, 'img%d' % slot, self.img)
      self.width  = self.img.width()
      self.height = self.img.height()

      C6C110_CardBlank.__init__(self, master=master, width=self.width,
                                height=self.height)
```

```
        xypos = [(10,180),(10,187),
                 (10,195),(10,203),
                 (10,210),(10,235),
                 (10,242)]

        self.canvas = Canvas(self.card_frame, width=self.width,
                      bd=0,highlightthickness=0,
                      height=self.height,selectborderwidth=0)
        self.canvas.pack(side="top", fill=BOTH, expand='no')
        self.canvas.create_image(0,0,anchor=NW,
                       image=eval('glb.img%d' % slot))

        for i, y in [(0, 0.330), (1, 0.619)]:
            setattr(self, 'j%d' % i, Enet10baseT(self.card_frame,
                 fid="10Base-T-%d", port=i, orient=HW_LEFT,
                 status=STATUS_OFF, xwidth=15, xheight=12))
            getattr(self, 'j%d' % i).j45_frame.place(relx=0.52,
                 rely=y, anchor=CENTER)

        for i in range(len(xypos)):
            xpos,ypos = xypos[i]
            setattr(self, 'led%d' % (i+1), CLED(self.card_frame,
                 self.canvas, shape=ROUND, width=4, status=STATUS_ON,
                 relx=xpos, rely=ypos))

class C6C110_Chassis:
    def __init__(self, master):
        self.outer=Frame(master, borderwidth=0, bg=Color.PANEL)
        self.outer.forget()

        self.img = PhotoImage(file='images/6c110_chassis.gif')
        self.width  = self.img.width()
        self.height = self.img.height()

        self.canvas = Canvas(self.outer, width=self.width,
                      height=self.height,selectborderwidth=0)
        self.canvas.pack(side="top", fill=BOTH, expand='no')
        self.canvas.create_image(0,0,anchor=NW, image=self.img)

        self.rack = Frame(self.outer, bd=0, width=self.width-84,
                    height=self.height-180,
                    bg=Color.CHASSIS, highlightthickness=0)
        self.rack.place(relx=0.081, rely=0.117, anchor=NW)

        x = 0.0
        for i in range(12):
            if i in [0,1,2,3,4,5]:
                card =C6C110_FDDI(self.rack, slot=i)
            elif i in [6,7,8,9]:
```

❷ LED Positions

❸

❹

❺

❻ Create LEDs

❼

```
            card =C6C110_ENET(self.rack, slot=i)
        else:
            card =C6C110_PSU(self.rack, slot=i)
        card.card_frame.place(x=x, y=0, anchor=NW)
        x = x + card.width
```



代码注解

❶ 大多数用于绘制面板组件的代码都很相似。下面的代码比较短，因为没有必要建立大量的组件。

```
self.img = PhotoImage(file='images/6c110_enet.gif')
setattr(glb, 'img%d' % slot, self.img)
self.width  = self.img.width()
self.height = self.img.height()
```

在_init_方法中，我们创建了一个照片图像的实例，保持其在给定范围很重要。如果图像被删除，你会在以前希望有图片的地方看到空空的背景。为了建立面板，必须得到图像的尺寸（以像素为单位）。

❷ 正如你所料，我们建立了一个存放已计算出的发光二极管位置的元组表单。

```
xypos = [(10,180),(10,187),
...
```

❸ 所有的边界、高亮部分，以及选区边界的宽度必须为0，以保证面板能够连接在一起。

```
self.canvas = Canvas(self.card_frame, width=self.width,
                bd=0,highlightthickness=0,
                height=self.height,selectborderwidth=0)
```

❹ 通过使用描画过的照片图像，图片就在底层画布上被创建出来了。

```
self.canvas.create_image(0,0,anchor=NW,
                image=eval('glb.img%d' % slot))
```

❺ J45 插头被画在图画上有插头的地方，这为其他的被动器件增加了导航功能和状态属性。

```
for i, y in [(0, 0.330), (1, 0.619)]:
    setattr(self, 'j%d' % i, Enet10baseT(self.card_frame,
        fid="10Base-T-%d" % i, port=i, orient=HW_LEFT,
        status=STATUS_OFF, xwidth=15, xheight=12))
    getattr(self, 'j%d' % i).j45_frame.place(relx=0.52,
        rely=y, anchor=CENTER)
```

插头的尺寸被送入构建函数,这为 J45 插头增加了功能。而 J45 插头在这章的前面部分就已经介绍过了。

❻ 发光二极管被画在画布的指定位置，值得注意的是，这些发光二极管用的不是 LED 类，而是 CLED 类。因为它们是直接被画在画布上，而不是画在框架里的。如果使用了 LED 类，那么在我们要填充与控件相连的长方形框架及背景的颜色时会出现问题。

```
for i in range(len(xypos)):
    xpos,ypos = xypos[i]
    setattr(self, 'led%d' % (i+1), CLED(self.card_frame,
    self.canvas, shape=ROUND, width=4, status=STATUS_ON,
    relx=xpos, rely=ypos))
```

还必须注意：我们把 cardframe 和画布均送入了构建函数。这让可使控件基类实现闪烁效果的 after 方法变得易于取得。

● 最后，我们要填充卡架。为了便于描述，2 个 FDDI 卡已被以太卡所代替。虽然这对 ATM 交换机并没有什么意义，但它却充分证明了使用这种方法会使安排卡的放置更为轻松。

```
x = 0.0
for i in range(12):
    if i in [0,1,2,3,4,5]:
        card =C6C110_FDDI(self.rack, slot=I)
    elif i in [6,7,8,9]:
        card =C6C110_ENET(self.rack, slot=i)
    else:
        card =C6C110_PSU(self.rack, slot=i)
    card.card_frame.place(x=x, y=0, anchor=NW)
    x = x + card.width
```

注意 就像在先前的例子中一样，下一个卡如何放置取决于真实的卡的宽度，而不是计算出来的。

运行 EC6110.py 屏幕显示如图 9.9 的右图所示。左图中显示的是没有安装卡时卡架的样子，就像在先前的例子中那样，为使组件更加生动的准备工作已做好。点击围绕着卡的底盘的任何地方，图像就会变得生动。具体内容这里就不再赘述了，一切由你去尝试吧。

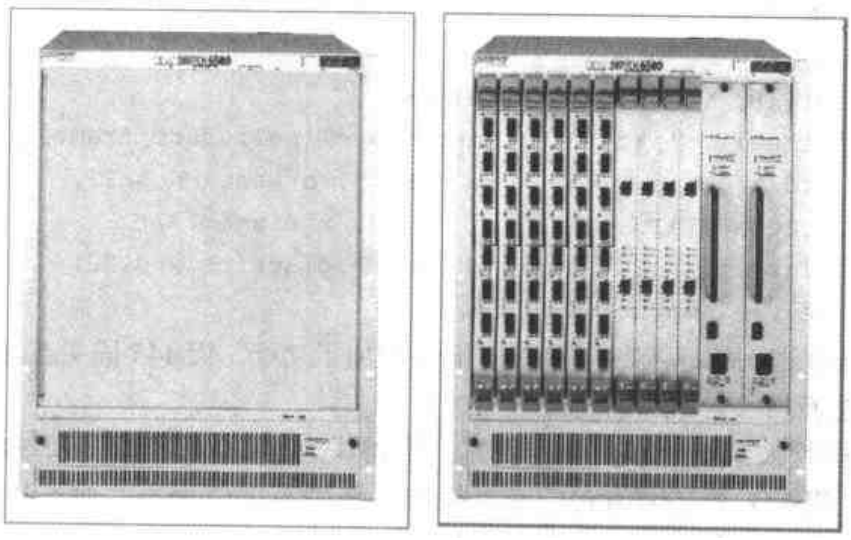


图 9.9 带有 GIF 面板及放置组件的卡架

9.5 一个更完整的例子

以前的例子中已经描述了使 GUI 显示面板及其他器件的所有方法。为了进一步探讨这个主题，让我们来看一个简单却非常有用的例子。

许多数字万用表都有可与计算机相连的连续界面。我有一个 RadioShack 22-168A 万用表，形如图 9.10 所示。该表有 24 个量程来测交流/直流电压或电流，还具有测量电阻、电容、频率等功能。这个万用表实行了一个简单的协议，该协议允许检测当前显示的数值。使用一个简单的编码系统可减小当前的选择范围。



图 9.10 数字万用表

执行一个 GUI 来显示万用表的当前显示状态并非难事。而要显示被选择的量程对我们来说却是一个挑战。一种解决办法是在万用表的顶部只显示 LCD 面板，而在 LCD 或其他什么地方用文本显示当前的量程。然而，这并非为取得超现实的效果，也决非为读者制作一个特别有趣的例子。

我们要采用的方法，是使万用表上的圆按钮具有像真按钮那样的功能，这样，它就能向使用者反映万用表的真实外观，这需要大量的工作来完成。但是，最后你将看到，这将创造出一个多么吸引人的 GUI。

下面就是我们为选取准备一系列 GIF 图层所实行的步骤，如图 9.11 所示：

1. 在某个位置获得有选择器的图像；
2. 从矩形选区中剪切下选择器；
3. 修描图像，去掉选择器周围的多余像素；
4. 添入便于区分的背景色；
5. 将图片旋转 15°；
6. 保持图像原有大小，并将其剪切至原来的选区；

7. 将与选择器背景一致的颜色实体作为透明色，再把图像保存为透明的 GIF 图片。图 9.11（7）给出了层叠图像的显示效果。

你很可能会认为我是一个疯子，竟然意图通过作出 24 张经过旋转的 GIF 图来显示真实万用表的精确外观。可能你是对的，但是请你最好保留自己的最终评判，直到你看到最终的效果！

注意 当我开始做这个工作的时候，我遇到了无法与万用表保持连续通信的难题，当我仔细研究过之后，我决定继续做下去。我通过建立一个简单的测试装备模仿设备的输入。我打算先显示测试说明，再加入一系列信息。这项技术，对于使应用开始是很有价值的。因为即使这些输入是你所无法取得的（或对你的老板来说签署这个购买建议书的花费太大了）但模仿设备的输入还是可以做到的。

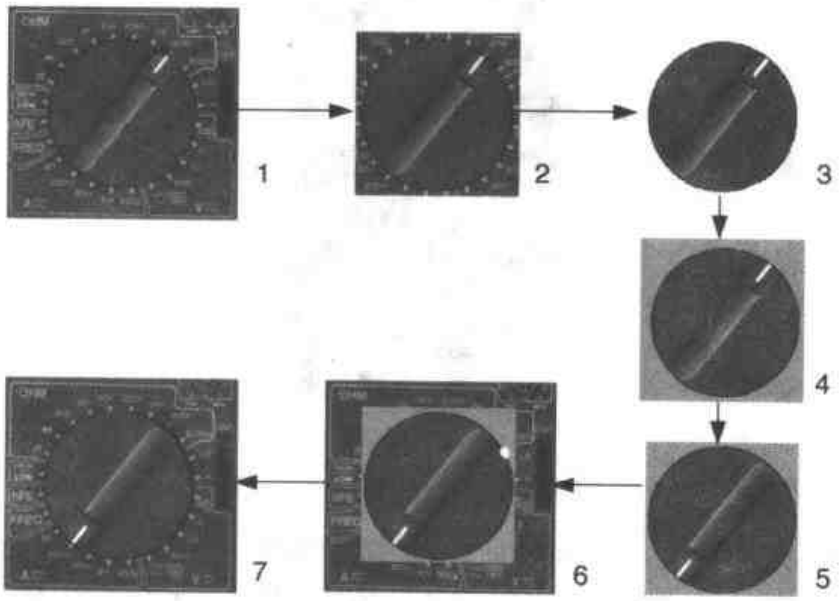


图 9.11 产生循环选择图片的步骤

下面就是执行万用表的测试说明的代码。首先，我们从数据文件开始，该文件定义了万用表的量程、控制表格、标签和其他的重要数据。

Example 9 1 data.py

```
#  Tag  Run  RFlag  Units   Key
PRIMARY_DATA = [                                    ●
  ('DI', 1, 'OL',  'mV',   'di'),
  ('DI', 0, '',    'mV',   'di'),
  ('OH', 1, 'OL.', 'Ohm',  'oh200o'),
  ('OH', 0, '4',   'Ohm',  'oh200o'),
  ('OH', 1, '.OL', 'KOhm', 'oh2ko'),                ●
```

```
  ('OH', 0, '2',   'KOhm', 'oh2ko'),
  ('OH', 1, 'O.L', 'KOhm', 'oh20ko'),
  ...
  ('CA', 0, '2',   'nF',   'calo'),
  ('CA', 0, '2',   'uF',   'cahi'),
  ('DC', 0, '4',   'mV',   'dc200mv'),
  ('DC', 0, '2',   'V',    'dc2v'),
  ('DC', 0, '3',   'V',    'dc20v'),
  ('DC', 0, '4',   'V',    'dc200v'),
  ('DC', 0, '',    'V',    'dc1000v'),
  ...
  ('AC', 0, '4',   'V',    'ac200v'),
  ...
  ('FR', 0, '2',   'Hz',   'frh'),
  ('FR', 0, '2',   'KHz',  'frk'),
  ('FR', 0, '2',   'MHz',  'frm'),
  ('HF', 0, '',    '',     'hfe'),
  ('LO', 0, '',    '',     'logic'),
  ('XX', 0, '',    '',     'cont')]

SECONDARY_DATA = {
#  Key       m  u  A  m  V  k  M  O  n  u  F  M  K  Hz AC Control, Label
'di':      (0, 0, 0, 0, 0,  0, 0, 0,  0, 0,  0, 0,  0,  0,  0, 'diode', 'DIO'),
'oh200o':  (0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0, '200Ohm', ''),
'oh2ko':   (0,  0,  0,  0,  0,  1,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0, '2KOhm', ''),
'oh20ko':  (0,  0,  0,  0,  0,  1,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0, '20KOhm', ''),
'oh200ko': (0,  0,  0,  0,  0,  1,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0, '200KOhm', ''),
'oh2mo':   (0,  0,  0,  0,  0,  0,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0, '2MOhm', ''),
'oh20mo':  (0,  0,  0,  0,  0,  0,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0, '20MOhm', ''),
....
'frh':     (0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0, 'freq', 'FREQ'),
'frk':     (0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  0, 'freq', 'FREQ'),
'frm':     (0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  1,  0, 'freq', 'FREQ'),
'hfe':     (0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 'hfe', 'HFE'),
'logic':   (0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 'logic', 'LOGI'),
'cont':    (0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 'cont', '')}

TESTDATA = ['DI   OL   mV', 'DI   OL   mV', 'DI   OL   mV',
            'DI   OL   mV', 'DI   OL   mV', 'DI   OL   mV',
            'DI  0422  mV', 'DI  0022  mV', 'DI  0003  mV',
            'DI  0001  mV', 'DI  0004  mV', 'DI  0001  mV',
            'DI  0001  mV', 'DI  0001  mV', 'DI  0892  mV',
...
            'OH  OL.  Ohm', 'OH  OL.  Ohm', 'OH  OL.  Ohm',
            'OH  OL.  Ohm', 'OH  OL.  Ohm', 'OH  007.8 Ohm',
            'OH  014.7 Ohm', 'OH  001.1 Ohm', 'OH  116.3 Ohm',
            'OH  018.3 Ohm', 'OH  002.9 Ohm', 'OH  003.0 Ohm',
            'OH  OL.  Ohm', 'OH  .OL KOhm', 'OH  .OL KOhm',
            'OH  .OL KOhm', 'OH  .OL KOhm', 'OH -0.010KOhm',
            'OH  0.085KOhm', 'OH  0.001KOhm', 'OH  0.001KOhm',
```

```
            'OH  0.047KOhm', 'OH  0.410KOhm', 'OH  0.277KOhm',
...
            'CA  0.000  nF', 'CA  0.000  nF', 'CA  0.000  nF',
            'CA  0.000  nF', 'CA  0.000  nF', 'CA  0.000  uF',
            'CA  0.000  uF', 'CA  0.000  uF', 'CA  0.000  uF',
            'DC  034.1  mV', 'DC  025.6  mV', 'DC  063.5  mV',
            'DC  072.3  mV', 'DC  040.7  mV', 'DC  017.5  mV',
            'DC  005.2  mV', 'DC  0.005   V', 'DC  0.002   V',
            'DC  0.001   V', 'DC  0.001   V', 'DC  0.001   V',
...
            'FR  0.000 KHz', 'FR  0.001 KHz', 'FR  0.001 KHz',
            'FR  0.002 KHz', 'FR  0.000 KHz', 'FR  0.001 KHz',
            'FR  0.001 KHz', 'FR  0.000 KHz', 'HF  0000     ',
            'HF  0000     ', 'HF  0000     ', 'HF  0000     ',
            'HF  0000     ', 'HF  0000     ', 'HF  0000     ',
            'HF  0000     ', 'LO   rdy     ', 'LO   rdy     ',
            'LO   rdy     ', 'LO   rdy     ', 'LO   rdy     ',
            'LO   rdy     ', 'LO - rdy     ' ]

PANEL_LABELS = [ (225,  80, 'Arial',  'M', 'mhz'),
                 (240,  80, 'Arial',  'K', 'khz'),
                 (255,  80, 'Arial',  'Hz', 'hz'),
                 (225,  95, 'Arial',  'm', 'ma'),
                 (240,  95, 'Symbol', 'm', 'ua'),
                 (255,  95, 'Arial',  'A', 'a'),
                 (240, 110, 'Arial',  'm', 'mv'),
                 (255, 110, 'Arial',  'V', 'v'),
                 (225, 125, 'Arial',  'K', 'ko'),
                 (240, 125, 'Arial',  'M', 'mo'),
                 (255, 125, 'Symbol', 'W', 'o'),
                 (225, 140, 'Arial',  'n', 'nf'),

                 (240, 140, 'Symbol', 'm', 'uf'),
                 (255, 140, 'Arial',  'F', 'f'),
                 (50,  110, 'Arial',  'AC','ac')]
```

代码注解

❶ 第一个表定义了量程，它可以作为 tuples 来支持或存储。标签决定了读数的类型。

```
# Tag  Run  RFlag  Units   Key
PRIMARY_DATA = [
  ('DI', 1, 'OL',  'mV',   'di'),
  ('DI', 0, '',    'mV',   'di'),
```

Key 用来获取控制屏幕上控制器的数据。

❷ 设定上限的指示器经过译码（通过改变小数点）来指示当前所选的量程。

```
('OH', 1, '.OL', 'KOhm', 'oh2ko'),
('OH', 1, 'O.L', 'KOhm', 'oh20ko'),
```

大多数的量程都有上限值。

❸ PRIMARY_DATA 中的关键字用来获取 SECONDARY_DATA 字典中的数组，该数组定义了各个信号器在显示时是否为“打开”状态。

```
# Key      m  u  A  m  V  k  M  O  n  u  F  M  K  Hz AC Control, Label
'di':     (0, 0, 0, 0, 0,  0, 0, 0,  0, 0,  0, 0,  0,  0,  0, 'diode', 'DIO'),
'oh200o': (0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  '200Ohm',  ''),
```

❹ TESTDATA 获取的数据的格式是由万用表的一系列协议定义的。这些数据是通过一个数据获取程序直接从万用表中获取的（还可用于排除上面提到过的问题中的故障）。

```
TESTDATA = ['DI    OL   mV', 'DI    OL   mV', 'DI    OL   mV',
            'DI    OL   mV', 'DI    OL   mV', 'DI    OL   mV',
            'DI  0422   mV', 'DI  0022   mV', 'DI  0003   mV',
            'DI  0001   mV', 'DI  0004   mV', 'DI  0001   mV',
            'DI  0001   mV', 'DI  0001   mV', 'DI  0892   mV',
```

Example 9_1.py

```
from Common import *
from Tkinter import *
from Example_9_1_data import *
import sys, time, string

class MeterServer:
  def __init__(self):
      # Open up the serial port
      pass                                                    ❶

  def poll(self):
      import random
      choice = random.choice(range(0, len(TESTDATA)-1))
      return TESTDATA[choice]                                 ❷

class MultiMeter:
  def __init__(self, master):
      self.root = master
      self.root.title("Digital Multimeter")
      self.root.iconname('22-168a')

      self.holdVal  = '0.0'
      self.curRange = None
      self.lineOpen = FALSE

      self.canvas = Canvas(self.root, width=300, height=694)
      self.canvas.pack(side="top", fill=BOTH, expand='no')

      self.img = PhotoImage(file='images/multimeter.gif')
      self.canvas.create_image(0,0,anchor=NW, image=self.img)
      self.buildRule()
```

```
        self.root.update()
        self.root.after(5, self.buildSymbols)
        self.dataReady = FALSE
        self.root.after(5, self.buildScanner)
        self.multimeter = MeterServer()
        self.root.after(500, self.doPoll)

    def buildSymbols(self):
        for x, y, font, txt, tag in PANEL_LABELS:
            self.canvas.create_text(x, y, text=txt,
                                    font=(font, 12),
                                    fill="gray75",
                                    anchor = CENTER,
                                    tags=tag)

    def buildRule(self):                                    ❶
        self.canvas.create_line(75,150, 213,150,
                                width=1,fill="#333377")
        self.Xincr = 140.0/40.0
        self.X = x = 75
        self.X1 = 213
        y = 150
        lbli = 0

        for i in range(40):
            lbl = ''
            if i in [0,9,19,29,39]:
                h   = 6
                lbl  = `lbli`
                lbli = lbli + 5
            elif i in [5,14,24,34]:
                h = 4
            else:
                h = 2
                self.canvas.create_line(x,y, x,y-h,
                                        width=1,fill="#333377")
            if lbl:
                self.canvas.create_text(x, y-5, text=lbl,
                                        font=("Arial", 6),
                                        fill="#333377",
                                        anchor = S),
            x = x + self.Xincr

    def startAnimation(self):
        self.animX = self.X
        self.action = TRUE
        self.root.after(30, self.animate)
```

❸

代码注解

❶ 我们的代码的检测说明并没有初始化连续界面。

```
def __init__(self):
    # Open up the serial port
    pass
```

❷ 模仿万用表计数的计数方法从 TESTDATA list 中恢复了随机实体。

```
def poll(self):
    import random
    choice = random.choice(range(0, len(TESTDATA)-1))
    return TESTDATA[choice]
```

❸ 这部分代码说明了怎样安排一个作为背景工作的操作发生。方法 buildSymbols 和 buildScanner 用来在几毫秒之后作为回调运行。

```
self.root.after(5, self.buildSymbols)
self.dataReady = FALSE
self.root.after(5, self.buildScanner)
self.multimeter = MeterServer()
self.root.after(500, self.doPoll)
```

buildScannel 把 GIF 文件转换为照片图像，这是一个费时的工作。通过把初始化移至背景，我们能立即显示基本的 GUI（虽然在运行前，用户必须等待所有图片被加载）。

❹ BuildRuler 构造了万用表的表盘，用以指示对应与量程选择刻度盘所选量程的当前测量值，并在测量值超过原有量程时激活图片以切换量程。

Example 9-1.py（续）

```
def animate(self):                    ❺
    if self.action:
        self.canvas.create_line(self.animX,155, self.animX,167,
                                width=2,fill="#333377",
                                tags='anim')
        self.animX = self.animX + self.Xincr
        if self.animX > self.X1:
            self.animX= self.X
            self.canvas.delete('anim')
            self.root.after(30, self.animate)
    else:
        self.canvas.delete('anim')

def stopAnimation(self):
    self.action = FALSE

def buildScanner(self):               ❻
    self.primary_lookup = {}
    for key, hasr, rfmt, un, sec in PRIMARY_DATA:
        if not self.primary_lookup.has_key(key):
            self.primary_lookup[key] = []
```

```
                self.primary_lookup[key].append((hasr, rfmt, un, sec))

        keys = SECONDARY_DATA.keys()
        for key in keys:
            img = SECONDARY_DATA[key][-2]
            try:
                if getattr(self, 'i%s' % key):
                    pass    # Already done...
            except:
                setattr(self, 'i%s' % key,
                    PhotoImage(file="images/%s.gif" % img))
        self.dataReady = TRUE

    def doPoll(self):
        if self.dataReady:
            result = self.multimeter.poll()
            if result:
                self.updateDisplay(result)
        self.root.after(1000, self.doPoll)

    def getRange(self, tag, val, units):          ❷
        matchlist = self.primary_lookup[tag]
        if not matchlist: return None
        gotIndex = None
        gotOpenLine = FALSE
        for hasr, rfmt, un, sec in matchlist:
            if hasr and (string.find(val, 'L') >= 0):
                if rfmt == string.strip(val):
                    gotIndex = sec
                    gotOpenLine = TRUE
            else:
                decimal = string.find(val, '.')
                if decimal > 0:
                    if rfmt == `decimal`:
                        gotIndex = sec
                else:
                    if not rfmt:  # No decimals
                        gotIndex = sec
            if gotIndex:
                if not string.strip(units) == string.strip(un):
                    gotIndex = None
            if gotIndex:
                break
        return (gotIndex, gotOpenLine)

    def updateDisplay(self, result):
        self.canvas.delete('display')
        tag   = result[:2]
        val   = result[3:9]
        units = result [9:13]
```

```
# display the hold value
redraw = FALSE
try:
    hold = string.atof(self.holdVal)
    nval = string.atof(val)
    if hold <= 0.0:
        if nval < 0.0:
            if nval < hold:
                self.holdVal = val
                redraw = TRUE
        else:
            hold = 0.0
    if hold >= 0.0 and not redraw:
        if nval >= 0.0:
            if nval > hold:
                self.holdVal = val
                redraw = TRUE
        else:
            self.holdVal = '0.0'
            redraw = TRUE
except ValueError:
    self.holdVal = '0.0'
    redraw = TRUE

if redraw:
    self.canvas.delete('holdval')
    self.canvas.create_text(263, 67, text=self.holdVal,
                    font=("Digiface", 16),
                    fill="#333377",
                    anchor = E,
                    tags="holdval")

range, openline = self.getRange(tag, val, units)
if range:    # Change the control to reflect the range
    if not self.curRange == range:
        self.curRange = range
        self.canvas.delete('control')
        self.canvas.create_image(146, 441, anchor=CENTER,
              image=getattr(self, 'i%s' % range),
              tags="control")
        self.holdVal = '0.0'  # reset
    if openline:
        self.startAnimation()
    else:
        self.stopAnimation()

    # Now we will update the units symbols pn the display
    ma,ua,a,mv,v,ko,mo,o,nf,uf,f,mhz,khz,hz,ac, ctrl, lbl = \
                                SECONDARY_DATA[range]
```

```
                for tag in ['ma','ua','a','mv','v','ko','mo','o',
                        'nf','uf','f','mhz','khz','hz','ac']:
                    self.canvas.itemconfig(tag,
                         fill=['gray75','#333377'][eval(tag)])
                    # Update the label field if there is one
                    self.canvas.delete('label')
                    if lbl:
                         self.canvas.create_text(55, 150, text=lbl,
                                        font=("Arial", 12),
                                        fill="#333377",
                                        anchor = CENTER,
                                        tags="label")

        # Finally, display the value
        self.canvas.create_text(214, 100, text=val,
                        font=("Digiface", 48),
                        fill="#333377",
                        anchor = E,
                        tags="display")

if __name__ == '__main__':
    root = Tk()
    multimeter = MultiMeter(root)
    multimeter.root.mainloop()
```

代码注解（续）

❾ 这种激活的方法显示了一组快速增长的垂直线。当线延长至显示画面的右边界时，将自动重置。

❿ Buildscanner 从 PRIMARY_DATA 建立了一个字典，来初步检查从万用表获取的信息。GIF 图片也当作照片图像被加载。

⓫ GetRange 分析从万用表获取的信息，从而决定要显示的量程与相应数值。

⓬ 万用表保留了测量的最高（或最低）值。这些代码显示了这些数据。代码比我们预想的要长，因为我们必须能够探知最大正值和最小负值。

⓭ 在这部分代码中，如果量程改变了，我们就要改变层叠的选择旋钮的位置。注意以前的被称为 control 的图像先被删除了。

⓮ 最后，我们根据当前所选的量程刷新信号器。为了达到简化的目的，我们用非常浅或非常深的灰色填充文本的笔画。

如果我们运行 Example9_1.py 我们将看到如图 9.12 所示的屏幕。当每一个值都被显示了，量程选择器就被激活而能为数值指示量程。这种显示的例子如 9.13 所示。

为了完成这个例子，我们只需要增加异步支持来连接万用表。这用到了 Python 的扩展模块 sio 模块。它可以很快地从 Python 语言站点（http://www.python.org）得到。sio 模块用了一个小小的变换来支持万用表握手协议的特性，但等一会儿，你就会了解得更为详尽。扩展模型的细节将出现在后面的章节里（从第 3 部分的“把它们全放在一起”开始。该模型使用了商业动态链接库，这作为一般通用的模型是可以获得的（细节见附



图 9.12　模拟测量

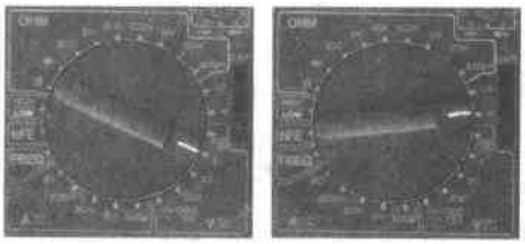


图 9.13　范围选择模拟

录 G 的“siomodule”)。代码必要的改变在 Example_9_2.py 中：

Example_9_2.py

```
from crilib import *
import sys, regex, serial, time, string, os
IGNORE   = '\n\r'

class RS232(serial.Port):
  def __init__(self):
       serial.Port.__init__(self)

   def open(self, cfg):
       self.debug = cfg['debug']
       self._trace('RS232.open')
       cfg['cmdsEchoed']  = FALSE
       cfg['cmdTerm']     = '\r'
       cfg['rspTerm']     = '\r'
       cfg['rspType']     = serial.RSP_TERMINATED
       serial.Port.open(self, cfg)

class MeterServer:
  def __init__(self):
      # Open up the serial port
      try:
          d = serial.PortDict()
          d['port']      = serial.COM1
          d['baud']      = serial.Baud1200
```

❶ Load serial

❶ Init UART

❸ Setup params

❹ Connection

```
            d['parity']    = serial.NoParity
            d['dataBits']  = serial.WordLength7
            d['stopBits']  = serial.TwoStopBits
            d['timeOutMs'] = 1500
            d['rspTerm']   = IGNORE
            d['rtsSignal'] = 'C'
            d['debug']     = FALSE
            self.fd = RS232()
            self.fd.open(d)
        except:
            print 'Cannot open serial port (COM1)'
            sys.exit()

    def poll(self):
        try:
            line = self.fd.write('D\r')
            # OK, read the serial line (wait maximum of 1500 mSec)
            inl = self.fd.readTerminated()
            return inl
        except:
            return 'XX   Off
```

Connection

代码注解

❶ 一系列的模块封装成了 sio 模块，sio 模块是一个动态链接库。

```
from crilib import *
import sys, serial, time, string, os
```

❷ 在我们要初始化 UART（通用异步接收模式）。

```
def __init__(self):
    serial.Port.__init__(self)
```

❸ 一系列协议的界限已被定义。虽然在此例中我们不能打开调试，但必要的初启程序已经给出，以帮助你重用代码。

❹ UART 的通信参数被设定。

```
d['port']      = serial.COM1
d['baud']      = serial.Baud1200
d['parity']    = serial.NoParity
d['dataBits']  = serial.WordLength7
d['stopBits']  = serial.TwoStopBits
```

注意 有些不正常的格式，放慢通信速度。但是，万用表是一种非常简单的设备，并且不太贵，因此我们在这儿例外了一次。

❺ 读取是模块化的，这对 GUI 来说是不常见的。因为没有使用者控制显示，我们不必担心事件循环暴露。所以，如果万用表不能对一个数据请求进行响应，我们可以检查 1 秒左右。

❶ 不正常的握手协议就是在这儿处理的。这就是我为代码建立检测说明的原因了，因为我不能一开始就让设备响应测试请求。我不得不使用软件来监听各个界面的控制线，以决定什么设备需要通信，不常有的情况是，在它发送任何信息前需要发送请求（RTS）来减慢速度。这一系列的模块，自从从 PythonFTP 站点获取后，就有 RTS 变高。一个小小的变化决定了这个问题。这里，我们强迫 RTS 减慢。

```
d['rtsSignal']='C'
```

❷ 必须检测万用表以获取当前测量值。

```
line = self.fd.write('D\r')
```

❸ 如果检测时间已过，我们假设万用表已关闭，我们伪造了一条消息来显示一个合适的值。

```
return 'XX   off     '
```

经过小许修改的 serial.py 源代码显示如下。

Serial.py（只给出修改过的部分）

```
        def __init__(self):
            self._dict = {}
...
            self._dict['rspType'] = RSP_BEST_EFFORT
            self._dict['rspFixedLen'] = 0
            self._dict['rtsSignal'] = 'S'
            self._dict['dtrSignal'] = 'S'
...
        def __setitem__(self, key, value):
...
            elif key == 'debug' or key == 'cmdsEchoed':
                if type(value) != IntType:
                    raise AttributeError, 'must be a boolean value'
            elif key == 'rtsSignal':
                if not value[:1] in 'CS':
                    raise AttributeError, 'Illegal rtsSignal value'
            elif key == 'dtrSignal':
                if not value[:1] in 'CS':
                    raise AttributeError, 'Illegal dtrSignal value'

            self._dict[key] = value
...
        def open(self, cfg):
            self._chkSioExec(SioRxClear, (port,))
            self._chkSioExec(SioDTR, (port, cfg['dtrSignal'][:1]))
            self._chkSioExec(SioRTS, (port, cfg['rtsSignal'][:1]))
            self._chkSioExec(SioFlow, (port, 'N'))
```

运行 Example_9_2.py 显示万用表。因为显示方法没有改变，显示与图 9.12 没有什么区别。因此，这里就不再给出相应的图了。

9.6 使用 POV-Ray 创建的虚拟设备

当然，一些应用可能没有物理设备，拿这些例子来说，创建一个有光照的图像（使用 ROV-Ray 或其他的随机系统）来创建真实的设备是可能的。

图 9.14 给出了一个有光照效果的 GUI 的实例。该实例已被用于商业应用。这个例子使用了同样的层叠技术，用以创建如图 9.9 的前端面板，只有这幅图像是完全伪造的。这个 GUI 支持的应用已经计划要给航空公司的飞行员使用，所以显示的画面被设计得有点像在飞行器中看见的仪表装置。阴影和高光使得这个 GUI 有了很强的三维效果。所有这些都是计算机用 POV-Ray 创造出来的。文字、发光二极管、导航按钮都是层叠的 Tkinter 控件。值得注意的很重要的一点是：这个应用与仪表架毫不相干，它实际上是一个数据库获取应用。这是一个很棒的应用。

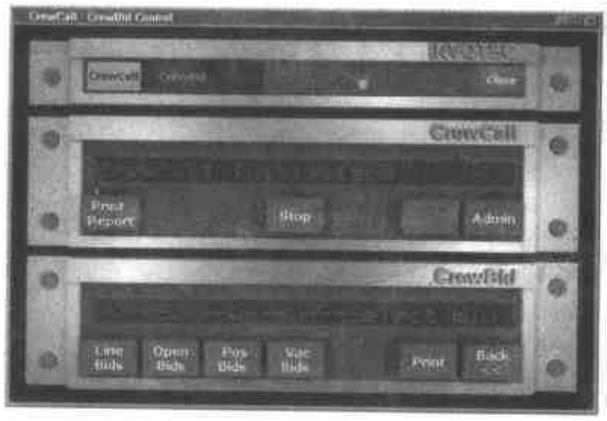


图 9.14 光线跟踪用户界面 （INVOTEC,Inc.许可）

9.6.1 现在看点完全不同的东西

我为使用这样的标题而感到抱歉，最后一个例子是要展示一下，用户界面可以变得多么的与众不同。对电脑游戏 Myst 和 Riven 比较熟悉的读者，将会与我分享这令人喜爱的使用光照效果的用户界面。这是一个这类界面的过于简化的版本。我不想像许多书那样，谈论光迹图像的细节。让我们从图 9.15 中的最基本的图像开始吧。

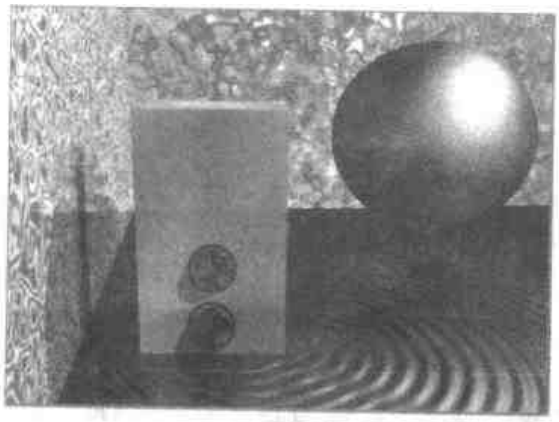


图 9.15 POV 光线产生的基景

这幅图若是彩色的看起来会更好，所以你一定很想从网上下载吧！既然你很希望解决这个例子所反映出来的简单的困惑，我会提供这个应用的一个代码片段。我要用这章中介绍的层技术来把该功能与显示出来的两个按钮捆绑在一起。这需要特殊处理，因为这两个按钮需要获得焦点，当它们获得焦点时就高亮显示，并收到按钮按下的事件。下列的代码摘录是来管理按钮的。

Example 9_3.py

```
class Machine:
  def __init__(self, master):
      self.root = master
...
      self.b1 = self.canvas.create_oval(216,285, 270,340, fill="",
                                        outline='#226644', width=3, tags='b_1')
      self.canvas.tag_bind(self.b1, "<Any-Enter>", self.mouseEnter)
      self.canvas.tag_bind(self.b1, "<Any-Leave>", self.mouseLeave)

      self.b2 = self.canvas.create_oval(216,355, 270,410, fill="",
                                        outline='#772244', width=3, tags='b_2')

      self.canvas.tag_bind(self.b2, "<Any-Enter>", self.mouseEnter)
      self.canvas.tag_bind(self.b2, "<Any-Leave>", self.mouseLeave)
      Widget.bind(self.canvas, "<1>", self.mouseDown)

      self.buttonAction = {'b_1': self.b1_action,
                           'b_2': self.b2_action}

  def mouseDown(self, event):
      # See if we're on a button. If we are, it
      # gets tagged as CURRENT for by Tk.
      if event.widget.find_withtag(CURRENT):
          tags = self.canvas.gettags('current')
          if '_' in tags[0]:
              self.buttonAction[tags[0]]()

  def mouseEnter(self, event):
      # The CURRENT tag is applied to
      # The object the cursor is over.
      tags = self.canvas.gettags('current')
      usetag  = tags[0]
      self.lastcolor = self.canvas.itemcget(usetag, 'outline')
      self.canvas.itemconfig(usetag,outline=Color.HIGHLIGHT)
      self.canvas.itemconfig(usetag,fill=self.lastcolor)

  def mouseLeave(self, event):
      tags = self.canvas.gettags('current')
      usetag  = tags[0]
      self.canvas.itemconfig(usetag, outline=self.lastcolor)
      self.canvas.itemconfig(usetag,  fill="")
```

```
    def b1_action(self):
        if self.inSet:
            value = eval(self.digits[self.curDigit])
            value = value + 1
            exec('%s = value' % self.digits[self.curDigit])
            self.makeTime()
            self.displaySet()

    def b2_action(self):
        if not self.inSet:
            self.inSet = TRUE
            self.displaySet()
            self.root.after(1000, self.displayTime)
        else:
            self.curDigit = self.curDigit + 1
            if self.curDigit > 3:
                self.inSet = FALSE
                self.canvas.delete('settag')
                self.mouseLeave(None)
                self.doCountdown()
```

代码注解

❶ 我们在显示图像上画两个圆圈，为了和整个画面相配，圆圈是有色的。注意，圆圈的内部并没有填充色，是透明的。

```
self.b1 = self.canvas.create_oval(216,285, 270,340, fill="",
                    outline='#226644', width=3, tags='b_1')
```

❷ 内部透明的圆圈只有这样的特性：透明的对象不能接收按钮事件，所以我们给圆圈的边线绑定进入（enter）和离开（leave）事件。

```
self.canvas.tag_bind(self.b2, "<Any-Enter>", self.mouseEnter)
self.canvas.tag_bind(self.b2, "<Any-Leave>", self.mouseLeave)
Widget.bind(self.canvas, "<1>", self.mouseDown)
```

我们还给整个画布加上了单击鼠标左键的事件。注意我们使用了以混合控件来绑定事件的方法。

❸ self.buttonAction 是一个非常简单的调度程序。

```
self.buttonAction = {'b_1': self.b1_action,
                     'b_2': self.b2_action}
```

❹ 当收到事件时通过使用画布的标签，mouseDown 分配了合适的功能。

```
def mouseDown(self, event):
    if event.widget.find_withtag(CURRENT):
        tags = self.canvas.gettags('current')
        if '_' in tags[0]:
            self.buttonAction[tags[0]]()
```

❺ mouseEnter 给圆圈填充了颜色以便它能接收按钮事件。

```
def mouseEnter(self, event):
```

```
        # the CURRENT tag is applied to
        # the object the cursor is over.
        tags = self.canvas.gettags('current')
        usetag  = tags[0]
        self.lastcolor = self.canvas.itemcget(usetag, 'outline')
        self.canvas.itemconfig(usetag,outline=Color.HIGHLIGHT)
        self.canvas.itemconfig(usetag,fill=self.lastcolor)
```

❶ 当光标离开按钮时，mouseLeave 清除了填充色。

```
def mouseLeave(self, event):
     tags = self.canvas.gettags('current')
     usetag  = tags[0]
     self.canvas.itemconfig(usetag, outline=self.lastcolor)
     self.canvas.itemconfig(usetag,  fill="")
```

❷ 最后，当我们遇到某种情况时，我们调用 mouseLeave 直接清除高光，即使光标在画布 item 上。

```
self.canvas.delete('settag')
self.mouseLeave(None)
```

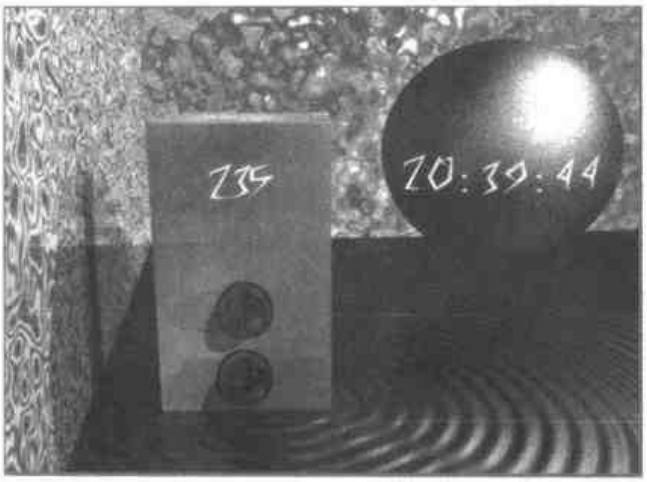


图 9.16　运行测试

9.7　小结

除了机械器件，这一章中的材料看起来似乎不适易使用。但是，我们没有理由认为，一个没有真正的前端面板的系统的信息就不能用抽象的界面来表示。如果这章中的技术被综合使用，就能为使用，创建出器件的展示是一件能给你带来满足感的事。

第 10 章　绘制点和标尺

本章包含一些创建绘图工具和用户界面的技术，并允许用户在 GUI 中创建和移动对象。但并不意味着这章就是完整的开发新颖的界面的指南，我只是要向大家介绍一些有用的模板和如何利用标尺以及如何在画布上重排对象。在前面的章节中已经讲过一些有关创建图形对象的方法，本章将讲述得更加详细。

本章中的一些例子是 Tkinter 实例的改写本，它们可以用于指导描述 Tcl/Tk。作者没有减少重复是为了有对比地体现 Tk 的优越性。

10.1　在画布上绘图

我们实际上已经接触过许多在画布上绘图的例子，但是这些程序所绘制的都是显示在前端的物理器件。现在我们来学习图和将图形绘制到画布上。

所有的绘图程序都要先定义包含所绘对象的界限框。它显示的是对象的左上角和右下角的坐标数组。线条则是一种特殊的对象，它由 start（头）和 end （尾）决定位置，同样也显示在对应的界限框里，它始终是一个左上角及右下角坐标的数组。值得注意的是：Tk 并不保证界限框精确界定对象，所以在要求比较严格的程序中应作出一些规定。这在图 10.1 中可以看到。

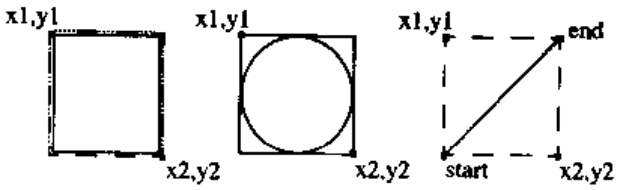


图 10.1　多边形、弧形和直线的外边界

曲线被视为由一系列的直线组成，它们有各自的边界框。因为在许多地方将会用到它们，所以务必引起重视。

让我们从一个简单例子开始吧！这个例子来自于 Douglas A.Young's 的 *The X Windows System: Programming and Applications with Xt*。本例可以绘制直线、矩形和椭圆并可任意选中它们。原先的程序是在 X Windows 下用 C 语言编写的，所以我们要先将它“Tk 化”。这无疑犹如用坚硬的铅笔在柔软的纸巾上画画一样——有一点麻烦。

draw.py

```
from  Tkinter import *
```

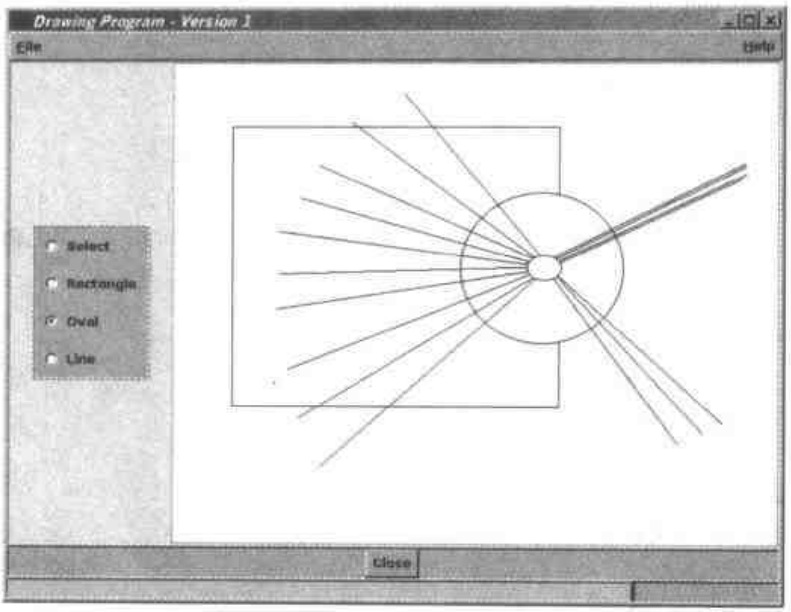


图 10.2　一个很简单的画图程序

```
import Pmw, AppShell, math
class Draw(AppShell.AppShell):
   usecommandarea = 1
   appname        = 'Drawing Program - Version 1'
   frameWidth     = 800
   frameHeight    = 600

   def createButtons(self):
        self.buttonAdd('Close', helpMessage='Close Screen',
                  statusMessage='Exit', command=self.close)

   def createBase(self):
        self.width  = self.root.winfo_width()-10
        self.height = self.root.winfo_height()-95
        self.command= self.createcomponent('command', (), None,
             Frame, (self.interior(),), width=self.width*0.25,
             height=self.height, background="gray90")
        self.command.pack(side=LEFT, expand=YES, fill=BOTH)

        self.canvas = self.createcomponent('canvas', (), None,
             Canvas, (self.interior(),), width=self.width*0.73,
             height=self.height, background="white")
        self.canvas.pack(side=LEFT, expand=YES, fill=BOTH)

    Widget.bind(self.canvas, "<Button-1>", self.mouseDown)
        Widget.bind(self.canvas, "<Button1-Motion>", self.mouseMotion)
        Widget.bind(self.canvas, "<Button1-ButtonRelease>", self.mouseUp)

       self.radio = Pmw.RadioSelect(self.command, labelpos = None,
```

```
            buttontype = 'radiobutton', orient = 'vertical',
            command = self.selectFunc, hull_borderwidth = 2,
            hull_relief = 'ridge',)
        self.radio.pack(side = TOP, expand = 1)

        self.func = {}
            for text, func in (('Select',    None),
                       ('Rectangle', self.drawRect),
                       ('Oval', self.drawOval),
                       ('Line', self.drawLine)):
             self.radio.add(text)
             self.func[text] = func
         self.radio.invoke('Rectangle')

    def selectFunc(self, tag):
        self.currentFunc = self.func[tag]

    def mouseDown(self, event):
        self.currentObject = None
        self.lastx = self.startx = self.canvas.canvasx(event.x)
        self.lasty = self.starty = self.canvas.canvasy(event.y)
        if not self.currentFunc:
            self.selObj = self.canvas.find_closest(self.startx,
                                        self.starty)[0]
            self.canvas.itemconfig(self.selObj, width=2)
            self.canvas.lift(self.selObj)

    def mouseMotion(self, event):
        self.lastx = self.canvas.canvasx(event.x)
        self.lasty = self.canvas.canvasy(event.y)
        if self.currentFunc:
            self.canvas.delete(self.currentObject)
            self.currentFunc(self.startx, self.starty,
                       self.lastx, self.lasty,
                       self.foreground, self.background)

    def mouseUp(self, event):
        self.lastx = self.canvas.canvasx(event.x)
        self.lasty = self.canvas.canvasy(event.y)
        self.canvas.delete(self.currentObject)
        self.currentObject = None
        if self.currentFunc:
            self.currentFunc(self.startx, self.starty,
                       self.lastx, self.lasty,
                       self.foreground, self.background)
        else:
            if self.selObj:
                self.canvas.itemconfig(self.selObj, width=1)
```

```
    def drawLine(self, x, y, x2, y2, fg, bg):
        self.currentObject = self.canvas.create_line(x,y,x2,y2,
                                                     fill=fg)

    def drawRect(self, x, y, x2, y2, fg, bg):
        self.currentObject = self.canvas.create_rectangle(x, y,
                                 x2, y2, outline=fg, fill=bg)

    def drawOval(self, x, y, x2, y2, fg, bg):
        self.currentObject = self.canvas.create_oval(x, y, x2, y2,
                             outline=fg, fill=bg)

    def initData(self):
        self.currentFunc   = None
        self.currentObject = None
        self.selObj    = None
        self.foreground    = 'black'
        self.background    = 'white'

    def close(self):
        self.quit()

    def createInterface(self):
        AppShell.AppShell.createInterface(self)
        self.createButtons()
        self.initData()
        self.createBase()

if __name__ == '__main__':
    draw = Draw()
    draw.run()
```

Draw.py 注解

❶ 本例是一个指示驱动,要依赖于将函数调用捆绑在鼠标的事件上才能工作。我们将点击、移动和松开几个事件与相应的函数绑在一起。

```
Widget.bind(self.canvas, "<Button-1>", self.mouseDown)
Widget.bind(self.canvas, "<Button1-Motion>", self.mouseMotion)
Widget.bind(self.canvas, "<Button1-ButtonRelease>", self.mouseUp)
```

❷ 这个简单的例子支持了三种简单的图形。创建 **Pmw.RadioSelect** 按钮来连接每一个图形和相应的绘图函数。另外，我们还定义一个选择项可以实现用鼠标直接点击画布。

❸ MouseDown 方法没有选中任何当前的可选对象，事件返回鼠标点击处的 x/y 坐标作为屏幕的坐标。再由画布控件的 canvasx 和 canvasy 方法将这组值与画布的相应值联系起来。

```
    def mouseDown(self, event):
        self.currentObject = None
        self.lastx = self.startx = self.canvas.canvasx(event.x)
```

```
self.lasty = self.starty = self.canvas.canvasy(event.y)
```

这种转换往往容易在初学编程的时候忘记。见图 10.3

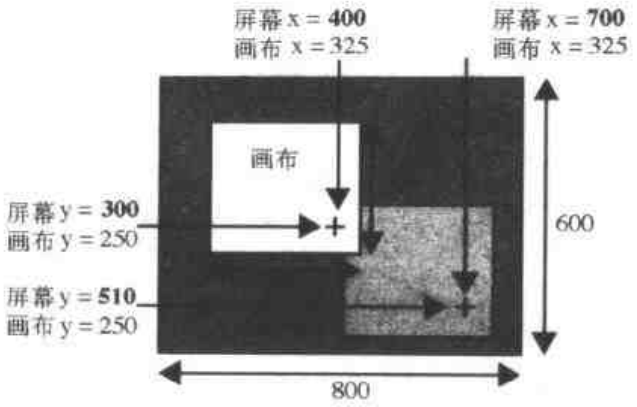


图 10.3　屏幕与画布坐标的关系

当用户点击画布时，立即返回点击处的坐标到事件中去，转换坐标的程序将它转换成与屏幕相对应的值，从而确定画布在什么地方。

❽ 如果没有选中任何一个图形就一直处于选择状态，并且区域选择最邻近的对象。这种方法可能不是对所有的应用程序都适用，因为它无论如何也要选中一个对象，不管你是不是点击了鼠标。这种情况会导致出错，所以最好将其设置为点击鼠标选中的状态。

```
if not self.currentFunc:
    self.selObj = self.canvas.find_closest(self.startx,
                                            self.starty)[0]
    self.canvas.itemconfig(self.selObj, width=2)
    self.canvas.lift(self.selObj)
```

在选择了所要的对象之后，加粗轮廓线并将它拖至绘图区。如图 10.4。

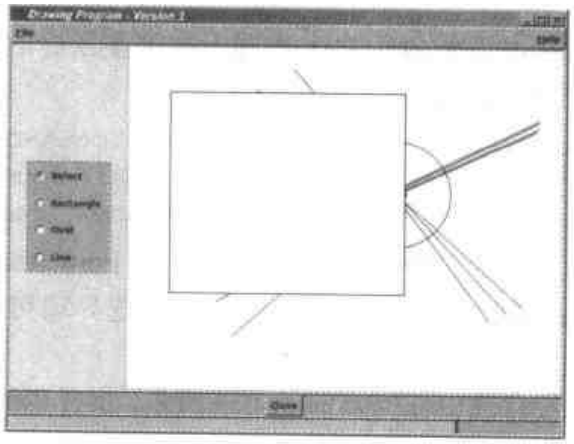


图 10.4　选择画布中的一个对象

❾ 每移动一次鼠标都将处理一系列的移动事件。每一次处理都将改变对象的边界框。在转换了 x，y 坐标之后，删除原有的图形，并用新函数和边界框来重绘。

```
self.canvas.delete(self.currentObject)
self.currentFunc(self.startx, self.starty,
          self.lastx, self.lasty,
          self.foreground, self.background)
```

❻ 绘图的方法很简单，只要在边界框中建立简单的画布就可以了。

```
def drawLine(self, x, y, x2, y2, fg, bg):
     self.currentObject = self.canvas.create_line(x,y,x2,y2,
                                        fill=fg)
```

10.1.1　移动画布对象

选中对象可以将它显示在最前面。但是如果选中一个较大的对象之后怎样才能不会影响其他小图片呢？所以我们还要提供一些方法可以巧妙地处理对象。有代表性的是用移动鼠标来拖曳对象。把这个功能加到程序中时非常简单的。下面是要添加到程序中的定义过程。

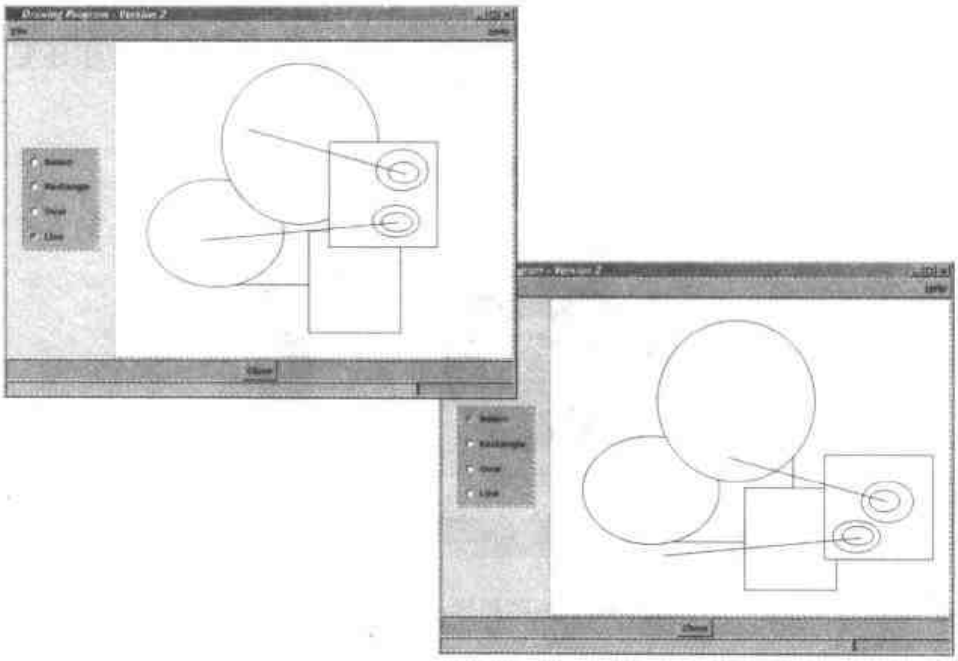


图 10.5　在画布中移动对象

Draw2.py

```
def mouseMotion(self, event):
     cx = self.canvas.canvasx(event.x)
     cy = self.canvas.canvasy(event.y)
     if self.currentFunc:
          self.lastx = cx
          self.lasty = cy
          self.canvas.delete(self.currentObject)
          self.currentFunc(self.startx, self.starty,
                    self.lastx, self.lasty,
```

❶

❶

```
                    self.foreground, self.background)
        else:
            if self.selObj:
                self.canvas.move(self.selObj, cx-self.lastx,
                                 cy-self.lasty)
                self.lastx = cx
                self.lasty = cy
```

Draw2.py 注解

❶ 将 x，y 坐标存储在中间变量中，因为我们要确定与前一个位置相比鼠标移动了多少距离。

❷ 在绘图的时候将此 x，y 坐标作为第二个数组存放在边界框中。

❸ 在移动图形的时候，计算当前的位置与原先的位置的差别。

10.2 一个更一个更完整的绘图程序

前面的例子虽然给出了一些基本的方法，但一个真正功能完善的程序还要有更多的内容，我们首先来看看有哪些东西是要补充的。

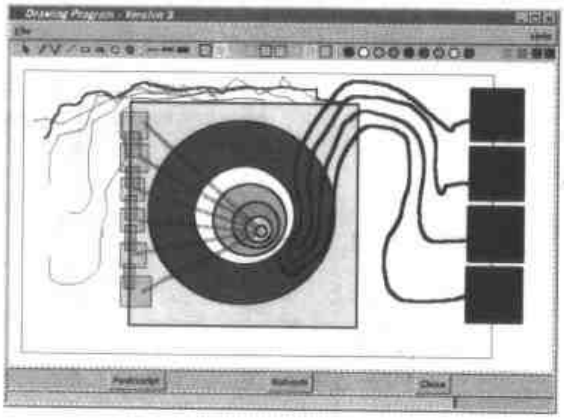


图 10.6 画图程序：扩展特性

1. 工具栏给出几种图形绘制工具和选择工具。

- 绘图工具如：自由曲线工具、平滑曲线工具、直线工具、矩形工具（填充/非填充）和椭圆工具（填充/非填充）。
- 设定线条或轮廓的颜色。
- 设定线条和轮廓的宽度。
- 设定填充色。
- 数量有限的透明遮罩。

2. 按下 Shift 键的同时使用矩形和椭圆工具将绘出正方形和圆。
3. 记录当前画布内容的选项。
4. 更新画布选项。
5. 帮助信息。

以下是具体的程序。

Draw3.py

```
from  Tkinter import *
import Pmw, AppShell, math, time, string

class ToolBarButton(Label):
   def __init__(self, top, parent, tag=None, image=None, command=None,
                statushelp='', balloonhelp='', height=21, width=21,
                bd=1, activebackground='lightgrey', padx=0, pady=0,
                state='normal', bg='grey'):
        Label.__init__(self, parent, height=height, width=width,
                       relief='flat', bd=bd, bg=bg)
        self.bg = bg
        self.activebackground = activebackground
        if image != None:
             if string.splitfields(image, '.')[1] == 'bmp':
                  self.Icon = BitmapImage(file='icons/%s' % image)
             else:
                  self.Icon = PhotoImage(file='icons/%s' % image)
        else:
              self.Icon = PhotoImage(file='icons/blank.gif')
        self.config(image=self.Icon)

        self.tag = tag
        self.icommand = command
        self.command  = self.activate
        self.bind("<Enter>",          self.buttonEnter)
        self.bind("<Leave>",          self.buttonLeave)
        self.bind("<ButtonPress-1>",   self.buttonDown)
        self.bind("<ButtonRelease-1>", self.buttonUp)
        self.pack(side='left', anchor=NW, padx=padx, pady=pady)

        if balloonhelp or statushelp:
            top.balloon().bind(self, balloonhelp, statushelp)
            self.state = state

   def activate(self):
        self.icommand(self.tag)

   def buttonEnter(self, event):
        if self.state != 'disabled':
             self.config(relief='raised', bg=self.bg)

   def buttonLeave(self, event):
        if self.state != 'disabled':
             self.config(relief='flat', bg=self.bg)
```

```
    def buttonDown(self, event):
        if self.state != 'disabled':
            self.config(relief='sunken', bg=self.activebackground)

    def buttonUp(self, event):
        if self.state != 'disabled':
            if self.command != None:
                self.command()
            time.sleep(0.05)
            self.config(relief='flat', bg=self.bg)

class Draw(AppShell.AppShell):
    usecommandarea = 1
    appname        = 'Drawing Program - Version 3'
    frameWidth     = 840
    frameHeight    = 600

    def createButtons(self):
        self.buttonAdd('Postscript',
                helpMessage='Save current drawing (as PostScript)',
                statusMessage='Save drawing as PostScript file',
                command=self.ipostscript)
        self.buttonAdd('Refresh', helpMessage='Refresh drawing',
                statusMessage='Redraw the screen', command=self.redraw)
        self.buttonAdd('Close', helpMessage='Close Screen',
                statusMessage='Exit', command=self.close)

    def createBase(self):
        self.toolbar = self.createcomponent('toolbar', (), None,
                Frame, (self.interior(),), background="gray90")
        self.toolbar.pack(fill=X)

        self.canvas = self.createcomponent('canvas', (), None,
                Canvas, (self.interior(),), background="white")
        self.canvas.pack(side=LEFT, expand=YES, fill=BOTH)

        Widget.bind(self.canvas, "<Button-1>", self.mouseDown)
        Widget.bind(self.canvas, "<Button1-Motion>", self.mouseMotion)
        Widget.bind(self.canvas, "<Button1-ButtonRelease>", self.mouseUp)
        self.root.bind("<KeyPress>", self.setRegular)
        self.root.bind("<KeyRelease>", self.setRegular)

    def setRegular(self, event):
        if event.type == '2' and event.keysym == 'Shift_L':
            self.regular = TRUE
        else:
            self.regular = FALSE
```

```
    def createTools(self):
        self.func = {}
        ToolBarButton(self, self.toolbar, 'sep', 'sep.gif',
                      width=10, state='disabled')
        for key, func, balloon in [
            ('pointer', None,                'Edit drawing'),
            ('draw',    self.drawFree,       'Draw freehand'),
            ('smooth',  self.drawSmooth,     'Smooth freehand'),
            ('line',    self.drawLine,       'Rubber line'),
            ('rect',    self.drawRect,       'Unfilled rectangle'),
            ('frect',   self.drawFilledRect, 'Filled rectangle'),
            ('oval',    self.drawOval,       'Unfilled oval'),
            ('foval',   self.drawFilledOval, 'Filled oval')]:
            ToolBarButton(self, self.toolbar, key, '%s.gif' % key,
                          command=self.selectFunc, balloonhelp=balloon,
                          statushelp=balloon)
            self.func[key] = func

    def createLineWidths(self):
        ToolBarButton(self, self.toolbar, 'sep', 'sep.gif', width=10,
                      state='disabled')
        for width in ['1', '3', '5']:
            ToolBarButton(self, self.toolbar, width, 'tline%s.gif' % \
                          width, command=self.selectWidth,
                          balloonhelp='%s pixel linewidth' % width,
                          statushelp='%s pixel linewidth' % width)

    def createLineColors(self):
    def createFillColors(self):
    def createPatterns(self):

# --- Code Removed ----------------------------------------------
```

❹

❺

Draw3.py 注解

❶ 工具栏按钮使用的是简单的图标按钮，具体可以使用位图或其他。

❷ 在按钮上绑定标签使得当鼠标停留在按钮上时出现生动明了的提示和说明。

❸ 绘制正方形和圆与 KeyPress 事件相联系，当得到返回值的时候，检查 Shift 键是否按下并设置标志位。

```
        self.root.bind("<KeyPress>", self.setRegular)
        self.root.bind("<KeyRelease>", self.setRegular)
    def setRegular(self, event):
        if event.type == '2' and event.keysym == 'Shift_L':
            self.regular = TRUE
```

❹ 为多样的对象类型建立一个任务表，填充显示名称、函数和帮助信息文本。

❺ 创建工具栏按钮的程序基本上是一致的故不再一一详举。

Draw3.py（续）

```
    def selectFunc(self, tag):
        self.curFunc = self.func[tag]
        if self.curFunc:
            self.canvas.config(cursor='crosshair')
        else:
            self.canvas.config(cursor='arrow')

    def selectWidth(self, tag):
    def selectBackground(self, tag):
    def selectForeground(self, tag):
    def selectPattern(self, tag):

# --- Code Removed ----------------------------------------------

    def mouseDown(self, event):
        self.curObject = None
        self.canvas.dtag('drawing')
        self.lineData = []
        self.lastx = self.startx = self.canvas.canvasx(event.x)
        self.lasty = self.starty = self.canvas.canvasy(event.y)
        if not self.curFunc:
            self.selObj = self.canvas.find_closest(self.startx,
                                        self.starty)[0]
            self.savedWidth = string.atoi(self.canvas.itemcget( \
                        self.selObj, 'width'))
            self.canvas.itemconfig(self.selObj,
                            width=self.savedWidth + 2)
            self.canvas.lift(self.selObj)

    def mouseMotion(self, event):
        curx = self.canvas.canvasx(event.x)
        cury = self.canvas.canvasy(event.y)
        prevx = self.lastx
        prevy = self.lasty
        if self.curFunc:
            self.lastx = curx
            self.lasty = cury

        if self.regular and self.canvas.type('drawing') in \
           ['oval','rectangle']:
            dx        = self.lastx - self.startx
            dy        = self.lasty - self.starty
            delta = max(dx, dy)
            self.lastx = self.startx + delta
            self.lasty = self.starty + delta
            self.curFunc(self.startx, self.starty, self.lastx,
                 self.lasty, prevx, prevy, self.foreground,
                 self.background, self.fillStyle, self.lineWidth,None)
```

```
        else:
            if self.selObj:
                self.canvas.move(self.selObj, curx-prevx, cury-prevy)
                self.lastx = curx
                self.lasty = cury

    def mouseUp(self, event):
        self.prevx = self.lastx
        self.prevy = self.lasty
        self.lastx = self.canvas.canvasx(event.x)
        self.lasty = self.canvas.canvasy(event.y)

        if self.curFunc:
            if self.regular and self.canvas.type('drawing') in \
              ['oval','rectangle']:
                dx       = self.lastx - self.startx
                dy       = self.lasty - self.starty
                delta = max(dx, dy)
                self.lastx = self.startx + delta
                self.lasty = self.starty + delta
            self.curFunc(self.startx, self.starty, self.lastx,
                    self.lasty, self.prevx, self.prevy, self.foreground,
                    self.background, self.fillStyle, self.lineWidth,
                    self.lineData)

            self.storeObject()
        else:
            if self.selObj:
                self.canvas.itemconfig(self.selObj,
                                       width=self.savedWidth)

    def drawLine(self,x,y,x2,y2,x3,y3,fg,bg,fillp,wid,ld):
        self.canvas.delete(self.curObject)
        self.curObject =  self.canvas.create_line(x,y,x2,y2,fill=fg,
                  tags='drawing',stipple=fillp,width=wid)

    def drawFree(self,x,y,x2,y2,x3,y3,fg,bg,fillp,wid,ld):
        self.drawFreeSmooth(x,y,x2,y2,x3,y3,FALSE,fg,bg,fillp,wid,ld)

    def drawSmooth(self,x,y,x2,y2,x3,y3,fg,bg,fillp,wid, ld):
        self.drawFreeSmooth(x,y,x2,y2,x3,y3,TRUE,fg,bg,fillp,wid,ld)

    def drawFreeSmooth(self,x,y,x2,y2,x3,y3,smooth,fg,bg,fillp,
                       wid,ld):
        if not ld:
            for coord in [[x3, y3, x2, y2], [x2, y2]][smooth]:
                self.lineData.append(coord)
                ild = self.lineData
        else:
            ild = ld
```

❼

❽

❾

```
        if len(ild) > 2:
            self.curObject = self.canvas.create_line(ild, fill=fg,
                stipple=fillp, tags='drawing', width=wid, smooth=smooth)

    def drawRect(self,x,y,x2,y2,x3,y3,fg,bg,fillp,wid,ld):
        self.drawFilledRect(x,y,x2,y2,x3,y3,fg,'',fillp,wid,ld)

    def drawFilledRect(self,x,y,x2,y2,x3,y3,fg,bg,fillp,wid,ld):
        self.canvas.delete(self.curObject)
        self.curObject =  self.canvas.create_rectangle(x,y,x2,y2,
                outline=fg, tags='drawing',fill=bg,
                stipple=fillp,width=wid)

    def drawOval(self,x,y,x2,y2,x3,y3,fg,bg,fillp,wid,ld):
        self.drawFilledOval(x,y,x2,y2,x3,y3,fg,'',fillp,wid,ld)

    def drawFilledOval(self,x,y,x2,y2,x3,y3,fg,bg,fillp,wid,ld):
        self.canvas.delete(self.curObject)
        self.curObject = self.canvas.create_oval(x,y,x2,y2,outline=fg,
                fill=bg,tags='drawing',stipple=fillp,width=wid)
```

代码注解（续）

❽ 每一次选择的返回值都用一个与工具栏按钮相关联的标志值(tag)来查找相应的函数、线条宽度及其他的按钮属性。

```
def selectFunc(self, tag):
    self.curFunc = self.func[tag]
    if self.curFunc:
        self.canvas.config(cursor='crosshair')
    else:
        self.canvas.config(cursor='arrow')
```

这里还是用了指针。

❾ 鼠标的反应与前面两个例子相同。

❿ 如果设置了恰当的值，本例也可绘制正方形、圆角矩形和椭圆。

⓫ Draw 方法与前面的例子非常相似，是将对象的属性存为一个列表实现的。

Draw3.py 列表

```
def storeObject(self):
    self.objects.append(( self.startx, self.starty, self.lastx,
            self.lasty, self.prevx,  self.prevy, self.curFunc,
            self.foreground, self.background, self.fillStyle,
            self.lineWidth, self.lineData ))

def redraw(self):
    self.canvas.delete(ALL)
    for startx, starty, lastx, lasty, prevx, prevy, func, \
        fg, bg, fill, lwid, ld,  in self.objects:
        self.curObject = None
```

```
                func(startx, starty, lastx, lasty, prevx, prevy,
                    fg, bg, fill, lwid, ld)

    def initData(self):
        self.curFunc        = self.drawLine
        self.curObject      = None
        self.selObj         = None
        self.lineData       = []
        self.savedWidth     = 1
        self.objects        = []
        self.foreground     = 'black'
        self.background     = 'white'
        self.fillStyle      = None
        self.lineWidth      = 1
        self.regular        = FALSE

    def ipostscript(self):
        postscript = self.canvas.postscript()
        fd = open('drawing.ps', 'w')
        fd.write(postscript)
        fd.close()

    def close(self):
        self.quit()

    def createInterface(self):
        AppShell.AppShell.createInterface(self)
        self.createButtons()
        self.initData()
        self.createBase()
        self.createTools()
        self.createLineWidths()
        self.createLineColors()
        self.createFillColors()
        self.createPatterns()

if __name__ == '__main__':
    draw = Draw()
    draw.run()
```

代码注解（续）

❿ StoreObject 的目的是将对对象的描述按照它们创建的顺序存放在列表中以便对象可以按正确的顺序更新。

● Redraw 删除所有的对象并按原样重绘。

● Tk 的一大优点就是可以自己进行描述，从而可以将它保存成为文件和图片，以便打印或者用适当的软件进行浏览。

10.3 滚动画布

通常情况下，图形会比画布大，要想使画布扩大就要创建滚动的画布。手动的滚动条如 X Window 系统中的滚动条，所用的代码比较适当。Tkinter 则可以使滚动的功能对代码的依赖小一些。请看下面这个由 Tkinter 源程序直接得到的例子。

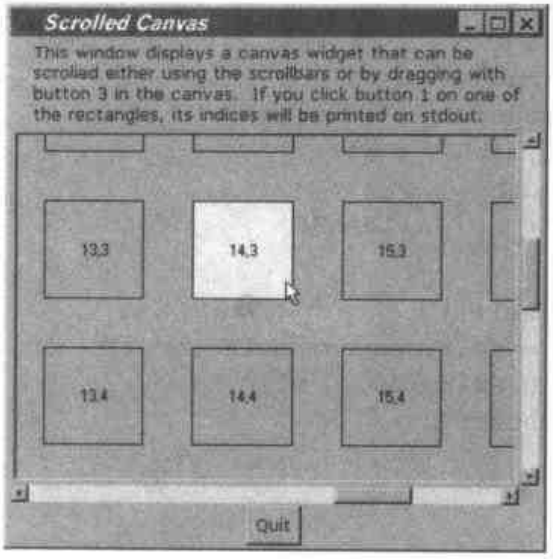


图 10.7 管理滚动画布

cscroll.py

```
from Tkinter import *

class ScrolledCanvas:
   def __init__(self, master, width=500, height=350):
       Label(master, text="This window displays a canvas widget "
             "that can be scrolled either using the scrollbars or "
             "by dragging with button 3 in the canvas.  If you "
             "click button 1 on one of the rectangles, its indices "
             "will be printed on stdout.",
             wraplength="4i", justify=LEFT).pack(side=TOP)
       self.control=Frame(master)
       self.control.pack(side=BOTTOM, fill=X, padx=2)
       Button(self.control, text='Quit', command=master.quit).pack()

       self.grid = Frame(master)
       self.canvas = Canvas(master, relief=SUNKEN, borderwidth=2,
                            scrollregion=('-11c','-11c','50c','20c'))
       self.hscroll = Scrollbar(master, orient=HORIZONTAL,
                                command=self.canvas.xview)
       self.vscroll = Scrollbar(master, command=self.canvas.yview)
```

```
        self.canvas.configure(xscrollcommand=self.hscroll.set,
                              yscrollcommand=self.vscroll.set)

        self.grid.pack(expand=YES, fill=BOTH, padx=1, pady=1)
        self.grid.rowconfigure(0, weight=1, minsize=0)
        self.grid.columnconfigure(0, weight=1, minsize=0)
        self.canvas.grid(padx=1, in_=self.grid, pady=1, row=0,
                  column=0, rowspan=1, columnspan=1, sticky='news')
        self.vscroll.grid(padx=1, in_=self.grid, pady=1, row=0,
                  column=1, rowspan=1, columnspan=1, sticky='news')
        self.hscroll.grid(padx=1, in_=self.grid, pady=1, row=1,
                  column=0, rowspan=1, columnspan=1, sticky='news')
        self.oldFill = None

        bg = self.canvas['background']
        for i in range(20):
            x = -10 + 3*i
            y = -10
            for j in range(10):
                self.canvas.create_rectangle('%dc'%x, '%dc'%y,
                          '%dc'%(x+2), '%dc'%(y+2), outline='black',
                          fill=bg, tags='rect')
                self.canvas.create_text('%dc'%(x+1), '%dc'%(y+1),
                              text='%d,%d'%(i,j), anchor=CENTER,
                              tags=('text', 'rect'))
                y = y + 3

        self.canvas.tag_bind('rect', '<Any-Enter>', self.scrollEnter)
        self.canvas.tag_bind('rect', '<Any-Leave>', self.scrollLeave)
        self.canvas.bind_all('<1>', self.scrollButton)
        self.canvas.bind('<3>',
                  lambda e, s=self: s.canvas.scan_mark(e.x, e.y))
        self.canvas.bind('<B3-Motion>',
                  lambda e, s=self: s.canvas.scan_dragto(e.x, e.y))

    def scrollEnter(self, event):
        id = self.canvas.find_withtag(CURRENT)[0]
        if 'text' in self.canvas.gettags(CURRENT):
            id = id-1
        self.canvas.itemconfigure(id, fill='SeaGreen1')

    def scrollLeave(self, event):
        id = self.canvas.find_withtag(CURRENT)[0]
        if 'text' in self.canvas.gettags(CURRENT):
            id = id-1
        self.canvas.itemconfigure(id, fill=self.canvas['background'])
```

```
    def scrollButton(self, event):
        ids = self.canvas.find_withtag(CURRENT)
        if ids:
            id = ids[0]
            if not 'text' in self.canvas.gettags(CURRENT):
                id = id+1
            print 'You clicked on %s' % \
                  self.canvas.itemcget(id, 'text')

if __name__ == '__main__':
    root = Tk()
    root.option_add('*Font', 'Verdana 10')
    root.title('Scrolled Canvas')
    scroll = ScrolledCanvas(root)
    root.mainloop()
```

❹

Cscroll.py 注解

❶ 创建一个 61cm×31cm 的对象，显然它不可能显示在 500 像素×350 像素这样小的画布上。水平滚动条和垂直滚动条直接由画布的 position （位置）方法创建和绑定。

```
self.canvas = Canvas(master, relief=SUNKEN, borderwidth=2,
              scrollregion=('-11c','-11c','50c','20c'))
self.hscroll = Scrollbar(master, orient=HORIZONTAL,
               command=self.canvas.xview)
self.vscroll = Scrollbar(master, command=self.canvas.yview)
```

❷ 滚动条设置为可以追踪画布的移动。

```
self.canvas.configure(xscrollcommand=self.hscroll.set,
                      yscrollcommand=self.vscroll.set)
```

❸ can_mark 和 scan_dragto 两种方法绑定在鼠标点击上就可以很轻松地实现按住鼠标右键并拖拽的功能。

```
self.canvas.bind('<3>',
          lambda e, s=self: s.canvas.scan_mark(e.x, e.y))
self.canvas.bind('<B3 Motion>',
          lambda e, s=self: s.canvas.scan_dragto(e.x, e.y))
```

❹ 最后，ScrollButton 回调是值得注意的，它描述了使用标志值定义对象的容易程度。

```
ids = self.canvas.find_withtag(CURRENT)
if ids:
    id = ids[0]
    if not 'text' in self.canvas.gettags(CURRENT):
        id = id+1
    print 'You clicked on %s' % \
          self.canvas.itemcget(id, 'text')
```

首先，我们找出所有有 CURRENT 标志的 ids（这会是一个小方格或者在中心有一个文字标记）。只需注意第一个标志。

然后，看看它是不是文本对象，如果不是那么下一个 ids 就是文本，因为我们是先

定义的矩形对象。

最后，得到文本对象的内容，它给出了行列的数组。

10.4　标尺工具

另一个有用的绘图工具是标尺，它可以用来显示停止或其他的强制图标，它还可以使拖拽产生更直观位置变换的效果。这个例子也是由 Tk 源代码而来。

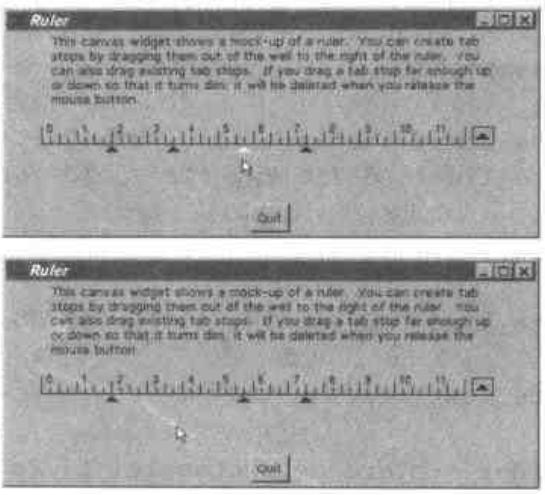


图 10.8　简单标尺工具

ruler.py

```
from Tkinter import *

class Ruler:
    def __init__(self, master, width='14.8c', height='2.5c'):
        Label(master, text="This canvas widget shows a mock-up of a "
              "ruler.  You can create tab stops by dragging them out "
              "of the well to the right of the ruler.  You can also "
              "drag existing tab stops.  If you drag a tab stop far "
              "enough up or down so that it turns dim, it will be "
              "deleted when you release the mouse button.",
              wraplength="5i", justify=LEFT).pack(side=TOP)
        self.ctl=Frame(master)
        self.ctl.pack(side=BOTTOM, fill=X, padx=2, pady=2)
        Button(self.ctl, text='Quit', command=master.quit).pack()
        self.canvas = Canvas(master, width=width, height=height,
                             relief=FLAT, borderwidth=2)
        self.canvas.pack(side=TOP, fill=X)

        c = self.canvas
        self.grid  = '0.25c'
        self.left  = c.winfo_fpixels('1c')
        self.right  = c.winfo_fpixels('13c')
```

```
        self.top  = c.winfo_fpixels('1c')
        self.bottom  = c.winfo_fpixels('1.5c')
        self.size  = c.winfo_fpixels('.2c')
        self.normalStyle   = 'black'
        self.activeStyle   = 'green'
        self.activeStipple = ''
        self.deleteStyle   = 'red'
        self.deleteStipple = 'gray25'

        c.create_line('1c', '0.5c', '1c', '1c', '13c', '1c',
                  '13c', '0.5c', width=1)
        for i in range(12):
            x = i+1
            c.create_line('%dc'%x, '1c', '%dc'%x, '0.6c', width=1)
            c.create_line('%d.25c'%x, '1c', '%d.25c'%x,
                      '0.8c', width=1)
            c.create_line('%d.5c'%x, '1c', '%d.5c'%x,
                      '0.7c', width=1)
            c.create_line('%d.75c'%x, '1c', '%d.75c'%x,
                      '0.8c', width=1)
            c.create_text('%d.15c'%x, '.75c', text=i, anchor=SW)

            wellBorder = c.create_rectangle('13.2c', '1c', '13.8c',
                                '0.5c', outline='black',
                                fill=self.canvas['background'])
            wellTab = self.mkTab(c.winfo_pixels('13.5c'),
                  c.winfo_pixels('.65c'))
            c.addtag_withtag('well', wellBorder)
            c.addtag_withtag('well', wellTab)

            c.tag_bind('well', '<1>',
                   lambda e, s=self: s.newTab(e.x, e.y))
            c.tag_bind('tab',  '<1>',
                   lambda e, s=self: s.selectTab(e.x, e.y))
            c.bind('<B1-Motion>',
                  lambda e, s=self: s.moveTab(e.x, e.y))
            c.bind('<Any-ButtonRelease-1>', self.releaseTab)

    def mkTab(self, x, y):
        return self.canvas.create_polygon(x, y, x+self.size,
                              y+self.size, x-self.size, y+self.size)

    def newTab(self, x, y):
        newTab = self.mkTab(x, y)
        self.canvas.addtag_withtag('active', newTab)
        self.canvas.addtag_withtag('tab', newTab)
        self.x = x
        self.y = y
        self.moveTab(x, y)
```

```
    def selectTab(self, x, y):
        self.x = self.canvas.canvasx(x, self.grid)
        self.y = self.top + 2
        self.canvas.addtag_withtag('active', CURRENT)
        self.canvas.itemconfig('active', fill=self.activeStyle,
                               stipple=self.activeStipple)
        self.canvas.lift('active')

    def moveTab(self, x, y):
        tags = self.canvas.find_withtag('active')                    ❹
        if not tags: return
        cx = self.canvas.canvasx(x, self.grid)                       ❶
        cy = self.canvas.canvasx(y)
        if cx < self.left:
            cx = self.left
        if cx > self.right:
            cx = self.right
        if cy >= self.top and cy <= self.bottom:
            cy = self.top+2
            self.canvas.itemconfig('active', fill=self.activeStyle,  ❺
                                   stipple=self.activeStipple)
        else:
            cy = cy-self.size-2
            self.canvas.itemconfig('active', fill=self.deleteStyle,  ❻
                                   stipple=self.deleteStipple)
        self.canvas.move('active', cx-self.x, cy-self.y)
        self.x = cx
        self.y = cy

    def releaseTab(self, event):
        tags = self.canvas.find_withtag('active')                    ❼
        if not tags: return
        if self.y != self.top+2:                                     ❽
            self.canvas.delete('active')
        else:
            self.canvas.itemconfig('active', fill=self.normalStyle,  ❿
                                   stipple=self.activeStipple)
            self.canvas.dtag('active')

if __name__ == '__main__':
    root = Tk()
    root.option_add('*Font', 'Verdana 10')
    root.title('Ruler')
    ruler = Ruler(root)
    root.mainloop()
```

Ruler.py 注解

❶ 指出如何在 Tkinter 允许的显示范围内指定对象的尺寸，并转换成为路径（此时的数据类型为浮点型）。

```
self.right  = c.winfo_fpixels('13c')
```

❷ 类似地，我们可以用绝对尺度尺寸来创建对象（一般用 cm 为单位），这在直接用支持绘图时很有用处。

```
c.create_line('1c', '0.5c', '1c', '1c', '13c', '1c',
              '13c', '0.5c', width=1)
```

❸ Tkinter 有时隐藏了加下划线的函数的功能，在这个例子中我们为新创建的对象 newTab 加入标签、动作和标志值。

```
newTab = self.mkTab(x, y)
self.canvas.addtag_withtag('active', newTab)
self.canvas.addtag_withtag('tab', newTab)
```

addtag_withtag 方法隐藏了一点，那就是这里略过了 withtag 变量适用的标签和地址。

❹ moveTab 很有用处因为它必须支持用户的创建和对已存在的对象进行修改和删除。此处如果不是要仿效 Ousterhout 的例子的话，我想程序会更简单一些。

❺ 标尺可将标签隔开，canvasx 方法含有可选择的变量，它定义画布上的对象如何被转换。

```
cx = self.canvas.canvasx(x, self.grid)
```

❻ 如果指针移动到标尺的前后两端之间，就从顶点所在之处将其分开，涂上不同的颜色。

```
cy = self.top+2
self.canvas.itemconfig('active', fill=self.activeStyle,
                                 stipple=self.activeStipple)
```

❼ 如果指针移出了前后两个端点，则进一步向前移动，并将它填充上别的颜色的点使其成为虚像。

```
cy = cy-self.size-2
self.canvas.itemconfig('active', fill=self.deleteStyle,
                                stipple=self.deleteStipple)
```

❽ 与移动标签一样，松开鼠标也对应一系列的动作。

❾ 一旦标签指明 deletion（删除），对象就被删除了。

```
if self.y != self.top+2:
    self.canvas.delete('active')
```

❿ 另外，用一般的颜色填充标签并删除 active 标志。

```
self.canvas.itemconfig('active', fill=self.normalStyle,
                  stipple=self.activeStipple)
self.canvas.dtag('active')
```

10.5 缩放画布对象

绘图程序的最普通的模块就是使已存在的对象变形，这要求可以让用户点击或拖拽以重定义对象大小的“手柄”。在讲述具体的方法之前，先来看一个简单的例子。这个小程序通过改变两个属性：width 和 arrowshape 来决定箭头的大小。你会发现这是一个很有用的工具，使你可以随心所欲地绘制箭头。

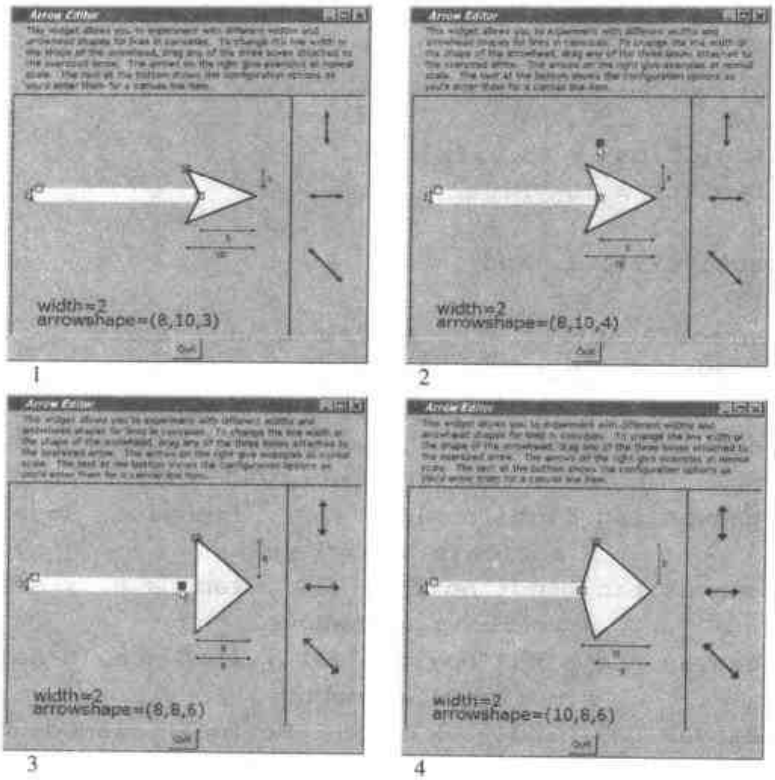


图 10.9　伸长画布对象

arrow.py

```
from Tkinter import *

class ArrowEditor:
   def __init__(self, master, width=500, height=350):
        Label(master, text="This widget allows you to experiment "
              "with different widths and arrowhead shapes for lines "
              "in canvases.  To change the line width or the shape "
              "of the arrowhead, drag any of the three boxes "
              "attached to the oversized arrow.  The arrows on the "
              "right give examples at normal scale.  The text at "
              "the bottom shows the configuration options as you'd "
              "enter them for a canvas line item.",
             wraplength="5i", justify=LEFT).pack(side=TOP)
        self.control=Frame(master)
        self.control.pack(side=BOTTOM, fill=X, padx=2)
        Button(self.control, text='Quit', command=master.quit).pack()
        self.canvas = Canvas(master, width=width, height=height,
                             relief=SUNKEN, borderwidth=2)
        self.canvas.pack(expand=YES, fill=BOTH)

        self.a      = 8      # Setup default values
        self.b      = 10
        self.c      = 3
```

```
        self.width = 2
        self.motionProc = None
        self.x1     = 40
        self.x2     = 350
        self.y      = 150
        self.smallTips  = (5,5,2)
        self.bigLine = 'SkyBlue2'
        self.boxFill = ''
        self.activeFill = 'red'

        self.arrowSetup()         # Draw default arrow
        self.canvas.tag_bind('box', '<Enter>', lambda e, s=self:
                    s.canvas.itemconfig(CURRENT, fill='red'))
        self.canvas.tag_bind('box', '<Leave>', lambda e, s=self:
                    s.canvas.itemconfig(CURRENT, fill=''))
        self.canvas.tag_bind('box1', '<1>', lambda e, s=self:
                    s.motion(s.arrowMove1))
        self.canvas.tag_bind('box2', '<1>', lambda e, s=self:
                    s.motion(s.arrowMove2)   )
        self.canvas.tag_bind('box3', '<1>', lambda e, s=self:
                    s.motion(s.arrowMove3))
        self.canvas.tag_bind('box', '<B1-Motion>', lambda e,
                    s=self: s.motionProc(e))
        self.canvas.bind('<Any-ButtonRelease-1>', lambda e,
                s=self: s.arrowSetup())

    def motion(self, func):
        self.motionProc = func

    def arrowMove1(self, event):
        newA = (self.x2+5-int(self.canvas.canvasx(event.x)))/10
        if newA < 0: newA = 0
        if newA > 25: newA = 25
        if newA != self.a:
            self.canvas.move("box1", 10*(self.a-newA), 0)
            self.a = newA

    def arrowMove2(self, event):
        newB = (self.x2+5-int(self.canvas.canvasx(event.x)))/10
        if newB < 0: newB = 0
        if newB > 25: newB = 25
        newC = (self.y+5-int(self.canvas.canvasx(event.y)+ \
                        5*self.width))/10
        if newC < 0: newC = 0
        if newC > 20: newC = 20
        if newB != self.b or newC != self.c:
            self.canvas.move("box2", 10*(self.b-newB),
                        10*(self.c-newC))
            self.b = newB
            self.c = newC
```

❶

❷

```
def arrowMove3(self, event):
    newW = (self.y+2-int(self.canvas.canvasx(event.y)))/5
    if newW < 0: newW = 0
    if newW > 20: newW = 20
    if newW != self.width:
        self.canvas.move("box3", 0, 5*(self.width-newW))
        self.width = newW

def arrowSetup(self):
    tags = self.canvas.gettags(CURRENT)
    cur = None
    if 'box' in tags:
        for tag in tags:
            if len(tag) == 4 and tag[:3] == 'box':
                cur = tag
                break
    self.canvas.delete(ALL)
    self.canvas.create_line(self.x1, self.y, self.x2, self.y,
                width=10*self.width,
                arrowshape=(10*self.a, 10*self.b, 10*self.c),
                arrow='last', fill=self.bigLine)
    xtip = self.x2-10*self.b
    deltaY = 10*self.c+5*self.width
    self.canvas.create_line(self.x2, self.y, xtip, self.y+deltaY,
                self.x2-10*self.a, self.y, xtip, self.y-deltaY,
                self.x2, self.y, width=2, capstyle='round',
                joinstyle='round')
    self.canvas.create_rectangle(self.x2-10*self.a-5, self.y-5,
                        self.x2-10*self.a+5, self.y+5,
                        fill=self.boxFill, outline='black',
                        tags=('box1', 'box'))
    self.canvas.create_rectangle(xtip-5, self.y-deltaY-5,
                        xtip+5, self.y-deltaY+5,
                        fill=self.boxFill, outline='black',
                        tags=('box2', 'box'))
    self.canvas.create_rectangle(self.x1-5,
                        self.y-5*self.width-5, self.x1+5,
                        self.y-5*self.width+5,  fill=self.boxFill,
                        outline='black', tags=('box3', 'box'))
    if cur:
        self.canvas.itemconfig(cur, fill=self.activeFill)
    self.canvas.create_line(self.x2+50, 0, self.x2+50,
              1000, width=2)

    tmp = self.x2+100
    self.canvas.create_line(tmp, self.y-125, tmp, self.y-75,
                width=self.width, arrow='both',
                arrowshape=(self.a, self.b, self.c))
    self.canvas.create_line(tmp-25, self.y, tmp+25, self.y,
```

```
                    width=self.width, arrow='both',
                    arrowshape=(self.a, self.b, self.c))
        self.canvas.create_line(tmp-25, self.y+75, tmp+25, self.y+125,
                    width=self.width, arrow='both',
                    arrowshape=(self.a, self.b, self.c))

        tmp = self.x2+10
        self.canvas.create_line(tmp, self.y-5*self.width, tmp,
                    self.y-deltaY, arrow='both', arrowshape=self.smallTips)
        self.canvas.create_text(self.x2+15, self.y-deltaY+5*self.c,
                    text=self.c, anchor=W)
        tmp = self.x1-10
        self.canvas.create_line(tmp, self.y-5*self.width, tmp,
                    self.y+5*self.width, arrow='both',
                    arrowshape=self.smallTips)
        self.canvas.create_text(self.x1-15, self.y,
                    text=self.width, anchor=E)
        tmp = self.y+5*self.width+10*self.c+10
        self.canvas.create_line(self.x2-10*self.a, tmp, self.x2, tmp,
                    arrow='both', arrowshape=self.smallTips)
        self.canvas.create_text(self.x2-5*self.a, tmp+5,
                    text=self.a, anchor=N)
        tmp = tmp+25
        self.canvas.create_line(self.x2-10*self.b, tmp, self.x2, tmp,
                    arrow='both', arrowshape=self.smallTips)
        self.canvas.create_text(self.x2-5*self.b, tmp+5,
                    text=self.b, anchor=N)

        self.canvas.create_text(self.x1, 310, text="width=%d" % \
                    self.width, anchor=W, font=('Verdana', 18))
        self.canvas.create_text(self.x1, 330,
                    text="arrowshape=(%d,%d,%d)" % \
                    (self.a, self.b, self.c),
                    anchor=W, font=('Verdana', 18))

if __name__ == '__main__':
    root = Tk()
    root.option_add('*Font', 'Verdana 10')
    root.title('Arrow Editor')
    arrow = ArrowEditor(root)
    root.mainloop()
```

---

arrow.py 注解

❶ 这个例子还需要创建点别的：

- <Enter>为手柄填色。
- <Leave>删除颜色。
- <Button-1><(1)>对应每一个手柄。
- <Bl-Motion>对应每个手柄的一般值。

● <Any-ButtonRelease-1>按手柄的最终状态处理。

❸ 三个“arrowMove”方法各自的功能是使相应的变化后的值生效并在相应的位置生成新的图形。

❹ 由于此处有三个 box：box1、box2、box3。就需要用一个简单的查找运算来确认到底是那一个 box 创建了对象。

```
if 'box' in tags:
    for tag in tags:
        if len(tag) == 4 and tag[:3] == 'box':
            cur = tag
            break
```

❺ 用用户提供的 width 和 arrowshape 值创建了一条直线。

余下的代码主要负责修改屏幕的尺寸值及绘制样本箭头。既然这个例子诠释了从 Tk 到 Tkinter 一对一的转换，那么就不再尝试去简化它了，其实它还可以写得更精炼一些。

10.6　一些已完成的小玩意

下面我们来扩展一下 draw3.py 的功能，加上一些函数和特性以便将它当作模版来使用。这些是要加的东西：

(1) 菜单项——用于新建或打开文件。

(2) 菜单项——将图片保存为已存在的某个文件。

(3) 菜单项——将图片保存。

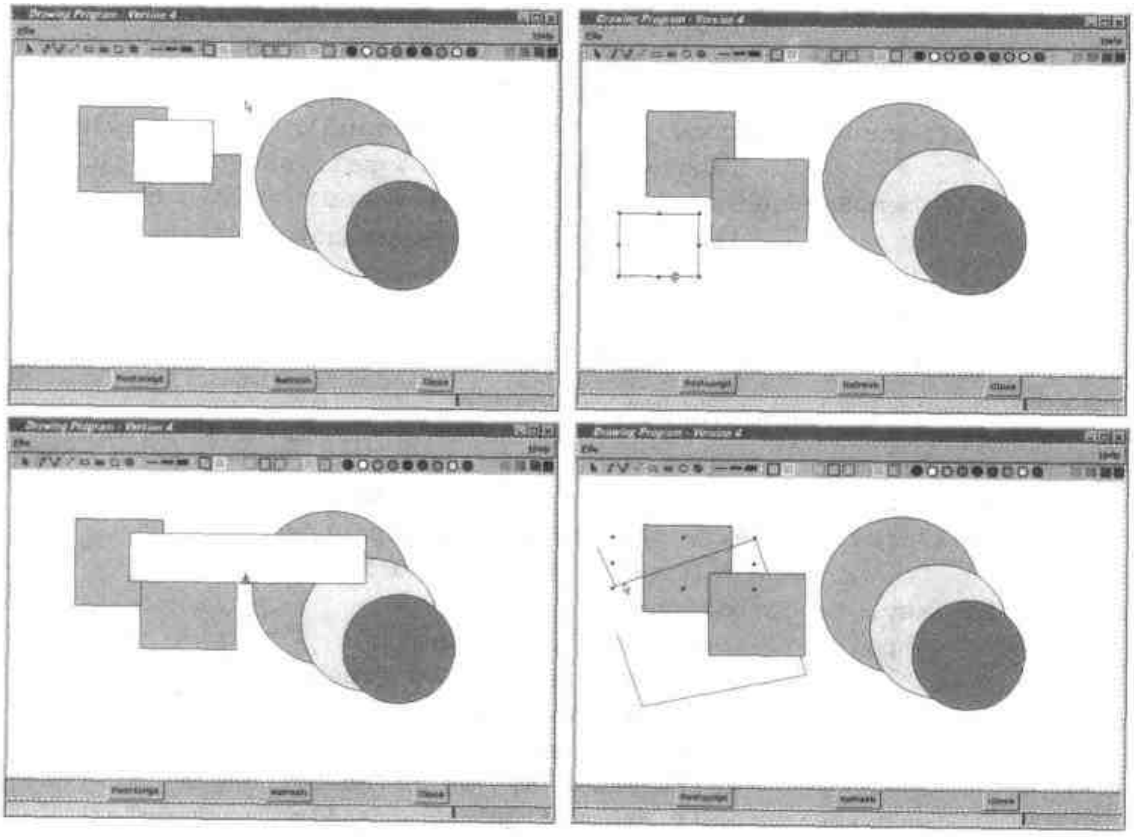


图 10.10

(4) 移动操作，可以将对象在画布上移动。

(5) 有八个手柄的缩放工具。

下面的程序来源于 draw3.py，见 10.2 节。在此删除了一些重复的代码以使程序看起来简短一些，但是注解部分还是很详细。

Draw4.py

```
from  Tkinter import *
import Pmw, AppShell, math, time, string, marshal
from cursornames import *
from toolbarbutton import ToolBarButton
from tkFileDialog import *

transDict = { 'bx': 'boundX', 'by': 'boundY',
        'x':  'adjX',   'y':  'adjY',
        'S':  'uniqueIDINT' }

class Draw(AppShell.AppShell):

# --- Code Removed ----------------------------------------------------

  def createMenus(self):
      self.menuBar.deletemenuitems('File')
      self.menuBar.addmenuitem('File', 'command', 'New drawing',
                    label='New', command=self.newDrawing)
      self.menuBar.addmenuitem('File', 'command', 'Open drawing',
                    label='Open...', command=self.openDrawing)
      self.menuBar.addmenuitem('File', 'command', 'Save drawing',
                    label='Save', command=self.saveDrawing)
      self.menuBar.addmenuitem('File', 'command', 'Save drawing',
                    label='SaveAs...', command=self.saveAsDrawing)
      self.menuBar.addmenuitem('File', 'separator')
      self.menuBar.addmenuitem('File', 'command', 'Exit program',
                    label='Exit', command=self.quit)

  def createTools(self):
      self.func      = {}
      self.transFunc = {}
      ToolBarButton(self, self.toolbar, 'sep', 'sep.gif',
                width=10, state='disabled')
      for key, func, balloon in [
          ('pointer', None, 'Edit drawing'),
          ('draw',    self.drawFree, 'Draw freehand'),
          ('smooth',  self.drawSmooth, 'Smooth freehand'),
          ('line',    self.drawLine, 'Rubber line'),
          ('rect',    self.drawRect, 'Unfilled rectangle'),
          ('frect',   self.drawFilledRect, 'Filled rectangle'),
          ('oval',    self.drawOval, 'Unfilled oval'),
```

```
        ('foval',   self.drawFilledOval, 'Filled oval')]:
        ToolBarButton(self, self.toolbar, key, '%s.gif' % key,
                      command=self.selectFunc, balloonhelp=balloon,
                      statushelp=balloon)
        self.func[key]     = func
        self.transFunc[func] = key

# --- Code Removed ----------------------------------------------------
```

Draw4.py 注解

❶ 工具栏按钮已被分开。

❷ transDict 方法用来分列出一个手柄的数据。见❶。

❸ transFunc 创建成反向查找，以便可以得到与特殊的函数相关的关键字。

Draw4.py（续）

```
    def mouseDown(self, event):
        self.curObject = None
        self.canvas.dtag('drawing')
        self.lineData = []
        self.lastx = self.startx = self.canvas.canvasx(event.x)
        self.lasty = self.starty = self.canvas.canvasy(event.y)
        self.uniqueID = 'S*%d' % self.serial                    ❶
        self.serial = self.serial + 1

        if not self.curFunc:
            if event.widget.find_withtag(CURRENT):
                tags = self.canvas.gettags(CURRENT)
                for tag in tags:
                    if tag[:2] == 'S*':
                        objectID = tag
                if 'grabHandle' in tags:
                    self.inGrab = TRUE
                    self.releaseGrab = FALSE
                    self.uniqueID = objectID
                else:                                           ❸
                    self.inGrab = FALSE
                    self.addGrabHandles(objectID, 'grab')
                    self.canvas.config(cursor='fleur')
                    self.uniqueID = objectID
        else:
            self.canvas.delete("grabHandle")
            self.canvas.dtag("grabHandle")
            self.canvas.dtag("grab")

    def mouseMotion(self, event):
        curx = self.canvas.canvasx(event.x)
        cury = self.canvas.canvasy(event.y)
```

```
        prevx = self.lastx
        prevy = self.lasty
        if not self.inGrab and self.curFunc:
            self.lastx = curx
            self.lasty = cury
            if self.regular and self.curFunc in \
               [self.func['oval'], self.func['rect'],
               self.func['foval'],self.func['frect']]:
                dx      = self.lastx - self.startx
                dy      = self.lasty - self.starty
                delta = max(dx, dy)
                self.lastx = self.startx + delta
                self.lasty = self.starty + delta
            self.curFunc(self.startx, self.starty, self.lastx,
                    self.lasty, prevx, prevy, self.foreground,
                    self.background, self.fillStyle, self.lineWidth,None)
    else self.inGrab:
            self.canvas.delete("grabbedObject")
            self.canvas.dtag("grabbedObject")
            tags = self.canvas.gettags(CURRENT)
            for tag in tags:
                if '*' in tag:
                    key, value = string.split(tag, '*')
                    var = transDict[key]
                    setattr(self, var, string.atoi(value))
            self.uniqueID = 'S*%d' % self.uniqueIDINT
            x1, y1, x2, y2, px, py, self.growFunc, \
                fg, bg, fill, lwid, ld= self.objects[self.uniqueID]
            if self.boundX == 1 and self.adjX:
                x1 =  x1 + curx-prevx
            elif self.boundX == 2 and self.adjX:
                x2 =  x2 + curx-prevx
            if self.boundY == 1 and self.adjY:
                y1 = y1 + cury-prevy
            elif self.boundY == 2 and self.adjY:
                y2 = y2 + cury-prevy
            self.growFunc(x1,y1,x2,y2,px,py,fg,bg,fill,lwid,ld)
            self.canvas.addtag_withtag("grabbedObject",
                                self.uniqueID)
            self.storeObject(x1,y1,x2,y2,px,py,self.growFunc,
                        fg,bg,fill,lwid,ld)
            self.lastx = curx
            self.lasty = cury
        else:
            self.canvas.move('grab', curx-prevx, cury-prevy)
            self.lastx = curx
            self.lasty = cury

    def mouseUp(self, event):
        self.prevx = self.lastx
```

```
    self.prevy = self.lasty
    self.lastx = self.canvas.canvasx(event.x)
    self.lasty = self.canvas.canvasy(event.y)
    if self.curFunc:
        if self.regular and self.curFunc in \
           [self.func['oval'], self.func['rect'],
            self.func['foval'],self.func['frect']]:
            dx = self.lastx - self.startx
            dy = self.lasty - self.starty
            delta = max(dx, dy)
            self.lastx = self.startx + delta
            self.lasty = self.starty + delta
        self.curFunc(self.startx, self.starty, self.lastx,
                self.lasty, self.prevx, self.prevy, self.foreground,
                self.background, self.fillStyle, self.lineWidth,
                self.lineData)
        self.inGrab = FALSE
        self.releaseGrab = TRUE
        self.growFunc = None
        self.storeObject(self.startx, self.starty, self.lastx,
                self.lasty, self.prevx, self.prevy, self.curFunc,
                self.foreground, self.background, self.fillStyle,
                self.lineWidth, self.lineData)
    else:
        if self.inGrab:
            tags = self.canvas.gettags(CURRENT)
            for tag in tags:
                if '*' in tag:
                    key, value = string.split(tag, '*')
                    var = transDict[key]
                    setattr(self, var, string.atoi(value))
            x1,y1,x2,y2, px, py, self.growFunc, \
                    fg,bg,fill,lwid,ld = self.objects[self.uniqueID]
            if self.boundX == 1 and self.adjX:
                x1 =  x1 + self.lastx-self.prevx
            elif self.boundX == 2 and self.adjX:
                x2 =  x2 + self.lastx-self.prevx
            if self.boundY == 1 and self.adjY:
                y1 = y1 + self.lasty-self.prevy
            elif self.boundY == 2 and self.adjY:
                y2 = y2 + self.lasty-self.prevy
            self.growFunc(x1,y1,x2,y2,px,py,fg,bg,fill,lwid,ld)
            self.storeObject(x1,y1,x2,y2,px,py,self.growFunc,
                    fg,bg,fill,lwid,ld)
            self.addGrabHandles(self.uniqueID, self.uniqueID)
        if self.selObj:
            self.canvas.itemconfig(self.selObj,
                                   width=self.savedWidth)
    self.canvas.config(cursor='arrow')
```

```
    def addGrabHandles(self, objectID, tag):
        self.canvas.delete("grabHandle")
        self.canvas.dtag("grabHandle")
        self.canvas.dtag("grab")
        self.canvas.dtag("grabbedObject")

        self.canvas.addtag("grab", "withtag", CURRENT)
        self.canvas.addtag("grabbedObject", "withtag", CURRENT)
        x1,y1,x2,y2 = self.canvas.bbox(tag)
        for x,y, curs, tagBx, tagBy, tagX, tagY in [
             (x1,y1,TLC,       'bx*1','by*1','x*1','y*1'),
             (x2,y1,TRC,       'bx*2','by*1','x*1','y*1'),
             (x1,y2,BLC,       'bx*1','by*2','x*1','y*1'),
             (x2,y2,BRC,       'bx*2','by*2','x*1','y*1'),
             (x1+((x2-x1)/2),y1,TS, 'bx*0','by*1','x*0','y*1'),
             (x2,y1+((y2-y1)/2),RS, 'bx*2','by*0','x*1','y*0'),
             (x1,y1+((y2-y1)/2),LS, 'bx*1','by*0','x*1','y*0'),
             (x1+((x2-x1)/2),y2,BS, 'bx*0','by*2','x*0','y*1')]:
             ghandle = self.canvas.create_rectangle(x-2,y-2,x+2,y+2,
                    outline='black', fill='black', tags=('grab',
                    'grabHandle', tagBx, tagBy, tagX,
                    tagY,'%s'%objectID))
             self.canvas.tag_bind(ghandle, '<Any-Enter>',
                  lambda e, s=self, c=curs: s.setCursor(e,c))
             self.canvas.tag_bind(ghandle, '<Any-Leave>',
                  self.resetCursor)
             self.canvas.lift("grab")

# --- Code Removed ----------------------------------------------------
```

代码注解（续）

❹ 每一个对象都有其唯一的描述，它是一个与对象相关的标志值。这里我们就定义了一个。

```
self.uniqueID = 'S*%d' % self.serial
self.serial = self.serial + 1
```

❺ 用标志值从对象或其变换手柄上获得它的描述，然后决定是否对手柄或对象进行操作。在这种情况下，将指针设成可移动对象的状态。

```
for tag in tags:
     if tag[:2] == 'S*':
          objectID = tag
```

❻ 这里分列出一个手柄的数据，每一个手柄都有它自己的处理过程编码，这样就减少了缩放对象所要求的代码，见❽。

```
for tag in tags:
     if '*' in tag:
        key, value = string.split(tag, '*')
        var = transDict[key]
        setattr(self, var, string.atoi(value))
```

这里有必要解释一下。看看图 10.11，每一个对象四个角上的缩放手柄定义了缩放作用区的尺寸，分别为 BB1 和 BB2 两个 x，y 数组。所以，可以看到，左下角的坐标就是 bx*1 和 by*2。另外，四个边上的手柄用来拖放四个方向，所以将其轴的参数设置为 free（自由）。

❾ 因为在放大或移动时要随时看到对象当前的尺寸，所以随时返回数据。

❿ 松开鼠标时，重新计算对变换区尺寸并存储对象的数据。这对代码与❽中的有关鼠标动作的代码相似。

⓫ AddGrabHandles 函数取消手柄的显示。

⓬ 创建手柄首先创建的是变换区，然后再从列表中获取数据创建手柄。

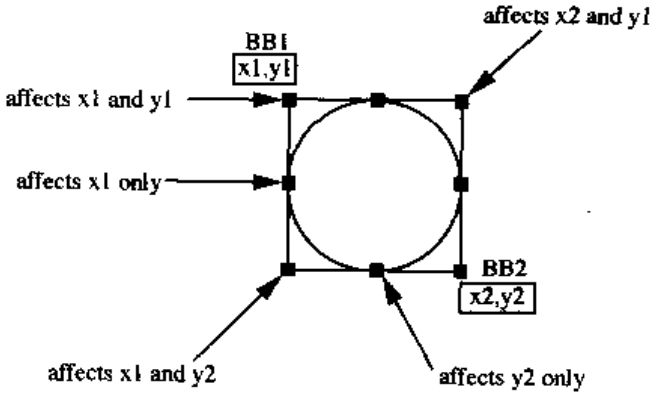


图 10.11　手柄与相关的边界框坐标

Draw4.py 续

```
        x1,y1,x2,y2 = self.canvas.bbox(tag)
        for x,y, curs, tagBx, tagBy, tagX, tagY in [
             (x1,y1,TLC,        'bx*1','by*1','x*1','y*1'),
             (x2,y1,TRC,        'bx*2','by*1','x*1','y*1'),
             ---- code removed -----
             ghandle = self.canvas.create_rectangle(x-2,y-2,x+2,y+2,
                    outline='black', fill='black', tags=('grab',
                    'grabHandle','%s'%tagBx,'%s'%tagBy,'%s'%tagX,
                    '%s'%tagY,'%s'%objectID))

    def storeObject(self, x1,y1,x2,y2,px,py,func,fg,bg,fill,lwid,ld):
        self.objects[self.uniqueID] = ( x1,y1,x2,y2,px,py,func,fg,bg,
                              fill,lwid,ld )

    def redraw(self):
        self.canvas.delete(ALL)
        keys = self.objects.keys()
        keys.sort()
        for key in keys:
            startx, starty, lastx, lasty, prevx, prevy, func, \
                 fg, bg, fill, lwid , ld= self.objects[key]
```

```
            self.curObject = None
            self.uniqueID = key
            func(startx, starty, lastx, lasty, prevx, prevy,
                 fg, bg, fill, lwid, ld)

    def newDrawing(self):
        self.canvas.delete(ALL)
        self.initData()

    def openDrawing(self):
        ofile = askopenfilename(filetypes=[("PTkP Draw", "ptk"),
                                ("All Files", "*")])
        if ofile:
            self.currentName = ofile
            self.initData()
            fd = open(ofile)
            items = marshal.load(fd)
            for i in range(items):
                self.uniqueID, x1,y1,x2,y2,px,py,cfunc, \
                            fg,bg,fill,lwid,ld = marshal.load(fd)
                self.storeObject(x1,y1,x2,y2,px,py,self.func[cfunc],
                            fg,bg,fill,lwid,ld)
            fd.close()
       self.redraw()

    def saveDrawing(self):
        self.doSave()

    def saveAsDrawing(self):
        ofile = asksaveasfilename(filetypes=[("PTkP Draw", "ptk"),
                                ("All Files", "*")])
        if ofile:
            self.currentName = ofile
            self.doSave()

    def doSave(self):
        fd = open(self.currentName, 'w')
        keys = self.objects.keys()
        keys.sort()
        marshal.dump(len(keys), fd)
        for key in keys:
            startx, starty, lastx, lasty, prevx, prevy, func, \
                 fg, bg, fill, lwid , ld= self.objects[key]
            cfunc = self.transFunc[func]
            marshal.dump((key, startx, starty, lastx, lasty, prevx,\
                        prevy, cfunc, fg, bg, fill, lwid, ld), fd)
        fd.close()

    def initData(self):
        self.curFunc = self.drawLine
```

```
        self.growFunc = None
        self.curObject = None
        self.selObj = None
        self.lineData = []
        self.savedWidth = 1
        self.savedCursor = None
        self.objects = {}     # Now a dictionary
        self.foreground = 'black'
        self.background = 'white'
        self.fillStyle = None
        self.lineWidth = 1
        self.serial = 1000
        self.regular = FALSE
        self.inGrab = FALSE
        self.releaseGrab = TRUE
        self.currentName = 'Untitled'
# --- Code Removed ----------------------------------------------
```

代码注解（续）

❶ 打开现存文件时用的是标准的 tkFileDialog 对话框。然后读取文件数据从而可以在对象数据字典中获取相应数据以重绘对象*。

由于不能直接读取到类的子函数，我们将函数的关键字保存下来再用❸中的反向查找得到相应的方法。

```
fd = open(ofile)
items = marshal.load(fd)
for i in range(items):
    self.uniqueID, x1,y1,x2,y2,px,py,cfunc, \
                fg,bg,fill,lwid,ld = marshal.load(fd)
    self.storeObject(x1,y1,x2,y2,px,py,self.func[cfunc],
                fg,bg,fill,lwid,ld)
```

❷ doSave 方法实现文件存储。图 10.12 打开对话框。注意 Tk 8.0 以上的版本才支持这种对话框。

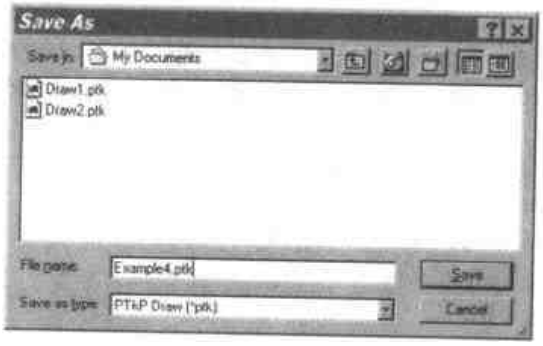


图 10.12　另存为对话框

* Marshaling 是个使任意的 Python 数据串行化到一种易于读写格式文件的方法。不是所有的 Python 类型都可以串行化，而这时可以使用别的方法，如 pickle 或 shelve（存储为数据对象）。为相对简单的字典提供持续性是足够的。

10.7 快速绘制

一般来说，创建画布对象效率比较高而且很少会出问题。但是对于非常复杂的图像来说，必须注意到画布的时滞问题，尤其是当画布上的对象非常多或者对象的线条非常复杂的时候。

提高画布效率的一种方法是将画布作为一幅图片。第 5 章介绍过的 Python 图片库就有直接将其弄成 gif 格式图片的工具。下面我们用这个来绘制一个非常有水准的图形。我们常常看到一般用来生成分形画的 Mandelbrot 作图器，Douglas A.Young's 的 *The X Window System: Programming and Applications with Xt* 的封面就是一幅分形画。下面就是一个 Python、Tkinter 和 PIL 的分形画改写本。

fractal.py

```
from   Tkinter import *
import Pmw, AppShell, Image, ImageDraw, os

class Palette:
  def __init__(self):
      self.palette = [(0,0,0), (255,255,255)]

  def getpalette(self):
      # flatten the  palette
      palette = []
      for r, g, b in self.palette:
          palette = palette + [r, g, b]
      return palette

  def loadpalette(self, cells):
      import random
      for i in range(cells-2):
         self.palette.append((
              random.choice(range(0, 255)),  # red
              random.choice(range(0, 255)),  # green
              random.choice(range(0, 255)))) # blue

class Fractal(AppShell.AppShell):
  usecommandarea = 1
  appname        = 'Fractal Demonstration'
  frameWidth     = 780
  frameHeight    = 580

  def createButtons(self):
      self.buttonAdd('Save',
                 helpMessage='Save current image',
                 statusMessage='Write current image as "out.gif"',
                 command=self.save)
```

```
        self.buttonAdd('Close',
                       helpMessage='Close Screen',
                       statusMessage='Exit',
                       command=self.close)

    def createDisplay(self):
        self.width  = self.root.winfo_width()-10
        self.height = self.root.winfo_height()-95
        self.form = self.createcomponent('form', (), None,
                                Frame, (self.interior(),),
                                width=self.width,
                                height=self.height)
        self.form.pack(side=TOP, expand=YES, fill=BOTH)
        self.im = Image.new("P", (self.width, self.height), 0)
        self.d = ImageDraw.ImageDraw(self.im)
        self.d.setfill(0)
        self.label = self.createcomponent('label', (), None,
                                Label, (self.form,),)
        self.label.pack()

    def initData(self):
        self.depth = 20
        self.origin = complex(-1.4, 1.0)
        self.range = 2.0
        self.maxDistance = 4.0
        self.ncolors = 256
        self.rgb = Palette()
        self.rgb.loadpalette(255)
        self.save = FALSE

    def createImage(self):
        self.updateProgress(0, self.height)
        for y in range(self.height):
            for x in range(self.width):
                z = complex(0.0, 0.0)
                k = complex(self.origin.real + \
                        float(x)/float(self.width)*self.range,
                        self.origin.imag - \
                        float(y) / float(self.height)*self.range)
                # calculate z = (z +k) * (z + k) over and over
                for iteration in range(self.depth):
                    real_part = z.real + k.real
                    imag_part = z.imag + k.imag
                    del z
                    z = complex(real_part * real_part - imag_part * \
                            imag_part, 2 * real_part * imag_part)
                    distance  = z.real * z.real + z.imag * z.imag
                    if distance >= self.maxDistance:
                        cidx = int(distance % self.ncolors)
                        self.pixel(x, y, cidx)
```

```
                        break
                self.updateProgress(y)
            self.updateProgress(self.height, self.height)
            self.im.putpalette(self.rgb.getpalette())
            self.im.save("out.gif")
            self.img = PhotoImage(file="out.gif")
            self.label['image'] = self.img

        def pixel(self, x, y, color):
            self.d.setink(color)
            self.d.point((x, y))

        def save(self):
            self.save = TRUE
            self.updateMessageBar('Saved as "out.gif"')

        def close(self):
            if not self.save:
                os.unlink("out.gif")
            self.quit()

        def createInterface(self):
            AppShell.AppShell.createInterface(self)
            self.createButtons()
            self.initData()
            self.createDisplay()

    if __name__ == '__main__':
        fractal = Fractal()
        fractal.root.after(10, fractal.createImage())
        fractal.run()
```

❹

❺

fractal.py 注解

❶ 调色板类负责生成一个任意的颜色并处理 GIF 文件的 RGB 色表。

❷ 创建一幅新的图片，指定为像素模式，然后使用绘图类，它提供了基本的绘图函数。用黑色填充图片。

```
self.im = Image.new("P", (self.width, self.height), 0)
self.d = ImageDraw.ImageDraw(self.im)
self.d.setfill(0)
```

❸ 在计算过程中选择颜色并为颜色设置相应的像素模式。

```
cidx = int(distance % self.ncolors)
self.pixel(x, y, cidx)
```

❹ 完成之后，加入调色板，存为 GIF 文件，再载入图形为 PhotoImage。

```
self.im.putpalette(self.rgb.getpalette())
self.im.save("out.gif")
self.img = PhotoImage(file="out.gif")
self.label['image'] = self.img
```

❺ 像素模式非常简单，只要设置颜色然后将像素定义为一系列的 x，y 数组。

用运算速度比较快的工作站运行 Fractal.py，两三分钟后就生成一幅 800 像素×600 像素大小的图片。

```
def pixel(self, x, y, color):
    self.d.setink(color)
    self.d.point((x, y))
```

感兴趣的读者可以在我们的网站上找到 slowfractal.py 的程序，它由 Tkinter 的画布方法生成。但对于 PC 来说运行要用相当长的时间。

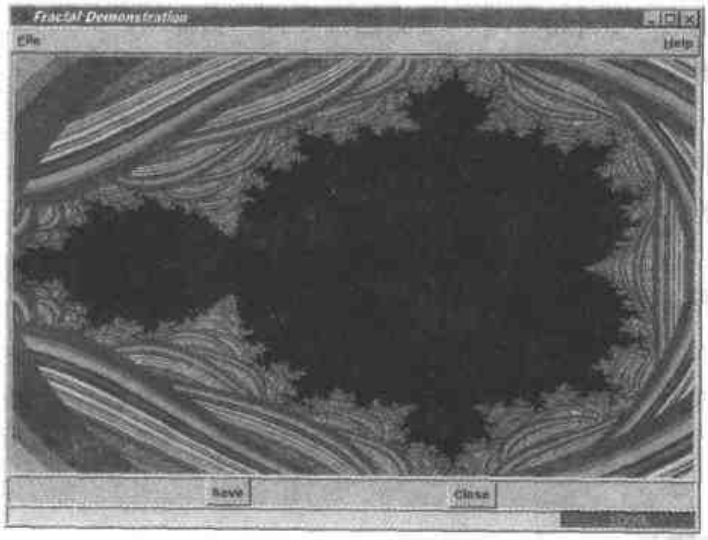


图 10.13　产生分形

10.8　小结

这一章对于希望巧妙绘图的读者来说也是非常重要的。无论是设计绘图软件还是设计 UML 的编辑系统，原则都是相似的，你会发现很多技术都是可以互相转换的。

在设计这些界面的时候有一点是很重要的，那就是仔细考虑你的系统使用的定位设备。如果用的是鼠标，拖放对象就很容易，但如果用的是键盘嵌入式的或是滚珠式的鼠标，那就不是很容易了。

第 11 章　图形和图表

有一段时间里图形（graphics）一词便包含了图（graphs）的意思，在这一章里我们要重新引入这个术语。虽然说图、直方图和饼图并不是对每种应用程序都合适，但它们确实给观众提供了一种获取大量信息的手段。我们将给出曲线图（linegraphs）、直方图和饼图这样一些典型图式（graphic formats）的例子。进一步再给出一些复杂的例子，如阈值报警器和指示器。

11.1　简单图形

我们先不考虑构造图形类以及增加太多的特征，来构建一个非常简单的图形，这样我们就能看到给一个应用程序加上图形是如此简单。我们将在以后再添加更多的功能。

simpleplot.py

```
from Tkinter import *
root = Tk()
root.title('Simple Plot - Version 1')

canvas = Canvas(root, width=450, height=300, bg = 'white')
canvas.pack()

Button(root, text='Quit', command=root.quit).pack()

canvas.create_line(100,250,400,250, width=2)
canvas.create_line(100,250,100,50,  width=2)

for i in range(11):
  x = 100 + (i * 30)
  canvas.create_line(x,250,x,245, width=2)
  canvas.create_text(x,254, text='%d'% (10*i), anchor=N)

for i in range(6):
  y = 250 - (i * 40)
   canvas.create_line(100,y,105,y, width=2)
   canvas.create_text(96,y,text='%5.1f'% (50.*i), anchor=E)
```

画坐标轴

结 y 轴加上刻度和标签

```
for x,y in [(12,56),(20,94),(33,98),(45,120),(61,180),
      (75,160),(98,223)]:
   x = 100 + 3*x
   y = 250 - (4*y)/5
   canvas.create_oval(x-6,y-6,x+6,y+6, width=1,
                      outline='black', fill='SkyBlue2')

root.mainloop()
```

● 绘数据点

代码注解

❶ 这里我们给 x 轴加上刻度和标签。注意所用的值是固定的——我们并没有为以后的使用考虑太多。

```
for i in range(11):
   x = 100 + (i * 30)
   canvas.create_line(x,250,x,245, width=2)
   canvas.create_text(x,254, text='%d'% (10*i), anchor=N)
```

注意我们是怎么进行设置使 x 以 10 为单位增加的。

如图 11.1 所示，这段很短的代码所产生的效果改变不很明显。通过增加点之间的连线，我们可以很容易地改进这幅图，结果如图 11.2 所示。

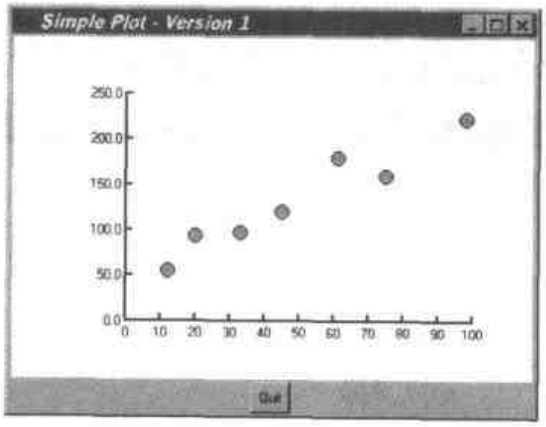


图 11.1　简单二维图

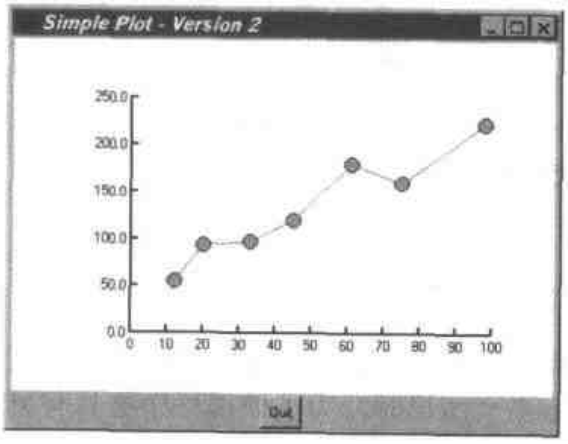


图 11.2　在简单图上添加直线

Simpleplot2.py

```
scaled = []
for x,y in [(12, 56), (20, 94), (33, 98), (45, 120), (61, 180),
            (75, 160), (98, 223)]:
    scaled.append(100 + 3*x, 250 - (4*y)/5)

canvas.create_line(scaled, fill='royalblue')

for x,y in scaled:
    canvas.create_oval(x-6,y-6,x+6,y+6, width=1,
                       outline='black', fill='SkyBlue2')
```

代码注解

❶ 这样我们不必通过在一个简单循环里迭代数据，就可以构造出一串 x 和 y 坐标值来建立这条连线（坐标数据串就可以输入到 create_line 方法中）。

❷ 我们先画出线段。注意画在画布上的那些项是层叠的，所以我们希望线条出现在圆点的下面。

❸ 接下来就是圆点。

这就出现了一个白送的东西！我们可以不需要额外的付出就可以使线条平滑！如果我们打开平滑功能，我们便可以得到一个额外三次样条曲线，如图 11.3 所示。

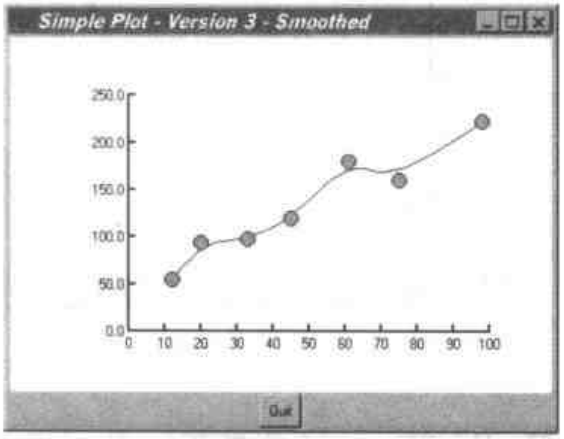


图 11.3 平滑直线

```
canvas.create_line(scaled,fill='black',smooth=1)
```

我不认为这里需要解释！

11.2 图形控件

前面的例子说明了可以非常容易地用一小段代码产生一幅简单的图形。然而，当需要在同一个坐标中显示多个图形时，为了使得程序更具有灵活性，而足以应付所有的情况，则在写程序的时候会非常得繁琐。前些时候 Konrad Hinsen 做了一个用于 Python 的有效的图形控件，此控件专门配合 NumPy*使用。经过 Hinsen 的同意，我把

它修改了使之可以用于标准 Python 发行版本，并且我还使之扩展了可以支持其他的显示格式。图 11.4 给出了一个输出的例子。在下面的代码清单中我们除去了一些重复的代码。读者可以在网上查到全部的源代码。

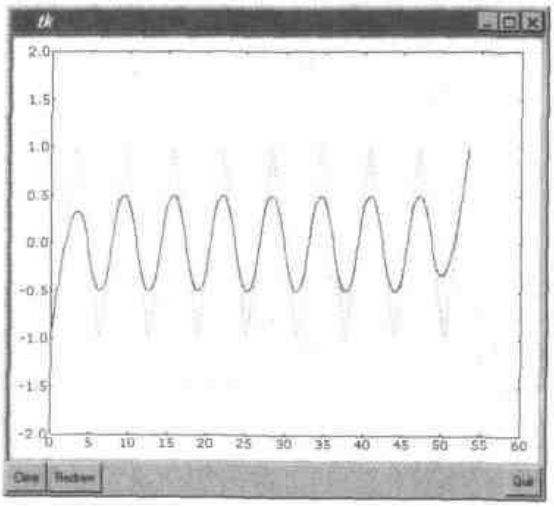


图 11.4　简单图形控件：线

plot.py

```
from Tkinter import *
from Canvas import Line, CanvasText
import string, math
from utils import *
from math import pi

class GraphPoints:
  def __init__(self, points, attr):
      self.points = points
      self.scaled = self.points
      self.attributes = {}
      for name, value in self._attributes.items():
          try:
              value = attr[name]
          except KeyError: pass
          self.attributes[name] = value

  def boundingBox(self):
      return minBound(self.points), maxBound(self.points)

  def fitToScale(self, scale=(1,1), shift=(0,0)):
      self.scaled = []
      for x,y in self.points:
          self.scaled.append((scale[0]*x)+shift[0],\
                             (scale[1]*y)+shift[1])
```

❶ ❷ ❸

* NumPy 即 Numeric Python，是一个专门用于简化数值计算的附加模块集。

```
class GraphLine(GraphPoints):
  def __init__(self, points, **attr):
      GraphPoints.__init__(self, points, attr)

  _attributes = {'color':  'black',
                 'width':       1,
                 'smooth':      0,
                 'splinesteps': 12}

  def draw(self, canvas):
      color  = self.attributes['color']
      width  = self.attributes['width']
      smooth = self.attributes['smooth']
      steps  = self.attributes['splinesteps']
      arguments = (canvas,)

      if smooth:
          for i in range(len(self.points)):
              x1, y1 = self.scaled[i]
              arguments = arguments + (x1, y1)
      else:
          for i in range(len(self.points)-1):
              x1, y1 = self.scaled[i]
              x2, y2 = self.scaled[i+1]
              arguments = arguments + (x1, y1, x2, y2)

      apply(Line, arguments, {'fill': color, 'width': width,
                              'smooth': smooth,
                              'splinesteps': steps})

class GraphSymbols(GraphPoints):
  def __init__(self, points, **attr):
      GraphPoints.__init__(self, points, attr)

  _attributes = {'color': 'black',
                 'width': 1,
                 'fillcolor': 'black',
                 'size': 2,
                 'fillstyle': '',
                 'outline': 'black',
                 'marker': 'circle'}

  def draw(self, canvas):
      color    = self.attributes['color']
      size     = self.attributes['size']
      fillcolor = self.attributes['fillcolor']
      marker   = self.attributes['marker']
      fillstyle = self.attributes['fillstyle']
```

```
            self._drawmarkers(canvas, self.scaled, marker, color,
                    fillstyle, fillcolor, size)

    def _drawmarkers(self, c, coords, marker='circle',
             color='black', fillstyle='', fillcolor='', size=2):
        l = []
        f = eval('self._' +marker)
        for xc, yc in coords:
            id = f(c, xc, yc, outline=color, size=size,
                  fill=fillcolor, fillstyle=fillstyle)
            if type(id) is type(()):
                for item in id: l.append(item)
            else:
                l.append(id)
        return l

    def _circle(self, c, xc, yc, size=1, fill='', outline='black',
            fillstyle=''):
        id = c.create_oval(xc-0.5, yc-0.5, xc+0.5, yc+0.5,
                    fill=fill, outline=outline,
                    stipple=fillstyle)
        c.scale(id, xc, yc, size*5, size*5)
        return id

# --- Code Removed ----------------------------------------------
```

代码注解

❶ GraphPoints 类定义了单个图形的点和属性。正如读者将会看到的，属性是由设计者处理的，并且根据线条的类型而不同。注意 self._attributes 定义是子类的必备条件。

❷ boundingBox 在点数据中扫过后返回左上角和右下角的坐标值。在 utils.py 中有方便的函数。

❸ fitToScale 修改坐标值使之能符合图中全部线条所规定的尺寸。

```
def fitToScale(self, scale=(1,1), shift=(0,0)):
    self.scaled = []
    for x,y in self.points:
        self.scaled.append((scale[0]*x)+shift[0],\
                (scale[1]*y)+shift[1])
```

注意我们定义 scale 和 shift 为元组型的数据。前一个值给 x 而后一个给 y。

❽ GraphLine 类定义由现有的坐标值绘制连线的方法。

❾ draw 方法首先由属性字典中提出合适的参数。

❿ 我们根据是否需要进行平滑，来提供线段的端点坐标值（非平滑）或者一串坐标值（平滑）。

⓫ 我们接下来给画布的 Line 方法提供参数和关键字。记住 Line 的参数的格式是这样的：

```
Line(*args, **keywords)
```

❶ GraphSymbols 和 GraphLine 相似，但它给每一个 x 和 y 坐标输出多种类型的填充形状。

❷ draw 方法通过类属_drawmarkers 方法调用合适的标记例程：

```
self._drawmarkers(canvas, self.scaled, marker, color,
                  fillstyle, fillcolor, size)
```

❸ _drawmarkers 运行所选的标记方法，接下来再建立一个生成的记号的列表。

```
f = eval('self._' +marker)
for xc, yc in coords:
     id = f(c, xc, yc, outline=color, size=size,
            fill=fillcolor, fillstyle=fillstyle)
```

❹ 我们这里仅给出了一个图形控件可以绘出的形状。其余的源代码可在网上查询。

plot.py（续）

```
    def _dot(self, c, xc, yc, ... ):
    def _square(self, c, xc, yc, ... ):
    def _triangle(self, c, xc, yc, ... ):
    def _triangle_down(self, c, xc, yc, ... ):
    def _cross(self, c, xc, yc, ... ):
    def _plus(self, c, xc, yc, ... ):

# --- Code Removed ----------------------------------------------------

class GraphObjects:
  def __init__(self, objects):
      self.objects = objects

  def boundingBox(self):
          c1, c2 = self.objects[0].boundingBox()
       for object in self.objects[1:]:
            c1o, c2o = object.boundingBox()
            c1 = minBound([c1, c1o])
            c2 = maxBound([c2, c2o])
       return c1, c2

  def fitToScale(self, scale=(1,1), shift=(0,0)):
       for object in self.objects:
            object.fitToScale(scale, shift)

  def draw(self, canvas):
       for object in self.objects:
            object.draw(canvas)

class GraphBase(Frame):
  def __init__(self, master, width, height,
            background='white', **kw):
      apply(Frame.__init__, (self, master), kw)
```

```
        self.canvas = Canvas(self, width=width, height=height,
                             background=background)
        self.canvas.pack(fill=BOTH, expand=YES)
        border_w = self.canvas.winfo_reqwidth() - \
                   string.atoi(self.canvas.cget('width'))
        border_h = self.canvas.winfo_reqheight() - \
                   string.atoi(self.canvas.cget('height'))
        self.border = (border_w, border_h)
        self.canvas.bind('<Configure>', self.configure)
        self.plotarea_size = [None, None]
        self._setsize()
        self.last_drawn = None
        self.font = ('Verdana', 10)

    def configure(self, event):
        new_width = event.width-self.border[0]
        new_height = event.height-self.border[1]
        width = string.atoi(self.canvas.cget('width'))
        height = string.atoi(self.canvas.cget('height'))
        if new_width == width and new_height == height:
            return
        self.canvas.configure(width=new_width, height=new_height)
        self._setsize()
        self.clear()
        self.replot()

    def bind(self, *args):
        apply(self.canvas.bind, args)

    def _setsize(self):
        self.width = string.atoi(self.canvas.cget('width'))
        self.height = string.atoi(self.canvas.cget('height'))
        self.plotarea_size[0] = 0.97 * self.width
        self.plotarea_size[1] = 0.97 * -self.height
        xo = 0.5*(self.width-self.plotarea_size[0])
        yo = self.height-0.5*(self.height+self.plotarea_size[1])
        self.plotarea_origin = (xo, yo)

    def draw(self, graphics, xaxis = None, yaxis = None):
        self.last_drawn = (graphics, xaxis, yaxis)
        p1, p2 = graphics.boundingBox()
        xaxis = self._axisInterval(xaxis, p1[0], p2[0])
        yaxis = self._axisInterval(yaxis, p1[1], p2[1])
        text_width = [0., 0.]
        text_height = [0., 0.]

        if xaxis is not None:
            p1 = xaxis[0], p1[1]
            p2 = xaxis[1], p2[1]
            xticks = self._ticks(xaxis[0], xaxis[1])
```

```
            bb = self._textBoundingBox(xticks[0][1])
            text_height[1] = bb[3]-bb[1]
            text_width[0] = 0.5*(bb[2]-bb[0])
            bb = self._textBoundingBox(xticks[-1][1])
            text_width[1] = 0.5*(bb[2]-bb[0])
        else:
            xticks = None
        if yaxis is not None:
            p1 = p1[0], yaxis[0]
            p2 = p2[0], yaxis[1]
            yticks = self._ticks(yaxis[0], yaxis[1])
            for y in yticks:
                bb = self._textBoundingBox(y[1])
                w = bb[2]-bb[0]
                text_width[0] = max(text_width[0], w)
                h = 0.5*(bb[3]-bb[1])
                text_height[0] = h
                text_height[1] = max(text_height[1], h)
        else:
            yticks = None
        text1 = [text_width[0], -text_height[1]]
        text2 = [text_width[1], -text_height[0]]
        scale = ((self.plotarea_size[0]-text1[0]-text2[0]) / \
              (p2[0]-p1[0]),
              (self.plotarea_size[1]-text1[1]-text2[1]) / \
              (p2[1]-p1[1]))
        shift = ((-p1[0]*scale[0]) + self.plotarea_origin[0] + \
              text1[0],
              (-p1[1]*scale[1]) + self.plotarea_origin[1] + \
              text1[1])
        self._drawAxes(self.canvas, xaxis, yaxis, p1, p2,
                       scale, shift, xticks, yticks)
        graphics.fitToScale(scale, shift)
        graphics.draw(self.canvas)

# --- Code Removed ------------------------------------------------
```

代码注解（续）

- GraphObjects 类为每一个图形定义了图形符号体系的集合。特别地，它负责确定所有线段的共同边界框。
- fitToScale 依据算好的边界框来对每条线段定标。
- 最后，draw 方法给复合体中的每一个图形着色。
- GraphBase 是基本的控件类，它包含了每一个复合体。正如读者以后将要看到的，我们可以组合使用不同的图形控件来产生需要的效果。
- 一旦父容器的尺寸改变，该控件将重新绘制图形，这是其最重要的特征。这使得用户可以随意地伸缩图像。我们把一个 configure 事件捆绑到 configure 回调上。

plot.py（续）

```
        self.canvas.bind('<Configure>', self.configure)
if __name__ == '__main__':
    root = Tk()
    di = 5.*pi/5.
    data = []

    for i in range(18):
        data.append((float(i)*di,
                     (math.sin(float(i)*di)-math.cos(float(i)*di))))
    line  = GraphLine(data, color='gray', smooth=0)
    linea = GraphLine(data, color='blue', smooth=1, splinesteps=500)

    graphObject = GraphObjects([line, linea])

    graph  = GraphBase(root, 500, 400, relief=SUNKEN, border=2)
    graph.pack(side=TOP, fill=BOTH, expand=YES)

    graph.draw(graphObject, 'automatic', 'automatic')

    Button(root, text='Clear',  command=graph.clear).pack(side=LEFT)
    Button(root, text='Redraw', command=graph.replot).pack(side=LEFT)
    Button(root, text='Quit',   command=root.quit).pack(side=RIGHT)

    root.mainloop()
```

代码注解（续）

● 使用图形控件非常容易。首先，我们建立我们想要绘制的线段/曲线：

```
for i in range(18):
    data.append((float(i)*di,
                 (math.sin(float(i)*di)-math.cos(float(i)*di))))
line  = GraphLine(data, color='gray', smooth=0)
linea = GraphLine(data, color='blue', smooth=1, splinesteps=500)
```

● 接下来，我们创建 GraphObject 完成所需的定标：

```
graphObject = GraphObjects([line, linea])
```

● 最后，我们创建图形控件，并且使之与 GraphObject 相联系。

```
graph  = GraphBase(root, 500, 400, relief=SUNKEN, border=2)
graph.pack(side=TOP, fill=BOTH, expand=YES)
graph.draw(graphObject, 'automatic', 'automatic')
```

11.2.1　添加条形图

现在我们已经有了基本的图形控件，便可以很容易地添加新的图像类型。条形图（也叫做直方图）是一个常用的显示数据的手段，尤其是当用来描绘数据的大小的时候，这是由于条形相对于曲线下方的视觉体积而言有着实际体积。图 11.5 给出了几张典型的条形图，有时也结合线段图形。注意，我们可以很容易地建立图形控件的复合

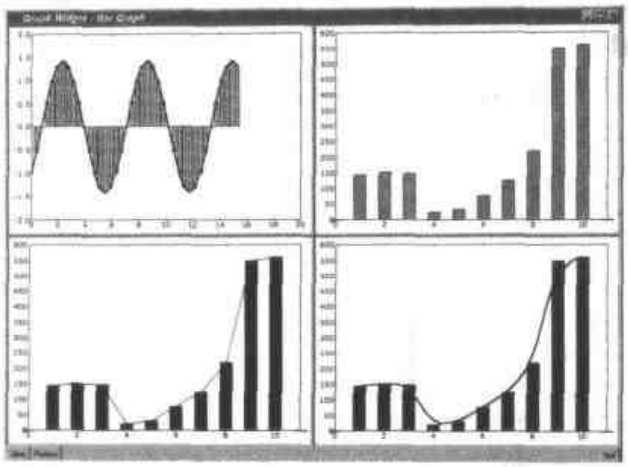


图 11.5　在图形控件上添加条形图

实例。

plot2.py

```
from Tkinter import *
from Canvas import Line, CanvasText, Rectangle

class GraphPoints:

# --- Code Removed ----------------------------------------------------

def fitToScale(self, scale=(1,1), shift=(0,0)):
       self.scaled = []
       for x,y in self.points:
          self.scaled.append((scale[0]*x)+shift[0],\
                             (scale[1]*y)+shift[1])
       self.anchor = scale[1]*self.attributes.get('anchor', 0.0)\ ❶
                           + shift[1]

# --- Code Removed ----------------------------------------------------

class GraphLine(GraphPoints):
  def __init__(self, points, **attr):
       GraphPoints.__init__(self, points, attr)
        _attributes = {'color': 'black',
                    'width': 1,
                    'fillcolor': 'yellow',
                    'size': 3,
                    'fillstyle': '',
                    'outline': 'black'}

  def draw(self, canvas):
       color  = self.attributes['color']
       width  = self.attributes['width']
       fillstyle = self.attributes['fillstyle']
```

❷

```
        outline = self.attributes['outline']
        spread  = self.attributes['size']
        arguments = (canvas,)
        p1, p2 = self.boundingBox()
        for i in range(len(self.points)):
             x1, y1 = self.scaled[i]
             canvas.create_rectangle(x1-spread, y1, x1+spread,
                                     self.anchor, fill=color,
                                     width=width, outline=outline,
                                     stipple=fillstyle)

# --- Code Removed ----------------------------------------------------

if __name__ == '__main__':
   root = Tk()
   root.title('Graph Widget - Bar Graph')

   di = 5.*pi/40.
   data = []
   for i in range(40):
       data.append((float(i)*di,
                    (math.sin(float(i)*di)-math.cos(float(i)*di))))
   line1  = GraphLine(data, color='black', width=2,
                      smooth=1)
   line1a = GraphBars(data[1:], color='blue', fillstyle='gray25',
                      anchor=0.0)

   line2 = GraphBars([(0,0),(1,145),(2,151),(3,147),(4,22),(5,31),
                      (6,77),(7,125),(8,220),(9,550),(10,560),(11,0)],
                     color='green', size=10)

   line3 = GraphBars([(0,0),(1,145),(2,151),(3,147),(4,22),(5,31),
                      (6,77),(7,125),(8,220),(9,550),(10,560),(11,0)],
                     color='blue', size=10)
   line3a = GraphLine([(1,145),(2,151),(3,147),(4,22),(5,31),
                       (6,77),(7,125),(8,220),(9,550),(10,560)],
                      color='black', width=1, smooth=0)

   line4 = GraphBars([(0,0),(1,145),(2,151),(3,147),(4,22),(5,31),
                      (6,77),(7,125),(8,220),(9,550),(10,560),(11,0)],
                     color='blue', size=10)
   line4a = GraphLine([(1,145),(2,151),(3,147),(4,22),(5,31),
                       (6,77),(7,125),(8,220),(9,550),(10,560)],
                      color='black', width=2, smooth=1)

   graphObject  = GraphObjects([line1a, line1])
   graphObject2 = GraphObjects([line2])
```

❸

❹

```
graphObject3 = GraphObjects([line3a, line3])
graphObject4 = GraphObjects([line4, line4a])

f1 = Frame(root)
f2 = Frame(root)
graph  = GraphBase(f1, 500, 350, relief=SUNKEN, border=2)
graph.pack(side=LEFT, fill=BOTH, expand=YES)
graph.draw(graphObject, 'automatic', 'automatic')

graph2  = GraphBase(f1, 500, 350, relief=SUNKEN, border=2)
graph2.pack(side=LEFT, fill=BOTH, expand=YES)
graph2.draw(graphObject2, 'automatic', 'automatic')

graph3  = GraphBase(f2, 500, 350, relief=SUNKEN, border=2)
graph3.pack(side=LEFT, fill=BOTH, expand=YES)
graph3.draw(graphObject3, 'automatic', 'automatic')

graph4  = GraphBase(f2, 500, 350, relief=SUNKEN, border=2)
graph4.pack(side=LEFT, fill=BOTH, expand=YES)
graph4.draw(graphObject4, 'automatic', 'automatic')

f1.pack()
f2.pack()
```

代码注解

❶ 这儿没有太多的解释，我们认为变化是白明的。不过，锚很值得简单介绍。对正弦/余弦曲线，我们需要条形由零开始，这便是锚点值。如果我们不对它设置，则不论其取什么样的值都将由 x 轴开始绘图。

```
self.anchor=scale[1]*self.attributes.get('anchor',0.0)+shift[1]
```

❷ 条形图有一些稍微不同的选项需要设置。

❸ 条形图仅绘制一个矩形。

❹ 数据的定义和线段所用的方法有些相似。注意我们省略了第一个数据点，这样就不会和 y 轴重合了：

```
    linela = GraphBars(data[1:], color='blue', fillstyle='gray25',
anchor=0.0)
```

11.2.2 饼图

就像 Emeril Lagesse*说的那样，“让我们来踢出个缺口！”如果条形图很容易地添加了，那么添加饼图也就不会很难了。饼图似乎在管理报告中找到了落脚地，因为它们非

* Emeril Lagasse 是美国 New Orleans 和 Las Vegas 的一个很受欢迎的厨师/餐馆经营者。他是一家正规有线电视烹饪节目的充满活力的主持人。观众们大声的叫着“Bam! 让我们来踢出个缺口！”进入到 Emeril 的节目中去，而他则把他自己的精华融入到创造中。

常好地表达了某些类型的信息。正如在图 11.6 中看到的那样，我添加了一些小细节来增加一点额外的用途。首先是给饼图定标，如果它是和其他的图形组合在一起画出的话——这可以避免饼图成为坐标轴那样(然而，我不赞成把饼图或者是直方图组合起来使用)。其次，如果饼图的高和宽不相等，我增加了点小装饰，使得效果和三维图形一样。

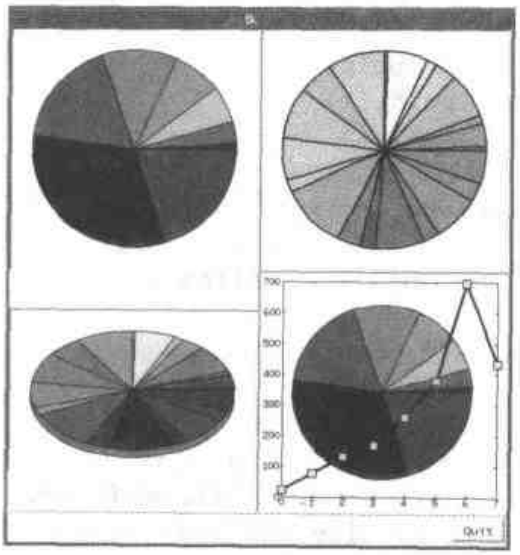


图 11.6　在图形控件中加入饼图

Tk 的 8.0/8.1 版有一个问题。在 Windows 下运行的时候，弧线对象的点描(stipple)——如果有的话——被忽略了；在 Unix 下，图形就被捕获了。下面是创建饼图时的更改：

plot3.py

```
# --- Code Removed ----------------------------------------------------
class GraphPie(GraphPoints):
   def __init__(self, points, **attr):
       GraphPoints.__init__(self, points, attr)

       _attributes = {'color': 'black',
                      'width': 1,
                      'fillcolor': 'yellow',
                      'size': 2,
                      'fillstyle': '',
                      'outline': 'black'}

   def draw(self, canvas, multi):
       width = self.attributes['width']
       fillstyle = self.attributes['fillstyle']
       outline = self.attributes['outline']
       colors = Pmw.Color.spectrum(len(self.scaled))    ❶
       arguments = (canvas,)

       x1 = string.atoi(canvas.cget('width'))
```

```
        y1 = string.atoi(canvas.cget('height'))
        adj = 0
        if multi: adj = 15
        xy  = 25+adj, 25+adj, x1-25-adj, y1-25-adj
        xys = 25+adj, 25+adj+10, x1-25-adj, y1-25-adj+10
        tt = 0.0
        i = 0
        for point in self.points:
            tt = tt + point[1]
        start = 0.0
        if not x1 == y1:
            canvas.create_arc(xys, start=0.0, extent=359.99,
                              fill='gray60', outline=outline,
                              style='pieslice')

        for point in self.points:
            x1, y1 = point
            extent = (y1/tt)*360.0
            canvas.create_arc(xy, start=start, extent=extent,
                              fill=colors[i], width=width,
                              outline=outline, stipple=fillstyle,
                              style='pieslice')
            start = start + extent
            i = i+1

class GraphObjects:
    def __init__(self, objects):
        self.objects  = objects
        self.multiple = len(objects)-1

# --- Code Removed ----------------------------------------------

if __name__ == '__main__':
    root = Tk()
    root.title('Graph Widget - Piechart')

    pie1   = GraphPie([(0,21),(1,77),(2,129),(3,169),(4,260),(5,377),
                       (6,695),(7,434)])
    pie2   = GraphPie([(0,5),(1,22),(2,8),(3,45),(4,22),
                       (5,9),(6,40),(7,2),(8,56),(9,34),
                       (10,51),(11,43),(12,12),(13,65),(14,22),
                       (15,15),(16,48),(17,16),(18,45),(19,19),
                       (20,33)], fillstyle='gray50', width=2)

    pie3   = GraphPie([(0,5),(1,22),(2,8),(3,45),(4,22),
                       (5,9),(6,40),(7,2),(8,56),(9,34),
                       (10,51),(11,43),(12,12),(13,65),(14,22),
                       (15,15),(16,48),(17,16),(18,45),(19,19),
                       (20,33)])
```

```
    pieline4  = GraphLine([(0,21),(1,77),(2,129),(3,169),(4,260),
                    (5,377),(6,695),(7,434)], width=3)
    pielines4 = GraphSymbols([(0,21),(1,77),(2,129),(3,169),(4,260),
                    (5,377),(6,695),(7,434)],
                    marker='square', fillcolor='yellow')

    graphObject1 = GraphObjects([pie1])
    graphObject2 = GraphObjects([pie2])
    graphObject3 = GraphObjects([pie3])
    graphObject4 = GraphObjects([pie1, pieline4, pielines4])

    f1 = Frame(root)
    f2 = Frame(root)

    graph1 = GraphBase(f1, 300, 300, relief=SUNKEN, border=2)
    graph1.pack(side=LEFT, fill=BOTH, expand=YES)
    graph1.draw(graphObject1)

# --- Code Removed ----------------------------------------------------
```

代码注解

❶ 饼图实现给图中的切片分配了一个颜色谱，每一个切片一个颜色，这对于少量的切片来说给出了合理的图像。

```
colors = Pmw.Color.spectrum(len(self.scaled))
```

❷ 在我们令饼图和其他图形同时显示的情况下，这段代码调整了饼图的位置。

```
adj = 0
if multi: adj = 15
xy  = 25+adj, 25+adj, x1-25-adj, y1-25-adj
xys = 25+adj, 25+adj+10, x1-25-adj, y1-25-adj+10
```

如果饼图要显示成一个倾斜的圆盘，则阴影圆盘（xys）将被用到。

❸ 阴影作为一个扇形区绘制出来，几乎具有一个完整的圆形切片（circular slice）。

```
if not x1 == y1:
     canvas.create_arc(xys, start=0.0, extent=359.99,
          fill='gray60', outline=outline, style='pieslice')
```

❹ 与增加直方图的情况相同，增添饼图也要求一个专用的 draw 例程。

❺ 比例因子由在同一控件中是否有多种图形来决定。

❻ self.multiple 传递到了图形对象的 draw 方法中去了。

正如你在这些例子中看到的那样，增加一个新的图形类型是很容易的，而且还能产生一些较为合理的直观的图形。我希望你也能利用它们，并且很有可能为 Python 家族创造出新的视觉样式。

11.3　三维图形

如果你有大量的数据，并且这些数据需要在同一坐标下（同一比例）检验其图形，

此时有很多方法来显示这些图形。其中的一个方法就是把它们一个接一个地并排分立地放在一起。如果你想仔细地检查单个的图形，这不失为一个好方法，但这样就不能显示出这些图形间的联系。为了显示出这种联系，你可以把所有曲线重叠起来用一幅图来表示，并且用不同的记号、线条或者两者都使用来区分。然而这会使曲线纠缠在一起，或者一个记号覆盖了另一个记号，整幅图形混乱不堪。

此时我总是喜欢使用三维图形。它们可以使观众得到对数据的整体拓扑形象，常常还会有其他的形式中不易察觉的特征突出出来。下面的一个例子就说明了这样一个图形（如图 11.7 所示）。其中我为了缩短代码采用了一些简化，例如，我没有考虑更改坐标方位和视角。这一点就留给热情的读者作为练习！

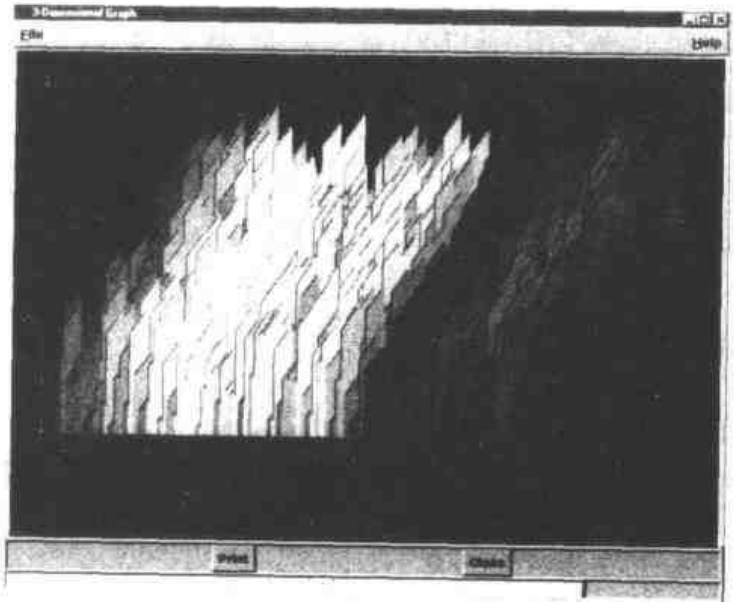


图 11.7　三维图形显示

3dgraph.py

```
from   Tkinter import *
import Pmw, AppShell, math

class Graph3D(AppShell.AppShell):
   usecommandarea = 1
   appname        = '3-Dimensional Graph'
   frameWidth     = 800
   frameHeight    = 650

   def createButtons(self):
       self.buttonAdd('Print',
                      helpMessage='Print current graph (PostScript)',
                      statusMessage='Print graph as PostScript file',
                      command=self.iprint)
```

```
        self.buttonAdd('Close',
                       helpMessage='Close Screen',
                       statusMessage='Exit',
                       command=self.close)

    def createBase(self):
        self.width  = self.root.winfo_width()-10
        self.height = self.root.winfo_height()-95
        self.canvas = self.createcomponent('canvas', (), None,
             Canvas, (self.interior(),), width=self.width,
             height=self.height, background="black")
        self.canvas.pack(side=TOP, expand=YES, fill=BOTH)

        self.awidth  = int(self.width  * 0.68)
        self.ahdight = int(self.height * 0.3)
        self.hoffset = self.awidth  / 3
        self.voffset = self.aheight +3
        self.vheight = self.voffset / 2
        self.hrowoff = (self.hoffset / self.rows)
        self.vrowoff = self.voffset / self.rows
        self.xincr   = float(self.awidth) / float(self.steps)
        self.xorigin = self.width/3.7
        self.yorigin = self.height/3
        self.yfactor = float(self.vht) / float(self.maxY-self.minY)

        self.canvas.create_polygon(self.xorigin, self.yorigin,
             self.xorigin+self.awid, self.yorigin,
             self.xorigin+self.awid-self.hoff, self.yorigin+self.voffset,
             self.xorigin-self.hoff, self.yorigin+self.voffset,
             self.xorigin, self.yorigin, fill='', outline=self.lineColor)

    self.canvas.create_rectangle(self.xorigin, self.yorigin-self.vheight,
         self.xorigin+self.awid, self.yorigin,
         fill='', outline=self.lineColor)

        self.canvas.create_polygon(self.xorigin-self.hoffset-5,
             self.yorigin+self.voffset, text='%d' % self.minY,
             fill=self.lineColor, anchor=E)
        self.canvas.create_text(self.xorigin-self.hoffset-5,
             self.yorigin+self.voffset-self.vheight, text='%d' % \
             self.maxY, fill=self.lineColor, anchor=E)

        self.canvas.create_text(self.xorigin-self.hoffset,
             self.yorigin+self.voffset+5, text='%d' % self.maxX,
             fill=self.lineColor, anchor=N)

def initData(self):
```

```
        self.minY =   0
        self.maxY = 100
        self.minX =   0
        self.maxX = 100
        self.steps = 100
        self.rows =  10
        self.spectrum = Pmw.Color.spectrum(self.steps,saturation=0.8,
                                           intensity=0.8, extraOrange=1)
        self.lineColor = 'gray80'
        self.lowThresh =  30
        self.highThresh =  70

    def transform(self, base, factor):
        rgb = self.winfo_rgb(base)
        retval = "#"
        for v in [rgb[0], rgb[1], rgb[2]]:
            v = (v*factor)/256
            if v > 255: v = 255
            if v < 0:   v = 0
            retval = "%s%02x" % (retval, v)
        return retval

    def plotData(self, row, rowdata):
        rootx  = self.xorigin - (row*self.hrowoff)
        rooty  = self.yorigin + (row*self.vrowoff)
        cidx   = 0
        lasthv = self.maxY*self.yfactor
        xadj   = float(self.xincr)/4.0
        lowv   = self.lowThresh*self.yfactor
        for datum in rowdata:
            lside = datum*self.yfactor
            color = self.spectrum[cidx]
            if datum <= self.lowThresh:
                color = self.transform(color, 0.8)
            elif datum >= self.highThresh:
                color = self.transform(color, 1.2)

            self.canvas.create_polygon(rootx, rooty, rootx, rooty-lside,
                      rootx-self.hrowoff, rooty-lside+self.vrowoff,
                      rootx-self.hrowoff, rooty+self.vrowoff,
                      rootx, rooty, fill=color, outline=color,
                      width=self.xincr)
            base = min(min(lside, lasthv), lowv)
            self.canvas.create_line(rootx-xadj, rooty-lside,
                    rootx-xadj-self.hrowoff, rooty-lside+self.vrowoff,
                    rootx-xadj-self.hrowoff, rooty+self.vrowoff-base,
                    fill='black', width=1)
            lasthv = lowv = lside
```

```
            cidx = cidx + 1
            rootx = rootx + self.xincr

    def makeData(self, number, min, max):
        import random
        data = []
        for i in range(number):
            data.append(random.choice(range(min, max)))
            return data

    def demo(self):
        for i in range(self.rows):
            data = self.makeData(100, 4, 99)
            self.plotData(i, data)
            self.root.update()

    def close(self):
        self.quit()

    def createInterface(self):
        AppShell.AppShell.createInterface(self)
        self.createButtons()
        self.initData()
        self.createBase()

if __name__ == '__main__':
    graph = Graph3D()
    graph.root.after(100, graph.demo)
    graph.run()
```

❹

代码注解

❶ 虽然图形很复杂，不过代码还是很简单的。有很多代码是用于绘制框架和文本标签的。

❷ 读者或许看到了 7.2.1 小节中的“将六边形螺帽加入类库”中用到的 transform 方法。其作用便是，当给定一个颜色时，计算一个或更加明亮或更加暗淡的色度值。

```
def transform(self,base,factor):
```

❸ 变换后的颜色用于突出那些超过了某个上限阈值的数据，同时也使那些低于某个下限阈值的数据不太突出。

❹ 对于这个例子，我们产生了十行的随机数据。

因为数据是随机产生的，故而效果显得很混乱。如果数据有着拓扑特征的话，就会比较平整了。图 11.8 就是这样的数据产生出来的这样一类三维图形。

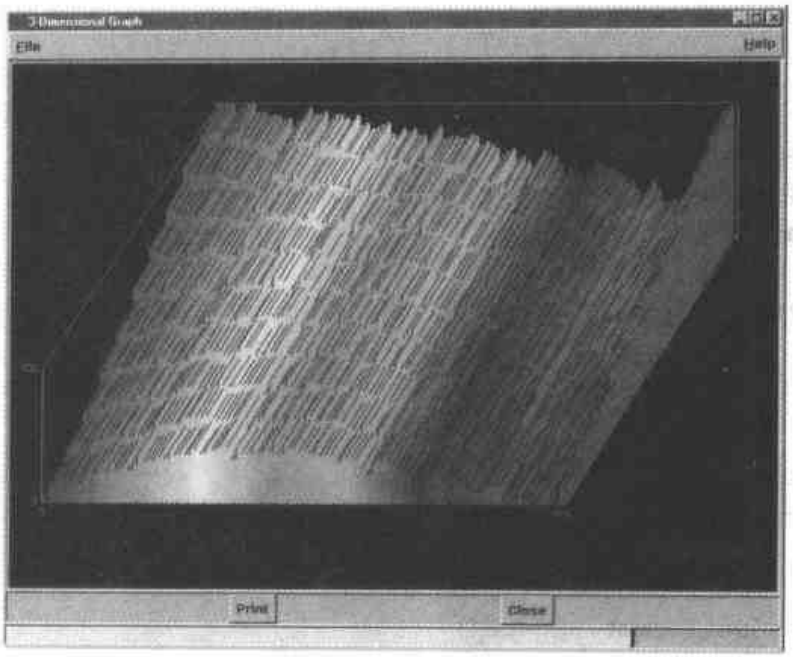


图 11.8　使用三维图形来表达拓扑数据

11.4　带形记录图

在最后一节里，我们将简要地看一下使用带形记录图来显示连续变化的数据。此时通常是在数据产生出来后或是按一定的时间间隔采样后，增量式地绘制图形，然后在空间被填满后重置图形。

带形记录图是显示动态数据的理想工具，这些数据有：传感器中的数据，例如温度、速度或者湿度；更为抽象的测量数据（比如食品店里每个顾客每小时平均购买的商品数），以及其他数据。带形记录图也可以作为设置阈值并在阈值达到时发出警报的工具。

最后一个是利用 METAR 数据启动天气监测系统的例子。这些经过编码的数据可以通过 FTP 由美国的 National Weather Service 得到，也可以从世界上其他类似的机构获得。我们不准备花很长时间去解说如何对 METAR*数据解码，因为这会需要单独的一章。对于本例，我甚至不准备给出源代码（这确实会占用很大的空间）。源代码可以在线查找，并且还可以看到这样一个简单的 FTP 抽样是如何连续地对数据采样的。

如图 11.9 所示，图中给出了大约从东部标准时间上午 8 点，持续约 9 个小时对 Florida 的 Tampa 海湾采集到的数据的处理结果。这些图形描绘了温度、湿度、高度（大气压）、能见度、风速、风向、10,000 英寸以上的云和 25,000 英寸以下的云。

带形记录图的形式就是为了与振荡器或别的类似的装置相似而设计的。通常反显示不是显示数据的最佳工具，不过这也许是一个例外。

例子中的代码实现了阈值的设定，这可使用户能够对在数据高于或者着低于取定的

* 如果你是一个气象爱好者或者是个人飞行员，你就会熟悉这个自动的、编码气象观测数据。在全世界，许多广播台都发送这些数据，包括一些大主要的航空港。基本上每隔一个小时更新一次（在气候变化较快的情况下，会更频繁一些），它们详细地包括了风向和速度、湿度、结露点、气压、云量和其他对于航行来说重要的数据。

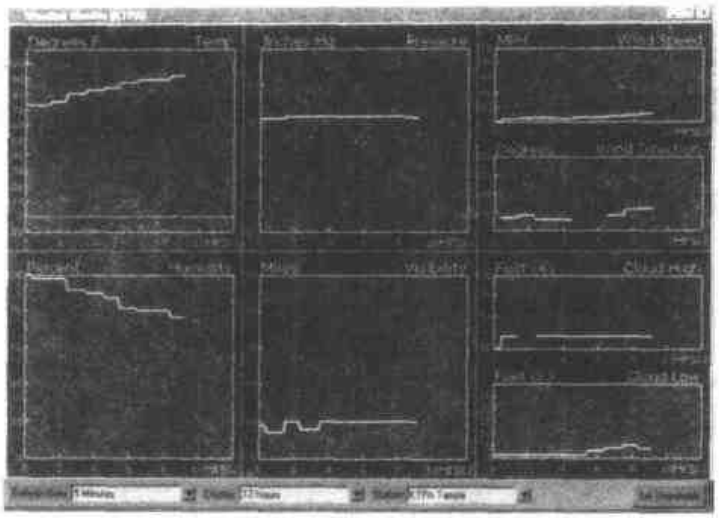


图 11.9　带状图来显示气象数据

阈值时发出警报或警告所用的值进行设定。如图 11.10 所示，其中说明了如何基于数据设置阈值。这些数据来自我的私人机场（Providence，Warwick，Rhode 岛），它显示了一场暴风雨之前的数据。

如果你看一下图 11.11，你就会观察到云层是如何突然下降到 5000 英尺以下，并引

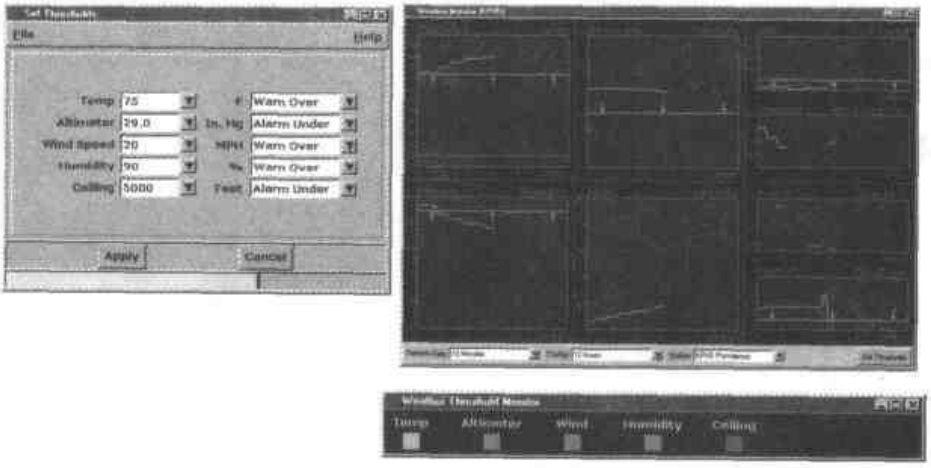


图 11.10　设置数值范围

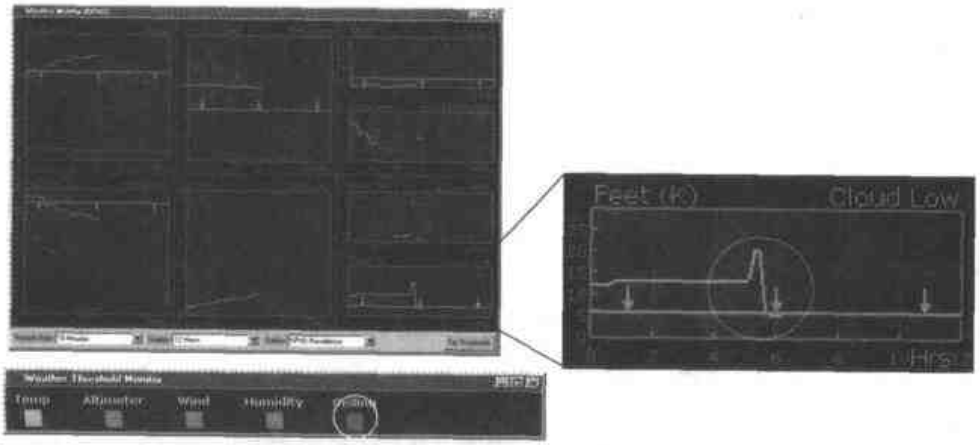


图 11.11　报警和警告阈值

发警报的。

如果你要使用这个例子，请不要把刷新的频率设置得很高。国家海洋和大气管理局（NOAA）网站的数据对于许多飞行员来说很重要——给他们留点带宽吧！

11.5 小结

对于许多应用程序来说绘图并不是很必要的，但是从不同的数据中产生引人注意的效果，这种功能在一些场合下是有用的。尽管有许多通用的绘图系统可以从任意的数据中产生图形，但是自己编写代码还是有不少乐趣的。

第12章 导 航

所有成功的GUI都提供兼容的、方便的图形元素之间的导航工具。在这短小的一章中我们将详尽地叙述各种方法。一些高级的导航方法也将讨论，并引导读者进入该专题。从给出的方法中，你必须找到你的应用程序的正确模式，并且使用相应的方法。

12.1 引言：导航方法

本章全都是关于焦点的。尤其是——键盘焦点——这是属于控件层次的，并且决定键盘事件将被发送到什么地方。窗口焦点严格地说是窗口管理器的一项功能，这将在第13章中的“窗口管理器”中做详尽的讨论。

通常有两种焦点模型：

Pointer 当某个控件包含指针时，所有的键盘事件都指向该控件。

Explicit 用户必须点击控件使由一个控件转向另一个控件，从而使焦点落到某个特定的控件上。

这两种模型都各有优缺点。许多用户喜欢指针模型，这是由于一个简单的鼠标移动就可以给指针定位，并且不需要点击。然而，如果鼠标意外地挪动了，键盘事件就会被指向一个不相干的控件了。显式型模型则需要用户点击每一个焦点将要指向的窗口和控件。但是，即便是指针挪到了别的控件上，键盘事件还是指向原来的控件上。在Win32和MacOS上，缺省使用的是显式型焦点，而在Unix下，缺省使用的是指针焦点。

Tk（当然也包括Tkinter）在一个给定的顶层外壳中使用的是显式型模型，而不论窗口管理器是如何配置的。然而，在一个应用程序里，窗口管理器能够越过焦点去处理系统层的操作，因此，焦点会丢失。

12.2 鼠标导航

大多数的电脑用户习惯于使用鼠标在屏幕上选择操作和目标。然而，有的时候不用键盘输入而只移动鼠标并不方便，一个习惯了敲击键盘的打字员会找不到位置，并且在继续之前得先恢复。因此，给用户提供一系列合理的跳转组（Tab group）使得用户可以在GUI中从一个控件移动到另一个控件，或着从一处移动到另一处，这常是一个很好的想法。在下一节中对此将做详细的讨论。

当鼠标用于选择目标，你必须考虑一些重要的人为因素：

1. 控件的对齐方式是很重要的。如果屏幕上控件毫无目标地排列着，那么在对鼠标定位时就需要多加小心了。如果控件按行和列放置，通常在一个或两个坐标轴里的移动就可以给指针重新定位了。

2. 一个控件的可点击元素需要很多，以保证用户不需要在想点击一个目标时先对指针重新定位。不过，这会干扰 GUI 的图像效率的。

3. 控件与控件之间必须留出空间来，这样用户就不会无意中选择一个相邻的控件了。

4. 记住不是所有的指向设备都是相同的。鼠标在使用上会比较容易地把指针指向一个可点击区域，然而，一些鼠标钮（在某些记事本电脑的键盘上装有的小按钮）则会比较难于控制。

除非 Tkinter（Tk）被指定了遵循严格的 Motif 规则，否则它就有一个很好的特性，这使得它在当指针进入到许多控件中后，可以更改这些控件的视觉属性。于是可以给用户提供很有价值的反馈，即某控件被定位了。

12.3 键盘导航："无鼠标导航"

人们很容易忘记提供其他可供选择的导航方法。如果你作为一个编程者，已经习惯了使用鼠标来指向焦点，你就会忽略了无鼠标导航。然而，有的时候具有使用 Tab 或 Arrow 键来遍历一个 GUI（系统）的功能也是很重要的。也有的时候鼠标不存在（我就碰到过这个问题，一只猫睡在我的电脑桌上，它的毛发干扰了鼠标！），或者一个应用程序是配置在一个不好的环境下，使得鼠标无法使用。

这个导航方法对你的 GUI 是如何创建的有一些规定。焦点在组与组之间以及组内移动的次序由控件创建的次序所决定。因此不注意对 GUI 的维护，就会导致不确定的行为，而焦点就会在整个屏幕上跳转。此外，有些控件不能接受焦点（比如它们已经被禁止了），而其他一些则在内部捆绑了导航键（例如 Text 控件就允许你键入 Tab 字符）。

有一点是必须要记住的，那就是指针始终指向相应于其下方控件的事件（比如 Enter、Leave 和 Button1），与键盘焦点被设置在什么地方无关。因此，如果一个控件本身不具有键盘焦点，那么你就有可能需要更换焦点到控件上。

12.4 建立应用程序的导航

让我们来看一个简单的例子，这个例子可以使你了解控件在焦点模型中以及某种特定的状态下是如何工作的。如果控件的 takefocus 选项置为 true，则该控件将被放到窗口的 Tab 组里，并且在按下 Tab 键后焦点从一个控件转向下一个。如果控件的 highlightthinkness 至少有一个像素，则你会看到现在是哪一个控件具有焦点。应该注意的是在 Tkinter 中，导航模型的可控制性有些差，一般说来这不是一问题，但是 X Windows 的编程者们会发现这种限制的缺陷。

Example 12 1 py

```
from Tkinter import *

class Navigation:
    def __init__(self, master):

        frame = Frame(master, takefocus=1, highlightthickness=2,
                      highlightcolor='blue')
        Label(frame, text='   ').grid(row=0, column=0,sticky=W)
        Label(frame, text='   ').grid(row=0, column=5,sticky=W)

        Self.B1=self.mkbutton(frame,'B1',1)
        Self.B1=self.mkbutton(frame,'B2',2)
        Self.B1=self.mkbutton(frame,'B3',3)
        Self.B1=self.mkbutton(frame,'B4',4)

        frame2 = Frame(master, takefocus=1, highlightthickness=2,
                       highlightcolor='green')
        Label(frame2, text='   ').grid(row=0, column=0,sticky=W)
        Label(frame2, text='   ').grid(row=0, column=4,sticky=W)
        Self.Disable=self.mkbutton (frame2,'Disable', 1, self.disable)
        Self.Enable =self.mkbutton (frame2,'Enable',  2, self.enable)
        Self.Focus  =self.mkbutton (frame2,'Focus',   3, self.focus)

        frame3 = Frame(master, takefocus=1, highlightthickness=2,
                       highlightcolor='yellow')
        Label(frame3, text='   ').grid(row=0, column=0,sticky=W)
        Label(frame2, text='   ').grid(row=0, column=4,sticky=W)
        self.text = Text(frame3, width=20, height=3,highlightthickness=2)
        self.text.insert(END, 'Tabs are valid here')
        self.text.grid(row=0, col=1, columnspan=3)

        frame.pack(fill=X, expand=1)
        frame2.pack(fill=X, expand=1)
        frame3.pack(fill=X, expand=1)

    def mkbutton(self,frame,button,column,action=None):
        button=Button(frame,text=button,highlightthickness=2)
        button.grid(padx=10,pady=6,row=0,col=column,sticky=NSEW)
        if action:
            button.config(command=action)
            return button

    def disable(self):
        self.B2.configure(state=DISABLED, background='cadetblue')
        self.Focus.configure(state=DISABLED, background='cadetblue')

    def enable(self):
        self.B2.configure(state=NORMAL,background=self.B1.cget('background'))
```

```
    def focus(self):
        self.Focus.configure(state=NORMAL,
                             background=self.B1.cget('background'))
    def focus(self):
        self.B3.focus_set()

root = Tk()
root.title('Navigation')
top = Navigation(root)
quit = Button(root, text='Quit', command=root.destroy)
quit.pack(side =BOTTOM,pady=5)

root.mainloop()
```

代码注解

❶ 为了显示键盘的焦点在什么位置，我们必须给出提示区的尺寸，因为 Frame 的缺省值是 0。color 也一样要设置，这样就可以很容易辨认出来了。

❷Text 控件也同样需要对 highlightthickness 设置。Text 控件所在的控件类不能传递 Tab 字符，因此你不可能使用 Tab 键来导航到 Text 控件以外（你必须使用 Ctrl+Tab）。

❸Buttons 与窗口系统有关。在 Win32 中，按钮用虚线显示其提示区，而 Motif 控件则需要你设置提示区的宽度。如果你的应用程序仅仅是针对 Win32 的，则你可以省略 highlightthickness 选项。

让我们运行例 12.1 的代码，来看一下 Tab 键的导航是如何工作的：

每一次你按下 Tab 键，焦点都会移向组内下一个控件。如果需要返回，则可以使用 Shift-Tab 键。在图 12.1 的第二个画面中，你可以看到其中显示了一个提示区。在 Tkinter 中，如果你确定了控件在几何管理器中按照你所预想的导航顺序排列，则 Tkinter 将会按照这个顺序来放置这些控件。如果你没有注意，则你将会在试图导航时使得焦点在 GUI 中跳一遍。

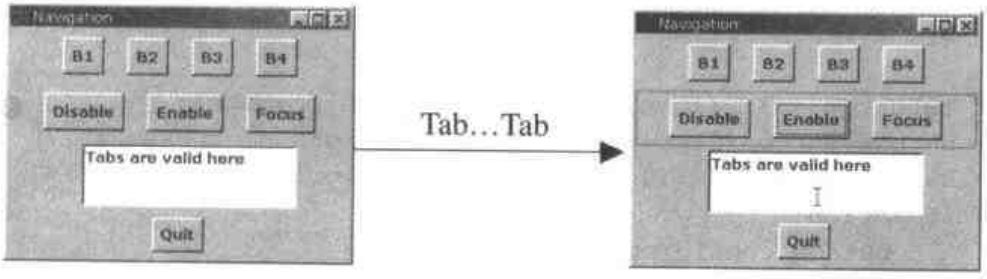


图 12.1　使用 Tab 键选择框架

一旦你得到了焦点，你就可以用空格键（无疑对于 Win32 和 Motif 来说，这就是缺省的选择键）或者用指针光标点击来激活控件。注意到在图 12.2 中，Enable 按钮表明，我们使用 Tab 键，它已经被导航到了，但是我们用指针光标选择 Disable。键盘焦点仍停留在 Enable 按钮上，因此按下空格键便会使按钮又可以使用。



图 12.2 键盘与指针光标焦点差异示范

Text 控件使用 Tab 键作为分隔符，因此默认的捆绑不会引起遍历到控件以外。在图 12.3 中你可以看到 Tab 被插入了文本中。为了能够移动到控件外，我们必须使用 Ctrl-Tab，因为这并没有捆绑到 Text 控件上。

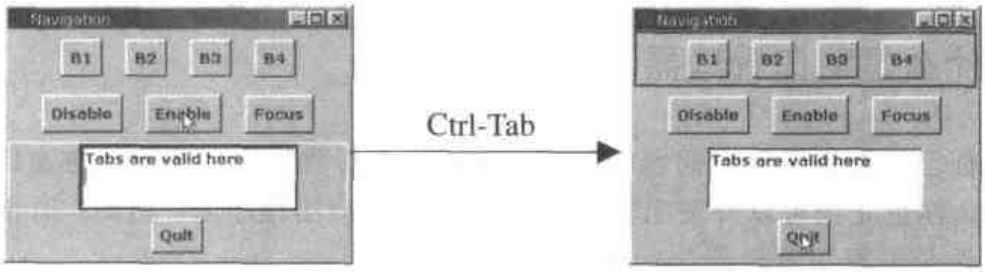


图 12.3 使用 Ctrl-Tab 来在文本控件中进行切换

12.5 图像映射

在 8.7 节的“图像映射”中，我们看了图像映射的一个实现。在那里使用了指针光标点击来检测图像中的区域并且选择映射区域。这项技术不能支持无鼠标的操作，因此如果是必要的话，你就需要做些事了。

其中的一个解决办法是，用可以获得焦点的对象来覆盖该图像，这样你就可以从一个对象跳到另一个对象了。这确实是可以的（9.6 节上的应用就是使用了这项技术来用按钮覆盖光迹图像上的按钮图像），不过这会需要很多的考虑和代码。

12.6 小结

本章又一次地说明了这样一个事实，即一项应用程序的开发者应该仔细地考虑到终端用户。如果你仅是在为建立在单一硬件平台上的一组用户开发一项应用程序，则可以不需要考虑提供导航 GUI 的替换方法。然而，如果你不能预测终端用户的环境，但如果能提供备用的导航模式，你将会在很大程度上改变他们对你的应用程序的看法。

第13章　窗口管理器

尽管可以不需要与窗口管理器直接通信，就可以建立一个应用程序，但是对窗口管理器在 X Window、Win32 和 MacOS 环境中的地位的了解还是会很有用处的。本章会讲到一些现有的 wm 应用程序以及使用它们的例子。

13.1　什么是窗口管理器

如果你已经知道了这个问题的答案，也许你会想跳过去。即使你对窗口管理器一无所知，你也完全可以开发出复杂的基于 GUI 的应用程序来。然而，有很多显示 GUI 的属性是由窗口管理器决定的。

窗口管理器对每一个操作系统都有相应的形式，举例来说有 mwm（Motif）、dtwm（CDE）和 ovwm（OpenView）。对于 Win32 来说，窗口管理器更是操作系统的一个组成部分，而不是一个单独的应用程序。窗口管理器基本上都支持的主要功能如下：

1. 对窗口的管理（显然的！），例如布局（placement）、调整大小（sizing）、图标化（iconizing）和最大化。
2. 在一个应用程序中，窗口的外观和行为，以及窗口间的关系。
3. 对一个或多个屏幕的管理。
4. 键盘焦点管理。
5. 窗口的装饰：标题、控件、菜单、尺寸和位置的控制。
6. 图标以及图标管理（例如 iconboxes 和 system tray）。
7. 整体的键盘捆绑（在应用程序的捆绑之前）。
8. 整体的鼠标捆绑。
9. 根窗口菜单。
10. 窗口层积和导航。
11. 默认行为和客户资源（.XDefaults）。
12. 通过窗口原语协调（几何管理器）尺寸及位置。
13. 设备配置，例如鼠标的双击时间、键盘重复和移动的门限。

不是所有的窗口管理器都支持相同的功能或者运作相同。然而，Tkinter 支持很多与窗口管理器相关的功能，它们会支持你的应用程序。很自然的，这些功能的名字是 Tk 特有的，因此你不会立刻认出其他的窗口管理器名字。

13.2　几何方法

几何方法用于设定窗口的位置和尺寸，以及对调整大小行为进行设置。非常重要的一点是，这些是窗口管理器可以分配某一给定量的空间，以及在屏幕上某一特定位置开窗口所必需的。但这并不能保证窗口管理器会注意到这个请求，因为一些尤为重要的因素会妨碍它。总的说来，如果你在使用几何方法时没有得到明显的效果，那么你很有可能是在做窗口管理器不能授权的事或者是在一个不合适的时间对它请求（要么是在窗口实现之前，要么就是太晚了）。

通常你是把窗口管理器方法用于 TopLevel 控件中。

用 geometry 可以控制窗口的尺寸和位置，参数为一个字符串，格式如下：

```
widthxheight+xoffset+yoffset
root.geometry('%dx%d+%d+%d' % (width, height, x, y))
```

注意，如果你想设置的参数是 widthxheight 或者+xoffset+yoffset 的话，你也可以把它们当作独立的参数。

毋庸置疑，self.geometry()返回一个如上格式的字符串。

注意　通常说来，在窗口第一次绘制的时候，你应该最多只发出一次 geometry 请求。在程序进行控制的时候，调整窗口的位置是不好的；这类定位应该是留给用户用窗口管理器控制来决定的。

设置窗口的最大和最小的尺寸常常是一个好主意。如果你设计的应用程序有很复杂的布局，那么给用户提供调整窗口大小的功能也许就不合适了。其实，如果你不限制这项功能的话，就很难保证 GUI 的完整性。然而，使用 Pack 或者 Grid 几何管理器的 Tkinter 的 GUI 比相对应的 X window 的 GUI 要容易配置。

```
Window.maxsize(width,height)
Window.minsize(width,height)
```

如果 window.minsize()和 window.maxsize()没有参数，则返回元组(width, height)。

你可以通过 resizable 方法控制调整大小的能力。这个方法设有两个布尔标志位；置 width 或者 height 的标志位为 false 就可以阻止相应的宽或者高不能调整大小。

```
Resizable(1,0)      #只允许改变宽度
Resizable(0,0)      #l 两个坐标方向都不允许改变
```

13.3　可见性方法

窗口管理器一般都提供给窗口加图标的功能，这样用户就不会在工作的时候茫然一片了。通常由程序控制来改变窗口的状态是合适的。例如，如果用户请求的窗口当前被图标化，我们可以替它 deiconify 这个窗口。一般来说最好在绘制一个复杂的 GUI 系统时隐藏窗口，其中的原因将在第 17 章的“性能编程”上解释；这样可以获得较快的生成

窗口速度。

用 iconify 方法给窗口图标化：

```
root.iconify()
```

用 withdraw 方法隐藏窗口：

```
self.toplevel.withdrew()
```

你可以用 state 方法查询当前窗口的状态。它会返回下列字符串中的一个：normal（窗口正在被实现）、iconic（窗口已经被图标化了）、withdrawn（窗口隐藏了）或者 icon（窗口为一个图标）。

```
State=self.toplevel.state()
```

如果一个窗口已经被图标化或者隐藏了，你可以使用 deiconify 方法恢复窗口。对当前正在显示的窗口取消图标化，这并不是错误的。不过一定要调用 lift 使得窗口被放在窗口堆栈的栈顶。

```
self.deiconify()
self.lift()
```

13.4 图标方法

图标方法应该说仅对 X Window 的窗口管理器有用。对于其他许多窗口管理器来说你对图标的控制很少。

使用 iconbitmap 可以设置一个双色图标：

```
self.top.iconbitmap(myBitmap)
```

如果要给图标取一个与窗口标题不同的名字的话，可以使用 iconname：

```
self.top.iconname('Example')
```

你可以给窗口管理器提示你想把图标放置在什么位置（然而，如果定义了一个 iconbox 的话，窗口管理器就可能把图标放在里面，否则放在需要的地方）：

```
self.root.iconposition(10,200)
```

如果你想要一个彩色位图，你必须先建立一个带图像的 Label，然后使用 iconwindow：

```
self.label=label(self,image=self.img)
self.root.iconwindow(self.label)
```

13.5 协议方法

遵守 ICCCM*协定的窗口管理器支持许多协议：

- WM_DELETE_WINDOW　窗口将要被删除。
- WM_SAVE_YOURSELF　保存客户数据。
- WM_TAKE_FOCUS　窗口刚获得焦点。

---

* ICCCM 即 Inter-Client Communication Conventions Manual（客户间通信约定）的缩写，这是在 X 环境下客户通信的手册，该约定从属于 Unix 系统，但是 Tk 模仿所有平台上的性能。

当用户没有使用应用程序的结束按钮，就选择了退出窗口菜单选项并且注销了窗口，则你通常要使用第一个协议来清除你的应用程序。

Self.root.protocol(WM_DELETE_WINDOW,self.cleanup)

在 Python 1.6 中，WE_DELETE_WINDOW 协议将自动捆绑在窗口的 destroy 方法中。

WM_SAVE_YOURSELF 就比较少碰到了，并且通常是在 WM_DELETE_WINDOW 发出之前发出的。在获得焦点时，WM_TAKE_FOCUS 可以被一个应用程序用于允许作某个特殊的操作（举例说，也许在窗口获得焦点时，更多是在执行一个轮询周期）。

13.6 其他 wm 方法

现在有许多种窗口管理器方法，而其中的很多都不需要了解。它们都整理在附录 B 中的“继承关系”上了。不过你会发现其中的一些很有用。

要把一个窗口在窗口堆栈中升或者降，可以使用 lift 或者 lower（你不能使用"raise"，因为这是 Python 的关键字）：

```
self.top.lift()              #放到栈顶
self.top.lift(name)          #放到栈中'name'上面
self.top.lower(self.spam) #放到栈中正好在 self.spam 之下
```

要查出你的窗口是在哪一个屏幕上（这仅对 X Window 有用），可以使用：

```
screen=self.rootscreen()
print screen
: 0.1
```

注意　Win32 浏览器　返回的数字指向显示中的 display 和 screen。X window 能够支持在同一系统下的多种显示。

第3部分

组合起来

第 3 部分讲述一些专题。注意所有的专题都直接和 Tkinter，甚至是和 Python 有关，而且很必要。我们在第 14 章中开始讲述对 Python 建立扩展。扩展是给 Python 增加新功能的一种有效手段，并且可以用来对你所用的本地 Python 增加纠错能力。

第 15 章讲述在 Python 和 Tkinter 中的调试技术，第 16 章则是关于 GUI 设计的。这些对于编程者来说会有些疑问，特别是那些 GUI 应用程序的新手们。

第 17 章讲述对应用程序的优化技术。这一块很可能对你所设计的界面在用户心目中的形象是好是坏产生奇妙的影响。

在第 18 章我们将考察多线程和其他的异步技术。这仅是一个入门性的介绍，但是它给那些对此感兴趣的读者一个起点。

最后在第 19 章我们罗列了一些给 Python 和 Tkinter 应用程序包装和发布的方法。虽然它不是毫无遗漏的，但还是可以帮助程序员，使得其所开发的 Python 应用程序能让终端用户不在对复杂的系统进行设置上费劲。

第 14 章　Python 扩展

Python 可以很容易地就使用一个明确的界面进行扩展以及建立模式。本章我将说明如何对外部系统构建界面，如同已经存在的扩展一样。另外，还要讲述不同的 Python 修订版间如何保留扩展的问题。我们还要讨论是在一个应用程序中嵌套 Python，还是开发一个独立 Python 应用程序的选择问题。

Python 所附带的文档对于这个专题来说是很好的资料，因此本章将尽量不只是做重复。我的目标是展示最重要的地方，使你可以很容易地构建界面。

14.1　写 Python 扩展

许多 Python 的程序员可能永远都没有必要去扩展 Python；现有的库模块就可以满足相当多的需求了。因此，除非有特殊的要求，否则标准的二进制发布就足够了。

有的情况下，在 C 或者 C++中做扩展是必要的：

1. 当计算的要求（例如高强度的数值运算）使得 Python 代码效率低时。
2. 当需要访问第三方软件时。这可以通过一些 API 或者是别的更为复杂的界面*。
3. 为那些没有给 Python 提供转换的传统软件提供入口。
4. 通过类似于上面第 2 和第 3 条的机制控制外部设备。

注意　对 Python 进行扩展需要有 Python 解释器的全部资源，以及 C 和 C++的入口（如果你想在 Windows 下操作，我建议你使用微软的 Visual C++ 5.0 版或者更高的）。除非你想对接一个库的应用程序接口或者有严重的执行问题，否则你都会想要完全避免去构建一个扩展。参阅第 17 章 “性能编程”，上面介绍了提高 Python 代码性能的办法。

让我们先来看一个关于把 C 的 API 连接到 Python 的例子（对于 C++的例子以后再说）。为了简化，我们假设这个 API 实现几个统计功能。其中的一项功能是对所提供的四个实数计算其最小值、平均值和最大值。在 Python 的代码中，我们将把这些值作为离散参数，并且返回一个元组。这看起来是一个很琐碎的例子，但不要担心——这仅仅是一个简单的例子！

* 如果你需要把 Python 接口提供给图书馆，你有可能想看一下 SWIG（请参看附录 G 中的 “SWIG”），在那里提供了一个建立接口的非常方便的途径。有的情况下，SWIG 也有可能仅由一个包含文件发展出一个接口。这当然节约了劳力和时间。

下面就是我们对 Python 这边想要做的了：

```
import statistics
....
minimum,average,maximum=statistics.mavm(1.3,5.5,6.6,8.8)
....
```

我们开始时先建立文件 statisticsmodule.c。对扩展模块大家所接受的命名习惯是 module module.c（在以后如果我们想建立动态装载模块这会是很重要的）。

所有的扩展模块都必须通过包含 Python.h 来调用 Python 的应用程序接口。这也包括一些标准 C 头文件，比如 stdio.h、string.h 和 stdlib.h。这样我们就定义了支持我们的 API 的 C 函数。

源代码大概是这样的：

statisticsmodule1.c

```
#include "Python.h"
static PyObject *
stats_mavm(self, args)
    PyObject *self, *args;
{
   double         value[4], total;
   double         minimum =  1E32;
   double         maximum = -1E32;
   int            i;

  if (!PyArg_ParseTuple (args, "ffff", &value[0], &value[1],
                                       &value[2], &value[3]))
     return NULL;

   for (i=0; i<4; i++)
   {
      if (value[i] < minimum)
          minimum = value[i];
      if (value[i] > maximum)
          maximum = value[i];
      total = total + value[i];
   }
   return Py_BuildValue("(fff)", minimum, total/4, maximum) ;
}
static PyMethodDef statistics_methods[] = {
       {"mavm",     stats_mavm,METH_VARARGS,  "Min, Avg, Max"},
       {NULL, NULL}};

DL_EXPORT(void)
initstatistics()
{
Py_InitModule("statistics", statistics_methods);
}
```

代码注解

❶ 正如前面提到的，所有的扩展模块都必须包含有对 Python API 的定义。

❷ 所有的接口项目都是 Python 的对象，因此我们定义的函数要返回一个 PyObject。

❸ 同样地，instance 和参数 args 都是 PyObject。

❹ Python API 提供了一个分析参数的函数，它可以把 Python 的对象转换为 C 的实体：

```
if (!PyArg_ParseTuple (args,"dddd",&value[0],&value[1],
                                                &value[2],&value[3]))
        return NULL;
```

PyArg_ParseTuple 使用所提供的格式串来分析对象 args。可以使用的选项在 14.5 节的“格式化字符串”上做了解释。注意，你必须提供被分析的值将要被赋给的变量的地址。

❺ 在这里我们处理我们的数据。正如你会注意到的，这确实是很困难！

❻ 使用与 PyArg_ParseTuple 类似的方式，Py_BuildValue 由 C 的实体来创建 Python 的对象。格式的含义相同。

```
return Py_BuildValue("(ddd)",minimum,total/4,maximum)
```

对这个例子，我们创建了一个元组（tuple）作为返回的对象。

❼ 所有的接口模块都必须定义一个方法表，其作用是把 Python 的函数名同对等的 C 函数联系起来。

```
Static PyMethodDef statistics_methods[]={
        {"mavm",    stats_mavm,METH_VARARGS,   "Min,Avg,Max"},
        {NULL,NULL}};
```

METH_VARARGS 定义了参数是如何提交给分析器的。文档域是可选的。注意方法表上的命名规则。虽然可以使用任意的名字，但最好还是依照规则命名。

❽ 方法表必须在一个初始化函数中向解释器注册。当模块第一次引入时，函数 initstatistics()被调用。

```
DL_EXPORT(void) initstatistics()
{
   Py_InitModule("statistics",statistics_methods);
}
```

再一次强调，命名规则必须遵守，因为 Python 会对每一个导入的模块试图调用 initmodulename。

14.2 建立 Python 扩展

在你可以使用 Python 扩展之前，你必须先编译和链接。这里你可以有多种选择，这取决于目标系统是 Unix 还是 Windows（对不起，这里没有考虑到 Macintosh 扩展）。

基本上，我们可以使模块成为 Python 解释器的一个驻留部分，这样就可以随时调用了，或者我们也可以动态地链接。不是所用的系统都有动态链接的，但是对于许多 Unix 系统和 Windows 系统来说确实比较好用。动态装载的好处是你不需要为扩展 Python 来修改解释器了。

14.2.1 在 Unix 下静态链接一个扩展

在 Unix 下静态地链接一个扩展是非常简单的。如果你没有经过配置和构建的 Python，可以按照附录 D 上的“创建与安装 Python，Tkinter”配置和构建。把你的模块（本例中的 statisticsmodule.c）拷贝到 Modules 目录下。然后，直接加在 Modules/Setup.local 后面（如果你愿意的话，你也可以添加一些注解）。

如果你的模块需要额外的库，例如 API，则在行末加上-lxxx 标记。注意，只有在前一个模块通过包含*shared*标志而构建为分享模块时，才需要有*static*标志。

```
*static*
statistics statisticsmodule.c
```

现在，只用在顶层 Python 目录下调用 make 就可以重构 python 使得其能在该目录下运行。

14.2.2 在 Windows 下静态链接一个扩展

在 Windows 下静态链接一个扩展比 Unix 下要麻烦些，但是如果你一步一步地来还是比较容易的。如果你现在还没有构建 Python，那就先按照附录 D 的“创建与安装 Python，Tkinter”上说的去做。

首先，编辑 PC/config.c。你会看到一个注解：

```
/* --ADDMODULE MARKER 1 --*/
extern void PyMarshal_Init();
extern void initimp();
extern void initstatistics();
extern void initwprint();
```

给 init 函数添加一个外部索引。然后找到第二个注释：

```
/* -- ADDMODULE MARKER 2 -- */
    /* this module"lives in "with marshal.c*/
    {"marshal",PyMarshal_Init},
    /* This lives it with import.c */
    {"imp",initimp},
    /* Statistics module (P-Tk-P) */
    {"statistics",initstatistics};
    /* window Print module */
    {"wprint",initwprint},
```

添加上模块名以及它的 init 函数。

接下来，编辑 PC/python15.dsp。在靠近最后的地方你就会看到 typeobject.c 的入口。

```
SOURCE=..\Objects\typeobject.c
# End Source File
#
# Begin Source File
SOURCE=..\modules\statisticsmodule.c
# End Source File
#
# Begin Source File
SOURCE=..\Modules\wprintmodule.c
# End Source File
```

把这些行插入 statisticsmodule.c 中。

最后，在 VC++中打开 PCbuild/pcbuild.dsw 工作台，选择合适的配置（见附录 G 的“创建与安装 Python，Tkinter”）并且构建工程。

14.2.3 在 Unix 下构建动态模块

生成可动态装载模块的样式有很多种。这里我仅给出一种对 Solaris 适用的方法，不过所有的由 SVR4 派生出来的 Unix 系统应该都提供了类似的接口。所有的工作都是在 make 程序的描述文件（makefile）中完成的，因此不需要修改代码。与静态链接时相同，首先应该构建和安装 Python 使得库和其他的一些项目都到位。

Makefile dyn

```
SRCS=        statisticsmodule.c
CFLAGS=      -DHAVE_CONFIG_H          ❶
C=           cc

# Symbols used for using shared libraries
SO=              .so
LDSHARED=        ld -G                ❷

OBJS=         statisticsmodule.o
PYTHON_INCLUDE= -I/usr/local/include/python1.5 \
                -I/usr/local/lib/python1.5/config          ❸

statistics: $(OBJS)
            $(LDSHARED) $(OBJS)
            -Bdynamic -o statisticsmodule.so               ❹

statisticsmodule.o:    statisticsmodule.c
                       $(C) -c $(CFLAGS) $(PYTHON_INCLUDE) \   ❹
                       statisticsmodule.c
```

代码注解

❶ CFLAGS 定义为 HAVE_CONFIG_H（在其他情况下，这定义了动态装载的模式），并不是所有的结构都需要，但往往包含这个定义。

❷ LDSHARED 定义为 ld 产生共享变量所必需的命令行，这将随着不同的结构而变化。

❸ PYTHON_INCLUDE 定义为 Python.h 和已安装的 config.h 的路径。

❹ 链接的目标或许需要库函数来提供更复杂的模块，-lxxx 标志要紧靠在$(OBJS)的后面。

❹ 编译的规则相当简单，仅仅需要加入 CFLAGS 和 PYTHON_INCLUDE 变量。

14.2.4 在 Windows 下构建动态模块

另外，在 Windows 下构建动态模块是非常棘手的，需要你编辑一些诸如包含 DO NOT

EDIT 之类注解的文件。但是尽管如此，它还是起作用的！因为带有静态链接，所以要先构建和安装 Python，这样这些库函数和其他项目才会被安装在适当的位置上。

首先在顶层 Python 目录下产生一个新的目录，并且和模块、语法分析器等在同一等级，把该目录和你的模块取为相同的名字。

下一步把支持你的模块所必需的文件拷贝到当前的目录，在本例中我们仅仅需要 statisticsmodule.c。

接下来在标准 Python 发布程序的 PC 机版本下，将有一个 example_nt 目录。把 example.def、example.dsp、example.dsw 和 example.mak 拷贝到模块目录下，并且根据你的模块名称来把这些文件的前缀重命名。

编辑每一个文件，把对 example 的引用改为模块的名称，大概有 50 多处需要修改。当你改变时，要注意 Python 库函数的路径（本例中为 python15.lib）。如果这与你已经安装的库函数不相符合，那么最好的修改方法就是先把它从项目中删除，再重新加入。

最后，选择 Debug 或者 Release 配置，选择 Build 菜单选项来编译 statistics.dll 以构建 dll。

14.2.5 安装动态模块

为了安装动态模块，你需要完成以下三件事情之一：你可以把 module.so 或者 module.dll 放置在任何被 PYTHONPATH 环境变量定义了的地方，你也可以在运行时把它的路径加入到 sys.path 的列表，或者把它拷贝到已安装的.../python/lib/lib-dynload 目录。三个方法可以达到相同的效果，但我通常在 Windows 下把 dll 文件和 python.exe 一起放置，在 Unix 下把.so 文件放到 lib-dynload 目录。你可以做自己的选择。

14.2.6 使用动态模块

操作和使用带有解释器的静态链接的模块和动态链接的模块是没有区别的。但是如果忘记把文件放到合适的位置或者没有把路径加入 PYTHONPATH，你将会碰到错误。

```
Python 1.5.2b2 (#17,Apr 7 1999,13:25:13) [C] on sunos5
Copyright 1991-1995 Stichting Mathematisch Centrum,Amsterdam
>>> import statistics
>>> statistics.mavm(1.3,5.5,6.6,8.8)
(1.3,5.55,8.8)
>>>
```

14.3 在扩展中使用 Python API

前面例子中的 mavm 例程实在太乏味了，让我们把输入改为列表，并在其上进行相同的操作。

Statisticsmodule2.c

```
#include "Python.h"
```

```
static PyObject *
stats_mavm(self, args)
    PyObject *self, *args;
{
   double          total     =   0.0;
   double          minimum   =  1E32;
   double          maximum   = -1E32;
   int            i, len;
   PyObject       *idataList;    = NULL;
   PyFloatObject *f     = NULL;                          ❶
   double         df;

   if (!PyArg_ParseTuple (args, "O", &idataList))        ❷
      return NULL;

   /* check first to make sure we've got a list */
   if (!PyList_Check(idataList))
   {                                                     ❸
      PyErr_SetString(PyExc_TypeError,
                 "input argument must be a list");
      return NULL;
   }
   len = PyList_Size(idataList);                         ❹

   for (i=0; i<len; i++)
   {
      f = (PyFloatObject *)PyList_GetItem(idataList, i);  ❺
      df = PyFloat_AsDouble(f);                           ❻
      if (df < minimum)
         minimum = df;
# ----------------------Remaining Code Removed ----------------------
```

代码注解

❶ 我们需要为那些被作为参数而传送的列表定义 Python 对象和为接收列表项而定义 PyFloatObject 对象（PyObject 的子类型）。

❻ 现在我们接收一个单独的对象（与之相反，前面的例子中使用离散值）。

❷ 我们检查对象确实在列表中，这实际上带来一个不足的地方——我们不能传送一个带有值的数组。如果有错误，我们使用 PyErr_SetString 来产生 PyExc_TypeError，给出特定的错误信息。

❹ 列表对象有 length 属性，所以我们能够得到它。

❺ 我们得到列表中的每一项。

❻ 对于每一项，我们把 PyFloat 转变为 C 的 double 型。其余的代码被省略了，因为前面已经看到过了。

上面的例子仅仅触及了 Python API 所能完成功能的很少一部分，一个发现它的使用方法的好地方是 Python 源目录下的 Modules 目录。这个主题很重要的一个原因是 Python

易于产生和处理字符串，特别是它们当中包括列表、数组或者字典。一个现实的情景就是使用 Python 产生这样的数据结构，接下来 C 调用进一步处理接口，也可以在迭代程序中使用 API 调用接口。使用这种方法，C 可以提供重要操作所需的速度，同时 Python 能够提供简洁的处理数据的能力。

14.4 在 C++中构建扩展

Python 是基于 C 语言的解释器。虽然调整源代码可以使其作为 C++来编译，但这将是很繁重的一个任务。它意味着从 C 的基础来调用 C++函数会引入一些特殊的问题。然而如果你能够在 C++的编译器（链接器）内链接 Python，问题会减少。

很清楚，许多 C++类库能够支持 Python 系统，技巧在于保持 Python 基本不被改变，并提供进入类库的接口。如果你能够对扩展模块使用动态链接，这将是一个毫无痛苦的经历。如果你必须静态链接，你可能要面对一些挑战。

因为每一个结构都涉及 C++编译器的巨大可变性，我将不会试着提供一个解决各式各样问题的大杂烩。但是，我会展示一些能够工作在 Solaris 上的代码段。

为了得到使用 C++来编译的模块，你必须定义 Python API 作为 C++编译器的一个 C 代码段：

```
extern "C" {
# include "Python.h"
```

接下来，init 函数必须给予同样的处理：

```
extern "C" {
  DL_EXPORT(void)
  Initstatistics()
  {
    Py_InitModule("statistics",statistics_methods);
  }
}
```

14.5 格式化字符串

格式化字符串提供了一项技巧，使得可以在向外部例程传递参数时指定 Python 数据类型的变化。字符串中的各项在数目和类型上必须匹配，还包括 PyArg_ParseTuple()调用提供的地址。虽然参数的类型由格式化字符串来检查，但是提供的地址却没有这样做。因此，这里的错误往往会对你的程序产生灾难性的影响。

既然 Python 提供了任意长度的长整型，所以有可能有的值无法使用 C 语言的长整型来存储。在所有接收域长度太小而无法存储值的情况下，最高位的比特将被自然截去。

字符“|”、“:”和“;”在格式化字符串中有特定的含义。

“|” 指示 Python 参数列表中余下的参数是任选的。对应于任选参数的 C 变量必须被初始化为系统指定缺省值，因为 PyArg_ParseTuple 认为对应于缺少参数的变量没有改变。

“:” 指示格式化单元的列表到此结束，冒号后面的字符串被用来作为错误信息当中的函数名称。

";" 指示格式化单元的列表到此结束，分号后面的字符串被用来作为替代系统指定缺省值的错误信息。

表 14.1 函数 PyArg_ParseTuple()对应的格式化字符串

格式化单元	Python 类型	C 类型	描　述
s	string	char *	把 Python 的字符串转化为 C 中指向字符串的指针。传递的地址必须是字符指针并且不需要提供存储。C 的字符串是以 null 结尾的，而 Python 字符串通常不包含嵌入式的 null 并且也不能为空。如果确实为空，一个 TyPeError 例程将被调用
s#	string	char *, int	存储到两个 C 变量中，第一个是指向字符串的指针，第二个是长度。Python 字符串可以包含嵌入式的 nulls
z	string 或者 None	char *	与 s 相似，但是 Python 对象可以也是 None，在这种情况下，C 指针被设置为 NULL
z#	string 或者 None	char *, int	与 s#类似
b	integer	char	把 Python 的整型转化为短整型，存储为 C 的字符型
h	integer	short int	把 Python 的整型转化为 C 的短整型
i	integer	int	把 Python 的整型转化为 C 的整型
l	integer	long int	把 Python 的整型转化为 C 的长整型
c	长度为 1 的 string	char	把 Python 的字符（长度为 1 的字符串）转化为 C 的字符型
f	float	float	把 Python 的浮点型转化为 C 的浮点型
d	float	double	把 Python 的浮点型转化为 C 的双精度型
D	complex	Py_complex	把 Python 的复数型转化为 C 的 Py_complex 结构体
O	object	PyObject *	把 Python 对象（不带有变化）存储为 C 的对象指针。C 的接口接收到传递的实际对象。对象的引用计数并没有增加
O!	object	typeobject,PyObject *	把 Python 对象存储为 C 的对象指针。这和 O 相似，但产生两个 C 参数：第一个是指定所需 Python 类型的地址，第二个是对象指针存储的 C 变量（PyObject *类型）地址。如果 Python 对象不具有所需要的类型，将产生一个 TypeError 错误例程
O&	object	function , variable	通过转换函数把 Python 对象转化为 C 变量，这产生两个参数：第一个是函数，第二个是 C 变量（任意类型，默认指定为 void *）的地址。转化函数如下调用：status = function(object,variable)，其中 object 是将要被转换的 Python 对象，variable 是将要传递给 PyArg_ConvertTuple()的 void * 参数。如果转换成功，返回状态为 1；若失败则返回 0，并引起错误例程
S	string	PyStringObject *	与 O 相似，但是期望 Python 对象是字符串对象，否则返回一个 TypeError 错误例程

（续）

格式化单元	Python 类型	C 类型	描　述
(items)	sequence	matching items	对象必须是 Python 序列，其长度是项目中格式化单元的数目。C 参数必须对应于项目中单独的格式化单元。序列的格式化单元可以嵌套

为了向调用扩展的 Python 程序返回值，我们使用 Py_BuildValue，它和 PyArg_ParseTuple 使用相似的格式化字符串。但 Py_BuildValue 还是有些不同：首先，调用的参数是值，而不是地址；其次，只有在以下情况才生成数组：有两个或更多的格式化字符串，或者将空的或单个格式化单元放入括弧中。

表 14.2 函数 Py_BuildValue()对应的格式化字符串

格式化单元	C 类型	Python 类型	描　述
s	char *	string	把以 null 结尾的 C 字符串转化为 Python 对象。如果 C 字符串指针为空，则返回 None
s#	char *, int	string	把 C 字符串和它的长度转化为 Python 对象。如果 C 字符串指针为空，则忽略长度返回 None
z	char *	string 或者 None	和 s 一样，如果 C 字符串指针为空，则返回 None
z#	char *, int	string 或者 None	和 s# 一样，如果 C 字符串指针为空，则返回 None
i	int	integer	把 C 的 int 型转化为 Python 的整型对象
b	char	interger	同 i 一样
h	short int	interger	同 i 一样
l	long int	integer	把 C 的长整型转化为 Python 的整型对象
c	char	长度为 1 的 string	把代表字符的 C 的整型转化为 Python 长度为 1 的字符串
d	double	float	把 C 的双精度型转化为 Python 的浮点型
f	float	float	和 d 一样
O	PyObject *	object	传递增加引用数目的 Python 对象。如果传递的是空指针，可以认为是产生参数的调用发现错误并设置错误例程而引起的，因此，Py_BuildValue()将返回空，但并不产生错误例程，并且置位 PyExc_SystemError
S	PyObject *	object	和 O 一样
N	PyObject *	object	和 O 相似，但是引用计数并不增加
O&	function , variable	object	把 variable 通过转换函数转换为 Python 对象，函数通过 variable（与 void *兼容）作为参数而被调用，并返回一个新的 Python 对象，当错误发生时，返回 NULL
(item)	matching items	tuple	把 C 的序列转化为相同长度的 Python 数组
[item]	matching items	list	把 C 的序列转化为相同长度的 Python 列表

（续）

格式化单元	C 类型	Python 类型	描　述
{item}	matching items	dictionary	把 C 的序列转化为相同长度的 Python 字典。每一个连续的 C 赋值对都加到字典当中的一项，使用第一个值作为关键词，第二个为转化的值

14.6 引用计数

你可能已经注意到在前面格式化字符串的论述里我们多次提到引用计数的概念。如果你对 Python 不熟悉，特别是对扩展的编程和 Python 的 API 不熟悉，这可能是一个学习的重点。

对于扩展和 API，Python 文档提供了必须要掌握的知识的完整图景。如果你想要看到完全的解释，我推荐你研究 Python 文档。

大多数 API 函数带有 PyObject *类型的返回值，这是一个指向任意 Python 对象的指针。所有的 Python 对象具有相似的基本行为，可以被单独的 C 类型所代表。你不可以声明一个变量为 PyObject 型，仅仅能够声明为 PyObject *，而指向实际的存储单元。所有的 PyObjects 都具有一个引用计数。

在这里，我们需要特别的小心：当一个对象的引用计数变为零，对象就会被释放。如果当前被释放的对象包含有对其他对象的引用，则其他对象的引用计数也将会减少。当他们的计数减为零，自然也会被释放。

问题通常发生在接口从列表中分离出对象，使用一会儿这个引用（或者更糟的情况，把引用传递回调用者）的时候。类似的情况，在传递数据给调用者之前使用 Py_DECREF() 将导致灾难性的后果。

Python 文档建议程序扩展应该使用那些带有 PyObject、PyNumber、PySequence 和 PyMapping 前缀的 API 函数，因为这些操作总是增加引用计数。当没有进一步的引用需要时，应该由调用者来负责调用 Py_DECREF()函数。

这里给出一个总体的原则：如果你在编写 Python 扩展时经常会在返回值或者退出程序时产生问题，那可能是把引用计数搞错了。

14.7 嵌入式的 Python

当有需要把 Python 功能模块加入 C 或者 C++程序当中的时候，嵌入 Python 解释器是可能的。如果你需要在 C 程序当中产生 Python 对象，或者使用字典作为数据结构，那么嵌入将是毫无价值的。在同一个程序中把扩展和嵌入组合起来是同样可能的。

Python 文档提供了关于这个 API 的详细信息，你应当进一步参考这些材料里的细节。使用 API 的所有工作就是在使用 API 调用之前从你的程序调用一次 Py_Initialize()。

一旦解释器初始化之后，你就可以使用 PyRun_SimpleString()来执行 Python 字符串，或者使用 PyRun_SimpleFile()来执行完成了的文件。另一个可选择的方案是通过解释器进行精确的控制。

下面举一个简单的例子，说明 Python 功能块从 C 中被存取的方法。我们将存取在 C 中使用简单的 Python 脚本生成的一个字典。这对于 C 程序提供了一个强有力的机制来实现批处理查询数据，而无需编写特殊的代码。大多数的代码完全使用 C 代码来运行，这意味着性能是很好的。

Dictionary.c

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <stdlib.h>
#include "Python.h"

PyObject *rDict = NULL;  /* Keep these global */                    ❶
PyObject *instanceDict;
/*
**   Initializes the dictionary
**   Returns TRUE if successful, FALSE otherwise
*/
int
initDictionary(char *name)
{
  PyObject *importModule;
  int retval = 0;

  /*   ***************** Initialize interpreter ***************   */
  Py_Initialize();

  /* Import a borrowed reference to the dict Module  */              ❷
  if ((importModule = PyImport_ImportModule("dict")))
  {
    /* Get a borrowed reference to the dictionary instance */        ❸
    if ((instanceDict= PyObject_CallMethod(importModule, "Dictionary",
                  "s", name)))
    {
      /* Store a global reference to the dictionary */               ❹
      rDict = PyObject_GetAttrString(instanceDict, "dictionary");
      if (rDict != NULL)
        retval = 1;
    }
    else
    {
        printf("Failed to initialize dictionary\n");
    }
  }
  else
  {
        printf("import of dict failed\n");
  }
```

```
  return (retval);
}
/*
**   Finalizes the dictionary
**   Returns TRUE
*/
int
exitDictionary(void)
{
  /*   ***************** Finalize interpreter *********************/
  Py_Finalize();
  return (1);
}

/*
**   Returns the information in buffer (which caller supplies)
*/
void
getInfo(char *who, char *buffer)
{
  PyObject *reference;
  int      birthYear;
  int      deathYear;
  char    *birthPlace;
  char    *degree;
  *buffer = '\0';

  if (rDict)
  {
    if ((reference = PyDict_GetItemString( rDict, who )))          ❺
    {
      if (PyTuple_Check(reference))                                ❻
      {
        if (PyArg_ParseTuple(reference, "iiss",
            &birthYear, &deathYear, &birthPlace, &degree))         ❼
        {
            sprintf(buffer,
         "%s was born at %s in %d. His degree is in %s\n",
            who, birthPlace, birthYear, degree);
  if (deathYear > 0)
            sprintf((buffer+strlen(buffer)),
         "He died in %d\n", deathYear);
            }
        }
    }
    else
        strcpy(buffer, "No information\n");
  }
  return;
}
```

```
main()
{
  static char  buf[256];
  initDictionary("Not Used");
  getInfo("Michael Palin", buf);
  printf(buf);
  getInfo("Spiny Norman", buf);
  printf(buf);
  getInfo("Graham Chapman", buf);
  printf(buf);
  exitDictionary();
}
```

代码注解

❶ 这个例子意味着代表一种长期查找服务，所以我们挂起调用之间的访问来减少开销。

❷ 我们引入 dict.py 模块，调用 PyImport_ImportModule 与使用 Python 的表达式 import dict 完全类似。

❸ 我们使用 PyObject_CallMethod 产生一个字典的例子，现在虽然类初始化并没有调用方法，但 Python 的实现方法使得它起到了一个方法调用的功能。下面就是简短的 Python 模块 dict.py：

```
class Dictionary:
def init(self,name=None):
self.dictionary={
    'Graham Chapman':    (1941,1989,'Leicester','Medicine'),
    'John Cleese':       (1939,-1,'Weston-Super-Mare','Law'),
    'Eric Idle':     (1943,-1,'South Shieleds','English'),
    'Terry Jones':       (1942,-1,'Colwyn Bay','English'),
    'Michael Palin':     (1943,-1,'Sheffield','History'),
    'Terry Gilliam':     (1940,-1,'Minneapolis','Political Science').
}
```

❹ 接下来我们得到 self.dictionary 的引用，存储以备以后使用。

❺ PyDict_GetItemString（rDict，who）等价于表达式 tuple = self.dictionary[who]

❻ 这是一段有些奇异的代码，但我们确实重新生成了一个数组。

❼ 最后，我们使用格式化字符串来分解数组。

为了编译和连接，应当使用如下方法（对于 Win32 系统）：

```
cl -c dictionary.c -I\pystuff\python-1.5.2\Include\
        -I\pystuff\python-1.5.2\PC -I.
Link dictionary.obj \pystuff\python-1.5.2\Pcbuild\python15.lib \
        -out:dict.exe
```

如果运行 dict.exe，将得到类似于图 14.1 的输出。

```
C:>dict
Michael Palin was born at Sheffield in 1943. His degree is in History
No information
Graham Chapman was born at Leicester in 1941. His degree is in Medicine
He died in 1989
```

图 14.1　Python 内嵌于 C 应用程序中

14.8 小结

用户可能无需构建 Python 扩展或在程序中嵌入 Python，但在一些情况下，这是唯一可以与特定系统接口或经济地开发代码的方法。虽然本章包含的内容在编写时是准确的，但我还是建议你访问 www.python.org 来获取最新的版本。

第15章 调试程序

调试是一个错综复杂的领域。我知道我将要从一些读者那里得到严厉的批评，但我还是不得不说一些会激怒他们的话。如果在引起人们怀疑的代码内有策略性地插入 print 语句，那么 Python 是易于调试的。

并不是得不到一些优秀的调试工具，包括 Python 自带的集成编译环境——IDLE，但是就像我们将要看到的那样，在一些情况下，调试器将会起到阻碍作用并引入人为因素。

在本章中，我们将要考察一些简单实用的技术，我还将向那些没有为调试程序开发过方法的读者提供一些建议。

15.1 为什么使用 print 表达式

使用调试器调试简单的 Python 程序很少会引起问题——你可以随意地在整个代码中单步运行，当数据发生改变时将它们打印出来，你还可以用未知的值来改变数据的赋值以进行试验。当你运行图形用户界面、网络程序或者包含定时事件的任何代码时，是很难准确地预测应用程序的行为的。这就是为什么对于图形用户界面，由主循环在一个特定的定时序列里调度事件的原因；也是为什么对于网络程序包含定时事件的原因（特别是那些超时设置很短，以至于无法在单步运行时表现出来）。

既然我们编制的许多程序都属于这个范围，所以我开发了一个通常可以避免前面所描述缺陷的方法。我说通常，是因为即使在程序中加入 print 语句也仅稍微增加了整个运行时间，但它还是在 CPU 占用，输出到文件或者 stdout 以及代码的整体长度上产生了影响。

无论如何试着去使用那些可以得到的工具——它们非常优秀。如果产生的结果你令你迷惑、失望、甚至恼怒时，试着使用 print 语句！

15.2 一个简单的例子

在本书的许多例子中，使用了 try...except 语句以便在执行时捕获错误。这是一个编写坚实代码的好方法，但当存在错误时，常常给程序员平添麻烦。因为当错误发生时，会引起 Python 程序跳转到 except 分支，从而可能掩盖了真实的问题。

Python 有一个最后阶段来测试程序：即使 Python 编译字节代码时没有什么语法错误，但在运行时还可能会不正确，因此我们要定位这些错误并改正它们。我将要诊断本书中前面举过的一个例子，来介绍一些运行时产生的错误。代码位于没有定义 except 分支的 try...except 语句内。这并不是一个特别好的主意，但的确对这个例子有用，因为确

实很难找到错误发生的位置。

让我们试着运行 debug1.py：

```
C:> python debug1.py
An error has occurred!
```

除了增加伤害外，我们仍旧得到部分期望的输出，如图 15.1 所示。

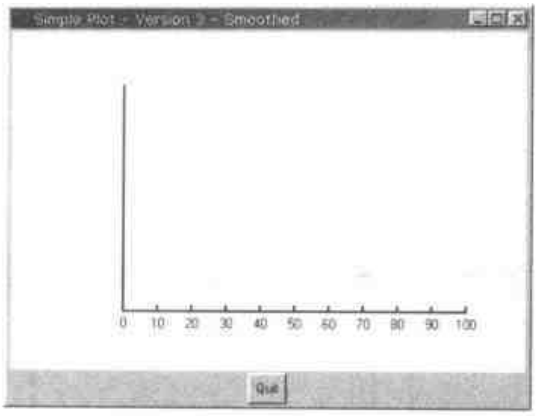


图 15.1　调试阶段 1

为了启动调试，我们有两种选择：要么在代码中放入 print 语句以找到最后执行正确的位置，要么注释掉 try...except 语句。为了完成这一点，快速像这样编辑文件：

```
    if 1:
#   try:
# ---------Code removed-----------------------------------------
#   except:
#       print 'An error has occurred!'
```

放入 if 1: 语句，并注释掉 except 分支意味着你不需要重新跳转——跳转往往会引起更多的错误。现在运行 debug2.py 将得到下面的结果：

```
C;>python debug2.py
Traceback (innermost last):
    File "debug2.py",line 41, in ?
        main()
    File "debug2.py", line 23, in main
        Canvas.create_line(100,y,105,y,width=2)
NameError:y
```

于是，我们来观察一下代码段，可以看到如下内容：

```
for I in range(6):
    x=250-(I+40)
    canvas.create_line(100,y,105,y,width=2)
    canvas.create_text(96,y,text='%5.1f'%(50.*i),anchor=E)
```

好，这是我的错！我剪切和粘贴了一些代码，但忘记修改变量 x。下面我把 x 改为 y，并恢复 try...except 语句（很显然这是唯一的问题）。

现在再来运行 debug3.py：

```
C:> python debug3.py
```

没有错误，但还不是我想要得到的结果，屏幕如图 15.2 所示！很明显，y 轴产生的不正确，于是让我们打印出正在被计算的值：

```
for I in range(6):
    y=250-(I+40)
    print "I-%d,y=%d"%(I,y)
    canvas.create_line(100,y,105,y,width=2)
    canvas.create_text(96,y,text='%5.1f'%(50.*i),anchor=E)
```

现在运行 debug4.py：

```
C:>python debug4.py
i=0,y=210
i=1,y=209
i=2,y=208
i=3,y=207
i=4,y=206
i=5,y=205
```

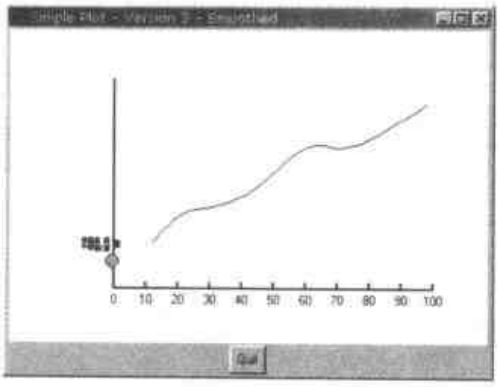


图 15.2　调试阶段 2

那不是我的本意！每次减少一个像素根本不起作用！如果观察 print 语句前面的一行代码，可以看到我是想乘以 40 而不是加上 40。改正之后，运行 debug5.py：

```
C:> python debug5.py
```

又没错误了，但不是我想要的（见图 15.3），小圆圈应该在线上。让我们看一下画点的那一段代码：

```
scaled=[]
for x,y in [(12,56),(20,94),(33,98),(45,120),(61,180),
            (75,160),(98,223)]:
    scaled.append(100+3*x,250-(4*y)/5)
```

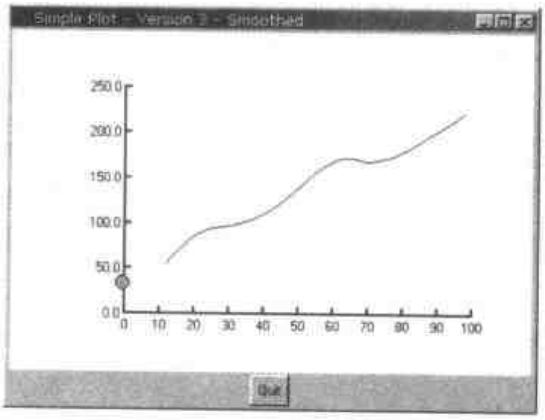


图 15.3　调试阶段 3

看起来没有问题，但还是让我们调试一下：

```
scaled=[]
    for x,y in [(12,56),(20,94),(33,98),(45,120),(61,180,)
                (75,160),(98,223)]:
        s=100+3*x,250-(4*y)/5
        print "x,y=%d,%d"%(s[0],s[1])
        scaled.append(s)
```

现在运行 debug6.py：

```
C:>python debug6.py
x,y-136,206
x,y-160,175
x,y-199,172
x,y-235,154
x,y-283,106
x,y-325,122
x,y-394,72
```

是的，现在看起来对了。我们需要进一步观察：

```
canvas.create_line(scaled,fill='black',smooth=1)
for xs,ys in scaled:
    canvas.create_oval(x-6,y-6,x+6,y+6,width=1,
                outline='black',fill='SkyBlue2')
```

这个例子上，我会使你避免更多的痛苦。很明显错误是故意造成的，并且我也不总是这么得粗心。我想使用 xs 和 ys 来作为定标过的坐标，但在 canvas.create_oval 方法中我使用了 x 和 y。因为前面我使用了 x 和 y，而且没有得到 NameError，所以我们仅仅使用最后的值。我知道如果发生了变化，你会相信我的话，代码现在会正常运行！

15.3　如何调试

调试的能力和编写代码的能力实际上同等重要。一些程序员在调试错误代码时都碰到过问题，特别是当代码很复杂时。调试中关键的技术是忽略次要的而把精力集中到重要的问题上，接下来是学会如何熟练掌握这项关键技术。

实际上，如果你想要证实代码产生了你所期望的值，并且按照你所指定的路径运行，那么使用 print 语句（或正确的使用调试器）来显示变量、指针和类似元素就是一个不错的主意。同样的，在你的代码中放入跟踪代码来显示哪一条语句正在被执行，可以帮助你快速把精力集中到有问题的区域。

如果你精通一种编辑器，例如 emacs，那么在你的代码中插入调试代码将十分容易。如果你有时间和兴趣，你可以编制自己准备调试的代码。加入当一个变量被设置时才会执行的代码，例如：

```
if debug >2:
        print('location:var=%d,var2=%s'%(var,var2))
```

很清楚，如果当你启动运行你的代码时遇到问题，你可以加入更多的细节、时间信息和其他有助于你分析的数据。在你编写代码时考虑调试的需要一般要比在发布的压力下考虑简单一些。

15.4 Tkinter 开发器

在第 3 章“建立一个应用”里，为了隐藏调试器的入口，我曾建议使用过一种技术。它通常工作得很好，因为你有进入工具的直接入口，所以不需重启程序。我在这里包含了一个例子（代码在网上可获得），当一个错误输出到 stderr 时，它将弹出来。接下来，你可以观察一下源代码，在 namespace 中改变变量或者甚至可以下载程序的其他版本来调试。这里给出一系列屏幕的显示：

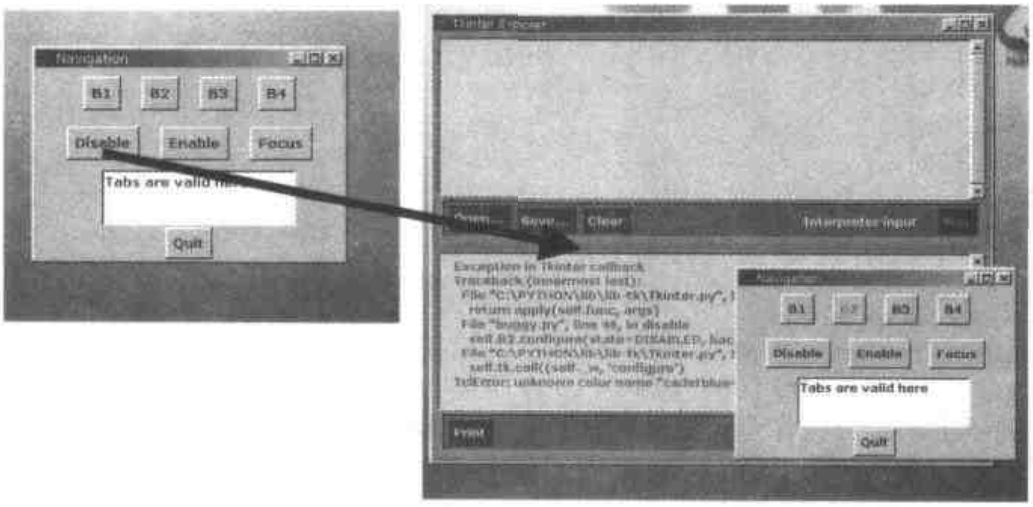


图 15.4 内嵌调试工具

接着，在其顶层窗口中打开源代码来寻找错误：

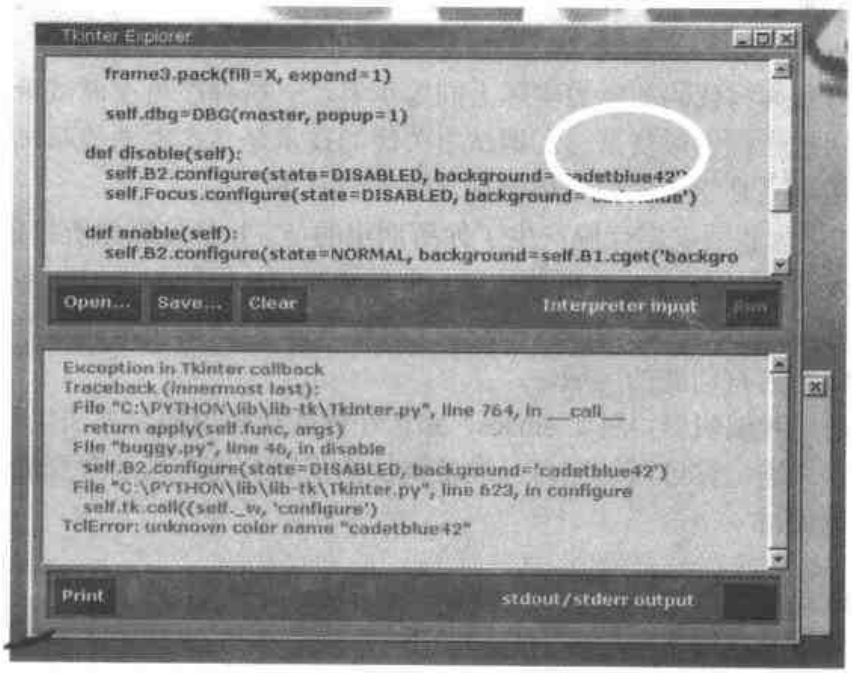


图 15.5 定位代码错误

如果我们改正了错误并单击运行，就执行了程序的另外一个版本，可以看到问题已经解决了。图 15.6 左上角的窗口给出了运行时的情形。

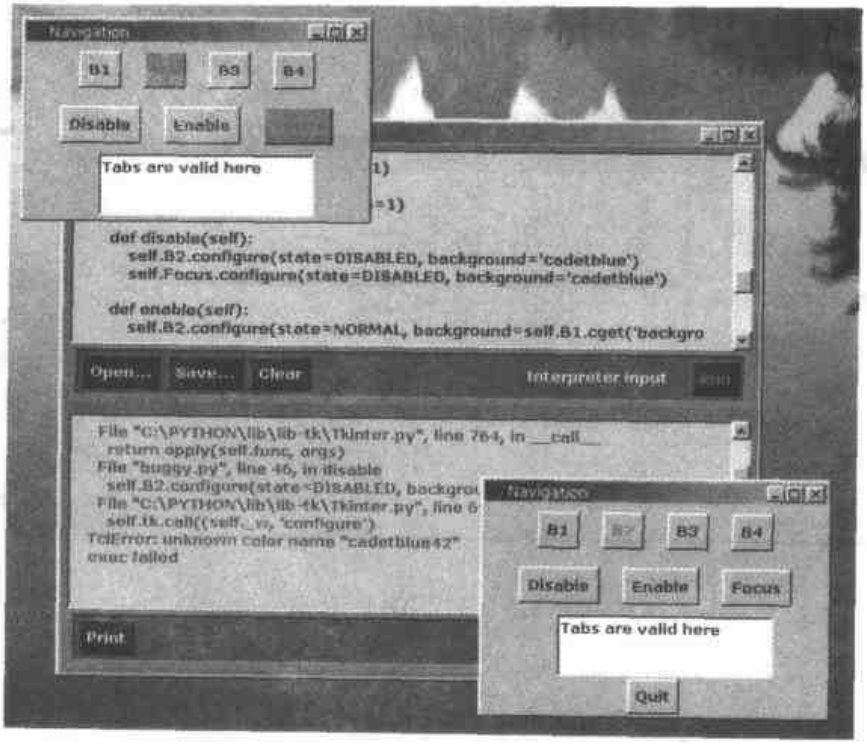


图 15.6　执行应用第二版本

15.5　pdb

如果喜欢打字，对你来说这将是一个不错的调试工具！pdb 是一个基本的调试器，它可以允许你设置断点、得到数据值、检查堆栈以及除了少许元件之外的命名。它的缺点是为了得到感兴趣的特定的点，我们不得不输入大量的数据。另外，当要重新运行时，你必须再一次输入信息！

这个工具并不适合所有的用户，但是如果你的调试风格适合 pdb，那就使用它。

15.6　IDLE

IDLE 是一种新兴的 Python 集成编译环境，它有一些限制，但是当不顾及时间和使用的增加，这些限制将会减少。你已经可以在 Python 新闻组看到关于各种各样功能类型的要求。随着更多的用户发现这种工具里需要什么时，这种需求将会继续增加。确保你得到的是最新的代码而不是使用 Python 1.5.2 中的代码包。因为已经有了许多变化，且改正了一些错误。

下面我将提供一个 IDLE 会话的屏幕显示来给你一个关于它的整体印象。试试这个工具，看看它是否适合你的需要，如图 15.7 所示。



图 15.7　使用 IDLE 调试

15.7　DDD

DDD 是一个图形前端调试器，可以支持大量语言。它使用了 pdb 的修改版本，得到程序员的广泛支持。我再次提供一个会话的例子，如图 15.8 所示，让你来决定它是否对你有用！

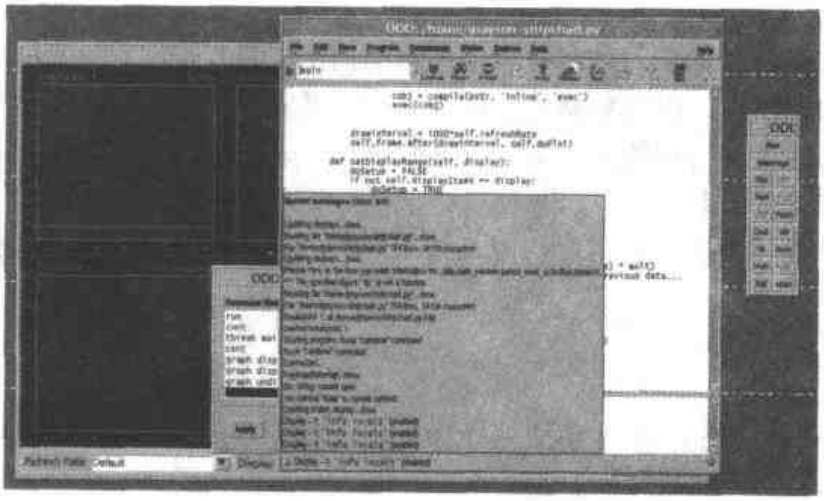


图 15.8　使用 DDD 调试

第 16 章　设计有效的图形应用程序

图形用户界面对于大多数需要和用户直接通信的程序来说已经成为必不可少的部分。本章将描述好的 GUI 设计的本质——但是请记住，这一部分是高度主观化的，将涉及到字体和颜色模型的考虑，同样包括尺寸、环境构造等人为因素。最后介绍了可选择的 GUI 设计，包括超现实主义、虚拟现实和其他新兴的技术。

关于界面设计已经公布的标准在参考书目中提到。许多公司有自己内部的标准，但可能会和已经公布的标准相冲突。一个 GUI 程序员的生活并不总是很轻松，审美学在一些情况下要屈从于标准。

如果非要选择一个好的 GUI 设计的决定因素的话，那就是简单性。一个精致的界面将显示引导用户进行输入和观察输出所需的最少的信息。就像我们将要看到的，有些情况下增加一点复杂度将改善整个界面，但总的来说，还是简单点好。

16.1　友好界面设计的元素

用户界面实际上很简单：显示一个屏幕，让用户输入数据并单击一个或两个按钮，并且显示用户动作所产生的结果。这看起来容易，但除非你注意细节，否则终端用户对于程序的满意程度将很容易地被一个设计得很差的 GUI 毁掉。当然，对于 GUI 的评价是相当主观化的，并不能使每一个人满意。在本章中，我们将要开发大量的界面，并检查它们相对的优点。这些例子的源代码将不会以文本的形式出现，但是如果你想要重现这些例子或者是把它们用作你自己界面的模板时，你可以从网上得到。从整本书中你可以看到更多的 Tkinter 代码！如果你一定要用提供的源代码，请确定你所用的都是好的例子。

看一看第一个 GUI（图 16.1），你可能不相信我，但我确实在商业软件上看到过类似的 GUI。这个屏幕完成了所有它想要完成的功能，但是却一点也没有考虑终端用户和

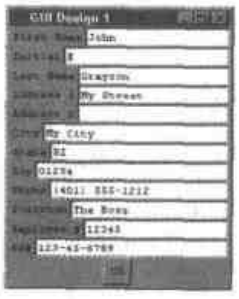


图 16.1　一个粗糙的界面

人为因素。在我们讨论过如何构建一个更好的 GUI 之后，你可能会重新审视这个例子。

即使我说这是一个劣质的 GUI，但这种屏幕在特定的上下文环境中还是可以接收的。如果所需要仅仅是一个简单窗口来给一个程序输入调试信息，那么用来支持这个屏幕所需的代码就非常少。然而不要将这种屏幕给你的终端用户使用，特别是想要他们付钱的时候！

因此，在决定如何改进这个屏幕之前，让我们迅速地看看它的缺点。

1．参差不齐的标签和项目边界看起来很费劲，使得用户不得不在 GUI 当中随机地浏览。

2．字体的选择不当：Courier 是一种固定衬线间距的字体。在这种情况下，它没有足够的字重和尺寸来方便地浏览。

3．虽然在项目输入的黑色字母和白色背景之间的对比很明显，但是在框架和它的标签背景之间的对比太大了，以至于形成刻板的界面。

4．拥挤的域使得用户很难定位标签和域的位置。

5．域的长度任意，用户无法决定应该输入多长的数据。

6．数据分组不当：Position 和 Employee #类型的数据与个人名字的关系比个人地址的关系更加密切，应该相应分组。

让我们试着更正图 16.1 的问题，并且看看我们能否改善这个界面。图 16.2 给出了改变 GUI 某些特性之后的结果。

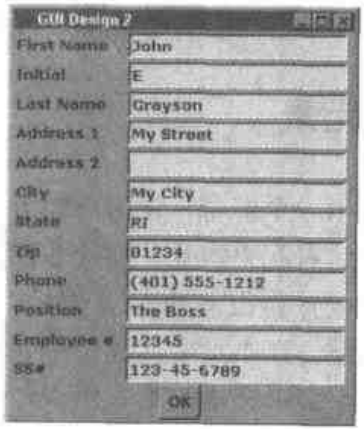


图 16.2 一个稍好些的界面

1．这个界面使用了网格（grid）管理器，而先前的界面使用的是 pack 管理器（对于几何管理器的细节参考 5.3 节）。这修正了标签和域的参差排列。

2．字体已经换成更大的衬线字体，与前面的例子相比，提高了可读性，见 16.2.1 小节的 “选择字体”。

3．项目输入域的背景色和标签域的对比已经减少。

4．少量的填充应用到域的周围，这样使得浏览更容易。

界面看起来好一些了，但它仍旧可以进一步改进。按照逻辑分组来排列域的位置，设置一些域为合适的尺寸将有助于用户填充信息。同样，把一些域排列到同一行将减少垂直方向的层，这样更适合人的浏览习惯，因为我们更愿意水平的浏览而不是垂直的——这就是我们怎样开始读的。但是，这不能推广到印刷字符要垂直阅读的情况。

下面的例子给出了实现上面讨论的各点。

图 16.3 所示的 GUI 开始显露出改进的迹象，因为域被逻辑分组，并且输入的宽度也符合它们所支持的数据的宽度。Position 输入改用了组合框（Pmw 窗口），允许用户从一系列可选值当中进行挑选（见图 16.4）。

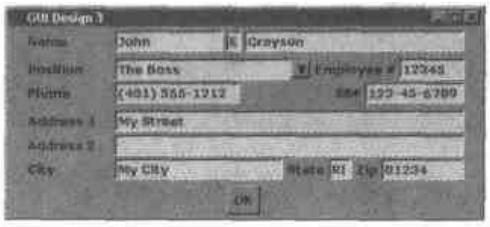


图 16.3　一个逻辑分组的 GUI 界面

把适当的域分为子域将有助于我们进一步改进界面。例如，社会保障号码通常是 3-2-4 格式。我们可以通过两种方式来处理：

1. 对每一个子域使用三个分离的域。

2. 使用智能化控件，自动在数据当中插入连字符。智能化控件在 6.8.3 小节中的“格式化（智能化）的控件”进行了讨论。

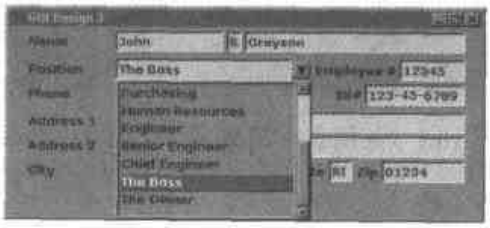


图 16.4　组合框控件

图 16.5 给出了把适当的域分为子域的情况。

不幸的是这些变化使得界面混乱，各种组合将使得终端用户感到迷惑。我们不得不按照逻辑域分组的方法来完成进一步的调整，以利于用户浏览界面。在图 16.6 中，我们在每三个域中插入一个空格。

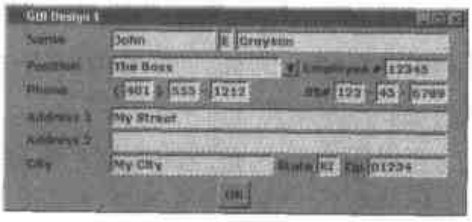


图 16.5　把域分为子域

这达到了想要的效果，但为了进一步改进，可以在每一个逻辑分组周围画一幅图。这可以通过很多方法实现：

1. 使用可得到的带有容器的 3-D 浮雕图形选项。例如，Tkinter 框架允许 sunken、raised、groove 和 ridge。

2. 在分组周围画一个框架。在 Motif 的 GUI 中可以经常看到，通常在框架里有相适合的标签。

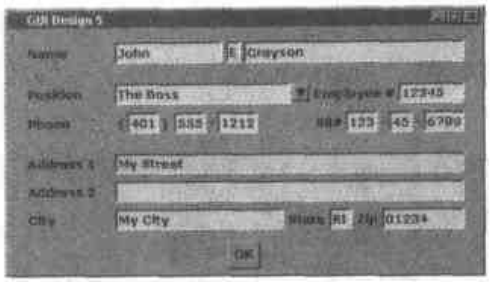


图 16.6　用白色空格分隔逻辑组

3. 安排框架外部的背景色和框架里面的显示不同，这种不同仅仅适用有限范围的界面类型。

图 16.7 解释了一个在框架中分组域周围 3-D 开槽的应用。

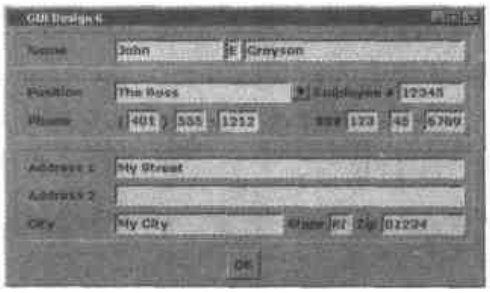


图 16.7　使用 3-D 图形来构建逻辑组框架

16.2　人为因素

关于人为因素技术的扩展文档使用科学的术语描述了 GUI 的设计（见 Reference 章节）。特别的，字体的选择可以基于一个字符对离该字符（视觉位置）20 英寸的一个点所产生的夹角的计算。在我所从事的工程中，实际测量这些角度以保证和指定的要求相适应是必需的，但通常不需要如此的精细。

但你设计一个包含 GUI 的程序时，应当考虑下列人为因素：

1. 保证程序适合终端用户的期望。例如，如果当前使用了文件（paper）系统，那么 GUI 程序应当至少模拟其中的一些操作。

2. 保持界面简单并且仅仅要求用户输入必需的数据；相对应的，仅仅显示相关的数据。

3. 选择可以使界面有效工作的字体。在一个单独的屏幕里，使用尽可能少的不同字体。

4. 按照逻辑序列对信息分层，适当的划分区域，这样可以通过平滑的方式来引导用户，而不是在关键的区域之间跳转。

5. 节省使用颜色来达到特定的目的。例如，那些用户必须填写的域可以使用颜色亮化，以区别于任选域。

6. 在 GUI 中提供多种浏览方法。要求使用鼠标进行浏览并不总是合适的；在域之间使用 Tab 键可能更加自然一些。提供两种方法，用户可以选择使用。

7. 保证程序的一致性。功能键和控制键的组合应当在单个程序和在两个程序当中产生相同的结果。

8. 一些平台的 GUI 格式可能限制了设计。即使意味着只能开发特定平台的版本，这些格式也是应当遵循的。

9. 在可能的地方加上帮助。浮动帮助*对初学者有用，但对专家可能是添麻烦。因此提供给用户关闭这项功能的方法是很重要的。

10. 用户界面应当直观，这样就无需提供文档。细致的域的标识，输入格式的提示，删除确认模式对这一点大有帮助。

11. 在任何可能的情况下，和用户一起测试用户界面的性能。在 Python 中，构建模板是很容易的；所以利用这一点，从对象用户那里得到关于你的工作的反馈信息。

16.2.1　选择字体

GUI 选择一种合适的字体对于保证界面的有效性是十分重要的，因为可读性是一个重要的因素。当一种字体在 640×480 的屏幕分辨率下清晰可读时，在 1024×768 或更大的屏幕分辨率下可能太小了而几乎无法阅读。因此，字体的尺寸应当基于屏幕分辨率来计算或者由终端用户来选择。然而在后一种情况下，你应当确保给定用户一个程序能够显示的字体的范围，而避免在改变屏幕的设置时引起溢出、交迭以及其他问题。

为了实现有效的界面，字体的选择是至关重要的。总体上说，衬线**体应该避免。我们很多人习惯于阅读印刷材料上的衬线体，因为我们相信对于浏览无衬线字体来说，我们能够更快的识别出衬线体。不幸的是，很多衬线体在屏幕上的显示不如它们打印出来清楚，因为很多打印机的分辨率要比显示器的高。

看一看图 16.8，它显示了 1024×768 的分辨率下的一个屏幕抓取，使用的是 Times New Roman 和 Verdana 的三号字体。虽然文本的整体宽度更大了，但无衬线字体导致了干净的图像。在大多数情况下，在 Windows 下选择 Arial 或者 Verdana 字体比较容易，而在 Unix 和 MacOS 下则选择 Helvetica，因为这些字体通常已经被各自的系统所安装。图 16.9 给出了这些字体。

This is an 8pt serif font　　This is an 8pt sans serif font
This is a 10pt serif font　　This is a 10pt sans serif font
This is a 12pt serif font　　This is a 12pt sans serif font

图 16.8　Serif 字体和 Sans Serif 字体比较

This is Arial　　**This is Arial Bold**
This is Verdana　　**This is Verdana Bold**
This is Switzerland　　**This is Switzerland Bold**

图 16.9　Arial, Verdanta 和 Switzerland 字体

*浮动帮助只要光标停留在一个域或者控件上不动，一会后就会出现了。该帮助如果可能的话会作为一条信息出现在窗口的状态栏里，也可以出现在靠近域或者控件的一个弹出式球中。

**衬线体就是这样一种字体，它的每一个字母都有一点给出轮廓突起和脚的艺术性花体。通常的衬线体有 Times，Bookman，Palatino 和 Garamond。无衬线字体就是那些没有这些额外花体的字体。通常的无衬线字体有 Helvetica，Arial，Geneva，Gothic 和 Swiss。

现在，让我们看一看如果选择了并不适合于 GUI 的字体，将会发生什么。当它们被打印出来的时候，这个 GUI 使用的字体看起来大了一些，。但是当在屏幕上显示时，产生的效果如图 16.10 所示。虽然字体的重量足够了，但在屏幕上看起来并不合适。

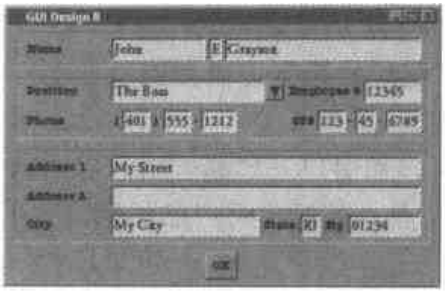


图 16.10　选择错误字体的效果

小结：

1．尽可能选择无衬线字体。

2．在一个单独的窗口中，使用最小数量的字体类型、尺寸、重量和格式。

3．尽可能适应不同分辨率的窗口。

4．如果用户能够从一个主题或者格式里选择字体，那就使用那种字体，即使违反了审美学。

16.2.2　在 GUI 中使用颜色

为一个 GUI 选择颜色模式是高度主观化的，在很多情况下，允许终端用户完成对程序所使用颜色的控制是最好的。当前系统（Windows 和 UNIX CDE）推断合适的可选颜色的趋势是允许用户选择一个主题或模式，接着应用选择了的颜色。如果你使用灰色背景设计你的 GUI 看起来不错，接着基于用户选择的模式来计算颜色，这个 GUI 将会工作得很好。那就是允许用户有权选择各种奇异的颜色组合方式，毕竟凭什么要由我们来判断呢？

对于颜色应该遵循的基本准则如下：

1．如果需要区分一个图形项目，按照最后一种方法选择颜色。

2．在屏幕上使用颜色画出用户最重要的特性。

3．记住你的用户可能患有色盲*，他们不能像你一样观察颜色对比。

4．在图形元件之间产生最优的对比，但在大多数情况下，应当避免 100%的对比**。

5．避免反色（黑底白字），除非在那些你认为必须引人注目的图形元件上。使用反色的一个很好的例子是指出输入域中数据有效性的错误。

---

* 相对而言很容易就能设计出针对有色盲用户的色彩。不要使用三原色，按不同的比例把红绿蓝三种颜色进行混合，以使得整体的色度有所不同。整体的效果对于正常色觉的用户来说没什么区别，但是对于色盲用户来说就会有很大不同。如果你的应用程序是在一个任务很关键的环境中，并且有可能被色盲者使用时，也许比较谨慎点的做法是让色觉不那么好的人来测试你的 GUI。

** 不是所有的显示器都能像显示灰底（或者别的浅色组合）黑字那样好的显示白底黑字。例如，试试看 90%灰色上的黑字（注意 90%的灰色其实是 90%的白色加 10%的黑色）。

为了说明颜色选择如何成为一个问题的，让我们看看图 16.11。虽然不可能直接看到本页上颜色的效果，但选择的颜色模式十分合意，使用了棕色、橄榄绿和米黄色这些暖色调。虽然短时间的显示可能看起来不错，但带有 GUI 的程序使用这些颜色来实现将使人产生疲倦的感觉。主要问题是在输入域使用了低对比度的反色影像。如果程序当中的颜色偶然地被应用到商业程序，我相信设计者一定为这些产生的结果而感到自豪。

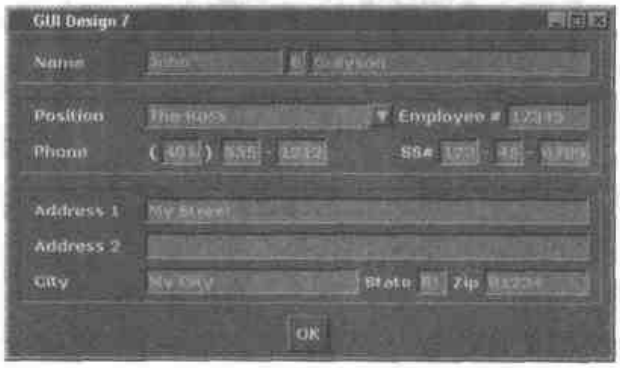


图 16.11　颜色组合的问题

16.2.3　尺寸的考虑

除了确保 GUI 能工作在各种不同的屏幕分辨率之外，还必须考虑是否允许用户改变显示的 GUI 尺寸。当屏幕已经设计的比例均衡之后，再允许用户随意改变显示尺寸将是很困难的。在很多情况下，不允许改变屏幕尺寸将更好一些，而不是不顾用户的要求来试图维持原来有效的 GUI（关于如何去做，见 13.2 节中“几何方法”）。

在本章前面给出的例子中，如果窗口变大时，则通过给一个域赋值使得每一行变长。缩小一个窗口将更难处理，甚至不可能。

如果屏幕不包含可滚动的窗口（文本、列表等），一般最好是固定窗口的尺寸并保持最优的层结构。

16.3　可选择的 GUI

前面描述的 GUI 设计准则适用于大多数程序，但是为了达到期望的效果，在一些特别的程序中需要打破这些准则。让我们举一个例子来研究。我曾经和一家航空公司签订合同，开发飞行员的机组调度系统。因为大多数航空公司的经理以前是飞行员，所以我想设计一个会打动这些人的界面。图 16.12 给出了其中的一个屏幕。它打破了前面提到的许多准则，但却被航空公司高度赞许。对于没有在飞机座舱里呆过的读者，我告诉你们这个界面是模拟飞机中的通信按键。在这个界面里，主要的显示采用了反色影像和系统中并不常用的像素字体。这个面板使用射线轨迹来提供极度的三维效果（这个技术在 9.6 节中涉及到）。虽然在黑白的效果下并不透明，但界面的确包含了许多有颜色的元素。而且，这和将要模仿的系统保持了一致性。

这儿给出一些建议，帮助你为程序选择合适的界面：

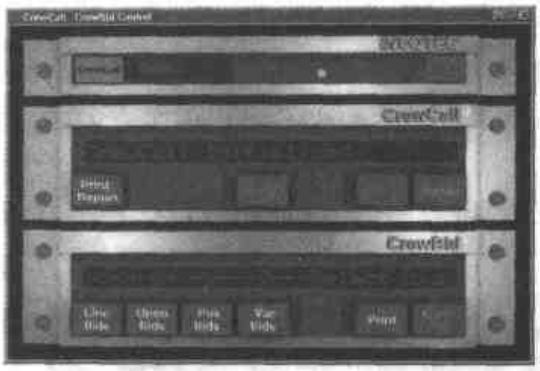


图 16.12　为特定目标的界面

1. 如果你将要实现的程序对应的人工系统已经存在，并且它使用纸的形式，那么就尽量在 GUI 中使用原有的元素。定购系统就是一个很好的例子。

2. 如果终端用户对于特定的显示比较熟悉，而程序又提供了这些方面的选项，那么就使用熟悉的形状、颜色和导航方法。由 SNMP 控制的前台面板是一个很好的例子。

3. 如果程序支持带有数字键盘的设备，而用户又很熟悉或者被训练使用过这种键盘，那就在你的界面上提供这种键盘的图形表示，说不定将来会发展成为触摸屏。

4. 如果终端用户习惯于使用控制面板而不是键盘，就考虑怎样提供更自然浏览方法，例如考虑使用触摸屏作为输入。

16.4　小结

本章的一些内容是相当主观的，当设计程序时你应当经常考虑一些相当好的准则，这些准则正被应用到人为因素技术当中。例如，使用小的或是大的字体将由你的用户立刻选择调用——你希望你的用户认可你的设计。然而，这并不意味着你必须严格遵循准则，一些最好的界面往往打破了其中的一些准则。

第17章 性能编程

当前的计算机系统有很强的能力来支持基于解释型的程序，例如本书中提到的Python/Tkinter 程序。本章既反驳了持怀疑态度的读者，也为持支持态度读者提供了证据。本章也包括了解释实际程序的例子以利于学习。既然 Tkinter 是一个有效的将 C 扩展转化为 Python 的好例子，那么我将介绍扩展构建方法来减轻复杂程序带来的性能下降。

17.1 每日加速

如果你尽心尽责，那么 Python 的性能很差将是没有理由的。不幸的是，你做的很多事会导致你的程序不能够达到用户可以接受的性能和响应性。然而，如果你努力研究以避免大量关键的错误区域，那么你生成的 Python 程序可以和编译过的 C++程序的性能相比。

17.1.1 程序组织

你可以做大量的事情来确保你的程序性能优越。让我们从你如何组织你的代码开始。不管怎样启动你的程序，也不管是否它要运行在 Unix、Win32 还是 MacOS，你需要确保的是 Python 代码的开始部分很短。如果调用 Python 解释器（在 Unix 上）并使用如下代码作为脚本的第一行：

```
#! /usr/bin/env python
```

那么后面的所有行在每次运行你的程序时都将被解析和编译。虽然脚本将被翻译成字节码，但不会生成任何字节码文件（.pyc 或者.pyo）。这意味着你不得不在每次调用程序时经过解析器。因此，你必须构建你的程序以保证在每次调用时仅解析最少数量的 Python 语句。让我们假设你已经像下面所示的样子构建好了程序：

```
from  Tkinter import *
form  tkSimpleDialog import Dialog
import  tkMessageBox
import  Pmw

class MyClass(Dialog):
def body(self,master):
  ...
det amethod(self):
 ...
```

```
root=Tk()
instance=MyClass(root)
```

当然，这里的代码并没有太多行，但你的程序可能有上千行。因此，让我们给模块命名为 ApplicationReal.py，并像下面一样改写最后两行：

```
def myApplicationReal():
   root=Tk()
   instance=MyClass(root)
```

接下来产生一个名为 myApplication 的简短 Python 脚本，并插入如下几行：

```
#! /usr/bin/env python

import myApplicationReal
myApplicationReal.myApplicationReal()
```

这将使用 myApplicationReal.pyc 或者 myApplicationReal.pyo 如果它们存在的话，否则就产生它们。这保证任何大程序在启动时性能有所提高。有些情况下，你能够在 Win32 下使用相同的文件。如果你构建了一个名为 myApp.bat 并包含下列行的批处理文件，你就可以在 Unix 和 Win32 下使用相同的模块：

```
python myApplication
```

你可以选择直接把.pyc 或者.pyo 传递给 Python 解析器。在 Unix 下，可以使用下列脚本：

```
#! /bin/sh
exec python script.pyo ${1+"$*"}
```

这项技术在 Win32 下可能不是非常有效，因为编制批处理文件和启动 Python 解析器所花费的时间将抵消使用.pyo 文件带来的速度上的增加。然而，如果你把浏览器的文件类型设置成双击.py 文件就执行 Python 的话，就根本不需要批处理文件。

17.1.2 使用 Python 优化器

如果使用最优化命令行选项设置（-o）来调用 Python，编译器将产生最优化的字节码。这意味着字节码将被优化而运行得更快，而这正是通过减少一些字节码指令达到的（例如从函数调用中删除 SET_LINENO，压缩 assert 语句）。它的确依赖于你的代码中调用了多少函数，和使用了多少 assert 语句。除了上述以外，也值得使用这个选项来提高程序的调用速度，哪怕是很少的百分比。

17.1.3 检查代码

Python 有一个不寻常的特性：如果你把新的代码在一边放了几天，通常你能够在重新检查时减少它的尺寸。经常有机会来压缩代码、展开循环或减少不必要的操作。你所做的是减少字节码操作的数量。因为通常不需要以降低性能为代价就可以修改代码，所以值得这样做。

17.2 Tkinter 性能

GUI 的性能将会造就或者毁坏程序。用户对于某些事情会多快的发生一般有下意识

的期待。即使简单的界面，例如经常看到的 ATM 机，对于不同操作的响应时间也不一样（通常每按一次键都有回应，比如一声蜂鸣声或确认少于 2s，在 10s 内完成交易）。一致性同样重要。假设一个 GUI 通常在 1s 内完成某件事，那么如果偶然需要两三秒钟，用户将很反感，这时应该提醒他们重复按键。

然而决不要在运行程序之前就试图进行优化。你应当想的是慢的代码可能在最终版本中工作得很好。如果一个循环执行的次数不多，那么努力提高它的性能将会使你无法注意到更重要问题。

17.2.1　让它保持简短

如果你能够在你的程序中减少代码行的数目而不降低任何人修改它的能力，那么你就提高它的性能。看一看下面简单的例子：

```
self.label=Label(frame,text='Password:',fg='black')
self.label.pack(side=LEFT,padx=10)
```

如果你不需要保持周围实例的存在，因为你并不打算改变它的内容、背景色或其他特性，则可以允许 Tkinter 产生内部的实例：

```
Label(frame,text='Password:',fg='black').pack(side=LEFT,padx=10)
```

17.2.2　删减本地变量

另一个值得注意的是那些并不一定需要赋值的一些本地变量，特别是在循环内部（如果只发生一次，就不是十分重要）。如果要在一个实例中产生许多特性变量，那么这一点就特别重要。如果不打算再次使用它，就不要生成。浏览你的代码，你会偶尔发现节省一些 CPU 周期的机会。当然，这也适用于规则的 Python 程序。看看下面的代码：

```
...
localx=event.x
locally=event.y

...
canvas.create_text(localx,locally,text='here',...)
```

意图是好的，但是如果你不需要重用数据的话，就删掉这些代码：

```
...
canvas.create_text(event.x,event.y,text='here', ...)
```

然而，本地变量并不总是不好。如果在循环中有不变化的子表达式或者特性引用，那就应该在进入循环之前采用本地副本。比较：

```
for I in range(30)
    self.list[i]=(self.attr2*i)+self.attr2
```

和

```
l=self.list
a1=self.attr1
a2=self.attr2
for I in range(30)
    l[i]=(a1*i)+a2
```

17.2.3 保持简单

这一点可能太明显了，但是记住一个简单的 GUI 通常会更快地初始化和消耗更少的系统资源。一些程序员（甚至是真正的工程师）对于复杂系统所进行的 GUI 设计有一个倾向，那就是在一些密集的屏幕上显示每一个可能的系统内部变量数据项。如果把数据安排到几个屏幕上，相互用少量高度相关的项联接，那么程序将会响应得更快，即使底下的代码执行相同的操作。

尽所有可能避免程序内部的波动。差的设计中，域之间的相互依赖会消耗大量系统资源，用户也会发现它的价值有限。

17.2.4 快速初始化

尽可能快地绘制你的屏幕是很重要的。用户通常由响应时间来判断一个程序；在一些情况下，如果在选择了一个操作后，用户等待了很长时间但程序没有丝毫变化，那么没有错误的程序也将被认为是有问题的。在一些例子中，我曾经评论过延迟删除框架窗口，直到它们所包含的窗口全部完成。这样做的目的是防止协调（即从内在几何管理器操作继承过来的方式）发生在每次产生新组件的时候。如果延迟协商直到所有的窗口都产生之后，那么启动时将会有更好的性能。

17.2.5 节省事件

窗口系统对于特定类型的动作会产生大量的事件。两个这样的事件是移动鼠标和按下 Shift 键（如果你的系统配置成了自动重复被按下的健）。试着运行 Example_6_2.py 并试试那些动作——你会惊讶于事件产生时运行的速度。如果所有你感兴趣的是鼠标的 x/y 位置，而且被触发的计算量也不大，那么这不是问题。然而，如果每个事件触发时，都要重画一个复杂的物体，那么你将面对性能问题。

你能够做大量的事情来达到高速度：

1. 截去事件以使得代码的响应更少。可以使用定时器做到这一点，这样每隔几百毫秒就可以更新一次。也可以使用计数器，每处理 10 个事件就更新一次。后一个方法更难于实现，通常需要定时器和计数器的组合使用。

2. 减少画图的开销。例如，如果事件是鼠标拖拉产生的结果，可以等到鼠标释放再画以进行简化。例如，在拖动过程中绘制物体的外形或阴影。

3. 压缩不相关的事件。例如，与鼠标经过物体相比，拖动物体可能引起产生大量的事件。除非它们是相关而拖动，否则最好忽略。

17.3 Python 技术

在 17.1 节中的“每日加速”中已经提到了一两项技术。在本节里，我们将研究一些 Python 特定的编码方法，它们通常生成更有效的代码。然而在你进入已经写好的每一个程序之前，一定注意除非万不得已，不要改动你的程序。如果没有发现错误，就不要改正什么！查看 17.4 节的“程序剖析”，来学习如何发现你程序的瓶颈。如果这个技术确

定了你的代码出现问题的地方，再进行修改。

本章的大多数建议都相当直接，你同样可以通过阅读 Python 新闻组来找到有用的建议（见附录 G 中的“Python 新闻组”）。

17.3.1　引用模块

可以在程序的任何位置执行 import 语句。一般情况下，大多数 import 语句在程序的开始位置。有时你可能想要延迟引用那些不经常使用，仅是在需要时才引用的模块。Python 能够正确处理重复引用相同模块的情况，但是这会引起少量的开销增加。如果可能，就避免引用相同的模块。让我们看一个假设的例子：

Example_17_1.py

```
import time
start = time.time()

def splitter():
    import string
    list = string.split('A B C D E F G H', ' ')

for i in range(123456):
    splitter()

print time.time() - start
```

与下面这个进行比较：

Example_17_2.py

```
import time
import string

start = time.time()

def splitter():
   list = string.split('A B C D E F G H', ' ')

for i in range(123456):
   splitter()

print time.time() - start
```

如果运行几次 Example_17_1.py 和 Example_17_2.py，就会发现 Example_17_2.py 的速度是 Example_17_1.py 的两倍多。另外，不要在 splitter 函数中调用 if not stringLoaded : import string ; stringLoaded = TRUE 这个结构。因为作用域的原因它不会工作，string 模块的引用在每次函数返回时就被去掉了。

17.3.2　字符串链接

直到 Python 的最近版本，文档才清楚地说明了格式化字符串的使用，因此字符串经

常被这样链接：

```
longstring=part1+' '+part2 +' '+part3
```

而使用格式化的字符串将更有效：

```
longstring='%s %s %s'(part1,part2,part3)
```

这样快得多，因为所有的工作由C代码来实现。依赖于你正在完成的链接类型，你应当至少看到使用格式化字符串会有25%的性能提高。

17.3.3 正确使用嵌套的循环

这一点前面已经提到，而且大多数经验丰富的程序员也肯定会碰到过这种情况，但仍旧值得再一次提起。当你不得不描述一个多维物体时，请注意那个变化不快的指标是在循环外面的。让我们看一个有100列25000行的二维数组。它的确与你存取它的方式有关：

```
1=[]
for row in range(25000):
    for col in range(100):
        1.append('%d.%d'%(row,col))
```

比下面的程序慢大约20%：

```
1=[]
for col in range(100):
    for row in range(25000):
        1.append('%d.%d'%(row,col))
```

然而，Guido van Rossum给我给出了再次提高20%的这个技巧：

```
rowrange=range(100)
for col in range(25000):
    for row in rowrange:
        1.append('%d.%d'%(row.col))
```

这达到了性能的提高，因为在-1到100之间的整数被缓存起来，所以相当得快。25000次从0到99的循环将产生24999个整数定位，而100次从0到24999的循环将产生100×24999个整数定位。

17.3.4 减少模块引用

每次使用模块引用时，都会增加少量开销。如果引用是在循环里面，将得到一些好处。比较：

```
import time,string
start=time.time()

for I in rage(5000):
    strtime=time.asctime(time.localtime(time.time()))
    lname=string.lower('UPPER')

print time.time()-start
```

和：

```
import time,string
start=time.time()
```

```
asct=time.asctime
ltime=time.localtime
now=time.time
lower=string.lower
for I in range(5000):
    strtime=asct(ltime(now()))
    lname=lower('UPPER')

print time.time()-start
```

虽然第二种格式需要更多的代码，但它运行得快一些。在一个带有合理结构的实际程序中，这些改进是值得的。

17.3.5 使用本地变量

现在真矛盾！前面我曾经鼓励你不要使用本地变量，但如果合理使用，你将会得到另外一个性能上的提高，因为本地变量的存取速度要比全局变量快。

如果我们使用包含本地变量的函数来重新编写最后一个例子，我们可以看到另一个小的性能提高：

```
import time,string
start=time.time()

def local():
    asct=time.asctime
    time=time.localtime
    now=time.time
    lower=string.lower

    for I in range(20000):
        strtime=asct(ltime(now()))
    lname=lower('UPPER')
Local()
print time.time()-start
```

类似的，如果在循环外面进行方法查询，还可能改进性能。比较：

```
r=[]
for item in biglist:
    r.append(item)
```

和：

```
r=[]
a=r.append
for item in biglist:
    a(item)
```

你可以通过缓存内建函数（例如 len）或全局量（例如相同模块中的其他函数）得到类似的提高。

17.3.6 使用异常处理

异常处理技术是一个能提高性能的有价值的工具。虽然 Python 的 1.5.2 版本已经通

过增加新的功能，推迟了下列技巧的发布，但是仍旧应用了其基本的原理。

这个例子着眼于使用字典。字典是很有价值的 Python 工具，因为当需要用关键词来存取数据时，字典能够带来程序性能的提高。然而，字典过去经常有一个问题：如果要存取的字典输入不存在，将得到一个 KeyError。这需要程序员使用如下代码来检查关键词是否存在：

```
if dictionary.has_key(key):
    ...
```

如果关键词通常都存在，那么使用异常处理来捕获偶尔的 KeyError 是不错的主意：

```
try:
    value=dictionary[key]
except KeyError:
    doErrorStuff()
```

在当前的 Python 版本中，你不需要做任何考虑，直接使用 get 字典方法即可：

```
value=dictionary.get(key,value_to_use_if_not_defined)
```

17.3.7 使用 map，filter 和 reduce

Python 支持三个允许列表操作的内建函数。使用这些函数的优点在于可以把大部分循环开销推到 C 代码，产生附带的性能提高。

map(function,sequence)把函数应用到列表中的每一项，并把结果返回到一个新的列表。下面就是一个简单的例子，把一列字符串中的 Y 字符变为 Ks：

```
from string import maketrans,translate

def Y2K(instr):
        returntranslate(instr,maketrans('yY','kK'))

list=['thirty','Year 2000','century','yellow']
print map(Y2K,list)

C:>python map.py
['thirtk','Kear 2000','centurk','kellow']
```

也可以使用 map 来把几个列表组合成一个列表数组。使用上面的例子：

```
print map(None,list,map(Y2K,list))
C:>python map.py
[('thirty','thirtk'),('Year 2000','Kear 2000'),('century','centurk'),
('yellow','kellow')]
```

filter (function,sequence)返回一个序列，其中包含所有函数返回值为真的项。下面的简单例子是选择所有以 7 结尾或能被 7 整除的项：

```
def func(n):
    return 'n'[-1]=='7' or (n%7==0) # Cocoricos!

Print filter(func,range(1,50))

C:>python filter.py
[7,14,17,21,27,28,35,37,42,47,49]
```

最后，reduce (function,sequence)返回一个单独项，它由连续应用序列当中的两项作

为函数参数而构建得到的。当求一组数之和时，这是很有用的方法：

```
...
    def sum(n1,n2):
        return n1+n2

seq=filter(func,range(1,50))
print reduce(sum,seq)

    C:>python reduce.py
    135
```

使用操作符.add 代替 sum 函数将进一步提高性能。操作符模块提供了一组用 C 实现的函数，它们对应于 Python 内在的操作符：

```
import operator
#   ...
print reduce(operator.add,seq)
```

然而，当函数是 λ 表达式或者函数是由下列方法之一清楚地构建时，注意到 map、filter 和 reduce 通常比内嵌代码执行速度慢是很重要的。

17.4　程序剖析

Python 有一个基本的剖析模块，允许你了解代码的瓶颈的位置。如果你的确发现代码中的某个地方花费了大量时间或者频繁执行，你就应该打算进行改进。

使用 profile 模块相当简单，并且仅要求对代码进行最小的改动。本质上，通过剖析器调用你的程序。如果你的程序由调用函数来启动，那么仅需要加上以下两行：

```
import profile
profile.run('start()')
```

当程序从 start 处退出时，剖析器（profiler）将打印一份关于所有函数和执行次数的报告。让我们来看一看书中前面的例子，在 10.7 节的 “速度绘制” 中，我们编了个程序片断。这是非常受计算量限制的，但我们可以进行一些改进。为了剖析这段代码，我们仅仅需要再加入以下的剖析器描述：

```
if name=='_main_'
    def start():
        fractal=Fractal()
        fractal.root.after(10,fractal.createImage())
        fractal.run()

import profile
profile.run('start()')
```

此时，当你退出程序，将会显示一满屏的统计信息。为了能够阅读，你必须把输出进行重定向。部分的输出如下所示：

```
655511 function calls (655491 primitive calls) in 419.062 CPU seconds
Ordered by:standard name
```

```
Ncalls  Tottime  Percal  Cumtime  Percall  Filename:lineno(function)
2       0.000    0.000   0.003    0.002    <string>:1(after)
1       0.000    0.000   0.001    0.001    <string>:1(after_cancel)
1       0.000    0.000   0.002    0.002    <string>:1(after_idle)
1       0.000    0.000   0.001    0.001    <string>:1(bind)
2       0.000    0.000   0.003    0.002    <string>:1(bind_class)
1       0.000    0.000   0.000    0.000    <string>:1(focus_set)
1       0.000    0.000   20.476   20.476   <string>:1(mainloop)
1       0.000    0.000   0.001    0.001    <string>:1(overideredirect)
1       0.000    0.000   0.000    0.000    ImageDraw.py:24(_init_)
214194  22.624   0.000   22.624   0.000    ImageDraw.py:34(setink)
1       0.000    0.000   0.000    0.000    ImageDraw.py:39(setfill)
214194  20.578   0.000   20.578   0.000    ImageDraw.py:51(point)
1       0.152    0.152   0.152    0.152    ImageFile.py:194(save)
1       0.000    0.000   0.465    0.465    P_fractal.py:55(initData)
1       287.474  287.474 394.599  394.599  P_fractal.py:65(createImage)
1       0.135    0.135   0.135    0.135    P_fractal.py:8(getpalette)
214194  49.528   0.000   92.730   0.000    P_fractal.py:96(pixel)
0       0.000            0.000            Profile:0(profiler)
```

你可以看到三个例程，每一个都被调用超过 200,000 次，而且几乎所有的时间都花在 createImage（这并不令人惊奇）。然而，的确在默认的运行中有太多的输出显得没什么用。

幸运的是剖析器可以工作在另外一种模式下，即统计信息能够被收集到一个文件中供你随意分析。你也可以对输出格式进行相当多的控制。为了解释这一点，我们需要做一些小改动。在这个例子中，我们将打印出累积次数最多的 20 项：

```
import profile
profile.run('start()','profile_results')

import pstats
p=pstats.Stats('profile_results')
p.sort_stats('cumulative').pring_stats(20)
```

现在，我们运行 p_fractal.py，我们应用程序存在时，得到如下输出

```
Tue Oct 05 12:14:25 1999  profile_results
655779  function calls(655759 primitive calls) in 374.967 CPU seconds
Ordered by:cumulative time
List reduced from 289 to 20 restriction<20>
Ncalls  Tottime  Percall  Cumtime  Percall  Filename:lineno(function)
1       0.003    0.003    374.969  374.969  Profile:0(start())
1       0.000    0.000    374.966  374.966  Python:0(279.C.32)
1       0.000    0.000    374.966  374.966  P_fractal.py:116(start)
```

1	271.083	271.083	366.943	366.943	P_fractal.py:65(createImage)
214194	47.581	0.000	87.823	0.000	P_fractal.py:96(pixel)
214194	21.319	0.000	21.319	0.000	C:\py15\PIL\ImageDraw.py:34(setink)
214194	18.923	0.000	18.923	0.000	C:\py15\PIL\imageDarw.py:51(point)
1	0.000	0.000	5.590	5.590	C:\Bookmaster\examples\common\AppShell.py:296(run)
1	0.000	0.000	5.590	5.590	C:\Bookmaster\examples\common\AppShell.py:291(main)
1	0.000	0.000	5.589	5.589	<string>:1(mainloop)
1	5.264	5.264	5.589	5.589	C:\py15\lib\lib-tk\Tkinter.py:486(mainloop)
489	0.345	0.001	5.301	0.011	C:\Bookmaster\examples\common\ProgressBar.py:50(update)
488	0.072	0.000	5.288	0.011	C:\Bookmaster\examples\common\AppShell.py:248(updateProgress)
488	0.073	0.000	5.217	0.011	C:\Bookmaster\examples\common\ProgressBar.py:44(updateProgress)
1	0.001	0.001	2.423	2.423	C:\Bookmaster\examples\common\AppShell.py:33(_init_)
1467	0.381	0.000	1.950	0.001	C:\py15\lib\lib-tk\Tkinter.py:1222(itemconfigure)
1956	1.841	0.001	1.841	0.001	C:\py15\lib\lib-tk\Tkinter.py:1058(_do)
490/489	1.800	0.004	1.804	0.004	C:\py15\lib\lib-tk\Tkinter.py:449(update_idletasks)
1	0.000	0.000	1.567	1.567	C:\py15\lib\lib-tk\Tkinter.py:1767(_init_)
1	1.567	1.567	1.567	1.567	C:\py15\lib\lib-tk\Tkinter.py:1717(_init_)

观察输出的信息，你可以发现先前看到的三个与绘图相关的例程。它们占用了整个累积时间的 20%，因此在我们的控制下，任何的缩减都将带来程序性能的确实提高。然而，你的程序可能还会有许多性能有待提高的地方。

你可能会使用一些很简单的基准（benchmarking）技术来测试一段代码能否通过特殊的方法来最优化。你仅需要对一段代码定时，例如：

```
import time
...
def function():
    ...
start=time.clock();function();print round(time.clock()-start,3)
```

因为 time.clock()能够提供 CPU 时间，而不是占用时间，所以一般用它来进行基准。

17.5 Python 扩展

在 14 章的“扩展 Python”,我们研究了构建 Python 扩展的方法，主要是扩展函数功能的一种方法。在性能提高方面，可以利用编译过的 C 来代替解释型的 Python。首先，你应该像上面显示的那样剖析你的程序。如果发现候选的例程，可以用 C 重写。记住，Python 的一些优势来自于高层的结构。因此，有的时候用 C 来重写这些功能是困难的，实际上 C 代码的效率可能会比 Python 还低。

然而，在很多地方加上扩展模块来减轻瓶颈效应是值得的。中等的浮点运算、三角法操作和矩阵操作都是这样的例子。在特定的情况下，如复杂的词典操作过程，使用 C 调用 Python 将比纯 Python 代码执行更快。

17.6 小结

记住对应用程序性能操作的最重要的准则：除非你或者你的使用者检测到错误，否则不要做任何事。因为有可能花费了大量的努力而只得到相对意义不大的提高，若把同样的时间用在界面的加工上会有更好的效果。你的用户常常会觉得界面的细节更有价值。但是，如果你确实发现有性能问题，我真心希望这些建议会有所帮助。

第 18 章　线程和异步技术

应用程序频繁地被用来处理诸如并行任务或耗费时间的操作，因此这一章涉及线程和其他支持后台处理的技术。关于这方面 Python 和 Tkinter 的编程会引起特殊问题，诸如防止系统被挂起和处理意想不到的行为。由于问题通常发生在系统全速运行时，所以脱离开调试器或者没有 print 语句调试起来会很困难。

18.1　线程

线程机制提供了一种允许多个任务或多个线程共享数据空间的方法。有时线程被认为是轻便进程，但我认为这会引起一点误解。如果把线程看作同一进程的子进程，那将很容易理解它们之间的关系。

Python 提供了两种线程接口：thread 模块提供了便于控制和同步线程的低级原语；threading 模块提供了建立在 thread 模块上层的高级接口。

在纯 Python 的环境（不涉及图形用户界面）下，使用线程变得相当直接。然而，增加了 Tkinter (或 CORBA 或 ILU)会引起一些特殊的问题。这样的系统依靠一个主循环来调度接受到的大量事件，并且线程可以使你的代码设计急剧般地复杂化。

18.1.1　非图形用户界面下的线程

让我们以一个不涉及或者至少不是直接涉及图形用户界面的简单例子开始。这个例子是一个骨干服务器，它用来接收大量的处理数据的请求（这些请求可能来自带有图形用户界面的客户机）。特别的，请求者不希望接收返回去的数据，或者至多接收一个成功/失败指示。

thread1.py

```
import thread, time

class Server:
    def __init__(self):
        self._dispatch = {}
        self._dispatch['a'] = self.serviceA
        self._dispatch['b'] = self.serviceB
        self._dispatch['c'] = self.serviceC
        self._dispatch['d'] = self.serviceD
```

```
    def service(self, which, qual):
        self._dispatch[which](qual)

    def serviceA(self, argin):
        thread.start_new_thread(self.engine, (argin,'A'))

    def serviceB(self, argin):
        thread.start_new_thread(self.engine, (argin,'B'))

    def serviceC(self, argin):
        thread.start_new_thread(self.engine, (argin,'C'))

    def serviceD(self, argin):
        thread.start_new_thread(self.engine, (argin,'D'))

    def engine(self, arg1, arg2):
        for i in range(500):
            print '%s%s%03d' % (arg1, arg2, i),
            time.sleep(0.0001)
        print

server = Server()

server.service('a', '88')       # These calls simulate receipt of
server.service('b', '12')       # requests for service.
server.service('c', '44')
server.service('d', '37')

time.sleep(30.0)
```

代码注解

❶ 构造函数（constructor）产生一个简单的调度字典来选择一个特别的方法。

❷ service 方法是骨干服务器的调度者，当然，一个实际的实现将更加复杂。

❸ 四个服务方法的每一个都会在新的线程里启动相同的新线程，用以传递输入变元和标识符。

```
def  serviceA(self ,argin)
    thread.start_new_thread(self,engine,(argin,'B'))
```

注意，start_new_thread 要求输入到方法内的变元是数组。

❹ 示例的服务引擎相当简单，仅仅是循环和打印到标准输出设备。

❺ 调用 time.sleep(0.0001)非常重要，因为这可以保证线程被阻塞很短的时间，以允许其他线程运行。在下面可以看到关于这个主题的进一步分析。

❻ 在示例的最后一行调用 time.sleep 也相当重要，原因将在下面的分析中给出。

如果运行 thread1.py，你将看到类似于图 18.1 的标准输出。输出虽然很难看，但可以用来解释一些关于线程的知识和代码设计影响行为的方法。

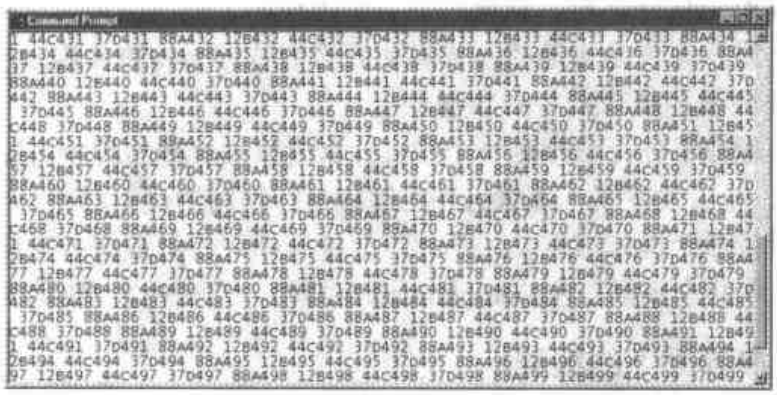


图 18.1　运行 thread1.py 输出结果

示例启动循环 500 次的四个线程，并在每个迭代之内打印一些字符。依赖于所使用操作系统时间调度的方式，每一个进程都将分到一个有限的时间片来运行。在无线程的系统中，执行输入/输出操作将挂起整个进程，直到操作完成，这样可以允许其他进程运行。对于线程，情况类似，除了同一进程中有一个线程进行输入/输出操作时，其他线程的行为会有所差别。然而，打印到标准输出设备并不被认为是阻塞式输入/输出（至少运行简单的 Python 程序是这样），所以线程继续处理指令直至完成分得的时间片或真正进行阻塞式输入/输出。

如果观察图 18.1 输出的片段，当例子运行时在每次打印之后都带有短的 sleep，我们将转向下一个线程并进行一次打印。结果就是四次打印在循环输出，每次的打印来自一个线程，如图 18.2 灰色方框中显示的那样。



图 18.2　运行 threads 并每次打印后有一个 sleep 的效果

如果在 print 语句后去掉 time.sleep 调用，将会改变例子运行时的行为，再次运行例子，我们会得到类似于图 18.3 的输出片段。注意观察每一个线程是如何在分得的时间片内完成循环的，其下一个线程又是如何顺序地输出字符。

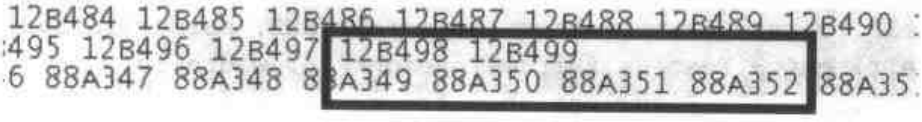


图 18.3　运行 threads，每次打印后没有一个 sleep 的效果

最后的 time.sleep 调用是非常重要的。因为没有图形用户界面，所以在程序中没有主循环。如果去掉这段休眠，当主线程退出时，所有的子线程将死掉。当你第一次开始学习线程编程，这一点是会令人恼怒的。在你意识到主线程已经退出之前，好像什么也没有工作，你会不断重复运行这个程序，有时会得到变化的结果。

这个十分简单的例子所依据的基础是线程间的数据共享，但并不是完全这样。实际上，它能运行是很幸运的。如果查看引擎方法，会发现 for 循环。变量 i 看起来被各个线

程共享，但实际并不是这样。对于变量，每一个线程产生一个引用，这样在一个线程堆栈中的变量相对于线程来说是本地的。依靠如此多的特性来构建线程程序看起来不是一个好主意，因为要遵循和破除太多的准则。

简单地说，问题在于你无法预测一个线程什么时候被其他线程抢占*。当你正在运行一个原子**操作时，抢占发生了，那么另外的线程执行相同的代码路径产生的结果是灾难性的。如果线程维持原子操作很重要，那就需要使用锁（信号量或者互斥体***），防止其他线程从获得的输入共享数据。在 Demo/thread 字典中，有一些典型的互锁 Python 程序。

18.1.2 图形用户界面下的线程

使用图形用户界面下的线程必须小心一些。总体说来，使用线程的系统必须保证包含主循环的主线程能不断地更新图形用户界面。下面的例子是根据标准 Python 程序中的一个 wpi.py 改写的。其中，一个主线程产生图形用户界面并开始主循环；同时，另一个线程连续不断地计算 π 值。另外，这个例子使用了 Tkinter 的事件循环来周期更新图形用户界面。

thread2.py

```
# Display digits of pi in a window, calculating in a separate thread.
# Compare with wpi.py in the Demo/threads/wpi.py

import sys
import time
import thread
from Tkinter import *

class ThreadExample:
    def __init__(self, master=None):
        self.ok               = 1
        self.digits           = []
    self.digits_calculated = 0
        self.digits_displayed  = 0
        self.master           = master
 thread.start_new_thread(self.worker_thread, ())                    ❶

        self.frame = Frame(master, relief=RAISED, borderwidth=2)
        self.text = Text(self.frame, height=26, width=50)
        self.scroll = Scrollbar(self.frame, command=self.text.yview)
        self.text.configure(yscrollcommand=self.scroll.set)
        self.text.pack(side=LEFT)
        self.scroll.pack(side=RIGHT, fill=Y)
        self.frame.pack(padx=4, pady=4)
        Button(master,text='Close',
command=self.shutdown).pack(side=TOP)
```

* 当一段正在运行的进程被停止来运行另一个进程（进程交换），这就是通常所指的被“抢占”。
** 原子操作在其他任何进程或者线程能够跟在同一个代码路径之前完成其全部操作。
*** 相互排斥的锁。

```
        self.master.after(100, self.check_digits)                    ❷

    def worker_thread(self):
    k, a, b, a1, b1 = 21, 41, 11, 121, 41
    while self.ok:
            # Next approximation
            p, q, k = k*k, 21*k+11, k+11
            a, b, a1, b1 = a1, b1, p*a+q*a1, p*b+q*b1
            # Print common digits
            d, d1 = a/b, a1/b1
            while d == d1:
                self.digits.append(`int(d)`)
                a, a1 = 101*(a%b), 101*(a1%b1)
                d, d1 = a/b, a1/b1
            time.sleep(0.001)                                        ❸

    def shutdown(self):                                              ❹
        self.ok =0
        self.master.after(100, self.master.quit)

    def check_digits(self):
        self.digits_calculated = len(self.digits)
        diff = self.digits_calculated - self.digits_displayed
        ix = self.digits_displayed
        for i in range(diff):

        self.text.insert(END, self.digits[ix+i])
    self.digits_displayed =  self.digits_calculated
    self.master.title('%d digits of pi' % self.digits_displayed)

    self.master.after(100, self.check_digits)                        ❷

root = Tk()
root.option_readfile('optionDB')
example = ThreadExample(root)
root.mainloop()
```

代码注解

❶ 连续计算 π 值的线程 worker_thread 如同先前的方法，即从 start_new_thread 函数开始执行。

❷ 我们使用 after 在事件队列里来放置一个定时事件，这样可以在不阻塞主循环的情况下加入延迟。

❸ 我们再一次在 work_thread 循环后加入短暂的 sleep 来保证主循环能够循环（详情见后）。

❶ 为了保证能够正常地关闭程序，我们使用了 ok 标志。这样可以导致 work_thread 退出，并在等待 100ms 后杀死图形用户界面进程。更实际的应用程序需要附加的方法。

❷ 由于 after 事件是只执行一次的事件，所以当定时事件需要重复执行时，必须放置返回队列的一个事件。

运行 thread2.py 将会得到类似于图 18.4 的结果，新得到的数字大约每隔 100ms 就附加到文本窗口。你可以使用或者注释掉 work_thread 中的 sleep 例程，当运行这段代码，将会发现单击 Close 按钮时程序会用较长的时间来响应。在上述情况下，work_thread 在被挂起前会使用一个完整的时间片来运行，因此图形用户界面的事件队列耗尽的频率会降低。

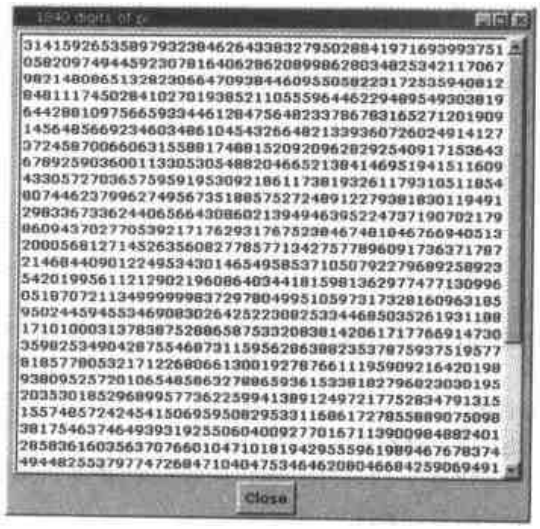


图 18.4 GUI 应用线程

当一个线程运行时，我们可以试着增加另外一个。由于这个例子并不是直接有用，所以仅仅是用来解释多重线程的概念。我们每隔 100ms 就随机地改变 Close 按钮的颜色，下面是需要改变的代码。

thread3.py

```
import time, thread, random
from Tkinter import *

class ThreadExample:
    def __init__(self, master=None):
# -------- Code Removed ----------------------------------------------

        self.btn.pack(side=TOP, pady=5)
        thread.start_new_thread(self.worker_thread2,())
        self.master.after(100,self.check_digits)

        def worker thread1(self):
# -------- Code Removed ----------------------------------------------

    def worker_thread2(self):
    while self.ok:
        self.btn.configure(background=self.color())
        time.sleep(0.1)
```

```
    def color(self):
    rc = random.choice
    return '#%2x%02x%02x' % (rc(range(0,255)),rc(range(0,255)),
                             rc(range(0,255))
# ------- Code Removed ----------------------------------------------
```

有两点值得注意：首先，第二个线程应该在窗口产生以后再开始运行。如果两个线程同时运行，Close 按钮将不会产生。这会阻止第二个线程运行，但不会引起程序崩溃。合理地运用线程这个重要特性，将会给编程带来方便。其次，在 worker_thread 中 100ms 的 sleep 事件是用来对线程进行周期定时，而不是仅仅释放线程。

如果运行 thread3.py 会看到类似于运行 thread2.py 时得到的窗口，只是 Close 按钮的颜色会快速地改变，如图 18.5 所示。

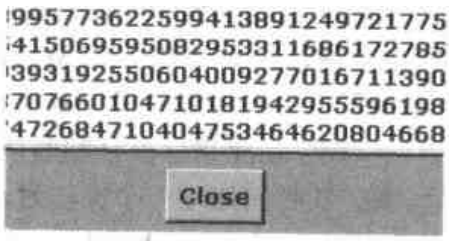


图 18.5　线程中的随机改变颜色

Python 提供了一个高层的界面——threading，可以管理许多线程间的协作，用户也不需要加入 sleep 调用来保证线程将释放控制权给其他线程。为了解释这一点，我们把 thread3.py 改变成 threading 模块。

Thread4.py

```
from threading import *

class ThreadExample:
    def __init__(self, master=None):
        self.ok              = 1
# ------- Code Removed ----------------------------------------------
        self.master          = master

self.thread1= Thread(target=self.worker_thread1)                    ❶
self.thread2= Thread(target=self.worker_thread2)

# ------- Code Removed ----------------------------------------------

        self.master.after(100, self.check_digits)
        self.thread1.start()                                        ❷
        self.thread2.start()

    def worker_thread1(self):
# ------- Code Removed ----------------------------------------------
###     time.sleep(0.001)                    ❸
# ------- Code Removed ----------------------------------------------
```

代码注解

❶ 我们为每一个工作线程产生一个线程实体。注意：这并不启动线程，需要在后面进行。

❷ 一旦图形用户界面产生了，我们就启动线程。

❸ 注意 work_thread1 中短暂的 sleep 事件已经注释掉了。

如果运行 thread4.py 会得到类似于 thread3.py 的输出。有趣的是，用 threading 来实现将比 thread 版本快 30% 。

18.2 “after”处理

after 方法被频繁地在例子中使用，包括上面的 threading 例子，它用来提供简单的警告回应。对于 X window 的程序员来说，after 类似于 XtAppAddTimeOut。after_idle 则类似于 XtAppAddWorkProc，它提供了一个简单的技术来定义后台任务，也就是指当事件队列中没有事件时将被执行的任务（当 after 事件被挂起时例外）。

下面举一个简单例子，即当系统长时间处理一个操作时，使用 after_idle 来实现一个 busy 光标。在开始操作之前，程序先显示一个 watch 光标*，并寄存工作过程。当 after_idle 回应过程被激活，光标又恢复到正常的状态。

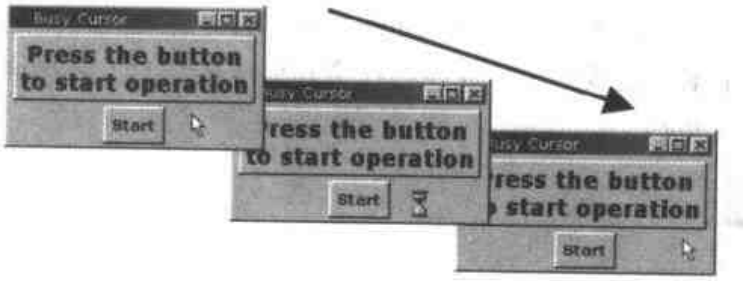


图 18.6 使用 after_idle 来实现 busy 光标

Busy cursor.py

```
import time
from Tkinter import *

class AfterIdleExample:
  def __init__(self, master=None):
    self.master          = master
    self.frame = Frame(master, relief=RAISED, borderwidth=2)
    Label(self.frame,text='Pressthebutton\ntostart operation').pack()
    self.frame.pack(padx-4, pady=4)
    Button(master, text='Start', command=self.startOP).pack(side=TOP)

  def startOP(self):
    self.displayBusyCursor()
```

* 在 Win32 中，watch 光标将显示给 watch 选择的当前光标。

```
        time.sleep(10.0)  # simulate a long operation

    def displayBusyCursor(self):
        self.master.configure(cursor='watch')
        self.master.update()
        self.master.after_idle(self.removeBusyCursor)

    def removeBusyCursor(self):
        self.master.configure(cursor='arrow')


root = Tk()
root.option_readfile('optionDB2')
root.title('Busy Cursor')
AfterIdleExample(root)
root.mainloop()
```



代码注解

❶ 函数 displayBusyCursor 将光标变为正确的形式，接着调用 update 保证光标在长时间操作开始之前能够显示。

❷ 方法 after_idle 寄存 removeBusyCursor，后者在事件队列为空时被调用。只有等到 sleep 结束之后，主循环才有机会循环下去。

❸ 函数 removeBusyCursor 的所有功能是恢复光标形状。

注意　在完整的实现过程中，为了使 busy 光标更通用，应该得到并保存当前光标以便于恢复成相同的光标。

可以使用其他方法实现异步操作。不幸的是，在当前的 Tcl/Tk 版本下，这种操作仅适合于 Unix。在 Tk 的 8.0 版本中，对于 createfilehandler，Win32 的支持撤销了，它的作用等同于 X Window 中的 XtAppAddInput。它允许用户在文件类操作发生时绑定正在运行的一个回调。这普遍地应用在套接字上。通常，我不愿使用仅对特定操作系统有效的方法，但是这一个却是例外。

许多程序要求能够响应外部事件，例如从套界面接收数据。用户有不同的选择来实现这种系统，但通常会在以下两种当中作出选择：要么阻塞，等待输入；要么基于周期性的查询，直到接收到数据。现在，在图形用户界面被激活时，当然不能够阻塞，除非对外部事件的响应在线程内部发生。然而，用户可能不愿意或者不能够使用线程，或者用户使用的系统根本就不支持线程，在这种情况下，使用 createfilehandle 提供了方便的技术。

为了便于解释，我们先给出一个实现时间服务器的简单的客户/服务器程序。它可以被用作某些类型的监视器，以保证程序中关键性的元件正在工作。服务器每隔一分钟就向一个给定的端口发送时间信息，而例子中的客户仅仅显示这些信息。我将客户和服务器在同一个程序中实现，这样可以节省代码。

Client server.py

```
from Tkinter import *
import sys, socket, time

class Server:
    def __init__(self):
        host = socket.gethostbyname(socket.gethostname())
        addr = host, 5000
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.bind('', 0)
        while 1:
            time.sleep(60.0)
            s.sendto(time.asctime(time.localtime(time.time())), addr)

class GUIClient:
    def __init__(self, master=None):
        self.master          = master
        self.master.title('Time Service Client')
        self.frame = Frame(master, relief=RAISED, borderwidth=2)
        self.text = Text(self.frame, height=26, width=50)
        self.scroll = Scrollbar(self.frame, command=self.text.yview)
        self.text.configure(yscrollcommand=self.scroll.set)
        self.text.pack(side=LEFT)
        self.scroll.pack(side=RIGHT, fill=Y)
        self.frame.pack(padx=4, pady=4)
        Button(master,text='Close',
                command=self.master.quit).pack(side=TOP)

        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind('', 5000)

        tkinter.createfilehandler(self.socket, READABLE, self.ihandler)

        self.master.after(5000, self.doMark)

    def ihandler(self, sock, mask):
        data, addr = sock.recvfrom(256)
        self.text.insert(END, '%s\n' % data)

    def doMark(self):
        self.text.insert(END, 'waiting...\n')
        self.master.after(5000, self.doMark)

if len(sys.argv) < 2:
    print 'select -s (server) or -c (client)'
    sys.exit(2)
if sys.argv[1] == '-s':
    server=Server()
elif sys.argv[1] == '-c':
```

```
root = Tk()
root.option_readfile('optionDB')
example = GUIClient(root)
root.mainloop()
```

代码注解

❶ 在本书中对于套接口涉及不多，所以一些读者可能对此不是十分熟悉。这里给出一个简单的解释：套接口是大多数操作系统上都可获得基本网络功能，我们就是通过建立套接口来发送消息。本质上，我们在特定的主机系统上连接一个带有数字编号的端口，接着使用 UDP（数据报）或者 TCP（流）协议来发送和接收消息。本例中我们采用数据报协议*。

❷ 本例的客户端使用和服务器相同的端口建立套接口，接下来它注册文件句柄，当套接口上三个事件之一发生时，文件句柄将会被调用。当一个套接口接收数据时，事件 mask 的可能值为 READABLE；当一个套接口被写入时，为 WRITABLE；当有错误发生时，为 EXCEPTION。我们建立 after 回调，每隔 5s 向窗口写入文本。

❸ 文件事件的句柄非常简单。mask 参数规定在调用中得到的文件操作的类型，因此单独一个句柄就能完成所有的文件操作。

为了运行这个例子，首先启动服务器进程：

```
% python client_server.py -s &
```

接下来启动客户进程：

```
% python client_server.py -c
```

图 18.7 给出了例子在运行时屏幕的情况。



图 18.7　时间-客户服务器

18.3　小结

本章简要地介绍了线程和异步技术，特别的是，直接引导用户熟悉在 threading 模块

*UDP 协议是一个非连接的协议，数据包被送到网络传输到一个具有规定端口号的特定的主机中。没有确认发送到发方以确认数据包已经接受到了，并且一旦出现问题，也没有自动的重发消息机制。因此，对于一个数据报是否发送到了目的地并没有保证。数据报一般是用在并不重要的消息上，通常是在 LAN 内以避免昂贵的连接开销。

下 Python 文件的信息，这提供了比 thread 模块更高级的线程接口。在这个领域内有许多可选的方案，一定要根据程序确切的本质来确定最好的解决方法。这里讲述的东西可以作为简介。

第 19 章　发布 Tkinter 程序

如果你连续几个星期关注 Python 的新闻组，你就会看到至少有一个问题经常被提到，那就是“怎样发布 Python 程序？”。本章学习一些 Unix 和 Win32 系统下发布程序的方法。Unix 下相对容易操作，而 Win32 下有几种可能的解决办法。

19.1　发布程序的一般问题

发布和安装 Python 程序的主要问题在于要保证终端用户拥有程序运行所必须的所有元件。要做到这一点有一些可选择的方法，但首先你必须确保用户的终端系统安装了大多数的元件：

1. 能够执行 Python（即系统安装了 python.exe）
2. 程序的启动装置（Unix 下的 xxx script，Win32 下的 xxx.bat）
3. Python 在 Unix 下的共享目标文件（xxx.so），在 Win32 下的动态链接库（xxx.dll）
4. 外部库函数（Python/Lib/lib-dynload/xxx.[soldll]）
5. Python 库文件（Python/Lib/xxx.py）
6. 程序指定的 Python 库文件（在正确的位置）
7. 运行 Tcl/Tk（Tcl/bin，Tcl/lib，Tk/bin，/Tk/lib 等）
8. Pmw（Python/Lib/Pmw）
9. 其他你的程序需要的扩展、数据文件和数据库等

你将要对怎样处理 Python 和 Tcl/Tk 的发布做出一个决定：要么让终端用户独立的安装这些软件，要么随你的程序一起发布。一般说来后一种方法较好，因为 Python 的普及程度仍旧在扩展，所以当前的用户终端不一定安装了这些软件。当然，我们完全期待在将来这种情况会改变。

如果你的确决定把 Python 和 Tcl/Tk 随程序一起发布，你将需要做出另一个决定：是公开的安装以便于终端用户方便地存取 Python 和（或）Tcl/Tk，还是本地安装，即通过你的程序来存取它们。虽然对于 Win32 系统使用本地安装更容易，但通常都是公开安装。

最后，你必须考虑程序将要支持的结构。如果仅仅是针对 Win32 系统，那么很简单，但是对于 Unix 却有多种结构，一般通过编制源代码来支持各种结构。要是你不想编制这些代码，就必须为指定的平台提供二进制代码。这是一个完全不同的问题，不论从商业还是从技术角度来说都是值得认真考虑的。这已经超过了本书这一简短章节要论述的范围。

19.2 发布 Unix 程序

一旦可以建立 Python 和 Tcl/Tk，那么支持你的程序通常很简单。一般说来，Unix 的终端用户能够建立和安装以上两个软件，所以你只需要要求终端用户注意使用即可。这样，安装你的程序就像从 tar 文件中解压缩或正确编辑用户环境一样简单。现在让我们假设已经是这种情况了，因此我们将注意力放在安装你的程序并运行。

首先，需要一个可执行程序来启动你的程序。当前的目标是使用最小的 Python 脚本来进入程序的 main 模块（注意：每次唤醒代码时都将执行它，所以要尽量地简短，见 17.1 节的“每天加速”）。

下面给出一个最小 Python 脚本：

```
#!/usr/bin/env python
    import myapplication
    myapplication.main( )
```

在某些情况下不能使用#!/usr/bin/env python，所以你不得不给出明确的路径，例如 /usr/local/bin/python。小提示：第一种形式空格是有意的，对于 Unix 的子进程来说自然而然地会把空格转换成斜线。接下来，你应该加入一些脚本来完成这项工作。脚本假设环境变量 PYTHONPATH 已经设置且把路径包含入../Python/Lib/，而 MyApplication.py 也安装了。你或许不愿意修改用户环境，但是却可以在你的脚本中来完成。

```
#! /usr/bin/env python

import sys
sys.path.insert(0,'/opt/tourapp/lib')

import myapplication
myapplication.main()
```

很清楚，还可以做进一步的完善，但目前的代码在实际中已经可以很好地工作了。

19.3 发布 Win32 程序

发布 Win32 程序要比 Unix 程序麻烦一些，因为你有多种可行方案。我一贯提倡使用最简单的方案，因为其他方案的确要与注册打交道。这意味着你将使用诸如 InstallShield 之类的安装工具，它可以自动地完成程序元件的安装与注册的全过程。更重要的是，它可能预备同时安装 uninstaller，这样能够不需用户的干预而自动地卸载注册信息和已安装的文件。

对于 Win32 系统，需要做出类似于 Unix 系统的选择。使用最小化的代码让程序运行的需求仍旧存在。实际的问题是 Win32 系统在从可点击的文件起运行 Python 代码将会打开一个 MS－DOS 窗口。经过一些努力，这一点可以避免。

首先，让我们决定怎样包装这个程序。在本例中，我们将产生一个独立的 Win32 程序，并且我们所需要的支持程序运行的所有元件都安装在一个单独的目录下（可以说万事具备）。我们不打算修改任何注册信息，而仅仅需要桌面上的一个图标，用户双击它就

可以启动程序。

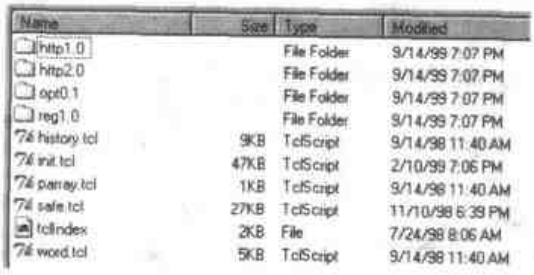
让我们先来看看显示在图 19.1 中顶层目录的内容：在这个目录下我们已经安装了 Python 的可执行程序（python.exe 和 pythonw.exe），系统的动态链接库文件（_tkinter.pyd，python15.dll，tcl80.dll，tk80.dll）和程序特有的动态链接库文件（例如 btrieve.dll 和 sio.pyd）。我们还有标准 Python 库的目录，其中包含 Pmw*（或许包含程序指定的文件）。



Name	Size	Type	Modified
doc		File Folder	9/14/99 7:07 PM
Lib		File Folder	9/14/99 7:07 PM
libs		File Folder	9/14/99 7:07 PM
tcl8.0		File Folder	9/14/99 7:07 PM
tk8.0		File Folder	9/14/99 7:07 PM
_tkinter.pyd	20KB	PYD File	4/13/99 10:55 AM
bsddb.pyd	69KB	PYD File	4/13/99 10:55 AM
btrieve.dll	59KB	Application	5/6/99 8:39 AM
elapsedTimer.pyd	6KB	PYD File	4/28/98 6:07 PM
parser.pyd	29KB	PYD File	4/13/99 10:56 AM
python.exe	5KB	Application	4/13/99 11:31 AM
python15.dll	545KB	Application	4/13/99 11:29 AM
pythonw.exe	6KB	Application	4/13/99 10:56 AM
sio.pyd	14KB	PYD File	4/28/98 7:11 PM
tcl80.dll	362KB	Application	3/8/99 6:17 PM
tclpip80.dll	24KB	Application	3/8/99 6:17 PM
tk80.dll	744KB	Application	3/8/99 9:31 PM
zlib.pyd	52KB	PYD File	4/13/99 10:56 AM

图 19.1　c:\python 目录下内容

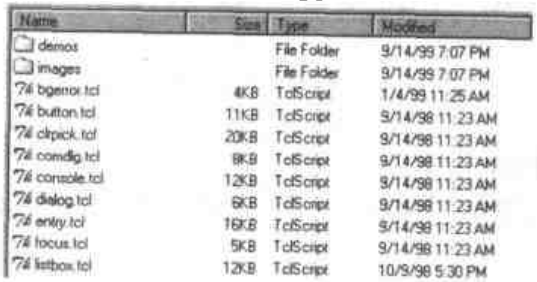
Tcl 也安装在顶层目录，其中包含了如图 19.2 所示的目录结构。



Name	Size	Type	Modified
http1.0		File Folder	9/14/99 7:07 PM
http2.0		File Folder	9/14/99 7:07 PM
opt0.1		File Folder	9/14/99 7:07 PM
reg1.0		File Folder	9/14/99 7:07 PM
history.tcl	9KB	TclScript	9/14/98 11:40 AM
init.tcl	47KB	TclScript	2/10/99 7:06 PM
parray.tcl	1KB	TclScript	9/14/98 11:40 AM
safe.tcl	27KB	TclScript	11/10/98 6:39 PM
tclIndex	2KB	File	7/24/98 8:06 AM
word.tcl	5KB	TclScript	9/14/98 11:40 AM

图 19.2　Tcl 目录下内容

相似地，Tk 目录安装在如图 19.3 所示的 python 目录下。



Name	Size	Type	Modified
demos		File Folder	9/14/99 7:07 PM
images		File Folder	9/14/99 7:07 PM
bgerror.tcl	4KB	TclScript	1/4/99 11:25 AM
button.tcl	11KB	TclScript	9/14/98 11:23 AM
clrpick.tcl	20KB	TclScript	9/14/98 11:23 AM
comdlg.tcl	8KB	TclScript	9/14/98 11:23 AM
console.tcl	12KB	TclScript	9/14/98 11:23 AM
dialog.tcl	6KB	TclScript	9/14/98 11:23 AM
entry.tcl	16KB	TclScript	9/14/98 11:23 AM
focus.tcl	5KB	TclScript	9/14/98 11:23 AM
listbox.tcl	12KB	TclScript	10/9/98 5:30 PM

图 19.3　Tk 目录下内容

现在，我们要生成一个批处理文件，用它来设置环境变量并启动我们程序简短的 Python 脚本。

---

* Pmw 不是标准 Python 发布。你可以到 http://www.dscpl.com.au/pmw/下载到 Pmw。

Start.bat

```
Set PYTHONPATH=C:\PYTHON\LIB;C:\PYTHON;C:\MyApplication\Common
Set TCL_LIBRARY=C:\PYTHON\TCL8.0
Set Tk_LIBRARY=C:\PYTHON\TK8.0
c:\Python\pythonw startApp.py
```

再没有被 Python 脚本简单的了。

startApp.py

```
Import myApplication
MyApplication.main()
```

现在，我们需要改变批处理程序 start.bat 的特性，我们用右键点击 start.bat，选择属性。

接下来，我们设置工作目录为包含 MyApplication.py 的目录，在 Run 组合框中选择 Minimized，并选中 Close on exit。这些选项显示在图 19.4 中。如果想要改变程序的图标，就点击 Change Icon...按钮，选择一个图标。选择代表程序的图标时要注意确保程序安装到了正确的位置。

现在，当你双击 Start.bat 文件时，你将不会得到带有 MS－DOS 窗口的程序。

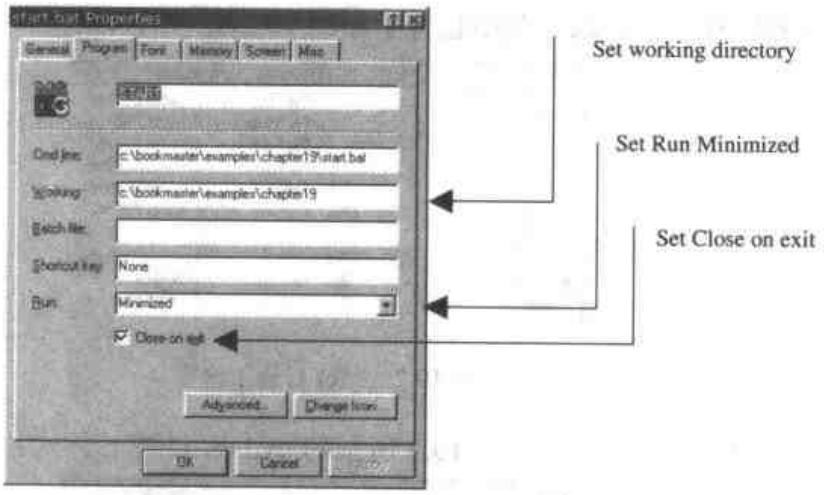


图 19.4　设置批处理文件属性

很明显，可以通过其他方式达到这个效果，如果经常浏览 Python 新闻组（见附录 G 的“Python 新闻组”），你会经常发现怎样去完成的一些其他建议。

19.4　Python 发布工具

程序开发者能够获得大量发布工具，而且还在不断的开发。由于将要覆盖如此之多的工具，所以我们不打算一一涉及。

一个值得一看的工具是 freeze，它可以把你的 Python 程序和所有必须的支持模块包装到一个嵌入的 C 程序中。时至今日，我并没有发现一定要使用它，但我确信对你们当

中的一些人来说这是一个可选的方案。

另一个是 Squeeze 工具，它可以把 Python 程序和所有所需的支持模块压缩到一个单独的被压缩的包当中。在附录 G 的“Python Works”中有更详细的说明。

还有一个 Distutils 专业组（SIG），致力于开发发布 Python 工具的工程师都可以去看看，这些工具可以带有 C 扩展也可以没有，可以适用于所有的平台。当这本书出来的时候，这些工具的早期版本都已经通过了大量的测试了。到 http://www.python.org/sigs/distutils-sig 上可以找到关于这方面发展的最新消息。

第4部分

附　录

附录 A　Tk 到 Tkinter 映射

此附录详细给出了 Tk 命令和参数到 Tkinter 方法和选项的映射。其中的顺序基本按照一本手册而定，该手册是 Paul 和 Jeff Trainer 为 Tcl/Tk 出版的一本书（*Tcl/Tk is a Nutshell:A Desktop Quick Reference* ,O'Reilly and Associates,Inc 出版）。不过，映射不包含任何 Tcl 信息。我假设你想直接把 Tk 译为 Tkinter。很多情况下，可能有更好的渠道完善 Tcl/Tk 代码序列到 Tkinter。Tkinter 实现很多 Tk 命令，如继承控件方法，这对许多初级编程人员可能引起些混淆。

一般 Tk 控件信息

所有控件都如下创建：

```
widget=Widget(master[,option=value[,option=value]])
```

这里 Widget 是设定的 Tkinter 控件类（如按钮 Button），widget 为实例。Widget 配置选项可以按参数传递到创建调用。选项可以通过配置方法修改，通过 cget 存取。一般地，存取可以通过控件字典关键字进行（value=widget['option']或 widget['option'] = value）。

一些公共控件选项支持这里描述的多个控件，为使附录简洁，它们不为单个控件重复。对于屏幕单位选项，值为像素，除非有一个可选的字母后缀：c(cm)，i(inch)，m(mm)或 p(points)。

表 A.1　标准控件选项

Tk	Tkinter
-activebackground color	activebackground=color
-activeborderwidth width	activeborderwidth=width
-activeforeground color	activeforeground=color
-anchor anchorPos	anchor=anchorPos
-background color	background=color
-bitmap bitmap	bitmap=bitmap
-borderwidth width	borderwidth=width
-command tclCommand	command=pythonCommand
-cursor cursor	cursor=cursor
-disabledforeground color	disabledforeground=color
-exportselection boolean	exportselection=boolean
-font font	font=font

（续）

Tk	Tkinter
-foreground color	foreground=color
-height height|textChars	height=height|textChars
-highlightbackground color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-image image	image=image
-insertbackground color	insertbackground=color
-insertborderwidth width	insertborderwidth=width
-insertofftime milliseconds	insertofftime=milliseconds
-insertontime milliseconds	insertontime=milliseconds
-insertwidth width	insertwidth=width
-jump boolean	jump=boolean
-justify left|center|right	justify=LEFT|CENTER|RIGHT
-orient horizontal|vertical	orient=HORIZONTAL|VERTICAL
-padx width	padx=width
-pady height	pady=height
-relief flat|groove|raised|ridge|sunken	Relief=FLAT|GROOVE|RAISED| RIDGE|SUNKEN|SOLID
-repeatdelay milliseconds	repeatdelay=milliseconds
-repeatinterval milliseconds	repeatinterval=milliseconds
-selectbackground color	selectbackground=color
-selectborderwidth width	selectborderwidth=width
-selectforeground color	selectforeground=color
-setgrid boolean	setgrid=boolean
-state normal|disabled|active	state=NORMAL|DISABLED|ACTIVE
-takefocus focusType	takefocus=focusType
-text string	text=string
-textvariable variable	textvariable=variable
-troughcolor color	troughcolor=color
-underline index	underline=index
-width width|textChars	width=width|textChars
-wraplength length	wraplength=length
-xscrollcommand cmdPrefix	xscrollcommand=command
-yscrollcommand cmdPrefix	yscrollcommand=command

表 A.2 Tk 特别变量

Tk	Tkinter
tk_library	Not available
tk_patchLevel	Not available
tkPriv	Not available
tk_strictMotif	window.tk_strictMotif(Boolean)
tk_version	TkVersion

画布控件(Canvas Widget)

表 A.3 画布控件标准控件选项

Tk	Tkinter
-background color	background=color
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-height height	height=height
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness number	highlightthickness=number
-insertbackground color	insertbackground=color
-insertborderwidth width	insertborderwidth=width
-insertofftime milliseconds	insertofftime=milliseconds
-insertontime milliseconds	insertontime=milliseconds
-insertwidth width	insertwidth=width
-relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED| RIDGE|SUNKEN|SOLID
-selectbackground color	selectbackground color
-selectborderwidth width	selectborderwidth width
-selectforeground color	selectforeground color
-takefocus focusType	takefocus focusType
-width width	width width
-xscrollcommand tclCommand	xscrollcommand tclCommand
-yscrollcommand tclCommand	yscrollcommand tclCommand

表 A.4 画布控件特有选项

Tk	Tkinter
-closeenough float	closeenough float
-confine Boolean	confine Boolean
-scrollregion corners	scrollregion=(x1,y1,x2,y2)
-xscrollincrement distance	xscrollincrement distance
-yscrollincrement distance	yscrollincrement distance

表 A.5 画布方法

Tk	Tkinter
canvas addtag above tagOrId	canvas addtag above tagOrId
canvas addtag all	canvas addtag all(newtag)
canvas addtag below tagOrId	canvas.addtag_below(newtag,tagOrId)
canvas addtag closest x y[halo][start]	canvas.addtag_closest(newtag,x,y,[,halo][,start])
canvas addtag enclosed x1 y1 x2 y2	canvas.addtag_enclosed(newtag,x1,y1,x2,y2)
canvas addtag overlapping x1 y1 x2 y2	canvas.addtag_overlapping(newtag,x1,y1,x2,y2)
canvas addtag withtag tagOrId	canvas.addtag_withtag(newtag,tagOrId)
canvas bbox tagOrId [tagOrId ...]	canvas.bbox(tagOrId [,tagOrId ...])

（续）

Tk	Tkinter
canvas bind tagOrId [sequence[command]]	canvas.tag_bind(tagOrId [,sequence[,command]])
canvas canvasx screenx [gridspacing]	canvas.canvasx(screenx [,gridspacing])
canvas canvasy screeny [gridspacing]	canvas.canvasy [,gridspacing])
canvas cords tagOrId [x0 y0 ...]	canvas.cords(tagOrId [,x0,y0 ...]
canvas create TYPE x y [x y ...]	canvas.create_TYPE(x,y,[,x,y ...]
[option value ...]	[,option=value ...]
canvas dchars tagOrId first [last]	canvas.dchars(tagOrId,first [,last])
canvas delete [tagOrId ...]	canvas.delete([tagOrId ...])
canvas dtag tagOrId [tagToDelete]	canvas dtag(tagOrId [,tagToDelete])
canvas find above tagOrId	canvas.find_above(tagOrId)
canvas find all	canvas.find_all()
canvas find below tagOrId	canvas.find_below(tagOrId)
canvas find closest x y [halo][start]	canvas.find_closest(x,y,[,halo][,start])
canvas find enclosed x1 y1 x2 y2	canvas.find_enclosed(x1,y1,x2,y2)
canvas find overlapping x1 y1 x2 y2	canvas.find_overlapping(x1,y1,x2,y2)
canvas find withtag tagOrId	canvas.find_withtag(tagOrId)
canvas focus tagOrId	canvas.focus(tagOrId)
canvas gettags tagOrId	canvas.gettags(tagOrId)
canvas icursor tagOrId index	canvas.icursor(tagOrId,index)
canvas index tagOrId index	canvas.index(tagOrId,index)
canvas insert tagOrId beforeThis string	canvas.insert(tagOrId,string)
canvas itemcget tagOrId option	canvas.itemcget(tagOrId,option)
canvas itemconfigure tagOrId[option=value ...]	canvas.itemconfigure(tagOrId[,option=value ...])
canvas lower(tagOrId [belowThis]	canvas.lower(tagOrId [,belowThis])
canvas move tagOrId xAmount yAmount	canvas.move(tagOrId,xAmount,yAmount)
canvas postscript[option value ...]	canvas.postscript([option=value ...])
canvas raise tagOrId[aboveThis]	canvas.tkraise(tagOrId,[,aboveThis])
canvas scale tagOrId xOrigin yOrigin xScale yScale	canvas.scale(tagOrId,xOrigin,yOrigin,xScale,yScale)
canvas scan mark x y	canvas.scan_mark(x,y)
canvas scan dragto x y	canvas.scan_dragto(x,y)
canvas select adjust tagOrId index	canvas.select_adjust(tagOrId,index)
canvas select clear	canvas.select_clear()
canvas select from tagOrId index	canvas.select_from(tagOrId,index)
canvas select item	canvas.select_item()
canvas select to tagOrId index	canvas.select_to(tagOrId,index)
canvas type tagOrId	canvas.type(tagOrId)
canvas xview args	canvas.xview(args)
canvas xview moveto fraction	canvas.xview_moveto(fraction)
canvas xview scroll number units|pages	canvas.xview_scroll(number,UNITS|PAGES)
canvas yview args	canvas.yview(args)
canvas yview moveto fraction	canvas.yview_moveto(fraction)
canvas yview scroll number units|pages	canvas.yview_scroll(number,UNITS|PAGES)

画布项类型

表 A.6 创建弧

Tk	Tkinter
canvas create arc x1 y1 x2 y2	canvas.create_arc(x1,y1,x2,y2,
[option value ...]	[,option=value ...])

表 A.7 弧选项

Tk	Tkinter
-extent degrees	extent=degrees
-fill color	fill=color
-outline color	outline=color
-outlinestipple bitmap	outlinestipple=bitmap
-start degrees	start=degrees
-stipple bitmap	stipple=bitmap
-style pieslice|chord|arc	style=PIESLICE|CHORD|ARC
-tags tagList	tags=tagList
-width outlineWidth	width=outlineWidth

表 A.8 创建位图 bitmap

Tk	Tkinter
canvas create bitmap x y [option value ...]	canvas.create_bitmap(x,y,[,option=value ...])

表 A.9 位图 bitmap 选项

Tk	Tkinter
-anchor anchorPos	anchor=anchorPos
-background color	background=color
-bitmap bitmap	bitmap=bitmap
-foreground color	foreground=color
-tags tagList	tags=tagList

表 A.10 创建图像 image

Tk	Tkinter
canvas create image x y [option value ...]c	canvas.create_image(x,y,[,option=value ...])

表 A.11 图像 image 选项

Tk	Tkinter
-anchor anchorPos	anchor=anchorPos
-image image	image=image
-tags tagList	tags=tagList

表 A.12 创建线 line

Tk	Tkinter
canvas create line x1 y1…xN yN [option value…]	canvas.create_line(x1,y1,…xN,yN, [option=value…])

表 A.13 线 line 选项

Tk	Tkinter
-arrow none|first|last|both	arrow=NONE|FIRST|LAST|BOTH
-arrowshape shape	arrowshape=shape
-capstyle butt|projecting|round	capstyle=BUTT|PROJECTING|ROUND
-fill color	fill=color
-joinstyle bevel|miter|round	joinstyle=BEVEL|MITER|ROUND
-smooth boolean	smooth=boolean
-splinesteps number	splinesteps=number
-stipple bitmap	stipple=bitmap
-tags tagList	tags=tagList
-width outlineWidth	width=outlineWidth

表 A.14 创建椭圆 oval

Tk	Tkinter
canvas create oval x1 y1 x2 y2 [option value…]	canvas.create_oval(x1,y1,x2,y2 [,option=value…])

表 A.15 椭圆 oval 选项

Tk	Tkinter
-fill color	fill=color
-outline color	outline=color
-stipple bitmap	stipple=bitmap
-tags tagList	tags=tagList
-width outlineWidth	width=outlineWidth

表 A.16 创建多边形弧

Tk	Tkinter
canvas create polygon x1 y1…xN yN [option value…]	Canvas.create_polygon(x1,y1,…xN,yN, [option=value…])

表 A.17 多边形选项

Tk	Tkinter
-fill color	fill=color
-outline color	outline=color
-smooth boolean	smooth=boolean
-splinesteps number	splinesteps=number
-stipple bitmap	stipple=bitmap
-tags tagList	tags=tagList
-width outlineWidth	width=outlineWidth

表 A.18 创建矩形

Tk	Tkinter
canvas create rectangle x1 y1 x2 y2	canvas.create_rectangle(x1,y1,x2,y2
[option value…]	[,option=value…])

表 A.19 矩形选项

Tk	Tkinter
-fill color	fill=color
-outline color	outline=color
-stipple bitmap	stipple=bitmap
-tags tagList	tags=tagList
-width outlineWidth	width=outlineWidth

表 A.20 创建文本

Tk	Tkinter
canvas create text x y	canvas.create_text(x,y,
[option value…]	[,option=value…])

表 A.21 文本选项

Tk	Tkinter
-anchor anchorPos	anchor=anchorPos
-fill color	fill=color
-font font	font=font
-justify leftrightcenter	justify=leftrightcenter
-stipple bitmap	stipple=bitmap
-tags tagList	tags=tagList
-text string	text=string
-width lineLength	width=lineLength

表 A.22 创建窗口

Tk	Tkinter
Canvas create window x y	canvas.create_window(x,y,
[option value…]	[,option=value…])

表 A.23 窗口选项

Tk	Tkinter
-anchor anchorPos	anchor=anchorPos
-height height	height=height
-tags tagList	tags=tagList
-width width	width=width

表 A.24 画布 postcript 选项

Tk	Tkinter
-height height	height=height
-width width	width=width
-window pathname	window=pathname
-colormap varName	colormap=varName
-colormode color|grey|mono	colormode='colorTgreyTmono'
-file fileName	file=fileName
-fontmap varName	fontmap=varName
-height size	height=size
-pageanchor anchor	pageanchor=anchor
-pageheight size	pageheight=size
-pagewidth size	pagewidth=size
-pagex position	pagex=position
-pagey position	pagey=position
-rotate boolean	rotate=boolean
-width size	width=size
-x position	x=position
-y position	y=position

输入控件

表 A.25 输入控件标准选项

Tk	Tkinter
-background color	background=color
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-exportselection boolean	exportselection=boolean
-font font	font=font
-foreground color	foreground=color
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-insertborderwidth width	insertborderwidth=width
-insertofftime milliseconds	insertofftime=milliseconds
-insertontime milliseconds	insertontime=milliseconds
-insertwidth width	insertwidth=width
-justify left|center|right	justify=LEFT|CENTER|RIGHT
relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID
-selectbackground color	selectbackground=color

（续）

Tk	Tkinter
-selectborderwidth width	selectborderwidth=width
-selectforeground color	selectforeground=color
-state NORMAL|DISABLED|ACTIVE	state=NORMAL|DISABLED|ACTIVE
-takefocus focusType	takefocus=focusType
-textvariable variable	textvariable=variable
-width width	width=width
-xscrollcommand tclCommand	xscrollcommand=tclCommand

表 A.26 输入控件特有选项

Tk	Tkinter
-show char	show=char

表 A.27 输入索引

Tk	Tkinter
number	number(start,end)
anchor	ANCHOR
end	END
insert	INSERT
sel.first	SEL_FIRST
sel.last	SEL_LAST
@x-coord	"@x"

表 A.28 输入控件方法

Tk	Tkinter
entry bbox index	*entry.bbox(index)*
entry delete first [last]	entry.delete(first [,last])
entry get	entry.get()
entry icursor index	entry.icursor(index)
entry index index	entry.index(index)
entry insert index string	*entry.insert(index,string)*
entry scan mark args	entry.scan_mark(args)
entry scan dragto args	entry.scan_dragto(args)
entry selection adjust index	entry.selection_adjust(index)
entry selection clear	entry.selection_clear()
entry selection from index	entry.selection_from(index)
entry selection present	entry.selection_present()
entry selection rangs start end	entry.selection_range(start,end)
entry selection to index	entry.selection_to(index)
entry xview moveto fraction	entry.xview_moveto(fraction)
entry xview scroll number units|pages	entry.xviewscroll(number,units|pages)

列表框控件

表 A.29 列表框控件标准选项

Tk	Tkinter
-background color	background=color
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-exportselection boolean	exportselection=boolean
-font font	font=font
-foreground color	foreground=color
-height height	height=height
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID
-selectbackground color	selectbackground=color
-selectborderwidth width	selectborderwidth=width
-selectforeground color	selectforeground=color
-setgrid boolean	setgrid=boolean
-takefocus focusType	takefocus=focusType
-width width	width=width
-xscrollcommand tclCommand	xscrollcommand=pythonCommand
-yscrollcommand tclCommand	yscrollcommand=pythonCommand

表 A.30 列表框控件特有选项

Tk	Tkinter
-selectmode single|browse |multiple|extended	selectmode=SINGLE|BROWE |MULTIPLE|EXTENDED

表 A.31 列表框索引

Tk	Tkinter
number	number or (start,end)
active	ACTIVE
anchor	ANCHOR
end	END
@x,y	"@x,y"

表 A.32 列表框方法

Tk	Tkinter
listbox activate index	listbox.activate(index)
listbox bbox index	listbox.bbox(index)
listbox curselection	listbox.curselection()

（续）

Tk	Tkinter
listbox delete index1 [index2]	listbox.delete(index1 [,index2])
listbox get index [index2]	listbox.get(index1 [,index2])
listbox index index	listbox.index(index)
listbox insert index [element...]	listbox.insert(index [,element...])
listbox nearest y	listbox.nearest(y)
listbox scan mark args	listbox.scan_mark(args)
listbox scan dragto args	listbox.scan_dragto(args)
listbox see index	listbox.see(index)
listbox selection anchor index	listbox.selection_anchor(index)
listbox selection clear first [last]	listbox.selection_clear(first [,last])
listbox selection includes index	listbox.selection_includes(index)
listbox selection set first [last]	listbox.selection_set(first [,last])
listbox size	listbox.size()
listbox xview index	listbox.xview(index)
listbox xview moveto fraction	listbox.xview_moveto(fraction)
listbox xview scroll number units|pages	listbox.xview_scroll(number,units|pages)
listbox yview index	listbox.yview(index)
listbox yview moveto fraction	listbox.yview_moveto(fraction)
listbox yview scroll number units|pages	listbox.yview_scroll(number,units|pages)

菜单控件

表 A.33 菜单控件标准选项

Tk	Tkinter
-activebackground color	activebackground=color
-activeborderwidth width	activeborderwidth=width
-activeforeground color	activeforeground=color
-background color	background=color
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-disabledforeground color	disabledforeground=color
-font font	font=font
-foreground color	foreground=color
-relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED| RIDGE|SUNKEN|SOLID

表 A.34 菜单控件特有选项

Tk	Tkinter
-postcommand tclCommand	postcommand=pythonCommand
-selectcolor color	selectcolor=color
-tearoff boolean	tearoff=boolean
-tearoffcommand tclCommand	tearoffcommand=pythonCommand

（续）

Tk	Tkinter
-title string	title=string
-type type	type=type

表 A.35 输入类型

Tk	Tkinter
cascade	cascade
checkbutton	checkbutton
command	command
radiobutton	radiobutton
separator	separator

表 A.36 菜单索引

Tk	Tkinter
number	number or (start,end)
active	ACTIVE
last	LAST
none	NONE
@y-coord	"@y"
matchPattern	matchPattern

表 A.37 菜单控件方法

Tk	Tkinter
menu activate index	menu.activate (index)
menu add cascade	menu.add_cascade (option=value...)
menu add checkbutton	menu.add_checkbutton (otion=value...)
menu add command	menu.add_command (option=value...)
menu add radiobutton	menu.add_radiobutton (option=value...)
menu add separator	menu.add_separator (option=value...)
menu clone newMenuName [cloneType]	
menu delete index1 [index2]	menu.delete (index1 [,index2])
menu entryget index option	menu.entryget (index,option)
menu entryconfigure index [option value...]	menu.entryconfigure (index[,option=value...])
menu index index	menu.index (index)
menu insert index type [option value...]	menu.insert.(index,type[,option=value...])
menu insert index cascade [option value...]	menu.insert_cascade (index[,option=value...])
menu insert index checkbutton [option value...]	menu.insert_checkbutton (index[,option=value...])
menu insert index command [option value...]	menu.insert_command (index [,option=value...])
menu insert index radiobutton [option value...]	menu,insert_radiobutton (index [,option=value...])
menu insert index separtator [option value...]	menu.insert_separator (index [,option=value...])
menu invoke index	menu.invoke (index)
menu post x y	menu.post (x,y)
menu postcascade index	menu.postcascade (index)
menu type index	menu.type (index)

（续）

Tk	Tkinter
menu unpost	menu.unpost ()
menu yposition index	menu.yposition (index)

表 A.38 附加菜单属性

Tk	Tkinter
-accelerator string	accelerator=string
-command tclCommand	command=tclCommand
-columnbreak value	columnbreak=value
-hidemargin value	hidemargin=value
-indicatoron boolean	indicatoron=boolean
-label string	label=string
-menu pathname	menu=pathname
-offvalue value	offvalue=value
-onvalue value	onvalue=value
-selectcolor color	selectcolor=color
-selectimage image	selectimage=image
-value value	value=value
-variable variable	variable=variable

文本控件

表 A.39 文本控件标准选项

Tk	Tkinter
-background color	background=color
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-exportselection boolean	exportselection=boolean
-font font	font=font
-foreground color	foreground=color
-height height	height=height
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-insertbackground color	insertbackground=color
-insertborderwidth width	insertborderwidth=width
-insertofftime milliseconds	insertofftime=milliseconds
-insertontime milliseconds	insertontime=milliseconds
-insertwidth width	insertwidth=width
-padx width	padx=width
-pady height	pady=height
-relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN |SOLID

（续）

Tk	Tkinter
-selectbackground color	selectbackground=color
-selectborderwidth width	selectborderwidth=width
-selectforeground color	selectforeground=color
-setgrid boolean	setgrid=boolean
-state NORMALIDISABLEDIACTIVE	state=NORMALIDISABLEDIACTIVE
-takefocus focus Type	takefocus=focus Type
-width width	width=width
-xscrollcommand tclCommand	xscrollcommand=tclCommand
-yscrollcommand tclCommand	yscrollcommand=tclCommand

表 A.40　文本控件特有选项

Tk	Tkinter
-spacing1 size	spacing1=size
-spacing2 size	spacing2=size
-spacing3 size	spacing3=size
-tabs tabList	tabs=tabList
-wrap none|char|word	wrap=NONE|CHAR|WORD

文本索引

表 A.41　基文本索引

Tk	Tkinter
line.char	line.char
@x,y	"@x,y"
end	END
mark	mark
tag	tag
pathname (embedded window)	window
imageName (embedded window)	imageName

表 A.42　文本索引调节器

Tk	Tkinter
+ count chars	'+ count chars'
- count chars	'- count chars'
+ count lines	'+ count lines'
- count lines	'- count lines'
linestart	' linestart'
lineend	' lineend'
wordstart	' wordstart'
wordend	' wordend'

表 A.43 文本标签标准选项

Tk	Tkinter
-background color	background=color
-borderwidth width	borderwidth=width
-font font	font=font
-foreground color	foreground=color
-justify left|center|right	justify=LEFT|CENTER|RIGHT
-relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID

表 A.44 文本标签特有选项

Tk	Tkinter
-bgstipple bitmap	bgstipple=bitmap
-fgstipple bitmap	fgstipple=bitmap
-lmargin1 size	lmargin1=size
-lmargin2 size	lmargin2=size
-offset size	offset=size
-overstrike boolean	overstrike=boolean
-rmargin size	rmargin=size
-tabs tabList	tabs=tabList
-underline boolean	underline=boolean

表 A.45 文本内嵌窗口选项

Tk	Tkinter
-align top|center|bottom|baseline	align=TOP|CENTER|BOTTOM|BASELINE
-create script	create=script
-padx width	padx=width
-pady height	pady=height
-stretch boolean	stretch=Boolean
-window pathName	window=pathName

表 A.46 文本内嵌图像选项

Tk	Tkinter
-align top|center|bottom|baseline	align=TOP|CENTER|BOTTOM|BASELINE
-image image	image=image
-name imageName	name=imageName
-padx width	padx=width
-pady height	pady=height

表 A.47 文本控件方法

Tk	Tkinter
text bbox index	text.bbox (index)
text compare index1 op index2	text.compare (index1 op index2)

（续）

Tk	Tkinter
text delete index1 [index2]	text.delete (index1 [,index2])
text dlineinfo index	text.dlineinfo (index)
text dump [switches] index1 [index2]	text.dump (index1 [,index2][,option=value...])
text get index1 [index2]	text.get (index1 [,index2])
text image cget index option	text.image_cget index option
text image configure index [option [value [option value...]]]	text.image_configure (index [,option [value...]]
text image create index [option value...]	text.image_create (index [,option value...])
text image names	text.image_names()
text index index	text.index index
text insert index [string [tagList string tagList...]]	text.insert (index [,string [,tagList,string,tagList...]])
text mark gravity markName [left|right]	text.mark_gravity (markName [LEFT|RIGHT])
text mark names	text.mark_names()
text mark next | previous index	No mapping
text mark set markName index	text.mark_set (markName,index)
text mark unset markName [markName...]	text.mark_unset (markName [,markName...])
text scan mark x y	text.scan_mark (x,y)
text scan dragto x y	text.scan_dragto (x,y)
text search [switches] pattern index [stopIndex]	text.search (pattern,index, [,stopIndex][,switches])
text see index	text.see (index)
text tag add tagName index1 [index2]	text.tag_add (tagName,index1 [index2])
text tag bind tagName [sequence [script]]	text.tag_bind (tagName,sequence,script [,'+'])
text tag cget tagName option	text.tag_cget (tagName,option)
text tag configure tagName [option [value [option value...]]]	text.tag_configure (tagName [,option=value...])
text tag delete tagName [tagName...]	text.tag_delete (tagName [,tagName...])
text tag lower tagName [belowThis]	text.tag_lower (tagName [,belowThis])
text tag names [index]	text.tag_names ([index])
text tag nextrange tagName index1 [index2]	text.tag_nextrange (tagName,index1 [,index2])
text tag prevrange tagName index1 [index2]	text.tag_prevrange (tagName,index1 [,index2])
text tag raise tagName [aboveThis]	text.tag_raise (tagName [,aboveThis])
text tag ranges tagName	text.tag_ranges (tagName)
text tag remove tagName index1 [index2]	text.tag_remove (tagName,index1 [,index2])
text window cget index option	text.window_cget (index,option)
text window configure index [option value...]	text.window_configure (index [,option=value...])
text window names	text.window_names()
text xview	text.xview()
text xview moveto fraction	text.xview_moveto (fraction)
text xview scroll number units|pages	text.xview_scroll (number,units|pages)
text yview	text.yview()
text yview moveto fraction	text.yview_moveto (fraction)
text yview scroll number units|pages	text.yview_scroll (number,units|pages)
text yview [-pickplace] index	text.yview_pickplace (index)

按钮控件

表 A.48 按钮控件标准选项

Tk	Tkinter
-activebackground color	activebackground=color
-activeforeground color	activeforeground=color
-anchor anchorPos	anchor=anchorPos
-background color	background=color
-bitmap bitmap	bitmap=bitmap
-borderwidth width	borderwidth=width
-command tclCommand	command=tclCommand
-cursor cursor	cursor=cursor
-disabledforegrund color	disabledforegrund=color
-font font	font=font
-foreground color	foreground=color
-height height	height=height
-highlightbackground color	highlightbackground=color
-highlightthickness width	highlightthickness=width
-image image	image=image
-justify leftlcenterlright	justify=LEFT|CENTER|RIGHT
-padx width	padx=width
-pady height	pady=height
-relief flatlgroovelraisedlridgelsunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID
-state normallldisabledlactive	state=normallldisabledlactive
-takefocus focusType	takefocus=focusType
-text string	text=string
-textvariable variable	textvariable=variable
-underline index	underline=index
-width width	width=width
-wraplength length	wraplength=length

表 A.49 按钮控件特殊选项

Tk	Tkinter
-default normallldisabledlactive	default=NORMAL|DISABLED|ACTIVE

表 A.50 按钮方法

Tk	Tkinter
button flash	button.flash()
button invoke	button.invoke()

复选按钮控件

表 A.51 复选按钮控件标准选项

Tk	Tkinter
-actiebackground color	actiebackground=color
-activeforeground color	activeforeground=color
-anchor anchorPos	anchor=anchorPos
-background color	background=color
-bitmap bitmap	bitmap=bitmap
-borderwidth width	borderwidth=width
-command tclCommand	command=tclCommand
-cursor cursor	cursor=cursor
-disabledforeground color	disabledforeground=color
-font font	font=font
-foreground color	foreground=color
-height height	jeight=height
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-image image	image=image
-justify leftlcenterlright	justify=LEFT|CENTER|RIGHT
-padx width	padx=width
-pady height	pady=height
-relief flatlgroovelraisedlridgelsunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID
-state normalldisabledlactive	state=normalldisabledlactive
-takefocus focusType	takefocus=focusType
-text string	text=string
-textvariable variable	textvariable=variable
-underline index	underline=index
-width width	width=width
-wraplength length	wraplength=length

表 A.52 复选按钮控件特殊选项

Tk	Tkinter
-indicatoron boolean	indicatoron=boolean
-offvalue value	offvalue=value
-onvalue value	onvalue=value
-selectcolor color	selectcolor=color
-selectimage image	selectimage=image
-variable variable	variable=variable

表 A.53 复选按钮方法

Tk	Tkinter
checkbutton deselect	checkbutton.deselect()

（续）

Tk	Tkinter
checkbutton flash	checkbutton.flash()
checkbutton invoke	checkbutton.invoke()
checkbutton select	checkbutton.select()
checkbutton toggle	checkbutton.toggle()

框架控件

表 A.54 框架控件标准选项

Tk	Tkinter
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-height height	height=height
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID
-takefocus focusType	takefocus=focusType
-width width	width=width

表 A.55 框架控件特有选项

Tk	Tkinter
-background color	background=color
-class name	class=name
-colormap colormap	colormap=colormap
-container boolean	container=boolean
-visual visual	visual=visual

标签控件

表 A.56 标签控件标准选项

Tk	Tkinter
-anchor anchorPos	anchor=anchorPos
-background color	background=color
-bitmap bitmap	bitmap=bitmap
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-font font	font=font
-foreground color	foreground=color
-height height	height=height
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color

（续）

Tk	Tkinter
-highlightthickness width	highlightthickness=width
-image image	image=image
-justify left|center|right	justify=LEFT|CENTER|RIGHT
-padx width	padx=width
-pady height	pady=height
-relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID
-takefocus focusType	takefocus=focusType
-text string	text=string
-textvariable variable	textvariable=variable
-underline index	underline=index
-width width	width=width
-wraplength length	wraplength=length

菜单按钮控件

表 A.57　菜单按钮控件标准方法

Tk	Tkinter
-activebackground color	activebackground=color
-activeforeground color	activeforeground=color
-anchor anchorPos	anchor=anchorPos
-background color	background=color
-bitmap bitmap	bitmap=bitmap
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-disabledforeground color	disabledforeground=color
-font font	font=font
-foreground color	foreground=color
-height height	height=height
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-image image	image=image
-justify left|center|right	justify=left|center|right
-padx width	padx=width
-pady height	pady=height
-relief flat|groove|raised|ridge|sunken	relief=flat|groove|raised|ridge|sunken
-state normal|disabled|active	state=normal|disabled|active
-takefocus focusType	takefocus=focusType
-text string	text=string
-textvariable variable	textvariable=variable
-underline index	underline=index
-width width	width=width
-wraplength length	wraplength=length

表 A.58 菜单按钮特殊选项

Tk	Tkinter
-direction direction	direction=direction
-indicatoron boolean	indicatoron=boolean
-menu pathName	menu=pathName

消息控件

表 A.59 消息控件标准选项

Tk	Tkinter
-anchor anchorPos	anchor=anchorPos
-background color	background=color
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-font font	font=font
-foreground color	foreground=color
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-justify left|center|right	justify=LEFT|CENTER|RIGHT
-padx width	padx=width
-pady height	pady=height
-relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID
-takefocus focusType	takefocus=focusType
-text string	text=string
-textvariable variable	textvariable=variable
-width width	width=width

表 A.60 消息控件特有选项

Tk	Tkinter
-aspect integer	aspect=integer

单选按钮控件

表 A.61 单选按钮控件标准选项

Tk	Tkinter
-activebackground color	activebackground=color
-activeforeground color	activeforeground=color
-anchor anchorPos	anchor=anchorPos
-background color	background=color
-bitmap bitmap	bitmap=bitmap
-borderwidth width	borderwidth=width
-command tclCommand	command=tclCommand

（续）

Tk	Tkinter
-cursor cursor	cursor=cursor
-disabledforeground color	disabledforeground=color
-fot font	fot=font
-foreground color	foreground=color
-height height	height=height
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-image image	image=image
-justify left|center|right	justify=LEFT|CENTER|RIGHT
-padx width	padx=width
-pady height	pady=height
-relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID
-state normal|disabled|active	state=NORMAL|DISABLED|ACTIVE
-takefocus focusType	takefocus=focusType
-text string	text=string
-textvariable variable	textvariable=variable
-underline index	underline=index
-width width	width=width
-wraplength length	wraplength=length

表 A.62　单选按钮控件标准选项

Tk	Tkinter
-indicatoron boolean	indicatoron=boolean
-selectcolor color	selectcolor=color
-selectimage image	selectimage=image
-value value	value=value
-variable variable	variable=variable

表 A.63　单选按钮控件方法

Tk	Tkinter
radiobutton deselect	radiobutton.deselect()
radiobutton flash	radiobutton.flash()
radiobutton invoke	radiobutton.invoke()
radiobutton select	radiobutton.select()

比例控件

表 A.64　比例控件标准选项

Tk	Tkinter
-activebackground color	activebackground=color
-background color	background=color

（续）

Tk	Tkinter
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-font font	font=font
-foreground color	foreground=color
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-orient horizontal|vertical	orient=HORIZONTAL|VERTICAL
-relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID
-repeatdelay milliseconds	repeatdelay=milliseconds
-repeatinterval milliseconds	repeatinterval=milliseconds
-state normal|disabled	state=normal|disabled
-takefocus focusType	takefocus=focusType
-troughcolor color	troughcolor=color

表 A.65 比例控件特有选项

Tk	Tkinter
-bigincrement number	bigincrement=number
-command tclCommand	command=tclCommand
-digits integer	digits=integer
-from number	from=number
-label string	label=string
-length size	length=size
-resolution number	resolution=number
-showvalue boolean	showvalue=boolean
-sliderlength size	sliderlength=size
-sliderrelief relief	sliderrelief=relief
-tickinterval number	tickinterval=number
-to number	to=number
-variable variable	variable=variable
-width width	width=width

表 A.66 比例控件方法

Tk	Tkinter
scale cords [value]	scale.cords ([value])
scale get [x y]	scale.get ()
scale identify x y	scale.identify (x,y)
scale set value	scale.set (value)

滚动条控件

表 A.67 滚动条控件标准选项

Tk	Tkinter
-activebackground color	activebackground=color

（续）

Tk	Tkinter
-background color	background=color
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-jump boolean	jump=boolean
-orient horizontal|vertical	orient=HORIZONTAL|VERTICAL
-relief flat|groove|raised|ridge|sunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID
-repeatdelay milliseconds	repeatdelay=milliseconds
-repeatinterval milliseconds	repeatinterval=milliseconds
-takefocus focusType	takefocus=focusType
-troughcolor color	troughcolor=color

表 A.68　滚动条控件特殊选项

Tk	Tkinter
-activerelief number	activerelief=number
-command tclCommandPrefix	command=tclCommandPrefix
-elementborderwidth width	flementborderwidth=width
-width width	width=width

表 A.69　滚动条控件选项

Tk	Tkinter
scrollbar activate [element]	scrollbar.activate (element)
scrollbar delta deltaX deltaY	scrollbar.delta (deltaX,deltaY)
scrollbar fraction x y	scrollbar.fraction (x,y)
scrollbar get	scrollbar.get()
scrollbar identify x y	scrollbar.identify (x,y)
scrollbar set first last	scrollbar.set (first,last)

Toplevel 控件

表 A.70　Toplevel 控件标准选项

Tk	Tkinter
-borderwidth width	borderwidth=width
-cursor cursor	cursor=cursor
-height height	height=height
-highlightbackground color	highlightbackground=color
-highlightcolor color	highlightcolor=color
-highlightthickness width	highlightthickness=width
-relief flat|groove|reised|ridge|sunken	relief=FLAT|GROOVE|RAISED|RIDGE|SUNKEN|SOLID
-takefocus focusType	takefocus=focusType
-width width	width=width

表 A.71 Toplevel 控件特殊选项

Tk	Tkinter
-background color	background=color
-class string	class=string
-colormap colormap	colormap=colormap
-container boolean	container=boolean
-menu pathname	menu=pathname
-use windowID	use=windowID
-screen screen	screen=screen
-visual visual	visual=visual

图像类

表 A.72 图像方法

Tk	Tkinter
image create type [name][options value...]	image=PhotoImage|BitmapImage ([option=value...])
image delete name	del (image)
image hight name	image.height ()
image names	image_names ()
image type name	image.type ()
image types	image_types ()
image width name	image_width ()

位图图像类型

表 A.73 位图选项

Tk	Tkinter
-background color	background=color
-data string	data=string
-file filename	file=filename
-foreground color	foreground=color
-maskdata string	maskdata=string
-maskfile fileName	maskfile=fileName

照片图像类型

表 A.74 照片图像选项

Tk	Tkinter
-data string	data=string
-file filename	file=filename
-format formatName	format=formatName
-height number	height=number
-alette paletteSpaec	alette=paletteSpaec
-width number	width=number

表 A.75 照片图像方法

Tk	Tkinter
image blank	Image.blank ()
image copy sourceImage [option value...]	Image.copy ()
image copy sourceImage [-zoom x y]	Image.zoom (xscale [,yscale])
image copy sourceImage [-subsample x y]	Image.subsample (xscale [,yscale])
image get x y	Image.get (x,y)
image put deta [-to x1 y1 x2 y2]	Image.put (data [, 'to' x1 y1 x2 y2]_
image read file [option value...]	No mapping
image redither	No mapping
image write filename [option value...]	Image.write(filename [,formatName][,(x1,y1,x2,y2)])

窗口信息

表 A.76 窗口信息 winfo 方法

Tk	Tkinter
winfo allmapped window	No mapping
winfo atom [-displayof window] name	window.winfo_atom (name, [,win])
winfo atomname [-displayof window] id	window.winfo_atomname (id [,win])
winfo cells window	window.winfo_cells ()
winfo children window	window.winfo_children ()
winfo class window	window.winfo_class ()
winfo_colormapfull window	window.winfo_colormapfull ()
winfo containing [-displayof window] footX rootY	window.winfo_containing (rootX,rootY [,win])
winfo depth window	window.winfo_depth ()
winfo exists window	window.winfo_exists ()
winfo fpixels window number	window.winfo_fpixels (bumber)
winfo geometry window	window.winfo_geometry ()
winfo height window	window.winfo_height ()
winfo id window	window.winfo_id ()
winfo interps [-displayof window]	window.winfo_interps ([win])
winfo ismapped window	window.winfo_ismapped ()
winfo manager window	window.winfo_manager ()
winfo name window	window.winfo_name ()
winfo parent window	window.winfo_parent ()
winfo pathname [-displayof window] id	window.winfo_pathname(id, [,win])
winfo pointerx window	window.winfo_pointerx ()
winfo pointerxy window	window.winfo_pointerxy ()
winfo pointery window	window.winfo_pointery ()
winfo pixels window number	window.winfo_pixels (number)
winfo_reqheight window	window.winfo_reqheight ()
winfo reqwidth window	window.winfo_reqwidth ()
winfo rgb window color	window.winfo_rgb (color)
winfo rootx window	window.winfo_rootx ()
winfo rooty window	window.winfo_rooty ()

（续）

Tk	Tkinter
winfo server window	windowwindow.winfo_server ()
winfo screen window	window.winfo_screen ()
winfo screencells window	window.winfo_screencells ()
winfo screendepth window	window.winfo_screendepth ()
winfo screenheight window	window.winfo_screenheight ()
winfo screenmmheight window	window.winfo_screenmmheight ()
winfo screenmmwidth window	window.winfo_screenmmwidth ()
winfo screenvisual window	window.winfo_screenvisual ()
winfo screenwidth window	window.winfo_screenwidth ()
winfo toplevel window	window.winfo_toplevel ()
winfo visual window	window.winfo_visual ()
winfo visualsavailable window	window.winfo_visualsavailable ()
winfo vrootheight window	window.winfo_vrootheight ()
winfo vrootwidth window	window.winfo_vrootwidth ()
winfo vrootx window	window.winfo_vrootx ()
winfo vrooty window	window.winfo_vrooty ()
winfo width window	window.winfo_width ()
winfo x window	window.winfo_x ()
winfo y window	window.winfo_y ()

窗口管理器

表 A.77 wm 选项

Tk	Tkinter
wm aspect window [minNumer minDenom maxNumer maxDenom]	window.wm_aspect ([minNumer,minDenom, maxNumer,maxDenom])
wm client window [name]	window.wm_client([name])
wm colormapwindows window [windowList]	window.wm_colormapwindows ([windowList])
wm command window [value]	window.wm_command ([value])
wm deiconify window	window.wm_deiconify ()
wm focusmodel window [activelpassive]	window.wm_focusmodel (['active'|'passive'])
wm frame window	window.wm_frame(
wm geometry window [newGeometry]	window wm_geometry ([newGeometry])
wm grid window [baseWidth baseHeight widthInc heightInc]	window.wm_grid ([baseWidth,baseHeight, widthInc,heightInc])
wm group window [pathName]	window.wm_group ([pathname])
wm iconbitmap window [bitmap]	widow.wm_iconbitmap ([bitmap])
Wm iconify window	window.wm_iconify ()
Wm iconmask window [bitmap]	window.wm_iconmask ([bitmap])
wm iconify window [newName]	window.wm_iconname ([newName])
wm iconposition window [x y]	window.wm_iconposition ([x,y])
wm iconwindow window [pathname]	window.wm_iconwindow ([pathName])
wm maxsize window [width height]	window.wm_maxsize ([width,height])

（续）

Tk	Tkinter
wm minsize window [width height]	window.wm_minsize ([width,height])
wm overrideredirect window [boolean]	window.wm_overrideredirect ([boolean])
wm positionfrom window [program|user]	window.wm_positionfrom (['program'|'user'])
wm protocol window [name][command]	window.wm_protocol ([name],command]]
wm resizable window [widthBoolean heightBoolean]	window.wm_resizable ([widthBoolean,heightBoolean])
wm sizefrom window [program|user]	window.wm_sizefrom (['program'|'user'])
wm state window	window.wm_state ()
wm title window [string]	window.wm_title ([string])
wm transient window [master]	window.wm_transient ([master])
wm withdraw window	window.wm_withdraw ()

捆绑与虚拟事件

表 A.78　捆绑与虚拟事件方法

Tk	Tkinter
bind tag	widget.bind ()
bind tag sequence	widget.bind ('sequence')
bind tag sequence script	widget.bind ('sequence,script')
bindtags window [tagList]	window.bindtags ([tagList])
event add <<virtual>> sequence [sequence...]	window.event_add (virtual,sequence [,sequence...])
event delete <<virtual>>[sequence...]	window.event_delete (virtual [,sequence...])
event generate window event [-when when][option value...]	window.event_generate (sequence [,option=value...])

几何管理器

包（pack）命令

表 A.79　pack 方法

Tk	Tkinter
pack [configure] slave [slave...][options]	slave.pack ([option=value...])
pack forget slave [slave...]	slave.pack_forget ()
pack info slave	slave.pack_info ()
pack propagate master [boolean]	master.pack_propagete ([boolean])
pack slaves master	master.pack_slaves ()

位置（place）命令

表 A.80　place 选项

Tk	Tkinter
-anchor anchor	anchor=anchor
-bordermode inside|outside|ignore	bordermode=inside|outside|ignore

（续）

Tk	Tkinter
-height size	height=size
-in master	in=master
-relheight size	relheight=size
-relwidth size	relwidth=size
-relx location	relx=location
-rely location	rely=location
-width size	width=size
-x location	x = location
-y location	y = location

表 A.81 位置方法

Tk	Tkinter
place [configure] window option value [option value...]	window.place ([option=value..])
place forget window	window.place_forget ()
place info window	window.place_info ()
place slaves window	window.place_slaves ()

网格命令

表 A.82 网格选项

Tk	Tkinter
-column n	column=n
-columnspan n	columnspan=n
-in other	in=other
-ipadx amount	ipadx=amount
-ipady amount	ipady=amount
-padx amount	padx=amount
-pady amount	pady=amount
-row n	row=n
-rowspan n	rowspan=n
-sticky [n][s][e][w]	sticky=[N][S][E][W]

表 A.83 网格方法

Tk	Tkinter
grid [configure] slave [slave...][option value...]	slave.grid ([option=value...])
grid boox master [column row (column2 row2]]	master.grid_bbox ([column,row [,column2,row2]])
grid columnconfigure master columnList [options]	master.grid_columnconfigure (columnList [,options])
-minsize size	minsize=size
-pad amount	pad=amount
-weight int	weight=int
grid forget slave [slave...]	slave.grid_forget ()
grid info slave	slave.grid_info ()

（续）

Tk	Tkinter
grid location master x y	slave.grid_location (x,y)
grid propagate master [boolean]	master.grid_propagate ([boolean])
grid remove slave [slave...]	slave.grid_remove ()
grid rowconfigure master rowList [options]	master.grid_rowconfigure (rowList,[,options])
grid size master	master.grid_size ()
grid slaves master [-row row][-column column]	master.grid_slaves([row][,column])

字体

表 A.84 字体选项

Tk	Tkinter
-family name	family=name
-size size	size=size
-weight weight	weight=weight
-slant slant	slant=slant
-underline boolean	underline=boolean
-overstrike boolean	overstrike=boglean

表 A.85 字体方法

Tk	Tkinter
font actual fontDesc [-displayof window][option]	fontDesc.actual [option])
font configure fontname [option [value option value...]]	fontname.configure ([option=value...])
font create [fontname [option value...]]	font=Font ([master][,option=value...])
font delete fontname [fontname ..]	del (fontname)
font families [-displayof window]	famililies ([window])
font measure fontDesc [-displayof window] text	fontDesc.measure (text)
font metrics fontDesc [-displayof window][metric]	fontDesc.metrics ([metric])
font names	name ([window])

其他 Tk 命令

表 A.86 其他 Tk 命令方法

Tk	Tkinter
bell [-displayof window]	window.bell ([displayof=window])
clipboard clear [-displayof window]	window.clipbord_clear ([displayof=window])
clipboard append [-displayof win][-format fmt] [-type type] data	window.clipbord_clear (data [,option=value...])
destroy [window window...]	window.destroy ()
focus [-force] window	window.focus_force ()
focus [-displayof window]	window.focus_displayof ()
focus –lastfor window	window.focus_lastfor ()

（续）

Tk	Tkinter
grab current [window]	window.grab_current ()
grab release window	window.grab_release ()
grab set window	window.grab_set ()
grab set –global window	window.grab_set_global ()
grab status window	window.grab_status ()
lower window [belowThis]	window.lower ([belowThis])
option add pattern value [priority]	window.option_add (pattern,value,[,priority])
option clear	window option clear ()
option get window name class	window.option_get (name,class)
option readfile fileName [priority]	window.option_readfile (fileName [,priority])
raise window [aboveThis]	window.raise ([aboveThis])
selection clear [-displayof window]	window.selection_clear ([displayof=window]
[-selection selection]	[,selection=sel])
selection get [-displayof widow]	window.selection_get ([displayof=window]
[-selection selection][-type type]	[,-selection=sel][,type=type]
selection handle [-selection sel][-type type]	Window.selection_handle (cmd [,selection=sel]
[-format fmt] win cmd	[,type=type][,format=fmt]
selection own [-displayof window]	window.selection_own ([displayof=window]
[-selection selection]	[,selection=sel])
selection own [-selection selection]	window.selection_own_get ([selection=sel]
[-command command] window	[command=command])
send [-displayof window][-async] interp cmd [arg arg...]	window.send (interp,cmd [,arg...])
tk appname [newName]	No mapping
tk scaling [-displayof window][floatNumber]	No mapping
tkwait variable varName	window.wait_variable ([window])
tkwait visibility window	window.wait_variabvisibility ([window])
tkwait window window	window.wait_window ([window])
tk_bisque	window.tk_bisque ()
tk_chooseColor [option value...]	Use tkColorChooser
tk_dialog topw title text bitmap default string [string...]	Use Dialog
tk_focusNext window	window.tk_focusNext ()
tk_focusPrev window	window.tk_focusPrev ()
tk_focusFollowsMouse	window.tk_focusFollowsMouse ()
tk_getOpenFile [option value...]	No mapping
tk_getSaveFile [option value...]	No mapping
tk_messageBox [option value...]	box=MessageBox (master,text=text [,option...])
tk_optionMenu w varName value [value...]	menu=OptionMenu (master,varName,value [,value...])
tk_popup menu x y [entry]	menu.tk_popup (x,y [,entry])
tk_setPalette color	window.tk_setPalette (color)
tk_setPalette name color [name color..]	window.tk_setPalette (name=color [,name=color...])

附录 B　Tkinter 参考资料

关于此附录

附录 B 里的信息已经大量用使用 Tkinter 模块字典和 Tk 用户手册的 Python 程序生成。已经被制作成为 Python 用法的大词典。其产生的包含头、文本和表格的 ACSII 码文件，被引入编为此书。一些信息需要手工调整，但是大量的信息只需要在 FrameMaker 格式化就行了。脚本的开发时间不需很长。

你可以找很多 Tcl 和 Tk 参考资料。我在正文中适当的地方都给出了。值得记住的是 Tkinter 最终只是一个进入它们的界面，只有 Tcl/Tk 才决定了提供的参数是否有效和合适。

共享选项

很多控件接受与其他控件相同的选项。尽管在绝对值上可能有所差异，不过它们如此相近，所以在这里可以归为一组。通常，很多 Tk 的描述都是从用户手册中得来的，因为 Tkinter 提供了一个简单的到需要考虑选项的 Tk 控件的界面。Tkinter 提供了一个面向对象的到 Tk 的捆绑，一些描述要根据 Tkinter 上下文做相当的修改。

大部分控件共享选项

选项（别名）	说　明	单位	典型值	除此所有控件
background(bg)	当控件显示时给出正常颜色	color	'gray25' '#ff4400'	
borderwidth(bd)	设置一个非负值，该值显示画控件时的控件外围 3-D 边界的宽度；特别地由 relief 选项决定这项决定）。控件内部 3-D 效果也可以使用该值，该值可以使 Tkinter(Tk_GetPixels)接受的任何格式	pixel	3	
cursor	指定控件使用的鼠标光标，该值可以使(Tk_GetPixels)接受的任何格式	cursor	gumby	
font	指定控件内部文本的字体	font	'Helvetica' ('Verdana',8)	Canvas Frame Scrollbar Toplevel

（续）

选项（别名）	说　明	单位	典型值	除此所有控件
foreground (fg)	指定控件的前景色	color	'black' '#FF2244'	Canvas Frame Scrollbar Toplevel
highlightbackground	指出经过没有输入焦点的控件加亮区域颜色	color	'gray30'	Menu
highlightcolor	指出经过有输入焦点的控件周围长方区域加亮颜色	color	'royalblue'	Menu
highlightthickness	指出一个非负值，该值指出一个有输入焦点的控件周围加亮方形区域的宽度，该值可以具有 Tkinter(Tk_GetCursor)接受的任何格式。如果为 0，则不画加亮区域	pixel	2,1m	Menu
relief	指出控件 3D 效果。可选值为 RAISED, SUNKEN FLAT, RIDGE, SOLID 和 GROOVE。该值指出控件内部相对外部的外观式样，RAISED 意味着控件内部相对于外部突出	constant	RAISED GROOVE	
takefocus	决定窗口在键盘遍历时是否接收焦点（比如 Tab, Shift- Tab）。在设定焦点到一个窗口之前，遍历脚本检查 takefocus 选项的值，值 0 意味着键盘遍历时完全跳过，值 1 意味着只要有输入焦点（它及所有父代都映射过）就接收。空值由脚本自己决定是否接收，当前的算法是如果窗口被禁止，或者是没有键盘捆绑或窗口不可见时，跳过	boolean	1 YES	
width	指定一个整数，设置控件宽度，控件字体的平均字符数。如果值小于等于 0，控件选择一个能够容纳目前字符的宽度。	integer	32	Menu

很多控件共享的选项

选　项	说　明	单位	典型值	仅此类控件
activebackground	指定画活动元素的背景颜色。元素（控件或控件的一部分）在鼠标放在上并按动鼠标按钮引起某些行为的发生时，是活动的。如果严格的 Motif 一致性请求通过设置 tk_strictMotif 变量完成，该选项将被忽略，正常背景色将被使用。对 Windows 和 Macintosh 系统，活动颜色将只有在鼠标按钮 1 被按过元素时使用	color	'red' '#fa07a3'	Button Checkbutton Menu Menubutton Radiobutton Scale Scrollbar

（续）

选　项	说　明	单位	典型值	仅此类控件
activeforeground	指定画活动元素时的前景颜色。参见上面关于活动元素的定义	color	'cadetblue'	Button Menu Checkbutton Menubutton Radiobutton
anchor	指出控件信息（比如文本或位图）如何在控件中显示。必须为下面值之一：N，NE，E，SE，S，SW，W，NW 或 CENTER。比如 NW 指显示信息时使左上角在控件的左上端	constant		Button Checkbutton Label Message Menubutton Radiobutton
bitmap	指定一个位图在控件中显示，以 Tkinter (Tk_GetBitmap)接受的任何形式。位图显示的精确方式受其他选项如锚或对齐的影响。 典型地，如果该项被指定，它覆盖指定显示控件中文本的其他选项；bitmap 选项可以重设为空串以使文本能够被显示在控件上。在同时支持位图和图像的控件中，图像通常覆盖位图	bitmap		Button Checkbutton Label Menubutton Radiobutton
command	指定一个与控件关联的命令。该命令通常在鼠标离开控件之时被调用，对于单选按钮和多选按钮，tkinter 变量（通过变量选项设置）将在命令调用时更新	command	setupData	Button Checkbutton Radiobutton Scale Scrollbar
disabledforeground	指定绘画元素时的前景色。如果选项为空串（单色显示器通常这样设置的），禁止的元素用通常的前景色画，但是采用点刻法填充模糊化	color	'gray50'	Button Checkbutton Radiobutton Menu Menubutton
height	指定窗口的高度，采用字体选项中给定字体的字符高度为单位，至少为 1	integer	1　4	Button Canvas Frame　label Listbox Checkbutton Radiobutton Menubutton Text Toplevel

（续）

选 项	说 明	单位	典型值	仅此类控件
image	指定在控件中显示的图像，必须是用图像 create 方法产生的。如果图像选项设定，它覆盖已经设置的位图或文本显示；重新恢复位图或文本的显示需要设置图像选项为空串	image		Button Checkbutton Label Menubutton Radiobutton
justify	当控件中显示多行文本的时候，该选项设置不同行之间是如何排列的。其值为如下之一：LEFT，CENTER 或 RIGHT。LEFT 指每行向左对齐，CENTER 指每行居中对齐，RIGHT 指向右对齐	constant	RIGHT	Button Checkbutton Entry Label Menubutton Message Radiobutton
padx	指定一个非负值设置控件在 X 方向需要的边距。该值需为 Tkinter(Tk_GetPixels)接受的格式。当计算需要多大窗口的时候，控件会把此值加到正常大小之上（由控件中显示内容决定）；如果几何管理器能满足此请求，控件将在左端和/或右端得到一个给定的多余空边。大部分控件只用此项于文本，如果它们显示位图或图像，通常忽略空边选项	pixel	2m 10	Button Checkbutton Label Menubutton Message Radiobutton Text
pady	指定一个非负值设置控件在 Y 方向需要的边距。该值需为 Tkinter(Tk_GetPixels)接受的格式。当计算需要多大窗口的时候，控件会把此值加到正常大小之上（由控件中显示内容决定）；如果几何管理器能满足此请求，控件将在上面和/或下面得到一个给定的多余空边。大部分控件只用此项于文本，如果它们显示位图或图像，通常忽略空边选项	pixel	12 3m	Button Checkbutton Label Menubutton Message Radiobutton Text
selectbackground	指定显示选中项时的背景颜色	color	blue	Canvas Listbox Entry Text
selectborderwidth	指定一个非负值，给出选中项的三维边界宽度，值可以是任何 Tkinter(Tk_GetPixels)接受的格式	pixel	3	Canvas Entry Listbox Text

（续）

选　项	说　明	单位	典型值	仅此类控件
selectforeground	指定显示选中项时的前景颜色	color	yellow	Canvas Entry Listbox Text
state	指定控件下列两三个状态之一（典型是复选按钮）：NORMAL 和 DISABLED 或 NORMAL、ACTIVE 和 NORMAL。在 NORMAL 状态，控件有前景和背景显示；在 ACTIVE 状态，控件按 activeforeground 和 activeback ground 选项显示；DISABLED 状态下，控件不敏感，缺省捆绑将拒绝激活控件，并忽略鼠标行为，此时，由 disabled foreground 和 background 选项决定如何显示	constant	ACTIVE	Button Checkbutton Entry Menubutton Scale Radiobutton Text
text	指定控件中显示的文本，文本显示格式由特定控件和其他诸如锚和对齐选项决定	string	'Display This'	Button Checkbutton Label Menubutton Message Radiobutton
textvariable	指定一个变量名字。变量值被转变为字符串在控件上显示。如果变量值改变，控件将自动更新以反映新值，字符串显示格式由特定控件和其他诸如锚和对齐选项决定	variable	widgetContent	Button Checkbutton Entry Label Menubutton Message Radiobutton
underline	指定控件中加下划线字符的整数索引。此选项完成菜单按钮与菜单输入的键盘遍历缺省捆绑。0 对应控件中显示的第一个字符，1 对应第二个，依次类推	integer	2	Button Checkbutton Label Menubutton Radiobutton
wraplength	对于能够支持字符换行的控件，该选项指定行的最大字符数，超过最大字符数的行将转到下行显示，这样一行不会超过最大字符数。该值可以是窗口距离的任何标准格式。如果该值小于或等于 0，不换行，换行只有在文本中的换行符的地方才出现	pixel	4i,65	Button Checkbutton Label Menubutton Radiobutton

（续）

选 项	说 明	单位	典型值	仅此类控件
xscrollcommand	指定一个用来与水平滚动框进行信息交流的命令前缀，当控件窗口视图改变（或者别的任何滚动条显示的改变，如控件体的总尺寸的改变等），控件将通过把滚动命令和两个数连接起来产生一个命令。两个数分别为 0 到 1 之间的分数，代表文档中的一个位置，0 表示文档的开头，1.0 表示文档的结尾处，0.333 表示整个文档的三分之一处，如此等等。第一个分数代表窗口中第一个可见文档信息。第二个分数代表紧跟上一个可见部分之后的信息。然后命令把它传递到 Tcl 解释器执行。 典型地，xscrollcommand 选项由滚动条标识跟着 set 组成，如 self.x.scrollbar set 将引起滚动条在窗口中视图变化时被更新。如果此项没有指定，不执行命令	function		Canvas Entry Listbox Text
yscrollcommand	指定一个用来与垂直滚动框进行信息交流的命令前缀，当控件窗口视图改变（或者别的任何能够改变滚动条显示的，如控件体的总尺寸的改变等），控件将通过把滚动命令和两个数，连接起来产生一个命令。两个数分别为 0 到 1 之间的分数，代表文档中的一个位置，0 表示文档的开头，1.0 表示文档的结尾处，0.333 表示整个文档的三分之一处，如此等等。第一个分数代表窗口中第一个可见文档信息。第二个分数代表紧跟上一个可见部分之后的信息。命令然后把它传递到 Tcl 解释器执行。 典型地，yscrollcommand 选项由滚动条标识跟着 set 组成，如 self.y.scrollbar set 将引起滚动条在窗口中视图变化时被更新。如果此项没有指定，不执行命令	Function		Canvas Entry Listbox Text

继承方法

很多方法是从基类继承过来的，可以被所有的控件使用。除了这里列出的方法外，grid、pack 和 place 几何管理器方法也被所有的控件继承。这些方法将与控件独立开给出。

这些方法的参数是按照 Tkinter 所定义的方式给出的。在这里你将看到一个从 Tk 到 Tkinter 的映射，特别地，Tk 命令把窗口作为第一个参数。Tkinter 方法被用到当前控件

实例中，这些实例被解释为 tk 命令的窗口参数或附/上参数。

通用控件方法

after(ms,function=None,*args)

注册一个在 *ms* 毫秒之后被调用的回调。注意，这段时间并不保证精确，你可以假定等待时间至少是给定的时间，而它本身可能要长得多。这个方法返回 id 值，可以用来作为 after_cancel 的参数来取消回调。

after_cancel(id)

取消指定的回调。

after_idle(function,*args)

注册一个回调，使其在系统空闲（没有更多的事件在事件队列中）的时候被调用。每次调用 after_idle 就会调用回调函数一次。

bell(displayof=0)

在窗口的显示中鸣铃，不返回值。如果 displayof 选项忽略掉，应用的主窗口的显示为缺省的。此方法使用当前 bell 相关的设置来显示，可以通过诸如 xset 的程序来修改。该方法还能重新设置屏保，有的屏保将忽略此项，但有的会使屏幕重新恢复可见。

bind(sequence=None,function=None,add=None)

把事件句柄和事件关联起来。当 add 为+时，绑定被加到当前绑定中，缺省时替代当前绑定。

bind_all(sequence=None, function=None,add=None)

把该应用层的事件和时间句柄关联起来。当 add 为+时，绑定被加到当前绑定中，缺省时替代当前绑定。

bind_class(ClassName,sequence=None, function=None,add=None)

为指定的控件类把事件句柄和事件关联起来。当 add 为+时，绑定被加到当前绑定中，缺省时替代当前绑定。

bindtags(tagList=None)

当 bind 标签没有参数时，以元组形式返回当前的捆绑标签。如果为 binding 标签指定 tagList 参数，必须是正确的元组，窗口的标签改为 tagList 给定的标签。tagList 的元素可以完全是字符串，然而，任何一个以点开始的串都认为是一个窗口的名字。如果事件处理时不存在这样名字的窗口，标签被忽略。tagList 中元素的顺序决定响应事件的捆绑脚本执行顺序。

cget(key)

返回给定 key 值的配置选项的当前值。

clipboard_append(string)

在窗口显示的剪切板添加字符串。

clipboard_clear()

获得窗口显示剪贴板的所有权，并除去以前的内容。

configure(option=None)

查询或更改控件的配置。选项未指定时，返回一个描述该控件能用的所有选项的字典。当指定一个或多个选项值时，该方法修改控件选项使其值变为给定的值，这时无返回值。

destroy()

析构控件并从名字空间除去所有索引。

event_add(virtual,*sequences)

依据系列参数把虚拟事件 virtual 与物理事件 sequences 联系起来，以便只要一个系列存在，虚拟事件就可以触发。virtual 可以是任何字符串，sequences 则可以是任何可以连接方法的参数。如果 virtual 已经定义了，新的物理事件序列添加到已经存在的序列。

event_delete(virtual,*sequences)

删除通过 virtual 与虚拟事件联系在一起的每一个序列。virtual 可以是任何字符串，sequences 则可以是任何可以连接方法的参数。任何当前没有与 virtual 联系的序列都被忽略。如果没有序列参数，所有的物理事件序列都将从 virtual 中除去，以便虚拟事件不再被触发。

event_generate(sequence,option=value…)

产生一个窗口事件，使其如同是从 window 系统产生的。序列提供事件的一个基本描述，比如<shift_button-2>,sequence 可以是能够把方法联系起来的任何序列形式，只是它必须包含一种单一事件模式，而不是一个序列。Option 值对可以用来指定事件的额外属性，比如鼠标位置的 x,y 坐标。

event_info(virtual=None)

返回 virtual 信息。如果 virtual 参数没有，返回值是当前定义的所有虚拟事件的一个元组。当 virtual 值给定，返回值的元组值是 virtual 所定义的物理事件序列，如果 virtual 没有定义，则返回空值。

focus_displayof()

返回包含控件的显示中的聚焦窗口名。如果聚焦窗口不在应用中，返回空值。

focus_force()

把控件显示焦点设定到自身，即使应用当前没有控件的输入焦点。本方法应当保守地用。通常使用中，一个应用不该为自己认定焦点，应当让窗口管理为其指定。

focus_get()

如果应用在控件显示中有输入焦点，该方法返回有焦点窗口的标识。

focus_lastfor()

返回与 self 具有相同顶层的所有窗口中有输入焦点的最近窗口标识。如果在那个顶层没有窗口有输入焦点，或者如果最近焦点窗口被删除了，那么这个顶层窗口的 ID 被返回。返回值是当下次窗口管理器发送焦点到顶层时接收焦点的窗口。

focus_set()

当应用当前在控件显示中有输入焦点时，该方法重置控件显示的输入焦点到 self。如果应用当前没有控件显示有输入焦点，self 将被作为顶层焦点，下次焦点到达顶层时，Tk 将把它重定向到 self。

getboolean(string)

使用 Tcl 协定把字符串变为布尔量。

getvar(name='PY_VAR')

返回变量名。

grab_current()

为窗口显示返回应用的当前抓取窗口；或空值，如果不存在这样的窗口。

grab_release()

如果存在，释放 self 中的抓取，不然，不做任何事。

grab_set()

设置当前应用的所有事件抓取为 self。如果控件显示中该应用已经有抓取生效，它自动释放；如果 self 已经有抓取，该方法不做任何事。

grab_set_global()

设置整个屏幕中所有事件抓取为 self。如果控件显示中该应用已经有抓取生效，它自动释放；如果 self 已经有抓取，该方法不做任何事。使用抓取时务必小心。

grab_status()

如果窗口中没有设置抓取，返回空值；如果窗口中设置的是局部抓取，返回 local；如果窗口中设置的是全局抓取，返回 global。

image_names()

返回包含所有存在图像名的列表。

image_types()

返回包含创建的所有图像属性的列表。

keys()

返回一个元组，元组值为控件可用选项的名字。通过 self.cget 得到每个选项的当前值。

lower(belowThis=None)

改变控件在栈中的位置。如果 belowThis 参数略去，该方法把窗口位置在栈中降低到其同属窗口的最低点（它会被覆盖它的同属隐藏，但不隐藏它的同属窗口）。当 belowThis 被指定时，它必须是同属窗口或其子窗口的标识。这种情况下，below 方法会把窗口插入到栈中 belowThis（或是同属窗口 belowThis 的祖先）下面；这可能导致下降或上升了窗口位置。

mainloop
开始事件循环。调用之前不做任何更新，调用 quit 时才返回值。

nametowidget(name)
返回与控件名相对应的标识。

option_add(pattern,value,priority=None)
允许在 Tk 选项库中加入值。Pattern 包含指定的选项，由星号或点的名字和/或类组成，按照通常的 X 格式。value 为与 pattern 相关的文本串，此值是调用 Tkinter(Tk_GetOption)或者 option_get 方法的返回值。当 priority 指定时，它指此选项的优先级，默认值是 interactive。

option_clear()
清除 Tk 选项数据库。下次选项加入或除去时，缺省选项将自动被载入（从 RESOURE_MANAGER 属性或.Xdefaults 文件）。

option_get(name,className)
返回在 name 和 class 条件下的 self 选项指定的值。如果选项库中的输入有多个和 name 与 class 匹配，则该方法返回具有最高优先级的值。如果匹配者最高优先级的有几项，则返回最近进入选项库者。如果没有匹配者，返回空字串。

option_readfile(fileName,prority=None)
读文件名，它必须是具有如同.Xdefaults 式样的 X 资源数据库的标准格式。它把文件里指定的所有选项都加到选项库。当 priority 指定时，它指的是进入选项的优先级；缺省值为 interactive。

quit()
退出主循环。

selection_clear()
清楚当前控件中的选择。如果选择不在当前控件中，方法无效。

selection_get()
从窗体显示中获取选择（selection）值并将其返回。

selection_handle(handler)
为选择请求创造句柄，这样当窗口拥有该选择且该选择按给定的格式（如 type 在 selection_get 方法中指定）获取时，handler 就会被执行。seletion 默认为 PRIMARY,type

默认为 STRING，format 默认为 STRING。如果 handler 为空则任何已经存在的窗口、类型和选择的 handler 都将被删去。

selection_own()

使 self 成为窗口显示中选项新的所有者，同时返回空串。先前所有者（如果有的话），将会被通知失去选项。

selection_own_get()

返回应用程序中窗口的标识，在该窗口中拥有包含 self 的显示的选择，或者返回空串，如果应用中没有窗口包含选择。selection 缺省值为 PRIMARY，窗口缺省为根窗口。

send(interp,cmd,*args)

安排 cmd(带上参数)在名为 interp 的应用中执行。返回命令执行的结果或错误。interp 可是任何应用名，只要该应用的主窗口在包含发送者主窗口的显示中；它不需要在同一个过程中。如果没有 args 参数出现，那么要执行的命令完全在 cmd 参数之中。如果有一个或多个参数，则像 eval 命令一样，把它们串接起来构成命令执行。

setvar(name='PY_VAR',value='1')

设定变量 name 到提供的值。

tk_bisque()

提供向后兼容：它应用的颜色恢复到 Tk3.6 或以前版本的浅棕色(bisque)。

tk_focusFollwsMouse()

构造一个隐式焦点模式：它重新配置 Tk 使得每当有鼠标进入窗体时便聚焦。

tk_focusNext()，tk_focusPrev()

tk_focusNext 和 tk_focusPrev 完成了顶层窗口焦点顺序的处理。它们在缺省的绑定方式用于 Tab,Shift Tab。

tk-menuBar（*args）

不做任何事，因为该 tk 函数已经过时了。

tk_setPallete(*args)

改变 tk 颜色配置。通过已存控件颜色和改变选项库，这样以后的控件就可以使用该颜色配置了。如果 tk_setpallete 采用单个参数调用，则参数为用来做正常背景色的一个颜色名。Tk_pallete 从背景色中计算出一个调色板颜色。

同时，tk_setpalette 可以包含任何多个“名字—值”对，前一个参数是 tk 选项库的选项名，后一个是该选项将使用的新值。下面的选项库是目前支持的：

- activeBackground
- activeForeground
- background

- disabledForeground
- foreground
- highlightBackground
- highlightColor
- insertBackground
- selectcolor
- selectBackground
- selectForeground
- troughColor

tk_strictMotif(boolean=None)

boolean 缺省值为 0。如果应用程序把它设为 TRUE，则 Tk 试图尽可能接近地继承 Motif look-and-feel 标准。比如，按钮、滚动条等活动元素在鼠标移过的时候不会改变颜色。

tkraise(aboveThis=None)[lift(aboveThis=None)]

当没有 aboveThis 参数的时候，该方法把 self 移到栈中同属的最高位置（它不会被任何同属所覆盖，相反却覆盖了它们）。如果 aboveThis 被指定，则它必须是窗口的同属或同属的子窗口。这时该方法把 self 插到栈中且恰好在 aboveThis 之上（或者 aboveThis 同属窗口的父窗口），出现在窗口之上或者窗口之下。

unbind(sequence,funcid=None)

把所给 sequence 的绑定除去。如果事件句柄 funcid 一块被绑定到 sequence 上，句柄本身也将被除去。

unbind_all(sequence)

除去该应用层所有系列的全部绑定。

unbind_class(className,sequence)

为指定的类 className 提供的 sequence 除去所有绑定。

update()

处理事件列表中的所有到来事件。特别地，如果需要的话，完成所有几何安排和控件重画。需小心使用，因为它可能是问题存在的原因。不仅因为耗用 CPU 周期，还同时产生了竞争条件。

update_idletasks()

处理事件队列里的所有无响应事件。

wait_variable(name='PY_VAR')

等待提供的 Tkinter 变量、名字值的改变。注意，本方法进入一个局部事件循环，直到变量改变，这样应用的主循环可以继续。

wait_visibility(window=None)

等待特定的窗口变为可见。注意，本方法进入一个局部事件循环，直到变量改变，这样应用的主循环可以继续。

wait_window(window=None)

等待特定的窗口被析构。注意，本方法进入一个局部事件循环，直到变量改变，这样应用的主循环可以继续。

Winfo 方法

winfo_atom(name,dislayof=0)

返回一个整数，它给出名为 name 的原子的整数标识符，如果不存在名字为 name 的原子，则新建一个名为 name 的原子。如果 displayof 选项给定，则原子被锁定到窗口显示中，否则被锁定在应用的主窗口中。

winfo_atomname(id,displayof=0)

返回数字标识为 id 的原子的文本名字。如果 displayof 给定，则标识被锁定到窗口显示中，否则锁定到应用主窗口的显示中。此方法是 winfo_atom 的逆方法，如果不存在这样的原子，则产生一个错误。

winfo_cell()

返回一个窗口颜色映像中单元的数目。

winfo_children()

返回一个列表，该列表内容为所有子窗口的路径名。该列表按照栈的次序，底层窗口在先，顶层窗口作为它们的逻辑父窗口返回。

winfo_class()

返回一个窗口的类名。

winfo_colormapfull()

如果已知窗口颜色映像满了则返回 TRUE，否则 FALSE。窗口颜色映像被认为满，如果分配一个新颜色到该窗口的最后一次尝试失败，且因分配失败应用没有释放颜色映像中的任何颜色的话。

winfo_containing(根 X,根 Y,displayof=0)

返回包含点（根 X，根 Y）的窗口标识，根 X 和根 Y 是按根窗口的屏幕坐标单位设定的（如果使用了虚拟根窗口管理器，则采用虚拟根窗口管理器的坐标体系）。当 displayof 选项给定的时候，则坐标表示包含窗口的部分，否则指应用的根窗口屏幕。如果应用中没有窗口包含指定的点则返回 None。在选择包容窗口时，子代比父代有更高的优先级，且同属的栈中最高位置将被选择。

Winfo_depth()

返回一个窗口深度的数字（每像素的比特个数）。

winfo_exists()

如果 self 窗口存在，返回 TRUE，否则，返回 FALSE。

winfo_fpixels(number)

返回一个浮点数，该浮点数给出了距离为 number 的窗口中的像素的数目。number 可以用 tkinter 接受的任何形式（Tk_GetScreenMM）比如 2.0c 或 1i。返回值可能是是分数的，要得到整值，可用 winfo_pixels。

winfo_geometry()

返回一个窗口的几何属性，按照如下形式：widthxheight+x+y，所有均为像素。

winfo_height()

返回表示窗口高度像素值的整数。当一个窗口创建的时候，高度为 1 像素，高度最终将被几何管理器调整到满足要求。如果你需要在窗口创建之后立即得到高度，调用 update 强制几何管理器来处理，或者使用 winfo_reqheight 来得到窗口的被请求的高度而不是实际高度。

winfo_id()

返回一个整数，该整数为特定平台的底层窗口的标识

winfo_inerps(displayof=0)

返回一个列表，该列表成员为目前在特定显示中注册的所有 Tcl 解释程序的名字（所有基于 Tk 的应用）。如果 displayof 选项给定，则返回值为窗口显示，否则为主窗口显示。这对 Tk 应用有一定的用处。

winfo_ismapped()

如果 self 已映射返回 TRUE，不然返回 FALSE。

winfo_manager()

返回当前负责 self 窗口的几何管理器名字；或者是空串，如果没有几何管理器管理这个窗口的话。名字通常是几何管理器的 Tcl 方法名，比如说 pack 和 place。如果几何管理器是个控件，比如画布或文本，那么名字为控件类，即画布。

winfo_name()

返回窗口名（在其父窗口中的名字，而不是完整的路径名）。

winfo_parent()

返回窗口父窗口的名字；或者是空串，如果窗口本身即是应用程序的主窗口。

winfo_pathname(id,displayof=0)

返回标识为 id 的窗口名。id 必须是十进制、十六进制或八进制的，必须与调用应用中的窗口相对应。如果 displayof 选项给定，则标识到显示窗口中查找。否则在应用主窗口中查找。

winfo_pixels(number)

返回一个数，该数给出了距离为 number 的窗口中的像素的数目。number 可以用 Tkinter 接受的任何形式（Tk_GetScreenMM）比如 2.0c 或 1i。结果是近似到最近的整数值，要得到分数值，使用 winfo_fpixels。

winfo_pointerx()

如果鼠标指针和窗口在同一屏面里，返回鼠标的 x 坐标，尺度为根窗口的像素。如果采用了虚拟根窗口，则使用虚拟窗口的像素。如果鼠标和窗口不在同一个屏面里，返回-1。

winfo_pointerxy()

如果鼠标指针和窗口在同一屏面里，返回一个包含鼠标的 x 坐标和 y 坐标的元组，坐标尺度为根窗口的像素。如果采用了虚拟根窗口，则使用虚拟窗口的像素。如果鼠标和窗口不在同一个屏面里，元组的两个值都为-1。

winfo_pointery()

如果鼠标指针和窗口在同一屏面里，返回鼠标的 y 坐标，尺度为根窗口的像素。如果采用了虚拟根窗口，则使用虚拟窗口的像素。如果鼠标和窗口不在同一个屏面里，返回-1。

winfo_reqheight()

返回窗口必须的高度值，单位为像素。这是几何管理器用来计算几何位置用的值。

winfo_reqwidth()

返回窗口必须的宽度值，单位为像素。这是几何管理器用来计算几何位置用的值。

winfo_rgb(color)

返回一个元组，元组包含三个十进制数，分别为给定窗口中窗口颜色的红绿蓝深度。颜色可以按任何可以接受的颜色格式。

winfo_root x()

返回一个整数，该整数为屏面中根窗口左上角边界的 x 坐标（如果没有边界，则为窗口的）。

winfo_root y()

返回一个整数，该整数为屏面中根窗口左上角边界的 y 坐标（如果没有边界，则为窗口的）。

winfo_screen()

返回与窗口相关的屏面名字，按照 displayName 和 screenIndex 的格式。

winfo_screencells()

返回一个整数，该整数为窗口内缺省颜色映像的单元个数。

winfo_screendepth()
返回一个整数，该整数为根窗口中屏幕的颜色深度（以每像素多少位为单位）。

winfo_screenheight()
返回一个整数，该整数为窗口屏幕的高度（以像素为单位）。

winfo_screenmmheight()
返回一个整数，该整数为窗口屏幕的高度（以毫米为单位）。

winfo_screenmmwidth()
返回一个整数，该整数为窗口屏幕的宽度（以毫米为单位）。

winfo_screenvisual()
返回下列字符串之一，表示窗口屏幕的缺省类型：directolor, grayscale, pseudocolor, staticcolor, staticgray 或 truecolor。

winfo_screenwidth()
返回窗口屏幕的大小（以像素为单位）。

winfo_server()
返回一个字符串，该串包含关于窗口显示服务器的信息。该串的精确格式可能在不同平台间有差别。对于 X 服务器，字符串格式为 xmajorRminor vendor vendorVersion，这里 major 和 minor 是服务器提供的版本和修订号（例如 X11R5），vendor 是服务器销售商名，vendorRelease 是服务器提供的一个整数版本号。

winfo_toplevel()
返回包含该窗口的高层窗口标识。

winfo_viewable()
返回 TRUE，如果窗口和它所继承的所有窗口到最近的顶层窗口已映射。返回 FALSE，如果这中的某个窗口未被映射。

winfo_visual()
返回如下的一个表示窗口视类的字符串:directcolor, grayscale, pseudocolor, staticcolor, staticgray 或 truecolor。

winfo_visualid()
返回窗口视件的 X 标识。

winfo_visualsavailable(inludeids=0)
返回一个列表，列表的值是窗口屏幕可能的视值。每个元素包含一个视类跟上一个整数深度。类的格式和 winfo_visual 返回的格式相同。深度给出了视中每个像素的位数。另外，如果 includeids 参数给定，则深度由 visual 的 x 标识跟随。

winfo_v root height()

返回与窗口相关联的虚拟根窗口的高，如果存在的话,不然，返回窗口屏面高。

winfo_vroot width()
返回与窗口相关联的虚拟根窗口的宽，如果存在的话,不然，返回窗口屏面宽。

winfo_vroot x()
返回与窗口相关联虚拟根窗口相对于根窗口屏幕的 x 偏移量。要么是 0，要么是负数。如果没有虚拟根窗口返回 0

winfo_vroot y()
返回与窗口相关联虚拟根窗口相对于根窗口屏幕的 y 偏移量。要么是 0，要么是负数。如果没有虚拟根窗口返回 0。

winfo_width()
返回一个整数，给出窗口宽度的像素值。当一个窗口刚刚创建的时候，宽度为 1，宽度将由几何管理器在窗口需要的时候调整。如果你需要窗口宽度在创建的时候马上为真实宽度，调用 update 可以强制几何管理器来处理，或者使用 winfo_reqwidth 来得到窗口的必须宽度而不是实际宽度。

winfo_x()
返回一个整数，该整数给出窗口父窗口边界左上角 x 坐标（如果没有边界则为窗口本身）。

winfo_y()
返回一个整数，该整数给出窗口父窗口边界左上角 y 坐标（如果没有边界则为窗口本身）。

wm 方法

说明

wm 方法用于与窗口管理器交互，以控制诸如窗口标题、其几何位置或者尺寸增减。Tkinter 在根窗口(Tk)或所有顶层控件级别使用这些方法。wm 方法可以使用几种不同方法中的任何一种，具体根据所用参数而定。所有的格式都需要至少一个另外的参数，window，这必须是顶层窗口的路径。

Tkinter 定义了 wm 方法的同义词，虽然你可以自由使用 wm_prefix。wm 方法的格式如下。

aspect(*minNumer=None,minDemon=None,maxNumer=None,maxDenom=None*)
如果 MinNumer,minDemon,maxNumer 和 maxDnom 所有参数给定，参数将被传递给窗口管理器，窗口管理器用来强制一个可以接受的窗口高宽比例范围。窗口的高宽比将限在 minNumer/minDenom 和 maxNumer/maxDenom 之间。

如果 minNumer 等并没有全部给定，则没有任何的比例限制。如果 minNumer 等给定，返回 None,否则返回一个元组，该元组有四个值，为当前的 minNumer, minDenom, maxNumer 和 maxDenom（如果没有比例限制，返回 None）。

client(name=None)

当 name 指定时，该方法把 name（name 应该是应用执行的主机的名字）存储到窗口的 WM_CLIENT_MACHINE 属性，以便窗口管理器或进程管理器使用。如果名字 name 没有给定，方法返回上次设置到 wm_client 中的名字。如果给定的 name 为空串，方法从窗口中删除 WM_CLIENT_MACHINE 属性，该方法只有 X 系统使用。

Colormapwindows(*windowList)

用来操作 WM_COLORMAP_WINDOWS 属性，该属性为具有私有颜色映像的窗口提供信息到窗口管理器。如果 windowList 没有给定，该方法返回一个列表，列表元素值为 WM_COLORMAP_WINDOWS 属性中的窗口名字。如果 windowList 给定，它包含一个窗口路径名的列表。该方法用给定的窗口去覆盖 WM_COLORMAP_WINDOWS 属性，返回 None。此方法仅为 X 系统使用。

WM_COLORMAP_WINDOWS 属性通常应该包含一个内部窗口列表，这些窗口颜色映像有别于其父窗口。属性中窗口的顺序反应了它们的优先级：窗口管理器将在获得颜色映像焦点之后从列表头部开始安装尽可能多的颜色映像。如果窗口不包含在 windowList 的窗口中，Tk 隐式地把它加到 WM_COLORMAP_WINDOWS 属性的尾部，这样它的颜色映像拥有最低的优先级。如果 wm_colormapwindows 没有调用，Tk 将为每个顶层窗口自动设置属性为颜色映像与父窗口不同的内部窗口，紧跟着是顶层窗口本身；内部窗口的顺序没有指定。关于 WM_COLORMAP_WINDOWS 属性更多的信息可以参看 ICCCM 文档。

command(callback=None)

给按钮指定一个回调，该回调通常在鼠标离开按钮的时候调用。此方法仅为 X 系统所用。

deiconify()

使窗口按照正常形式显示（非图标形式）。这通过窗口映射来完成，如果此窗口还没有映射过，则该方法不会去映射它，但是它能保证在窗口第一次被映射的时候以非图标的形式显示，返回 None。

focusmode(model=None)

如果 active 或 passive 作为 model 的选项，它给出窗口的聚焦模式。这种情况下，方法返回空串。如果没有别的参数，则方法返回窗口当前聚焦模式。active 聚焦模式窗口会为自己或子窗口获取输入焦点，甚至当焦点正在别的应用程序中。passive 意味着窗口从不为自己主动获得焦点，窗口管理器将在适当的时候把焦点传给窗口。然而，一旦焦点送到窗口或其子窗口，应用程序将在子窗口中重新分配焦点。聚焦模式缺省为 passive。此方法仅为 X 系统所用。

frame()

如果窗口被窗口管理器重设为父窗口，成为修饰框架，方法返回一个基于平台的窗口标识给包含窗口（父代是根或虚拟根的窗口）的最外框架。如果窗口没有被窗口管理器所重设为父窗口，则方法返回一个基于平台的窗口标识。此方法仅为 X 系统所用。

geometry(newGeometry=None)

当 NewGeometry 给定时，窗口的 geometry 被改变，返回空串。不然，返回当前窗口的 geometry（最近手工调整的位置或通过调用 wm_geometry 产生的位置）。NewGeometry 的格式是=widthxheight+-x+-y，其中=,widthxheight 和+-x+-y 中的任何一项都是可以省略的。width 和 height 是正整数，给出窗口的大小。如果窗口是网格状的，则以网格为单位，否则以像素为单位。x 和 y 给出在屏幕上的位置，以像素为单位。如果 x 前为+，它给出了屏幕左端到窗口左边界之间的距离。如果为-，则给出了屏幕右端离窗口右边界的距离。如果 y 前为+，它给出了屏幕顶端到窗口上边界之间的距离。如果为一，则给出了屏幕底端离窗口下边界的距离。如果 newgeometry 设为空串，则任何用户指定的窗口 geometry 将被取消，而回到了内部控件给定的尺寸。

group(pathName=None)

如果 pathName 给定，它为一组相关窗口 leader 的路径名。窗口管理器利用这些信息来 unmap 一组 leader 已经图标化的窗口。pathName 可以是空串，这样能清除一组关联的窗口。如果 pathName 给定，返回空串，否则返回当前组 leader 的路径名或者空串，如果当前窗口不属于任何组的话。

iconbitmap(bitmap=None)

如果 bitmap 给定，则它给了一个 Tkinter(Tk_GetBitmap)接受的标准格式位图。此位图传送给窗口管理器，显示为窗口图标，返回空串。如果 bitmap 未给定，窗口当前的任何图标位图将被取消。如果 bitmap 给定，返回空串，否则返回当前窗口图标位图的名字，或者空串，如果当前窗口没有图标位图的话。

wm_iconify()

安排窗口图标化。如果窗口没有经过映射，该方法使其在映射之后图标化。

iconmask(bitmap=None)

如果 bitmap 指定，则它给了一个 Tkinter(Tk_GetBitmap)接受的标准格式位图。该 bitmap 被传递给窗口管理器用来掩码图标位图选项：这里掩码中 0 的地方不显示图标，1 的地方显示。如果 bitmap 没有给定，则任何当前图标掩码将取消（这等于给了一个全为 1 的 bitmap。如果 bitmap 给定，返回空串，否则返回与当前窗口相关联的位图掩码名字，或者空串，如果没有掩码的话。

iconname(newName=None)

如果 newName 给定，则被传送到窗口管理器，窗口管理器将把 newName 显示在与窗口相关的图标中。这种情况下，将返回一个空串做为结果。如果没有给出 newName，方法返回窗口当前图标名字，或者空串，如果窗口没有指定图标的话（这种情况下，窗

口管理器会正常显示窗口的标题，即 wm_title 所指定的）。

iconposition(x=None,y=None)

如果 x 和 y 给定，它们将传给窗口管理器决定在什么位置放置窗口图标，此时返回空串。如果 x 和 y 为空串，则存在的图标位置将被取消。如果 x 和 y 都没有给定，方法返回一个有两个元素的元组，给出当前图标位置（如没有，则返回空值）。

iconwindow(pathName=None)

如果 pathName 给定，此为窗口用做图标的路径名。当窗口图标化之后，pathname 被映射为图标，相反则 pathName 将被取消映射。如果 pathName 为空串，窗口已存在的图标将被取消。

当 pathName 给定时，返回空值，否则返回当前窗口图标的路径名，或空串，如果窗口当前没有指定图标的话。图标窗口使按钮事件无效，这个目的是为了让窗口管理器“拥有”事件。

注：并非所有的窗口管理器都支持图标窗口记号。

maxsize(width=None,height=None)

如果 width 和 height 给定，它们给出了窗口的最大可能尺寸。对网格窗口，采用网格单位，不然使用像素单位。窗口管理器会调整窗口的尺寸小于或等于这个尺度。如果 width 和 height 给定，方法返回 None，不然，返回一个元组，元组的值为当前最大 width 和 height。窗口的缺省最大尺寸为屏幕大小。如果大小功能使用了 wm_resizable 使其失效，则该方法不起作用。具体可参看本附录 B 中的几何管理中的 Grid 部分、Pack 部分以及 Place 部分。

minsize(width=None,height=None)

如果 width 和 height 给定，它们给出了窗口的最小可能尺寸。对网格窗口，采用网格单位，不然使用像素单位。窗口管理器会调整窗口的尺寸大于或等于这个尺度。如果 width 和 height 给定，方法返回 None，不然，返回一个元组，元组的值为当前最小 width 和 height。窗口的缺省最小尺寸为 1 像素。参看附录 B 中的几何管理中的 Grid 部分、Pack 部分以及 Place 部分。

overrideredirect(boolean=None)

如果 boolean 给定，它必须是合格的布尔量，override_redirect 标志就设置为此值。如果 boolean 值没有给定，则返回 TRUE 或 FALSE，给出窗口当前的 override 标志。

设置窗口 override 标志使窗口为窗口管理器所忽略，其他情况下，这意味着窗口不会被父化从根窗口进入修饰框架，用户也无法使用常规窗口管理器机制来对窗口进行操作。

positionfrom(who=None)

如果 who 给定，它必须是 program 或 user，或者二者之一的简称。它给出当前窗口是应用程序还是用户的请求。 很多窗口管理器忽略程序请求的初始状态，要求用户人为定位窗口。如果 user 设定，那么窗口管理器会按照给定的位置定位窗口而不是向用户请求。

如果 who 设定为空串，则当前定位资源被取消；如果 who 设定，则返回空串，否则返回 user 或窗口指定窗口当前位置源，或者空串，如果还没有设定源的话。大部分窗口管理器把 no source 解释为 program。当调用 wm_geometry 时，Tk 会自动设定位置源给 user，除非源被显式地设为 program。

protocol(name=None,function=None)

该方法用来管理窗口管理器协议，如 WM_DELETE_WINDOW。name 是与窗口管理器协议相对应的原子。比如 WM_DELETE_WINDOW 或 WM_SAVE_YOURSELF 或 WM_TAKE_FOCUS。如果 name 和 function 两者都给定，function 与 name 指定的协议有关。name 将被加到窗口管理器 WM_PROTOCOLS 属性来告诉窗口管理器应用程序有 name 的协议句柄。而 function 则将在窗口管理器发送消息给该协议的客户的时候被调用。这种情况下，返回一个空串。

如果 name 给定而 function 没有，则返回 name 的当前 function，或者返回空串，如果没有为 name 定义句柄。如果 function 设为空串，则当前 name 的句柄被删除，从窗口的 WM_PROTOCOLS 中移走,返回空串。

最后，如果 name 和 function 都没有给定，方法返回一个列表，列表值为当前定义句柄的所有协议。Tk 总是为 WM_DELETE_WINDOW 定义一个协议句柄，即使你没有请求一个 wm 协议。如果 WM_DELETE_WINDOW 消息到来而你还没有为之定义句柄，则 Tk 通过让接收消息的窗口销毁消息的方式进行处理。

resizable(width=None,height=None)

该方法控制用户是否能够交互式控制顶层窗口的大小。如果 width 和 height 给定，它们都是布尔值，决定用户是否能够更改宽高尺寸。此时返回空串。如果没有 width 和 height，返回一个含 TRUE/FALSE 的列表，说明当前窗口宽高能否改变。缺省情况下为宽高都是可变的。如果重定大小被禁止，窗口大小为最近交互式调整的大小或者是 wm_geometry 调用的大小。如果都没有这些操作，则为窗口的自然大小。

sizefrom(who=None)

如果 who 给定，它必须是 program 或 user，或者二者之一的简称。它给出当前窗口尺寸是 program 还是 user 的请求。 很多窗口管理器忽略程序请求尺寸，要求用户人为设定窗口调整。如果 user 设定，那么窗口管理器会按照给定的尺寸而不是向用户请求。

如果 who 设定为空串，则当前尺寸资源被取消。如果 who 设定，则返回空串，否则返回 user 或窗口指定的窗口当前尺寸源，或者空串，如果还没有设定当前源的话。大部分窗口管理器把没有源解释为 program。

state()

给按钮指定三个状态之一：NORMAL，ACTIVE，DISABLED。在 NORMAL 状态下，按钮使用前景和背景选项显示，在 ACTIVE 状态通常是在鼠标点击到按钮的时候。在 ACTIVE 状态下，按钮按激活前景和激活背景显示的，DISABLED 状态说明按钮不再激活，缺省 bindings 会拒绝激活控件，将忽略鼠标的按下。这种状态下 disbableforeground 和 background 选项决定按钮如何显示。

title(string=None)

如果 string 给定，则传送到窗口管理器用做窗口标题（窗口管理器把它显示在标题栏）。如果 string 没有给定，方法返回当前窗口标题，窗口缺省标题为名字。

transient(master=None)

如果 master 给定，窗口管理器将被告知窗口是代表 master（这里 master 为顶层窗口标识）的临时窗口（比如下拉菜单），窗口管理器将用这些信息来管理窗口。如果 master 设置为空串，则窗口被设为不再是临时的。如果 master 设定，方法返回空值，不然方法返回当前 master 窗口的路径名或者空串，如果不存在这样的窗口的话。

withdraw()

安排窗口从屏幕上撤消，这样使窗口失去映射，被窗口管理器忘记。如果窗口已经被映射，则方法使其成为撤消状态下的映射；不是所有的窗口管理器都知道处理撤消状态下映射的窗口。

注：有时候有必要撤消窗口，重新映射（如 wm_deiconify）以使某些窗口管理器注意到窗口属性诸如 group 等的变化。

Bitmap 类

说明

位图是那些像素可以使用二色或透明显示的图片。位图图像是通过前景色、背景色及称为源和掩码的两个位图定义的。其中每个位图是一个用 0/1 量来指示的方阵，两个位图维数必须相等。掩码是 0 的像素什么都不显示，产生透明效果；对其他像素，如果源数据是 1 则显示前景颜色，如果源数据是 0 则显示背景颜色。

继承关系

从 Image 继承。

共享选项

选项	缺省值
background	None
foreground	"black"

位图选项

选项（别名）	说 明	单位	典型值
data	指定源位图内容为一个串。该串必须遵守 X11 位图格式（比如位图程序产生的）。如果 data 和 file 选项都指定，data 选项优先	string	
file	filename 给出定义源位图的文件。该文件必须遵守 X11 位图格式（比如位图程序产生的）	string	'icon.xbm'

（续）

选项（别名）	说 明	单位	典型值
maskdata	指定位图掩码内容为一个串。该串必须遵守 X11 位图格式（比如位图程序产生的）。如果 maskdata 和 maskfile 选项都指定，data 选项优先	string	
maskfile	filename 给出定义位图掩码的文件。该文件必须遵守 X11 位图格式（比如位图程序产生的）	string	"imask.xbm"

方法

Bitmap(option…)
使用 option 中的选项值对产生位图实例。

cget(option)
返回 option 给定的当前配置选项值。option 可以是位图构造器认可的任何值。

configure(option=value…)
查询或更改位图配置项。如果没有指定 option，返回一个描述 imageName 所有可用选项的字典。如果指定 option 没有值，命令返回一个描述名为 option 的选项的字典（该字典与没有指定 option 返回值的相应子表相同）。

height()
返回一个给出图像高度的像素值。

type()
以字符串形式返回图像的类型（当图像被创建的时候类型值也创建）。

width()
返回一个给出图像宽度的像素值。

Button（按钮）



说明

按钮类定义了一个新窗口和一个按钮控件。另外的选项，如下面描述的那样，可以在方法调用或选项数据库中设置按钮的各个方面，如颜色、字体、文本、初始凸凹度等。按钮方法返回一个新控件标识。当方法调用的时候，父按钮必须存在。

按钮可以显示文本串、位图或图像。如果显示的是文本，必须是单一字体，但可以在多行显示（如果有新行或因为 wrapLength 选项而出现换行），且其中一个字母可以通过使用 underline 选项而带下划线。它可以使用三种方式显示：凸起、下凹和平面；还可以做成闪烁。当用户调用 button(通过按鼠标左键于按钮上)，这时候激活命令选项指定的回调函数。

继承关系

按钮从控件继承过来。

共享选项

选项	缺省值
activebackground	SystemButtonFace
activeforeground	SystemButtonText
anchor	center
background	SystemButtonFace
bitmap	
borderwidth(bd)	2
command	
cursor	
disabledforeground	SystemDisabledText
font	(('MS', 'Sans', 'Serif'), '8')
foreground(fg)	SystemButtonText
height	0
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	1
image	
justify	center
padx	1
pady	1
relief	raised
state	normal
takefocus	
text	
textvariable	
underline	-1
width	0
wraplength	0

按钮特有选项

选项（别名）	说　明	单位	典型	缺省
default	指定按钮的三个状态之一：NORMAL、ACTIVE 或 DISABLED。在 ACTIVE 状态，按钮按照特定平台的形式画为缺省式样。在 NORMAL 状态，按钮按照特定平台的形式画为缺省式样，留有足够的空间画缺省按钮外观。NORMAL 状态下的按钮会和 ACTIVE 状态下的按钮大小相等。在 DISABLED 状态，按钮按照特定平台的形式画为缺省式样，不留空间画缺省按钮外观。DISABLED 状态下的按钮会小于 ACTIVE 状态下的按钮	constant	NORMAL "disabled"	disabled

方法

flash()

使按钮闪烁。这伴随着重复显示按钮多次，不断在 active 和 normal 颜色下改变，在闪烁结束，按钮颜色和闪烁前颜色相同（active/normal）。如果按钮状态为禁止，此函数被忽略。

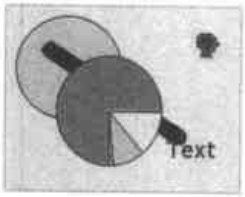
invoke（）

如果存在按钮，则调用与按钮相关的回调。返回值是回调返回值，或者是空串，如果回调不存在的话。如果按钮状态为禁止，此函数被忽略。

```
tkButtonDown(*ignored)
tkButtonEnter(*ignored)
tkButtonInvoke(*ignored)
tkButtonLeave(*ignored)
tkButtonUp(*ignored)
```

这些函数只有在写自己的事件控制按钮的时候才用。它们的功能是置按钮外观如缺省动作的那样。它们可以用来模拟用户的 GUI 交互程序。

Canvas 画布



说明

画布类定义了一个新窗口，产生一个画布控件实例。如下所述选项，可以在方法调用中或通过修改选项数据库来配置画布的各个方面，如颜色、3D 属性等。canvas 方法返回一个新控件标识，在 canvas 方法调用的时刻，其父画布必须存在。

画布控件实现了结构性图形。画布可以显示任何数量的项目，它们可以是方形、三角形、圆、直线、文字等。项目可以操作（比如移动或重上色），回调可以与项目关联，就如 bind 方法允许回调绑定到控件一样。例如，一个特定回调可以与<Button-1>事件关联，以便光标在一个项目上时按下 Button-1 激活回调。这意味着一个画布内的项目可以具有绑定到其上的 Tkinter 函数定义的性能。

继承关系

Canvas 继承自 Widget。

共享选项

选项	缺省值
background(bg)	SystemButtonFace
borderwidth(bd)	0
Cursor	
Height	7c

（续）

选项	缺省值
highlightback-ground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	2
relief	flat
selectbackground	SystemHighlight
selectborderwidth	1
selectforeground	SystemHighlightText
takefocus	
width	10c
xscrollcommand	

画布特有选项

选项（别名）	说　明	单位	典型	缺省值
closeenough	给出一个浮点值，指示鼠标要离一个项多远的距离才算鼠标在此项内，缺省为 1.0	float	0.5	1
confine	给出一个布尔值，指示是否允许在滚动区域参数定义的区域外设置画布。缺省为 TRUE，这意味着画布视图被限制在滚动区域内	Boolean	FLASE 1	1
insertbackground	插入光标位置的背景颜色，该颜色将覆盖控件的正常背景颜色（或选中部分背景颜色，如果插入光标碰巧落到选中部位上）	color	'yellow'	System ButonText
insertborderwidth	给定一个非负值，指示插入光标周围的 3D 的宽度，值可以是 Tkinter(Tk_GetPixels)认可的任何格式	pixel	2	0
insertofftime	给定一个非负整数，指示插入光标在每次闪烁之间离开的毫秒数，如果为 0，光标不闪烁，一直存在	integer	250	300
insertontime	给定一个非负整数，指示插入光标在每次闪烁之间保持的毫秒数	integer	175	600
Insertwidth	给定一个值，指示插入光标的总宽度。值可以是任何 Tkinter(Tk_GetPixels)认可的任何格式。如果插入光标已经有边界（通过 insertborderwidth 选项），边界将在 insertwidth 选项给定的宽度内	pixel	2	2
scrollregion	指定一个列表，列表元素为四个坐标，描述一个长方形的左、顶、右和底四个边。该区域用做滚动作用，是画布中信息的边界。每一个坐标都可以用下部分 COORDINATES 所说的格式	list	(10,10,200,250)	

（续）

选项（别名）	说　明	单位	典型	缺省值
xscrollincrement	指定一个水平滚动增量，为通常屏幕 distance 所允许的任何格式。如果该选项的值大于 0，窗口的水平视图将被限制成让画布在窗口左端的 x 坐标总是 xscrollincrement 的偶数倍，而且滚动(当滚动条的左右箭头选中时)的单位为 xscrollincrement。如果选项参数小于或等于 0，则水平滚动不被限制	distance	10m 200	0
yscrollcommand	指定一个与垂直滚动条通信的前缀，该选项与 xscrollcommand 的方法相同，除了它用在垂直滚动条且由支持垂直滚动条的控件提供，选项详细情况参见 xscrollcommand 说明	function		
yscrollincrement	指定一个垂直滚动增量，为通常屏幕 distance 所允许的任何格式。如果该选项的值大于 0，窗口的水平视图将被限制成让画布在窗口左端的 y 坐标总是 yscrollincrement 的偶数倍，而且，滚动(当滚动条的顶和底箭头选中时)的单位为 yscrollincrement。如果选项参数小于或等于 0，则水平滚动不被限制	distance	10m 200	0

方法

add tag_above(newtag, tagOrId)

在给定的 tagOrId 之上加入 newtag 到显示列表中。如果 tagOrId 代表多于一项，则使用这列表中的最顶层。

add tag_all(newtag)

把 newtag 加入到画布所有项之上。

add tag_below(newtag,tagOrId)

在给定的 tagOrId 之下加入 newtag 到显示列表中。如果 tagOrId 代表多于一项，则使用这列表中的最底层。

add tag_closest(newtag,x,y,halo=None,start=None)

在离给定的 x,y 最近处加上 newtag。如果在同样近距离有多于一个的项（意味着两个项在同一点重合），则使用这些项中最顶层者（显示列表中最后一个）。如果 halo 给定，它必须是非负值，任何到该点比 halo 近的项都将覆盖它。start 参数可以作为步长在所有的近邻项中循环地使用。如果 start 给定，它给出一个 tag 或 id 项（如果是 tag，选择显示列表中第一个）。它会选择列表中 start 之下的最顶层项而不是所有中的最顶层者，如果不存在这样的项，则按照没有给定 start 一样执行。

add tag_enclosed(newtag,x1,y1,x2,y2)

添加 newtag 到完全以 x1,y1,x2,y2 包围的矩阵内所有项。x1 必须不大于 x2，y1 不大于 y2。

add tag_overlapping(newtag,x1,y1,x2,y2)

添加 newtag 到完全以 x1,y1,x2,y2 包围或覆盖的矩阵内所有项。x1 必须不大于 x2，y1 不大于 y2。

add tag_withtag(newtag,tagOrId)

给 tagOrId 所有项添加标签。

bbox(tagOrId,bbox)

返回一个四元素元组，给出 tagOrId 参数附近的所有项的近似矩阵边界。元组按照 x1,y1,x2,y2 顺序排列，这样所画的元素将全部被以 x1 为左，x2 为右，y1 为上，y2 为下的区域包围。返回值会比实际值要高估通常几个像素。如果没有项匹配 tagOrId 参数，或者匹配的项边界框为空（没有可显示的东西），返回空串。

canvasx(screenx,gridspacing=None)

在画布 screenx 给出窗口的 x 坐标，此方法返回显示在这里的画布 x 坐标。如果 gridspacing 给定，坐标被截为 gridspacing 倍数最接近的一个。

canvasy(screenx,gridspacing=None)

在画布 screenx 给出窗口的 y 坐标，此方法返回显示在这里的画布 y 坐标。如果 gredspacing 给定，坐标被截为 gridspacing 倍数最接近的一个。

coords(tagOrId,x0,y0,x1,y1,…,xn,yn)

查询或更改一个项的坐标。如果没有指定坐标，方法返回一个元组，该元组值为项 tagOrId 的坐标；如果坐标给定，则替代 tagOrId 的坐标。如果 tagOrId 指多个项，则显示列表中第一个被使用。

下列方法产生一个画布项 item，但是作为单独的“控件”，这样它们自己的属性和行为可以更充分地解说，尽管事实上它们不是独立的控件。

create_arc(*)

参见后面的“Canvas Arc（画布弧形）”。

create_bitmap(*)

参见后面的“Canvas Arc（画布弧形）”。

create_image(*)

参见后面的“Canvas Arc（画布弧形）”。

create_line(*)

参见后面的“Canvas line（画布直线）”。

create_oval(*)

参见后面的“Canvas Arc（画布弧形）”。

create_polygon(*)
参见后面的“Canvas Arc（画布弧形）”。

create_rectangle(*)
参见后面的“Canvas Arc（画布弧形）”。

create_text(*)
参见后面的“Canvas Arc（画布弧形）”。

create_window(*)
参见后面的“Canvas Arc（画布弧形）”。

dchars(tagOrId,first=0,last=first)
tagOrId 给定的任何一个项，删除从 first 到 last 所界定的所有字母。如果 tagOrId 给的某个项不支持文本操作，则被忽略，first 和 last 是上面 INDICES 里描述的字符索引。如果没有 last，缺省为 first，该方法返回 None。

delete(tagOrId)
删除 tagOrId 给定的所有项，返回空串。

dtag(tagOrId,tagToDelete)
对 tagOrId 给定每个项，删除与这些项相关的由标签 Todelete 给出的标签，如果某项没有标签 Todelete 指出的标签，方法不影响这些项，如果标签 ToDelete 忽略，缺省为标签 OrId。该方法返回 None。

find_above(tagOrId)
找出显示列表中出正好在 tagOrId 之上的项，如果 tagOrId 代表不止一项，则最后（顶）一项被使用。

find_all()
返回一个画布中所有项的标识的列表。

find_below(tagOrId)
找显示列表中出正好在 tagOrId 之下的项，如果 tagOrId 代表不止一项，则最前（底）一项被使用。

find_closest(x,y,halo=None,start=None)
返回离 x,y 点最近的项。如果在同样近距离有多于一个的项（意味着两个项在同一点重合），则使用这些项中最顶层者（显示列表中最后一个）。如果 halo 给定，它必须是非负值，任何到该点比 halo 近的项都将覆盖它。start 参数可以作为步长在所有的近邻项中循环地使用。如果 start 给定，它给出一个标签或 id 项（如果是标签，选择显示列表中第一个）。它会选择列表中 start 之下的最顶层项而不是所有中的最顶层者，如果不存在

这样的项，则按照没有给定 start 一样执行。如果画布中有一个或多个项，该方法总是返回一项。

find_enclosed(x1,y1,x2,y2)

返回完全被 x1,y1,x2,y2 包围的矩阵内所有项。x1 必须不大于 x2，y1 不大于 y2。

find_overlapping(x1,y1,x2,y2)

返回完全被 x1,y1,x2,y2 包围或覆盖的矩阵内所有项。x1 必须不大于 x2，y1 不大于 y2。

find_with tag(tagOrId)

返回一个 tagOrId 给定项的标识列表。

focus(tagOrId)

设置画布控件的键盘焦点到 tagOrId，如果 tagOrId 代表多个项，则焦点定到显示列表中第一个支持插入光标的项，如果 tagOrId 不代表任何项，或者没有一项支持插入光标，则焦点不变。如果 tagOrId 是空串，则是焦点重设，所有项焦点都清除。如果 tagOrId 没有指定，方法返回当前焦点所在项的标识，或者是空串，如果焦点不在项上的话。

一旦焦点到了某项上面，该项就显示插入光标，所有的键盘事件都将定向到该项。画布内焦点项和屏幕中的焦点窗口（如 focus 所设置那样）是完全独立的。一个给定的项不具有焦点，除非它的画布是聚焦窗口并且焦点在该项上，建议使用控件聚焦方法来设置窗口焦点到画布上（如果不在的话）。

gettag(tagOrId)

返回一个列表，列表值为与 tagOrId 相关项的标签。如果 tagOrId 代表多于一个的项，返回显示列表中第一个项相关的 tag；如果 tagOrId 不代表任何项，或者所代表的项不具有任何 tag，返回空串。

icursor(tagOrId,index)

设置插入光标到 tagOrId 给出的项的正好在字符 index 之前的地方，如果某些或所有 tagOrId 给定的项不支持插入光标，则该方法对之无效。

注：插入光标只在当前有键盘焦点的项中（控件聚焦方法，参见下面），但是光标的位置可以在没有焦点的控件中设置。返回 None。

index(tagOrId,index)

返回一个整数，给出 tagOrId 与 index 相应的数值索引。Index 给出了期望位置的文本描述（如结尾）。返回的值保证在 0 和项中字符数之间，如果 tagOrId 代表多个项，则 index 使用第一个支持索引操作的项（在显示列表中）。

insert(tagOrId,beforeThis string)

对 tagOrId 给的每个项，如果该项支持文本插入，字符串 string 被插入到正好在索引为 beforethis 的字符之前。返回 None。

itemcget(tagOrId,option)

返回 tagOrId 给定项的名为 option 配置选项的当前值。这与 cget 控件方法有些相似，只是它是用于控件某些项而不是整个控件。Option 可以有任何被创建项时的控件创建方法所接受的值，如果 tagOrId 代表多于一个的项，第一个被使用。

itemconfigure(tagOrId,options)

该方法与 configure 控件方法相似，只是它更改 tagOrId 给定的项的与项相关 options，而不是更改整个画布控件的。如果 option 没有给出，返回一个描述 tagOrId 给出的所有项能够使用的 option 字典。如果 option 没有值，命令返回一个描述给定选项的字典（该字典与没有给定 option 返回的相应的子列表相同）。如果给了一个或多个选项值对，则方法更改控件选项，是 tagOrId 给定的项具有给出的值（值对）；这种情况下，方法返回 None。Option 和值和项创建时的控件创建方法允许的相同、合法的格式请看下面描述项类型的部分，那里有更详细的信息。

move(tagOrId,xAmount,yAmount)

把 tagOrId 给定的每个项在画布坐标系中通过增加 xAmount 和 yAmount 而使相应的 x 坐标和 y 坐标变到新的点。返回 None。

postscript(options)

为部分或全部画布产生 Postscript 表示。如果文件选项给定，则 Postscript 写一个文件，返回空串；否则 Postscript 作为方法的结果被返回。如果拥有画布的解释器被标为安全（safe），则操作失败，因为安全解释器不允许写文件。如果管道 option 给定，参数即已经打开的为写的管道选项名。Postscript 写到该管道，管道保持打开，为了在操作结束继续写入。Postscript 用文档结构化规约（Document Struturing Conuention 3.0 版按 Encapsulated PostScript）格式创建。

注意，缺省情况下，Postscript 仅仅为显示在屏幕画布窗口中的信息而产生的。如果画布刚刚产生，则它将保持 1 像素×1 像素的尺寸，这样 Postscript 中就什么也不显示，避免此问题的方法是调用 update 方法来等待画布窗口达到最终大小，或者使用 width 和 height 选项来指定画布打印的区域。选项一值参数对提供额外控制产生 Postscript 的信息，下面的选项是支持的。

选　项	值（单位）	说　明
colormap	varName(array)	varName 须是数列变量名，指定 Postscript 中使用的颜色映射。每个 varName 的元素必须是设置特定颜色（如 1.0　1.0　1.0 为黑色）的 Postscript 指令。当输出颜色信息到 Postscript,Tk 检查 varName 看是否有与颜色相同名字的元素，如果有，Tk 用来作为 Postscript 命令设置颜色。如果选项没有设定，或者 varName 中没有给出颜色，Tk 使用 X 颜色的红、绿、蓝强度
colormode	mode(string)	指出如何设置输入颜色信息。Mode 必须是 color（全色输出）或 gray（灰度均衡）或 mono（转为黑白）之一
file	fileName(string)	指出输出 Postscript 的文件名，如果没有指定此项，Postscript 被作为命令返回而不是写到文件

（续）

选 项	值（单位）	说 明
fontmap	varName(arry)	VarName 必须是一个数列变量名，用来指定到 Postscript 的字体映射。VarName 每个元素必须是二值 Tcl 列，为 Postscript 的字体名和字体大小。当为特定字体输出 Postscript 命令时，Tk 检测是否 varName 包含与字体同名的名字。如果有这样的元素，则元素中的字体被用在 Postscript 中。否则 Tk 猜测该使用什么样的字体。Tk 的猜测通常只有对常用的字体如 Times,Courier, 和 Helvtica 起效，而且只有在 X 字体名中没有忽略任何字体间的连接时符有效。如-Courier- Bold-R-Normal-*-120-*行,而 CourierBoldRnormal *120*则不行。Tk 需要连接符号把字体名分开
height	height(distance)	指定画布区域的高度以供打印，缺省高度为窗口高
pageanchor	constant	指定画布中的哪一点在页面（由 pagex 和 pagey 定位的）的设定点打印。比如 pageanchor n 指被打印画布区域顶中央（画布窗口中的位置）应该放在设定点，缺省为中央
pageheight	height(distance)	设定 Postscript 的 x 和 y 方向的比例以使打印区域高度尺寸为 Postscript 页的高度。Size 为浮点数加上 c 表示厘米，i 表示英寸，p 或什么也没有表示打印机点（1/72 英寸）。Height 的缺省值为窗口的打印区域。如果 pageheight 和 pagewidth 都设置，比例因子 scale factor 将被使用
pagewidth	width（distance）	设定 Postscript 的 x 和 y 方向的比例以使打印区域宽度尺寸为 Postscript 页的宽度。Size 同 pageheight 的。Height 的缺省值为窗口的打印区域。如果 pageheight 和 pagewidth 都设置，比例因子 scale factor 将被使用
pagex	position(integer)	给定 Postscript 页的 x 坐标位置，使用页码高度可以接受的任何形式，与 pagey 及 pageanchor 一起决定打印区域在 Postscript 页中的显示位置。缺省为页码中央
pagey	position(integer)	给定 Postscript 页的 y 坐标位置，使用页码高度可以接受的任何形式，与 pagex 及 pageanchor 一起决定打印区域在 Postscript 页中的显示位置。缺省为页码中央
rotate	boolean(boolean)	指定打印区域是否旋转 90 度。非旋转方式下，打印区域的 x 轴沿页面的短轴方向（人像方向)画。在旋转方式下，打印区域的 x 轴沿页面的长轴方向（地面方向）画。缺省为非旋转
width	width(distance)	指定画布区域的宽度以供打印，缺省宽度为窗口宽
x	position(integer)	指定画布区域 x 轴要打印的左部坐标，按照画布坐标，而不是窗口坐标。缺省为窗口的左部坐标
y	position(integer)	指定画布区域 y 轴要打印的顶部坐标，按照画布坐标，而不是窗口坐标。缺省为窗口的顶部坐标

scale(tagOrId,xOrigin,yOrigin,xSc)

对画布空间中 tagOrId 给定的项重新调整。xOrigin 和 yOrigin 为调整的原标识，xscale 和 yscale 为比例调整的 x 坐标方向和 y 坐标方向的比例。方法通常与鼠标移动事件相联系。

scan_dragto(x,y)

计算参数 x 与 y（通常是鼠标坐标）和上次 scan_mark 处 x 与 y 参数之间的差距。它再通过坐标差距 10 倍调整视图。该方法通常与控件上的鼠标动作事件，产生通过窗口高速拖动画布的效果。

返回值为空串。

scan_mark(x,y)

记录 x 和 y 及画布视图，与 scan_dragto 调用联系在一起用。典型地，此方法与鼠标按下控件联系起来，x 和 y 是鼠标的坐标，返回 None。

select_adjust(tagOrId,index)

定位 tagOrId 中离给定字符 index 最近的选择的末尾。调整该选择末尾到 index（包括但不超越 index）。选择的另一端在 select_to 调用的锚点。如果选择目前不在 tagOrId 中，该方法如 select_to 一样。返回 None。

select_clear()

清除选择，如果在本控件中。如果选择不在本控件中，方法不起作用。返回 None。

select_from(tagOrId,index)

设置锚点到 tagOrId 中离给定字符 index 最近的选择的前面。该方法不改变选择，仅为将来 select_to 设置固定的结束点。返回 None。

select_item()

返回选定项的 ID，如果选中项在本画布中。如果不在本画布中，返回空串。

select_to(tagOrId,index)

设置选择范围为包含 tagOrId 中从选择锚点到 index 之间的字符。新选择将包含 index 给定的字符；当且仅当 index 大于或等于锚点的时候才包含锚点字符。锚点由 select_adjust 和 select_from 调用本控件定位。如果当前控件锚点不在 tagOrId 中，在设置在 index 给定字符。返回 None。

tag_bind(tagOrId,sequence=None,function=None,add=None)

把 function 和 tagOrId 给定的项关联起来，这样当 sequence 给定的事件序列对某个事件存在时，function 将被调用。该方法同 bind 方法一样，除了它操作于画布中的项，而不是整个控件。如果所有参数给定，它产生一个新的捆绑，替代已经存在的任何 sequence 和 tagOrId 之间的捆绑（如果 function 的第一个符号是+，函数增加一个捆绑而不是替代它），此时，返回空串。如果 function 略去，方法返回 function 和 tagOrId 之间的捆绑（如果不存在这样的捆绑，出错）。如果 function 和 sequence 都略去，方法返回所有 tagOrId 定义的捆绑的列表。

捆绑可以指定的所有事件是鼠标和键盘事件，比如 Enter、Leave、ButtonPress、Motion 和 Keypress，或虚拟事件。当一项成为当前项或不再是当前项时，Enter 和 Leave 事件触发。注意这和窗口的 Enter 和 Leave 是不一样的。

鼠标关联的事件定向到当前项，如果有的话。键盘相关事件定位到激活项。如果虚拟事件被用到捆绑中，该虚拟事件只有在被鼠标或键盘所定义的情况下才能用。可能有几个捆绑来匹配特定的事件。这可能存在于，比如说，如果一个捆绑和项的 ID 关联，而另一个捆绑则和项的标签关联。当这存在时，所有的匹配捆绑都被调用。和所有标签关联的捆绑最先被调用，接下来是每个项的标签（按顺序），再下来是项的 ID。如果单个标签存在多个捆绑与之匹配，最特定的那个被调用。

事件句柄返回的“break”字符串终止该句柄，跳过该事件接下来的所有句柄，就如 bind 方法一样。如果用 bind 方法产生了画布捆绑，则它将被除了画布项的捆绑外的空间捆绑调用。项的捆绑将被对窗口做为一个整体先于任何任何捆绑被调用。

tag_lower(tagOrId,belowThis)

把 tagOrId 给定的所有项移动到显示列表中的一个新位置，让其正好在 belowThis 项之前。如果 tagOrId 代表多于一个项，则所有的都移动，但是它们之间的相对位置不变。BelowThis 是一个标签或 ID，如果它代表多于一个项，则显示列表中第一个（最底层）的将被用做目标项，以作为移动的参考。

注意：该方法对窗口项不起作用。窗口项总是覆盖其他类型的项，窗口项的栈顺序是通过 lower 或 raise 方法来决定的，而不是画布的 raise 和 lower 控件方法。该方法返回 None。

tag_raise(tagOrId,aboveThis)

把 tagOrId 给定的所有项移动到显示列表中的一个新位置，让其正好在 aboveThis 项之后。如果 tagOrId 代表多于一个项，则所有的都移动，但是它们之间的相对位置不变。aboveThis 是一个标签或 ID，如果它代表多于一个项，则显示列表中最后（最顶层）的将被用做目标项，以作为移动的参考。

注意，该方法对窗口项不起作用。窗口项总是覆盖其他类型的项，窗口项的栈顺序是通过 lower 或 raise 方法来决定的，而不是画布的 raise 和 lower 控件方法。该方法返回 None。

tag_unbind(tagOrId,sequence,funcid=None)

除去 tagOrId 项的所有事件序列 sequence 和事件句柄 funcid 之间的捆绑。如果提供 funcid，句柄将被破坏。

type(tagOrId)

返回 tagOrId 项的类型，比如 rectangle 或 text。如果 tagOrId 代表多于一个项，type 返回显示列表中第一个项的类型；如果 tagOrId 不代表任何项，返回空串。

xview_moveto(fraction)

调整窗口视图，以使画布的整个宽度离屏幕左端为 fraction，fraction 是 0 和 1 之间的一个数。

xview_scroll(number,what)

该命令使视图在窗口中根据 number 和 what 左移或右移，number 必须是整数，what 必须是 UNITS 或 PAGES。如果是 UNITS，且大于 0，视图按照 xScrollIncrement 选项为单位调整左右移动，不然为窗口宽度的 1/10。如果是 PAGES，视图按照窗口宽度 9/10 调整左右移动。如果 number 为负数，左部信息可见；为正，右部信息可见。

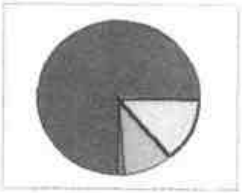
yview_moveto(fraction)

调整窗口视图，以使画布的整个宽度离屏幕上端为 fraction，fraction 是 0 和 1 之间的一个数。

yview_scroll(number,what)

该命令使视图在窗口中根据 number 和 what 上移或下移，number 必须是整数，what 必须是 UNITS 或 PAGES。如果是 UNITS，且大于 0，视图按照 yScrollIncrement 选项为单位调整上下移动，不然为窗口高度的 1/10。如果是 PAGES，视图按照窗口高度 9/10 调整上下移动。如果 number 为负数，上部信息可见，为正，下部信息可见。

Canvas Arc(画布弧形)



说明

类型 Arc 的项显示为弧形区域。弧是由两个角（由 start 和 extent 选项指定）界定的椭圆的一部分，可以有多种显示方式（由 style 选项指定）。

继承关系

从画布控件继承而来。

共享选项

选项	缺省值
Fill	Transparent
Width	1

弧特殊选项

属　性	说　明	单位	典型值	缺省值
extent	指出弧所包含的角度大小。角度从起始点按逆时针方向开始。角度可以为负，如果大于 360 度或小于-360 度，用角度模 360 之后的值	degree	10.0	
outline	指定一个颜色来画弧的边界，在 Tkinter(Tk_GetColor)中，有多种颜色供选择使用，默认为黑色。如果颜色为空，不画边界	color	RED 'black'	'black'
outlinestipple	指出弧边界线型。bitmap 指出将使用的线型，如果 outline 选项没有选中，该选项无效。如果 bitmap 为空串（缺省值），弧边界用粗线画	bitmap	'gray12'	None

（续）

属 性	说 明	单位	典型值	缺省值
start	指定弧所占角度地起始点，角度是按照顺时针方向测量的，角度大小，可以是正，也可以是负	degrees	0.0	0.0
stipple	指出弧填充点的式样。bitmap 指出使用的点填充方式，如果填充选项没有选中，则此功能不起作用。如果 bitmap 为空（缺省），使用粗点填充	bitmap	'gray25'	None
style	指定如何画弧。如果 style 是 PIESLICE（默认），则弧的区域为椭圆的部分周边和两条半径组成的区域，即为扇形。如果 style 是 CHORD，则弧的区域为椭圆的部分周边加上一条弦组成的区域。如果 style 是 ARC，则弧的区域仅仅是椭圆周边的一部分，最后这种情况，填充属性被忽略	constant	CHORD 'arc'	PIESLICE
标签	指定一组标签用于项。标签 List 包含一个标签名字的元组，它用来代替已经存在的标签	tuple	('标签 1', 'win')	None

方法

create_arc(x0,y0,x1,y1,*options)

x0,y0,x1,y1 给出包含定义弧的椭圆的矩形对角坐标。在坐标之后，可以是任何数量的选项-值对。其中的每组都设置了项的配置选项。同样的选项-值对可以用在 itemconfigure 中来修改配置。

delete(item)

删除一个 arc 项。

coords(item,x0,y0,x1,y1)

查询或修改一个项的坐标。如果没有给定坐标值，此方法返回一个列表，列表值为 item 命名的项的坐标。如果坐标给定，它更改命名的 item 的坐标。如果 item 代表多个项，显示列表中第一个项被使用。

itemconfigure(Item,*option)

更改一个或多个 arc 项的选项。

Canvas bitmap（画布位图）



说明

bitmap 类的项显示为二色（前景色和背景色）的图像。

继承关系

继承自 Widget，Canvas。

共享选项

选项	缺省值
anchor	CENTER
background	transparent
foreground	black

Canvas bitmap 特有选项

选项（别名）	说　明	单位	典型	缺省值
bitmap	指定在项中显示的位图，bitmap 可以是任何能被 Tkinter (tk_GetBitmap)接受的格式	bitmap	'info'	
tags	指定一组用于项的标签，TagList 由一组标签名字组成，取代项的已经存在的标签。TagList 可以为空	tuple	('tag1', 'bitmap')	None

方法

create_bitmap(x,y,*options)

参数 x+y 指定显示中用来定位位图的点的坐标（使用 anchor 选项）。在坐标后面可以是任何数量的选项数-值对。每对用来配置选项，这些相同的选项-值对可以用在 itemconfigure 调用中修改项的配置。

delete(Item)

删除位图项。

coords(item,x,y)

查询或修改一个项的坐标。如果没有给定坐标值，此方法返回一个列表，列表值为项的坐标。如果坐标给定，它更改给定的项的坐标。如果 item 代表多个项，显示列表中第一个项被使用。

itemconfigure(item,*options)

修改一个或多个位图项。

Canvas image（画布图像）



说明

image 类的项用来在画布中显示图像。

继承关系

继承自 Widget,Canvas。

共享选项

选项	缺省值
anchor	CENTER

图像特有选项

选项（别名）	说　明	单位	典型	缺省值
bitmap	指定在项中显示的图像名，必须是已经用 create_image 方法产生的	bitmap	'scene.gif'	
tags	指定一组用于项的标签，TagList 由一组标签名字组成，取代项的已经存在的标签。TagList 可以为空	tuple	('tag1', 'img')	None

方法

create_image(x,y,*options)

参数 x，y 指定显示中用来定位图像的点的坐标（使用 anchor 选项）。在坐标后面可以是任何数量的选项-值对。每对用来配置选项，这些相同的选项-值对可以用在 itemconfigure 调用中修改项的配置。

delete(Item)

删除图像项。

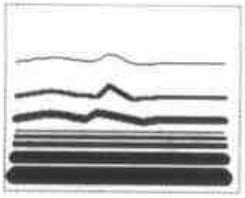
coords(item,x0,y0,x1,y1)

查询或修改一个项的坐标。如果没有给定坐标值，此方法返回一个列表，列表值为项的坐标。如果坐标给定，它更改给定的项的坐标。如果 item 代表多个项，显示列表中第一个项被使用。

Itemconfigure(item,*options)

修改一个或多个图像项。

Canvas line



说明

line 类的项用来在项中显示直线或曲线段。

继承关系

继承自 Widget、Canvas。

共享选项

选项	缺省值
fill	"black"
width	1

Line 特有选项

属　性	说　明	单　位	典型值	缺省值
arrow	指出是否需要在两端画行箭头。 where 必须是以下值之 ：None(无箭头)和 FIRST（首端有 一个箭头)，last(尾有箭头)或 both（两端有箭头）	constant	FIRST, 'last'	None
arrowtype	如何画箭头的选项。形状参数必须为 一个三元素元组，每一个指出一个 Tkinter 接受的距离形式。第 一个元素给出沿着直线从箭头颈到顶点的距离，第二个元素给出沿着直线从尾到顶点的距离。第 三个元素给出外边到拖尾的距离。如果没有给出参数，Tk 给出一个合理的形状	tuple	(6,8,3)	(8,10,3)
capstyle	指出线端格式。可以是 BUTT, PROJECTING, ROUND，如果有箭头，线端格式忽略	constant	BUTT, 'round'	BUTT
jointstyle	直线垂直相交时的相交方式。可以是 BEVEL，MITER 或 ROUND。如果直线只有两个点，则选项无关	constant	BEVEL 'miter'	ROUND
smooth	值须为布尔量，指出线是否是曲的。如果是，线由一系列的抛物样条线连成， 一个样条线为第 一和第二条直线， 一个样条线为第二和第 三条直线，依次类推。直线部分可以通过使两端重合而在曲线部分之间产生	Boolean	1, FALSE	FALSE
spinesteps	指出曲线光滑程度。每一个样条由数段直线近似，该选项只有 smooth 为 true 时才不被忽略	integer	20	12
stipple	指出椭圆填充点的式样。bitmap 指出使用的点填充方式，如果填充选项没有选中，则此功能不起作用。如果 bitmap 为空(缺省)，使用粗点填充	bitmap	'gray25'	None
tags	指定一组标签用于项。TagList 包含 一个标签名字的元组，它用来代替已经存在的标签	tuple	('tag1', 'line')	None

方法

create_line(x0,y0,x1,y1,…,xn,yn,*options)

参数 x0 到 yn 指定显示中一系列的两点或多点坐标，来描述一条线或一系列线。在数的后面可以是任何数量的选项-值对。每对用来配置选项，这些相同的选项-值对可以用在 itemconfigure 调用中修改项的配置。

delete(item)

删除线 item。

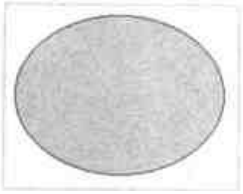
coords(item,x0,y0,x1,y1,…,xn,yn)

查询或修改一个项的坐标。如果没有给定坐标值，此方法返回一个列表，列表值为项的坐标。如果坐标给定，它更改给定的项的坐标。如果 item 代表多个项，显示列表中第一个项被使用。

itemconfigure(item,*options)

修改一个或多条线 item。

Canvas oval（画布椭圆）



说明

椭圆类项是在画布中某区域内的椭形带，每个椭圆可以有边界，填充或者二者都有。

继承关系

继承自 Widget，Canvas。

共享选项

选项	缺省值
fill	transparent
width	1

Oval 特有选项

属性	说　明	单位	典型值	缺省值
outline	指定一个颜色来画椭圆的边界，如果颜色为空，不画边界	color	RED 'black'	'black'
stipple	指出椭圆填充点的式样。bitmap 指出使用的点填充方式，如果填充选项没有选中，则此功能不起作用。如果 bitmap 为空（缺省），使用粗点填充	bitmap	'gray25'	None

（续）

属性	说　明	单位	典型值	缺省值
tags	指定一组标签用于项。TagList 包含一个标签名字的元组，它用来代替已经存在的标签	tuple	('tag1', 'oval')	None

方法

create_oval(x0,y0,x1,y1,*options)

x0,y0,x1,y1 给出包围椭圆形的矩形的对角坐标。在坐标之后，可以是任何数量的选项-值对。其中的每组都设置了项的配置选项。同样的选项-值对可以用在 itemconfigure 中来修改配置。

delete(item)

删除椭圆 item。

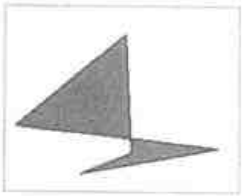
coords(item,x0,y0,x1,y1)

查询或修改一个项的坐标。如果没有给定坐标值，此方法返回一个列表，列表值为项的坐标。如果坐标给定，它更改给定的项的坐标。如果 item 代表多个项，显示列表中第一个项被使用。

itemconfigure(item,*options)

修改一个或多个椭圆项。

Canvas polygon （画布多边形）



说明

多边形项在显示中表现为一个多边形或折线填充区域。

继承关系

从控件、画布继承而来。

共享选项

选项	缺省值
fill	transparent
width	1

polygon 特有选项

属性	说　明	单位	典型值	缺省值
outline	指定一个颜色来画多边形的边界，如果颜色为空，不画边界	color	BLUE 'black'	'black'

（续）

属性	说　明	单位	典型值	缺省值
smooth	boolean 须为 Tkinter 接受的格式之一，指出多边形是否是曲周的。如果是，多边形周由一系列的抛物样条线连成，一个样条线为第一和第二条线段，一个样条线为第二和第三条线段，依次类推。一个平滑多边形中的直线段可以通过复制其端点产生	Boolean	1, FALSE	FALSE
spinesteps	指出曲线光滑程度。每一个样条由数段直线段近似，该选项只有 smooth 为 true 时才不被忽略	integer	20	12
stipple	指出多边形填充点的式样。bitmap 指出使用的点填充方式，如果填充选项没有选中，则此功能不起作用。如果 bitmap 为空（缺省），使用粗点填充	bitmap	'gray25'	None
tags	指定一组标签用于项。TagList 包含一个标签名字的元组，它用来代替已经存在的标签	tuple	('tag1', 'poly')	None

方法

create_polygon(x0,y0,x1,y1,…,xn,yn,*options)

参数 x0 到 yn 指定显示中一系列的二点或多点坐标，来描述一个封闭多边形。起点和终点可以相同也可以不同；不管相同与否，Tkinter 都画出封闭的多边形。在数的后面可以是任何数量的选项-值对。每对用来配置选项，这些相同的选项-值对可以用在 itemconfigure 调用中修改项的配置。

delete(item)

删除多边形 item。

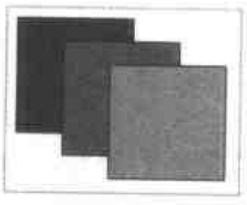
coords(item,x0,y0,x1,y1,…,xn,yn)

查询或修改一个项的坐标。如果没有给定坐标值，此方法返回一个列表，列表值为项的坐标。如果坐标给定，它更改给定的项的坐标。如果 item 代表多个项，显示列表中第一个项被使用。

itemconfigure(item,*options)

修改一个或多个多边形 item 选项。

Canvas rectangle（画布矩形）



说明

rectangle 类项是在显示中的一个长方形区域。长方形可以有边界、填充或者二者兼之。

继承关系

继承自 Widget，Canvas。

共享选项

选项	缺省值
fill	transparent
width	1

rectangle 特有选项

属性	说　明	单位	典型值	缺省值
outline	指定一个颜色来画长方形的边界，如果颜色为空，不画边界	color	RED 'black'	'black'
stipple	指出长方形填充点的式样。bitmap 指出使用的点填充方式，如果填充选项没有选中，则此功能不起作用。如果 bitmap 为空（缺省），使用粗点填充	bitmap	'gray25'	None
tags	指定一组标签用于项。TagList 包含一个标签名字的元组，它用来代替已经存在的标签	tuple	('tag1', 'rect')	None

方法

create_rectangle(x0,y0,x1,y1,*options)

参数 x0,y0,x1,y1 指定显示中长方形对坐标（长方形将包括左边和上边，而不是下边或右边）。在数的后面可以是任何数量的选项-值对。每对用来配置选项，这些相同的选项-值对可以用在 itemconfigure 调用中修改项的配置。

delete(item)

删除长方形 item。

coords(item,x0,y0,x1,y1)

查询或修改一个项的坐标。如果没有给定坐标值，此方法返回一个列表，列表值为项的坐标。如果坐标给定，它更改给定的项的坐标。如果 item 代表多个项，显示列表中第一个项被使用。

itemconfigure(item,*options)

修改一个或多个长方形 item 选项。

Canvas text（画布文本）

Canvas
Text

说明

文本项是在屏幕上的一行或多行的字符。文本项支持索引(indexing)和选中(selection)，文本有如下的画布方法：dchars,focus,icuror,index,insert

和 select。

继承关系

继承自 Widget，Canvas。

共享选项

选项	缺省值
anchor	CENTER
fill	transparent
width	1

Text 特有选项

属性	说　明	单位	典型值	缺省值
font	指定文本的字体。FontName 可以是任何 Tkinter 接受的字符串，如果此项没有指定，缺省为依赖于系统的字体	font	'Verdant'	(('MS', 'Sans', 'Serif'), '8')
justify	指定在边界区域内如何调整文本。必须是 LEFT,RIGHT, CENTER 中的一个，此选项只有在文本是多行的时候才重要	constant	RIGHT	LEFT
stipple	指出文本填充点的式样。bitmap 指出使用的点填充方式，如果填充选项没有选中，则此功能不起作用。如果 bitmap 为空（缺省），使用粗点填充	bitmap	'gray25'	None
标签	指定一组标签用于项。TagList 包含一个标签名字的元组，它用来代替已经存在的标签	tuple	('tag1', 'text')	None
text	在文本项中显示的字符。新行导致换行符，文本项中的字符可以在 insert 和 delete 之间切换	string	'Hello'	None

方法

create_text(x,y,*options)

参数 x,y 指定显示文本的点的坐标，在数的后面可以是任何数量的选项-值对。每对用来配置选项，这些相同的选项-值对可以用在 itemconfigure 调用中修改项的配置。

delete(Item)

删除文本 item。

coords(item,x0,y0)

查询或修改一个 item 项的坐标。如果没有给定坐标值，此方法返回一个列表，列表值为项的坐标。如果坐标给定，它更改给定的项的坐标。如果 item 代表多个项，显示列表中第一个项被使用。

itemconfigure(item,*options)

修改一个或多个文本 item。

Canvas window （画布窗口）



说明

window 类型使特定窗口在画布某一位置显示。

继承关系

继承自 Widget，Canvas。

共享选项

选项	缺省值
anchor	CENTER
height	window height
width	window width

窗口特有属性

属性	说　明	单位	典型值	缺省值
tags	指定一组标签用于项。TagList 包含一个标签名字的元组，它用来代替已经存在的标签	tuple	('tag1', 'win')	None
window	指定与项关联的 window，指定的 window 必须是画布控件或父代的子代。window 可以不是顶层窗口	window	mywin	None

方法

create_window(x,y,*options)

参数 x,y 指定显示窗口的点的坐标。在数的后面可以是任何数量的选项-值对。每对用来配置选项，这些相同的选项-值对可以用在 itemconfigure 调用中修改项的配置。

delete(Item)

删除窗口 item。

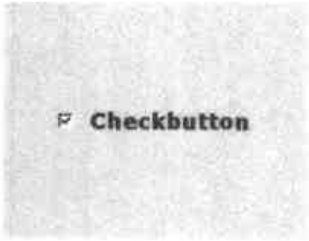
coords(item,x,y)

查询或修改一个项的坐标。如果没有给定坐标值，此方法返回一个列表，列表值为项的坐标。如果坐标给定，它更改给定的项的坐标。如果 item 代表多个项，显示列表中第一个项被使用。

itemconfigure(item,*options)

修改一个或多窗口 item 选项。

Checkbutton（复选按钮）



说明

Checkbutton 类定义了一个新窗口，并产生一个 Checkbutton 控件实例。别的选项，如下面所描述的，可以在方法调用中或选项数据库中设置复选按钮的相关选项，如颜色、字体、文本、初始凸凹度等。Checkbutton 方法返回新控件的标识。该方法调用时，复选按钮必须有父窗口存在。

复选按钮是一个显示文本、位图、图片的控件和一个方形标志器。如果是文本，必须是单一字体，但是可以占据屏幕的很多行（如果包含新行或发生换行（因为行长度选项）），其中一个字母可以通过 underline 选项加下划线。复选按钮具有简单按钮的所有行为，包括如下：它能根据 state 选项按三种方式之一显示，可以按凸起、下沉和平面显示，可以闪烁，当鼠标左键按下时，它调用回调函数。另外，复选按钮可以被选定。如果被选定，则按照被选定标记显示。与复选按钮相关的 Tkinter 变量被设置为特定的值（通常为 1）。在 Unix 下面，指示器画为下沉的，并有一定颜色。在 Windows 下，指示器里面有一个钩。

如果复选按钮没有选中，指示器按照没有选定方式画。与复选按钮相关的 Tkinter 变量被设置为特定的值（通常为 0）。在 Unix 下面，指示器画为突出的，没有颜色。在 Windows 下，指示器里面没有钩。

存储在复选按钮中的 on 和 off 可以通过命令行选项或选项数据库修改。配置选项也可以用来修改指示器显示（或是否显示）。缺省情况下，复选按钮通过鼠标点击能自动选中或反选。另外，复选按钮监控相关变量，当变量值变动时自动选中或反选。

继承关系

Checkbutton 继承自 Widget。

共享选项

选项	缺省值
activebackground	SystemButtonFace
activeforeground	SystemwindowText
anchor	center
background(bg)	SystemButtonFace
bitmap	
borderwidth(bd)	2
command	
cursor	
disabledforeground	SystemDisabledText
font	(('MS', 'Sans', 'Serif'), '8')
foreground(fg)	SystemWindowText
height	0

（续）

选项	缺省值
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	1
image	
justify	center
padx	1
pady	1
relief	flat
state	normal
takefocus	
text	
textvariable	
underline	-1
width	0
wraplength	0

复选按钮特有选项

选项（别名）	说　明	单位	典型值	默认值
indicatoron	规定指示器是否需要画出来。必须是一个正确的布尔值。如果是 FALSE，relief 选项将被忽略并且控件会在被选择的时候凹下去，而其他时候凸起	布尔值	0 TRUE	1
ofvalue	规定在该按钮被取消选定时与控件相关的变量所存储的值。默认值为 0	字符串	0 off	0
onvalue	规定在该按钮被选定时与控件相关的变量所存储的值。默认值为 1	字符串	1 On	1
selectcolor	规定在控件（通常是一个复选或者单选按钮）被选定的时候所用的背景颜色。如果 indicatoron 为 TRUE，则指示器取该颜色。在 Windows 下，不论控件的状态如何，该颜色都将作为指示器的背景颜色。如果 indicatoron 为 FALSE，只要控件被选择，该颜色将作为整个控件的背景颜色，取代 background 和 activeBackground。如果规定的是一个空字符串，则在控件被选择的时候不会有特殊的颜色出现	color	"red"	SystemWindow
selectimage	规定了在控件（通常是一个复选按钮）被选择的时候所显示的图像（取代了图像选项）。除非 image 选项已经被规定了，否则该选项将被忽略	image	"redcross"	
variable	规定一个 Tkinter 变量名来包含内容并且设定控件的内容	variable	myVariable	

方法

deselect()

取消复选按钮并且设置相关的变量取 off 值。

flash()

使复选按钮闪烁。这由重复显示复选按钮多次来实现，在活动颜色和正常颜色之间重复。在闪烁结束的时候，复选按钮恢复到方法被调用前的同一个正常/活动状态。如果复选按钮的状态被取消，则该方法将被忽略。

invoke()

一旦用户通过鼠标调用复选按钮，则如同预料的那样执行：切换按钮的选择状态并且调用与复选按钮相关的回调（如果有的话）。返回值是回调的返回值，若是没有回调的话，则返回一个空字符串。如果复选按钮的状态无效了，则该方法将被忽略。

select()

选择复选按钮并且设置相关的变量取 on 值。

toggle()

切换按钮的选择状态，重新显示按钮并且修改相关的变量使之能反映新的状态。

Entry（输入）

说明

输入类定义了一个新的窗口并且创建了一个输入控件的实例。下面将要谈到的外加选项可以在方法调用或者选项数据库中配置输入（entry）的特征，比如颜色、前端和凸凹情况。entry 方法返回这个新控件的 identity。当该方法被调用的时候，输入的父节点必须存在。

输入是这样一个控件，它可以显示一行文本字符串并且允许通过使用下面介绍的方法来编辑该字符串，通常以按键或者鼠标动作作为结束标志。首次创建的时候，输入的字符串为空。输入的一部分可以按如下所述选择。如果一个输入正在输出它的选择（参看 exportSelection 选项），那么它就要按照 X11 标准协议来处理选择；输入的选择以 STRING 类型出现。

输入也同样遵循标准 Tk 规则来处理输入焦点。当一个输入获得了输入焦点后，它就会显示一个插入光标来标明新的字符将在什么地方插入。

输入可以显示很长的字符串，而这个字符串不能全部出现在控件的窗口里。这种情况下，只有字符串的一部分被显示；下面描述的命令可以用来改变窗口的面貌。

输入使用标准 xScrollCommand 机制来与滚动条作用（详情参看 xScrollCommand 选项的描述）。如下所述，它们同样也支持搜索。

继承性

Entry 继承自 Widget。

共享选项

选项	缺省值
background(bg)	SystemWindow
borderwidth(bd)	2
cursor	xterm
font	(('MS', 'Sans', 'Serif'), '8')
foreground(fg)	SystemWindowText
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	0
justify	left
relief	sunken
selectbackground	SystemHighlight
selectborderwidth	0
selectforeground	SystemHighlightText
state	normal
takefocus	
textvariable	
width	
xscrollcommand	

Entry 特有选项

选　项	说　明	单 位	典型值	默认值
exportselection	规定了控件的一个选择是否也必须是 X 选择。取值可以是 Tcl_GetBoolean 所接受的任何一种形式，例如 true、false、0、1、yes 或者 no。如果选择输出了，则控件内选择将会取消当前的 X 选择，而控件外选择将会取消任何一个控件选择，并且当控件获得一个选择时会对选择检索请求做出反应。缺省的情况通常是控件输出选择	boolean	0 YES	1
insertbackground	规定插入光标所在区域使用的背景颜色。这个颜色一般情况下会覆盖控件的正常底色，以及插入光标碰巧落入选择时选择的底色	color	'yellow'	System WindowText
insertborderwidth	规定一个非负整数来表示插入光标周围所画的三维边界的宽度。这个值可以是 Tkinter（Tk_GetPixels）所接受的任何一种形式	pixel	2	0

（续）

选 项	说 明	单 位	典型值	默认值
insertofftime	规定一个非负整数来表示插入光标每次闪烁中停留在"off"状态的毫秒数。如果该选项为 0，则光标将不闪烁——它将一直都在	integer	250	300
insertontime	规定一个非负整数来表示插入光标每次闪烁中停留在"on"状态的毫秒数	integer	175	600
insertwidth	规定插入光标的总的宽度的数值。这个值可以是 Tkinter（Tk_GetPixels）所接受的任何一种形式。如果一个边界被专门用于插入光标（使用 insertBorderWidth 选项），则这个边界会在 insertWidth 选项所规定的宽度内绘制	Pixel	2	2
show	如果该选项被指定了，则输入的原文就不再在窗口中显示。输入值的每个字符都将被替换为该选项值的第一个字符，例如*。这是很有用的，举例来说，当输入被用于输入密码时。如果输入处的字符被选中并且复制到别的地方，则复制出的信息将会是所显示的东西，而不是输入处的原文	character	"*"	

方法

delete(first, last=None)

删除输入的一个或多个元素。first 是第一个被删除字符的索引，而 last 则是最后一个被删除字符的下一个字符的索引。如果 last 没有指定则缺省值是 first+1，即只删除一个字符。该方法返回 None。

get()

返回输入字符串。

icursor(index)

令插入光标在 index 所指示的字符前面显示。返回为 None。

index(index)

调整窗口中的视图使得 index 所指示的字符显示在窗口的左边。

insert(index,string)

把字符串 string 中的字符插入到 index 所指示的字符的前面。返回为 None。

scan_dragto(x)

计算其参数 x 与控件调用的上一个 scan_mark 方法的参数 x 的差值，然后调整视图在 x 轴方向左移或者右移这个差值的 10 倍。这条命令通常与控件中鼠标移动事件相关，这样可以产生在窗口中快速地拖拉输入的效果。返回值是一个空字符串。

scan_mark(x)

记录 x 和输入窗口中当前的视图；它在使用上和后面的 scan_dragto 方法相联系。通常这条命令和控件中按压鼠标钮相关。它返回一个空字符串。

selection_adjust(index)

对选择中离 index 所指示的字符最近的一端定位，并且调整选择的这一端使之位于 index 处（意思是包含而不越过 index）。而选择的另一端则会成为锚点以备后面的 select to 命令用。如果选择当前不在输入中，那么一个新的选择会被创建，并且包含 index 和最近的选择锚点之间的所有字符。返回为空字符串。

selection_clear()

如果选择在这个控件中则清除它。如果选择不在这个控件中则该方法不产生任何的效果。返回 None。

selection_from(index)

设置选择的锚点正好位于 index 所指示的字符前面。不要修改选择！返回 None。

selection_present()

如果输入中有字符被选择了，则返回 TRUE，若没有则返回 FALSE。

selection_range(start,end)

设置选择使得包含由 start 标记的字符开始到 end 前一个字符结束的全部字符。如果 end 所指的字符和 start 的一样或者还要靠前，则输入的选择将被清除。

selection_to(index)

如果 index 在锚点之前，则设置由 index 起，到锚点止（但不包括锚点）的那些字符为选择。如果 index 和锚点相同，则不作用。如果 index 在锚点之后，则设置由锚点起，到 index 止（但不包含 index）的那些字符为选择。锚点由控件中最近一次 select from 或 select adjust 命令来决定。如果选择不在这个控件中，而一个新的选择将被创建，并且使用指定给控件的最近的一个锚点。返回 None。

xview(index)

调整窗口中的视图，使得 index 所指的字符显示在窗口的左边界上。

xview_moveto(fraction)

调整窗口中的视图，使得位于文本全长的 fraction 部分处的那个字符出现在窗口的左边界上。fraction 必须是 0 到 1 之间的一个分数。

xview_scroll(number,what)

根据 number 和 what 在窗口中左移或者右移视图。number 必须是一个整数。what 必须是 UNITS 或者 PAGES 或者两个中一个缩写。如果 what 为 UNITS，则视图将向左或者右调整 number 个平均宽度的字符；如果是 PAGES，则视图将调整 number 个全屏。如果 number 为负数，则左前方的字符就会被显示；如果是正数，则右后方的字符就会被显示。

Font class（字体类）

继承性

无。

说明

Font 类提供了处理字体的一些工具，比如定义指定字体和检查字体的实际属性。这个类定义了若干个方法。

共享选项

无。

Font 特有选项

选项	说　明	单位	典型值	默认值
family	字体族名，与具体的情况关系不明显。Tk 保证可以使用这样一些字体族，Courier（一种等宽的“打字机”字体），Times（一种有衬线的“新闻”字体），还有 Helvetica（一种无衬线的“欧洲”字体）。在使用上面的任何一种字体族时，最为密切匹配的本地字体族将被自动代入。名字也可以是一个本地的，平台特定字体族的名字；如果是这样的话，则可能会在某个平台取得了预期的效果，而在别的平台则显示不正确。如果 family 没有被指定或者没有被识别出，则一个平台特定的缺省字体将被选择	string	'Times'	'MS'
font	X-字体格式中的字体说明符或者是一个（family，size，style）元组	font	'MS'	(('MS', 'Sans', 'Serif'), '8')
overstrike	取值为一个布尔型标志，用来规定在这种字体中是否有一条水平线需要从字符的中间画出。overstrike 的缺省值为 false	Boolean	1 FALSE	FALSE
size	想要得到的字体的大小。如果参数 size 是一个正数，则它就是以磅为单位的尺寸大小。如果 size 为一个负数，则它的绝对值就是以像素为单位的尺寸大小。如果一个字体不能按规定的大小显示，则选择一个相近的尺寸大小。如果没有规定 size 或者取为 0，则选择一个与平台相关的默认尺寸大小	integer	12, -16	
slant	字体中字符由垂直位置倾斜的数量。slant 的有效值为 roman 或者 italic。roman 字体是一种正常的直立的字体，而 italic 字体是一种由垂直位置倾斜一定角度的字体。与所指定的斜体字最接近的 slant 将被选择。slant 的默认值为 roman。	constant	ITALIC	NORMAL

（续）

选项	说　明	单位	典型值	默认值
underline	取值为布尔型标志，用来指定字体中的字符是否需要加下划线。	Boolean	TRUE 0	FALSE
weight	字体中字符指定的粗细。值 NORMAL 指定字符为正常的粗细，而 BOLD 则指定字体为粗字体。与所指定的 weight 最为接近的粗细将被选择。	constant	BOLD	NORMAL

方法

actual(option=None)

当字体在窗口中显示时，获取关于实际属性的信息，然后返回；所获得的真实属性有可能和由平台相关限制所要求的属性不同，比如字体族和点尺寸的有效性。如果 option 省略，返回所有实际字体属性作为字典。如果 option 指定了，则返回该属性的值。

cget(option)

查询当前字体所需要的属性 option。

configure(**options)

查询或者修改当前字体所需要的属性。如果没有一个 option 被指定，则返回一个描述 fontname 的所有选项和值的字典。如果某个 option 没有被指定值，则返回该属性的当前值。如果有一个或多个选项-值对被指定，则此方法修改指定的字体使具有指定的值。

copy()

返回实际字体的一个拷贝。

measure(text)

计算在当前显示字符串所需要的空间大小。返回值为文本的宽度，而不是扩大字体如“f”所占的多余空间。如果字符串包含新行或换行符或 tab，则那些字符不被扩展或专门地处理。

metrics(*options)

返回窗口中显示字体的规格（字体特有的数据）信息。如果 options 给定，返回其规格，如果略去，返回所有规格和值的字典。

函数

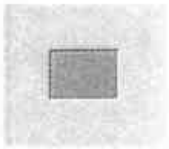
families(root=None)

返回所有字体族的列表。

names(root=None)

返回值为当前定义的命名字体的名字列表。

Frame（框架）



说明

Frame 类定义了一个新的窗口，并生成一个 Frame 控件的实例。下面描述的附加选项可以在方法调用或者可选项数据库中指定，用来配置 Frame 的特性，例如背景颜色和凸凹度。Frame 命令返回新窗口的路径名。

Frame 是一个简单控件。它的主要目的是作为复杂窗口页面的容器。Frame 仅有的特征是背景颜色和让框架上升或下沉的可选 3D 边界。

继承关系

Frame 继承自 Widget。

共享选项

选项	缺省值
background(bg)	SystemButtonFace
borderwidth(bd)	0
cursor	
height	0
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	0
relief	flat
takefocus	0
width	0

Frame 特有选项

选　项	说　明	单位	典型值	缺省值
class	指定窗口的一个类。该类在查询窗口其他选项的选项数据库时使用，以后在其他如捆绑这样的功能上也会用到。采用 configure 方法不能修改类选项。注意，class 是一个保留字，tkinter 必须使用_class	class		Frame
colormap	指定窗口颜色映射。值可以为 NEW，此时为窗口和子窗口创建一个新色彩映射，或其他窗口名（必须在同一屏幕上和相同的路径画面），此时新窗口将从指定窗口中使用颜色映射。如果窗口使用和父窗口相同的颜色映射，此选项不能被配置方法改变	colormap	New myWindow	

（续）

选　项	说　明	单位	典型值	缺省值
container	该值必须是 boolean。如果为 TRUE，它意味着该窗口将用做容器，在这里一些应用为内嵌的（比如 tkinter 顶层使用该选项内嵌其中）。窗口将对如几何请求之类的东西支持适当窗口管理协议。该应用中窗口不该拥有自己的子窗口。此选项不能被配置方法改变	Boolean	TRUE 0	0
visual	按 winfo.visual 格式指定新窗口视图信息，如果选项没有指定，新窗口将使用与父体相同的视图，视图选项不能通过 configure 方法修改	visual	monochrome	

方法

除了控件常用的方法如 configure 外，Frame 没有别的方法。

Grid geometry manager（网格几何管理器）

继承关系

没有从任何控件继承过来。

说明

Grid 用来与几何主控件（也称主窗口）通信，主窗口是安排另一窗口内控件行列关系的网格几何管理器。

共享选项

无

Grid 独有选项

选　项	描　述	单位	典型值	缺省值
column	插入从控件到网格第 n 列，列从 0 开始。如果选项没有给出，从控件被插入到上次调用网格时的从控件右面，或者第 0 列，如果它是第一个从控件的话。对每一个在从控件前的元素 x，列加 1。这里 x 代表网格该行中的一个空列	integer	1　4	0
columnspan	插入从控件到网格,使其占用 n 列空间	integer	2	1
in_	在给定主窗口插入从控件	widget	myWin	NONE
ipadx	数量指定了在从控件的每一边留下多少水平内部填充，此空间通常加到从控件边界内，数量必须是一个有效的屏幕距离，例如 2 或者.5c	distance	5m	0
ipady	数量指定了在从控件的每一边留下多少垂直的内部填充。此空间通常加到从控件边界内	distance	3m	0

（续）

选 项	描 述	单位	典型值	缺省值
minsize(row,column)	按照屏幕单位设置最小尺寸，对行和列均可	integer	25	None
pad	指定一个屏幕单位，该单位加到完整包含在该列的最大的窗口上（当几何管理器从包容窗口请求一个尺寸时）	distance	5	0
padx	数量指定了在从控件的每一边留多少水平外部填充。按照屏幕单位，此空间通常加到从控件边界外	distance	5	0
pady	数量指定了在从控件的每一边留下多少垂直外部填充。按照屏幕单位，此空间通常加到从控件边界外	distance	5	0
row	插入从控件到网格第 n 行，行从 0 开始。如果选项没有给出，从控件被插入到上次调用网格时的从控件右面，或者第 0 行，如果它是第一个从控件的话。对每一个在从控件前的元素 x，行加 1。这里 x 代表网格该行中的一个空行	integer	3	0
rowspan	插入从控件到网格,使其占用 n 行空间	integer	4	some row
sticky	如果从控件单元比要求的空间大，该选项用来把从控件放入单元中，STYLE 是一个包含 N，S，E，或 W 中一个或多个字符的字符串。字符串可以带有空格和逗号，但是它们被忽略，每一个字母分别代表（北，南，东，西）从控件将“粘贴”的方位。如果 N 和 S 同时给定（或 E 与 W）则拉为整个高度（宽度）。粘贴选项包含 pack 使用的 anchor 和 fill 组合	string	EW	CENTER
weight(row/column)	设定行列之间空间的相对大小，0 粗度表示该列不会偏离需要尺寸。当分配多余空格时，weight 为 2 的列将是 weight 为 1 的两倍	integer	2	0

方法

grid(option=value,…)

给 self 使用网格管理器。

grid_bbox(column=None,row=None,col2=None,row2=None)

没有参数时，返回网格的边界框（以像素格式）。返回值包括四个整数，前两个为网格左上角从主窗口的偏移量（先为 x，后是 y）。后两个为网格的宽和高（像素）。如果方法调用中指定单行或列，那个单元的边界框被返回。如果方法调用中指定多行或列，则所跨单元的边界框被返回。

grid_columnconfigure(index,options…)

查询或设置几何主窗口，master 的 index 列属性。有效选项为 minsize，weight 和 pad。

grid_configure(options…)

参数里包含了指定如何管理从窗口的参数对。

grid_forget()

把 self 从网格中移走，并取消其窗口映射。从窗口将不再由网格几何管理器管理。窗口的设置选项被忽略。因此当从窗口再次被网格几何管理器管理时，初始缺省选项将被使用。

grid_info()

返回一个值为当前从窗口设置状态的列表。元组前二值为 in master，这里 master 是从窗口的 master。

grid_location(x,y)

给定相对于主窗口的 x,y 值，返回 x,y 位置的行列值。网格左或上部位置，返回-1。

grid_propagate(flag=_noarg_)

如果 flag 为布尔真值，比如 1 或 on 则传送对 self 启用。如果 flag 为布尔假值，则传送对 self 禁止。如果 flag 略去，方法返回传递是否启用的布尔值 true 或 false。缺省情况下传送是启动的。

grid_rowconfigure(index,options…)

查询或设置几何主窗口，master 的 index 行属性。有效选项为 minsize，weight 和 pad。

grid_remove()

把从窗口从网格中移走，并取消其窗口映射，从窗口将不再由网格几何管理器管理，然而窗口的设置选项依然保留。因此当从窗口再次被网格几何管理器管理时，原先设定值将被使用。

grid_size()

返回 master 网格大小(行和列)。大小是通过从窗口所占行或列决定，或者是 minsize，weight，pad 中一个非零时的最大行或列决定。

grid_slaves(row=None,column=None)

如果没有给选项，给出 master 中所有从窗口的列表，最近管理的最先。选项可以是 row 或 column，只返回 value 指定的行或列的从控件。

Label（标签）

说明

Label 标签类定义了一个新的窗口，并生成一个标签控件的实例。下面描述的附加选项可以在方法调用或者可选项数据库中指定，用来配置标签的特性，例如颜色、字体、文本和凸凹度。标签方法返回新控件的标识。调用此方法时，标签父体必须存在。

标签是可以显示文本串、位图或图像的控件。如果显示的是文本，必须是单一字体，

但可以在多行显示（如果有新行或折行（因为 wraplength 选项）存在的话），且其一个字母可以通过使用下划线选项而带下划线。标签可以使用一些简单方法操作，比如通过标准控件选项改变凸凹度或文本。

继承关系

Label 继承自 Widget。

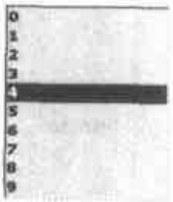
共享选项

选项	缺省值
anchor	center
background(bg)	SystemButtonFace
bitmap	
borderwidth(bd)	2
cursor	
font	(('MS', 'Sans', 'Serif'), '8')
foreground(fg)	SystemButtonText
height	0
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	0
image	
justify	center
padx	1
pady	1
relief	flat
takefocus	0
text	
textvariable	
underline	-1
width	0
wraplength	0

方法

没有 label 方法，除了如 configure 这样的普通控件方法。

Listbox（列表框）



说明

列表框类定义了一个新的窗口，并生成一个列表框控件的实例。下面描述的附加选项可以在方法调用或者可选项数据库中指定，用来配置列表框的特性，例如颜色、字体、文本和凸凹度。列表框方法返回新控件的标

识。调用此方法时，列表框父体必须存在。

列表框是显示一列文本串的控件，每行一个。当最初创建时，没有元素值。元素值可以通过下面描述的方法增删，另外，一行或多行字符可以如下面描述的被选定。

如果列表框导出选值(见 exportSelection 选项)，它将遵循处理选择的标准 X11 协议。列表框选择是作为字符串处理的，值为选择元素的文本，并以新行把元素分开。

没有必要让所有的元素在列表框中一次显示。下面描述的命令可以用来改变窗口的视图。列表框可以通过标准 xScrollcommand 和 yScrollcommand 选项使用滚动条上下滚动。它还支持浏览，如下所描述。

继承关系

Listbox 继承自 Widget。

共享选项

选项	缺省值
background(bg)	SystemButtonface
borderwidth	2
cursor	
font	(('MS', 'Sans', 'Serif'), '8')
foreground(fg)	SystemButtonText
height	10
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	1
relief	sunken
selectbackground	SystemHighlight
selectborderwidth	1
selectforeground	SystemHighlightText
takefocus	
width	20
xscrollcommand	

Listbox 特有选项

选 项	说 明	单位	典型值	缺省值
exportselection	规定了控件的一个选择是否也必须是 X 选择。取值可以是 Tcl_GetBoolean 所接受的任何一种形式，例如 true、false、0、1、yes 或者 no。如果选择输出了，则选择控件中的值将会取消当前的 X 选择，而选择控件外值将会取消任何一个控件选择，并且当控件获得一个选择时会对选择检索请求做出反应。缺省的情况通常是控件输出选择	boolean	0 YES	1
selectmode	指定操纵选择的几种方式之一。选项值可以是任意的，但是缺省捆绑却希望它是 SINGLE、BROWSE、MULTIPLE 或 EXTENDED，缺省值为 BROWSE	constant	SINGLE "browse"	browse

（续）

选 项	说 明	单位	典型值	缺省值
setgrid	指定一个布尔量，决定控件是否控制顶层窗口网格尺寸。此选项常用于文本控件中，这里控件中的信息有个自然尺寸（一个字符的大小），且它使窗口的大小为这个单元的整数倍，这些自然窗口尺寸组成网格。如果setGrid选项设置为true，则控件将与窗口管理器进行通信，以便当用户交互地调整包含控件的顶层窗口时，窗口将按网格单位显示于用户，窗口尺寸限制为网格单位的整数倍。	boolean	NO 1	0
yscrollcommand	指定一个与垂直滚动条通信的前缀，该选项与xscrollcommand的方法相同，除了它用在垂直滚动条且由支持垂直滚动条的控件提供，选项详细情况参见 xscrollcommand说明	function		

方法

activate(index)

设置 index 指定元素活动元。如果 index 在列表框范围之外，选择最近的一个。当控件有输入焦点时，它被画成带下划线。它的索引可以通过活动 index 获得。

bbox(index)

返回 index 给定文本的边界框的四个值。前两个为文本覆盖屏幕左上角的坐标（相对控件，采用像素单位），后两个为区域的宽和高，像素单位。如果 index 所给的没有一部分在屏幕上可见，结果为 None，如果元素部分可见，结果为元素整个区域，包括不可见部分。

curselection()

返回列表框中当前选定元素的数字索引。如果没有选定任何元素，返回空串。

delete(first,last=None)

删除列表框中一个或多个元素。first 和 last 指定删除的范围。如果 last 没有给定，缺省值为 first，单个元素被删除。

get(first,last)

如果 last 省去，返回 first 指定的列表框元素内容，或者空串，如果 first 指向一个不存在的元素的话。如果 last 给定，返回从 first 到 last 的完整的列表元素。first 和 last 都可以是任何标准索引格式。

index(index)

调整窗口视图，使 index 所指元素能在窗口中显示。

insert(index,*elements)

在 index 正前面插入零个或多个新元素。如果 index 设为 End，插入到列表尾。返回 None。

nearest(y)

给定列表窗口中的坐标 y，该方法返回离坐标 y 最近的（可见）元素值。

scan_dragto(x,y)

该方法计算 x 和 y 坐标（通常是鼠标坐标）和上次 scan_mark 标记的地方的 x 和 y 的差距，然后调整视图为 10 倍距离大小。该方法通常与控件中的鼠标移动事件相关，产生窗口中高速移动列表效果。返回值为空串。

scan_mark(x,y)

记录 x 和 y 及列表框当前视图，与上次 scan_dragto 调用相关。特别地，该方法把按下控件的鼠标按钮和鼠标的 x,y 联系起来。返回 None。

see(index)

调整列表框视图，以使 index 所给元素在列表框中可见。如果该元素已经可见，方法无作用，如果元素离窗口一端很近，列表框滚动使元素在此端视图可见，否则滚动中央。

selection_anchor(index)

在 index 元素处设置选择锚。

selection_clear(first,last=None)

如果 first 和 last 包括 last 指定的元素之间的元素被选定，方法让其取消选择。范围外的选定元素不受影响。

selection_includes(index)

返回 true，如果 index 元素被选定，不然为 false。

Selection_set(first,last=None)

选定在 frist 和 last 之间的元素，包括 last 指定的元素。不影响指定范围外的元素。

size()

返回列表框中元素行数的个数。

xview_moveto(fraction)

调整窗口中的视图，使得位于列表框总宽度的 fraction 部分处的那个字符出现在窗口的左边界上。fraction 必须是 0 到 1 之间的一个分数。

xview_scroll(number,what)

根据 number 和 what 在窗口中左移或者右移视图。number 必须是一个整数。what 必须是 UNITS 或者 PAGES 或者两个中一个缩写。如果 what 为 UNITS，则视图将向左或者右调整 number 个平均宽度的字符；如果是 PAGES，则视图将调整 number 个全屏。如果 number 为负数，则[illegible]前方的字符就会被显示；如果是正数，则右后方的字符就会被显示。

yview_moveto(fraction)

调整窗口中的视图，使得位于列表框总高度的 fraction 部分处的那个字符出现在窗口的上边界上。fraction 必须是 0 到 1 之间的一个分数。

yview_scroll(number,what)

根据 number 和 what 在窗口中上移或者下移视图。number 必须是一个整数。what 必须是 UNITS 或者 PAGES 或者两个中一个缩写。如果 what 为 UNITS，则视图将向上或者下调整 number 个平均宽度的字符；如果是 PAGES，则视图将调整 number 个全屏。如果 number 为负数，则上部字符就会被显示；如果是正数，则下部字符就会被显示。

Menu（菜单）



说明

菜单类定义了一个新顶层窗口和一个菜单控件实例。下面描述的附加选项可以在方法调用或者可选项数据库中指定，用来配置菜单特性，例如背景颜色和字体。Menu 方法返回新窗口的标识。此方法调用时，父体必须存在。

继承关系

Menu 继承自 widget。

共享选项

选项	缺省值
activebackground	SystemHighlight
activeforeground	SystemHighlightText
background(bg)	SystemMenu
borderwidth(bd)	1
cursor	arrow
disabledforeground	SystemDisabledText
font	('Georgia', '8')
foreground(fg)	SystemMenuText
relief	flat
takefocus	0

Menu 特有选项

选 项	说 明	单位	典型值	缺省值
activeborderwidth	设置一个非负值，该值显示画活动元素时的控件外围 3D 边界的宽度；该值可以是 Tkinter(Tk_GetPixels)接受的任何格式。此选项只有在同时显示多于一个元素时才有效（如菜单，但不是按钮）	pixel	2，1m	1

（续）

选项	说明	单位	典型值	缺省值
postcommand	如果此选项指定，则它提供一个 Tkinter 命令用在每次发送菜单时执行。命令在发送菜单时被控件执行，注意在 Macintosh 和 Windows 下的 8.0 版本，所有菜单系统的命令都在发送菜单前被执行。这是因为个人平台的菜单管理器的缺点造成的	command	display-Menu	
selectcolor	规定在控件（通常是一个复选或者单选按钮）被选定的时候所用的背景颜色。如果 indicatoron 为 TRUE，则指示器取该颜色。在 Windows 下，不论控件的状态如何，该颜色都将作为指示器的背景颜色。如果 indicatoron 为 FALSE，只要控件被选择，该颜色将作为整个控件的背景颜色，取代background 和 activeBackground。如果规定的是一个空字符串，则在控件被选择的时候不会有特殊的颜色出现	color	"red"	SystemMenu Text
tearoff	此选项必须是一个适当的布尔值。它指定了菜单上部是否包含一个下拉输入。如果包括，则存在于菜单的属于 0，其他输入从 1 开始编号。缺省菜单捆绑使菜单在调用带虚线框输入时下拉	boolean	TRUE 0	1
tearoffcommand	如果此选项为非空值，则它指定一个在菜单拉下时调用的命令，真正的命令包括选项值，紧跟着菜单窗口的名字，再跟着空格，接着是下拉菜单窗口的名字	command	myTearoff	
title	此串用做创建 shell 或拉下菜单时窗口创建的标题。对于菜单，如果标题是 NULL，则窗口标题为菜单按钮标题和调用此菜单的下拉项的文本	string	"WidgetTable"	
type	此选项可以是 MENUBAR, TEAROFF 或 NORMAL 中的一个，在菜单创建时被指定。尽管在修改选项后配置数据库返回的字串会改变，这不影响菜单控件的行为。这通常应用于克隆机制，在 Tk 库外不大被设置	constant	NORMAL	normal

方法

add_cascade(options…)
在菜单底部增加新的下拉菜单。

add_checkbutton(options…)
在菜单底上部增加新复选按钮。

add_command(options…)
在菜单底上部增加新命令。

add_radiobutton(options…)
在菜单底上部增加新单选按钮。

add_separator(options…)
在菜单底上部增加新分割符。

delete(index1,index2=None)
删除 index1 到 index2 之间的所有菜单项。如果 index2 略去，缺省为 index1。删除带虚线框菜单项的试图是被忽略的（否则，你应修改 tearOff 选项以删除带虚线框项）。

entrycget(index,option)
返回 index 所给项的当前配置选项。option 可以是任何 add 方法接受的值。

entryconfigure(index,options…)
与 configure 命令相同。除了它是对单个项 index 的选项，而 configure 则是用于菜单整体的选项。option 可以是任何 add 控件方法接受的值。如果 option 给定，option 被更新为方法指定的值，返回空值。如果没有指定 options 值，则返回一个描述项 index 当前选项的列表。

index(index)
返回 index 相对应的数值索引，或 None，如果 index 指定为 None。

insert_cascade(index,options…)
与 add_cascade 相同，除了它插入一个新级联菜单到 index 所给项正前面，而不是加到菜单之后。options 的诠释和 add 方法相同。如果菜单有带虚线框项，不可在其前面加入新菜单项。

insert_checkbutton(index,options…)
与 add_checkbutton 相同，除了它插入一个复选按钮到 index 所给项正前面，而不是加到菜单之后。options 的诠释和 add 方法相同。如果菜单有带虚线框项，不能在其前面加入新菜单项。

insert_command(index,options…)
与 add_command 相同，除了它插入一个新命令项到 index 所给项正前面，而不是加到菜单之后。options 的诠释和 add 方法相同，如果菜单有带虚线框项，不能在其前面加入新菜单项。

insert_radiobutton(index,options…)
与 add_radiobutton 相同，除了它插入一个新单选按钮项到 index 所给项正前面，而不是加到菜单之后。options 的诠释和 add 方法相同。如果菜单有带虚线框项，不能在其前面加入新菜单项。

insert_separator(index,options…)

与 add_separator 相同，除了它插入一个新分割符项到 index 所给项正前面，而不是加到菜单之后。options 的诠释和 add 方法相同，如果菜单有带虚线框项，不能在其前面加入新菜单项。

invoke(index)

调用菜单项 index 的动作。如果菜单项禁止，不发生任何事件。如果项有与之相关的回调，回调结果被作为控件调用的结果返回。

注意：调用菜单项不自动取消粘贴菜单，缺省捆绑通常在调用 invoke 控件回调前处理调用。

post(x,y)

安排菜单在根窗给定坐标 x 和 y 处显示。如果有必要，为了保证整个菜单都可以在屏幕上显示，这些坐标可以调整。该方法通常返回 None。如果 postCommand 选项已经被设定，则其值作为粘贴菜单之前的回调，然后把此脚本结果作为 post 方法的结果返回。如果执行方法的时候返回错误，则不执行粘贴菜单就把错误返回。

除了 tk_popup，下面的方法只有在你为菜单写自己的事件处理程序时才有用。它们的功能是设置菜单元素的状态如缺省行为发生过一样。它们在模拟用户与 GUI 交互也有用。

tk_bindForTraversal()
tk_firstMenu()
tk_getMenuButtons()
tk_invokeMenu()
tk_mbButtonDown()
tk_mbPost()
tk_mbUnpost()
tk_nextMenu(count)
tk_nextMenuEntry(count)

tk_popup(x,y,entry="")

在屏幕给定位置发送一个菜单，并配置 Tk，使菜单和子下拉菜单可以通过鼠标和键盘遍历。x 和 y 是显示菜单的根坐标。如果 entry 省去，或者是空串，菜单的左上角放置在该点，否则输入值给出菜单值的一个索引，菜单将放置在使得输入值恰好在给定点上。

tk_traverseToMenu(char)
tk_traverseWithinMenu(char)

type(index)

返回 index 给定菜单类型。这是当项创建时传递给 add 方法的类型参数。比如

command 或 separator 或用于带虚线框项的 tearoff。

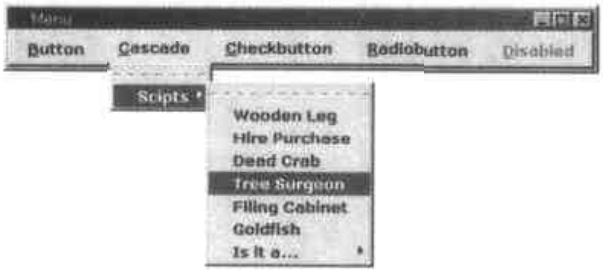
unpost()

取消窗口映射，这样它就不再显示了。如果低级级联菜单被粘贴，则 unpost 此菜单，返回空串。此方法对 Windows 和 Macitosh 无效，因为这些平台有它们自己的取消粘贴菜单。

yposition(index)

返回一个整数，给出菜单窗口中 index 指定的项最上层像素的 y 坐标。

Menubutton（菜单按钮）



说明

Menubutton 类定义了一个新的窗口，并生成一个菜单按钮控件的实例。下面描述的附加选项可以在方法调用或者可选项数据库中指定，用来配置菜单按钮的特性，例如颜色、字体、文本内容和初始凸凹情况。Menubutton 方法返回新控件的特性。在方法被激活时，菜单按钮的父进程必须存在。

菜单按钮是用来显示文本字符串、位图或者图像的控件，并且和一个菜单控件相联系。文本在显示时，必须使用同一种字体，但可以占据屏幕的多行（如果包含新行或者由于 wrapLength 选项而造成换行），并且任意一个字符都可以使用 underline 选项来添加下划线。

在一般使用时，当在菜单按钮上方单击鼠标左键，会使得相对应的菜单恰好弹出在按钮的下方。如果在释放鼠标按键之前把鼠标移动到菜单之上，那么释放鼠标将会导致下面的菜单选项被激活。当鼠标释放时，菜单将不再显示。菜单按钮通常排列成组——菜单条，以便于浏览：如果鼠标按键在一个菜单按钮上按下（导致它的菜单弹出），接着鼠标移动到同一菜单条中另一个按钮上，其间不释放鼠标，那么第一个按钮的菜单不再显示，取而代之的是新的按钮的菜单弹出显示。

菜单按钮和菜单之间有若干交互作用：请查看菜单项来获取各种菜单配置的信息，例如下拉菜单和可选菜单。

继承关系

Menubutton 继承自 Widget。

共享选项

选项（别名）	缺省值
Activebackground	SystemButtonFace

（续）

选项（别名）	缺省值
activeforeground	SystemButtonText
anchor	center
background （bg）	SystemButtonFace
bitmap	
borderwidth （bd）	2
cursor	
disableforeground	SystemDisabledText
font	(('MS', 'Sans', 'Serif'), '8')
foreground （fg）	SystemButtonText
height	0
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	0
image	
justify	center
padx	4p
pady	3p
relief	flat
state	normal
takefocus	0
text	
textvariable	
underline	-1
width	0
wraplength	0

Menubutton 特有选项

选项（别名）	说　明	单位	典型值	缺省值
direction	指定菜单弹出的方向：ABOVE 表示向菜单按钮上方弹出，BELOW 表示向下方弹出，LEFT 表示向左面弹出，RIGHT 表示向右面弹出，FLUSH 表示直接在菜单按钮的上面覆盖式弹出	constant	FLUSH "above"	below
indication	指定是否绘出指示器，它必须有一个合适的的布尔值。如果取为 FALSE，那么 relief（浮雕外形）选项将被忽略，控件在被选择时的外形为凹陷下去的，否则是凸起的	Boolean	0 TRUE	0
menu	指定和一个菜单按钮相关联的菜单的路径，而且该菜单必须是这个菜单按钮的子类。	string	subMenu-Action	

方法

menubutton(options…)
options 决定了 membutton 方法的确切行为。

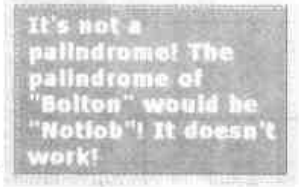
cget(option)

返回配置选项的当前值。

configure(options…)

查询或修改控件的配置选项。如果没有指定 option，则返回一个描述菜单按钮所有变量选项的字典。如果指定 option 没有值，那么该命令返回一个名为 option 的字典。如果指定了一个或多个选项值对，那么方法就按照当前的输入值修改给定的控件选项；在这种情况下，方法返回一个空的字符串。options 可以是菜单按钮可接收的任何值。

Message（消息）



说明

Message 类定义了一个新的窗口，并生成一个消息控件的实例。下面描述的附加选项可以在方法调用或者可选项数据库中指定，用来配置消息的方法，例如颜色、字体、文本内容和初始外形。Message 方法返回新控件的特性。在方法被激活时，消息的父进程必须存在。

消息是用来显示文本字符串的控件，它有以下三方面的特性。第一，为了产生一个适合窗口给定比例的外观，需要把字符串分割成行。在可能的情况下，在字符的边界分割（否则即使一个单独的字符放在一行里适合，它也要被分割）。字符串中的回车字符将强迫分行，例如在显示时可以留出一些空白。

消息控件的第二个特性是对齐。文本可以显示为左对齐（每一行均以窗口的左端作为起始），基于一行一行的中心对齐和右对齐（每一行均以窗口的右端作为结束）。

消息控件的第三个特性是对于控制符和不可打印字符的特殊处理。Tab 字符被替换为足够多的空白排列在接下来的 8 个字节边界。New Lines 导致换行。其他控制符（ASCII 码小于 0x20）和字体中没有定义的字符将作为 4 个字节的序列\xhhh 来显示，其中 hh 代表对应字符的两位十六进制数。在特殊情况下，如字体不包含 0123456789abcdefx 中所有的字符，那么控制符和未定义的字符将根本不会显示。

继承关系

Message 继承自 Widget。

共享选项

选项（别名）	缺省值
anchor	center
background （bg）	SystemButtonFace
borderwidth （bd）	2
cursor	
font	(('MS', 'Sans', 'Serif'), '8')
foreground （fg）	SystemButtonText

（续）

选项（别名）	缺省值
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	0
justify	left
padx	-1
pady	-1
relief	flat
takefocus	0
text	
textvariable	
width	0

Message 特有选项

选项（别名）	说　明	单位	典型值	缺省值
aspect	指定一个非负的整数值来指示想要得到的文本显示比例。显示比例指定为 100*width/heigt。100 表示文本的宽和高一样长，200 表示文本的宽是高的 2 倍，50 表示文本的高是宽的 2 倍等等。如果 width 选项没有指定，通常为文本选择行的长度，缺省值为 150	integer	50 75	150

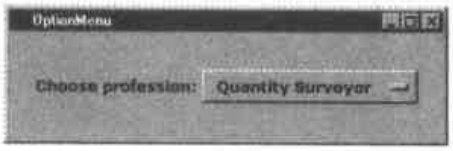
方法

message(options…)
options 决定了 message 方法的确切行为。

cget(option)
返回配置选项的当前值。

configure(options…)
查询或修改控件的配置选项。如果没有指定 option，则返回一个描述消息所有变量选项的字典。如果指定 option 没有值，那么该命令返回一个名为 option 的字典。如果指定了一个或多个可选项值对，那么方法就按照当前的输入值修改给定的控件选项，在这种情况下，方法返回一个空的字符串。options 可以是消息可接收的任何值。

OptionMenu class（任选菜单类）



说明

这个类把一个带有相关菜单的菜单按钮实例化。这可以允许用户选择一个由值域指定的值，当前的值将存储在 Tkinter 变量当中，它的名字将由构造函数给定，同样它将作为任选菜单的标签

而显示。用户可以点击菜单按钮来显示一个包含有所有可选值的菜单，并从其中选择一个新值。一旦新的值被选中，它将存储到变量里，并在任选菜单上显示。通过设置变量的值同样可以改变当前值。

继承关系

继承自 MenuButton。

共享选项

无。

控件特有的选项

无。

方法

OptionMenu(master , variable , value , *values)

产生任选菜单的一个实例。master 是父控件，variable 是 Tkinter 变量标识，value 是缺省值，而 values 是一系列插入到任选菜单的菜单当中的值。

Pack geometry manager（包几何管理器）

说明

pack 方法用来和 Packer 进行通信，而后者是一个几何管理器，可以按顺序在父控件边上排列它的子控件。

继承关系

没有继承，pack 没有从任何地方进行继承。

共享选项

没有。

pack 特有的选项

选　项	说　明	单位	典型值	缺省值
after	值必须是另外一个窗口。使用它的主控件作为从控件的主控件，并且恰好在包顺序中的 other 之后插入从控件	widger	label	
before	值必须是另外一个窗口。使用它的主控件作为从控件的主控件，并且恰好在包顺序中的 other 之前插入从控件	widger	self.entry	
expand	指定从控件是否应当消耗额外的空间在它们的主控件中扩展，布尔值可以是任意合适的值，例如 1 或者 NO	boolean	YES	0

（续）

选　项	说　明	单位	典型值	缺省值
Fill	如果从控件的包人于它所要求的空间，那么这个选项可以用来伸展从控件	constant	X 'both'	NONE
in_	在由 value 所给定的主控件窗口包序列的最后插入从控件	widget	container	parent
ipadx	数量指定了在从控件的每一边留下多少水平的内部填充。数量必须是一个有效的屏幕距离，例如 2 或者.5c	distance	2	0
ipady	数量指定了在从控件的每一边留下多少垂直的内部填充	distance	1m	0
padx	数量指定了在从控件的每一边留下多少水平的外部填充	distance	3	0
pady	数量指定了在从控件的每一边留下多少垂直的外部填充	distance	'2m'	0
Side	指定主控件的哪一边将被从控件捆绑，必须是 LEFT、RIGHT、TOP 或者 BOTTOM	constan t	LEFT 'top'	TOP

方法

pack(option = value , …)
参数包含怎样管理从控件的参数对

pack_forget()
从主控件的包序列中去除 self 并且取消窗口映射。从控件不再由 Packer 管理。

pack_info()
返回一个字典。它的元素是在相同的选项值格式里 self 当前的配置状态，其中格式指定给 pack_configure。

pack_propagate(flag = noarg_)
如果 flag 的布尔值为真，例如 1 或者 ON，那么传送将对 self 有效。如果 flag 的布尔值为假，那么传送将对主控件无效。如果省略了 flag，那么命令返回 FALSE 或者 TRUE 来指示当前传送是否对主控件有效。缺省状态下，传送是有效的。

pack_slaves()
返回主控件包序列中所有从控件的 ID 列表。其中的从控件的顺序和它们的包顺序是相同的。如果主控件没有从控件，那么返回 None。

PhotoImage class（照片影像类）

说明

照片是像素可以显示为任意颜色或者透明的图像。照片影像在内部存储为真彩色(每个像素 24bits)，在需要时可以使用抖动来进行显示。照片影像的数据可以从文件或者字

符串当中获得，而且还可以通过程序的接口从 C 语言的代码获得。目前，可支持的图像格式仅有 GIF 和 PPM/PGM，但是有一个接口，可以允许其他图像文件格式被很容易地加入。照片影像在没有提供图像数据的区域显示为透明。

继承关系

PhotoImage class 继承自 Image。

共享选项

选项	缺省值
width	被请求的宽度
height	被请求的高度

PhotoImage class 的特有选项

选项	说　明	单位	典型值	缺省值
data	把图像的内容指定为格式，必须是图像文件格式句柄能够接受的格式之一（目前是 GIF 格式）。如果 data 和 file 选项同时被指定，那么 file 选项优先级更高	string		
file	filename 给定了提供图像数据的待读取的文件名称。文件的格式必须是图像文件格式句柄能够读取的格式之一（目前是 GIF、PGM 和 PPM 格式）	string	"icon.gif"	
format	说明了数据对应的文件格式的名称，数据由 data 或者 file 选项指定	string		
gamma	指定分配给当前窗口进行显示图像的颜色应当根据 gamma 指数值对应的非线性显示来修改（在比较好的近似情况下，大多数 CRT 产生的密度为输入值的平方，gamma 为指数项，通常取为 2）。指定的值必须大于 0，缺省值为 1，表示没有修改。通常情况，当值大于 1 表示图像将会更亮，而小于 1 表示图像将会变暗	float	1.2	1.0
palette	指定显示图像的彩色的分辨率和色彩映射所包含的颜色的数目。palette-spec 既可以是一个单独的十进制数用来指定灰度值，也可以是三个用斜杠（/）分隔的十进制数，分别指定红蓝绿的值。如果采用第一种方法，将以单色来显示图像	integer 或 string	'255/220/125'	依赖系统

方法

PhotoImage(option…)
使用可选项当中的选项生成影像实例。

blank()
使图像变为空白，也就是把整个图像设置为没有值，因此显示为透明的，并且显示的任何窗口背景可见。

cget(option)

返回由 option 指定的当前配置值，option 可以使用影像构造函数所能接收的任意值。

configure(option=value…)

查询或者修改当前图像所需要的属性。如果没有一个 option 被指定，则返回一个描述图像的所有选项和值的字典。如果某个 option 没有被指定值，则返回描述某 option 的值的字典此同没有选项时返回的子列值一样)。如果有一个或多个选项-值对被指定，则此方法修改指定的字体使具有指定的值；此时返回空串。option 可以有照片影像构造函数认可的任何格式。

copy()

拷贝当前图像。Tkinter 方法简单化了 Tk 命令，允许在图像内部一个区域进行拷贝。

get(x,y)

返回坐标（x,y）处图像颜色的三值元组，值在 0 到 255 之间，对应红、绿、蓝。

height()

返回图像高度的像素个数的整数。

put(data,to=None)

设置图像中像素为 data 中给定的颜色。data 为拷贝图像像素的二维数组。data 被结构化为一个水平行列表，从顶到底，每一值为一个颜色列表，从左到右列出颜色。每个颜色可以是名字或十六进制格式（如#2376af)。to 选项用来指定将影响的边界方框。如果元组只有 x1 和 y1，则影响范围为(x1,y1)开始到 data 的大小。如果四个坐标都给定，它指定了对角矩形内的区域，data 中的数据拷贝填充矩形。

subsample(x,y=None)

通过只使用 x 方向的第 x 像素和 y 方向的第 y 像素减小图像大小。负值将引起 y 或 x 坐标倒转，相应地，如果 y 没有给定，缺省值同 x。

type()

返回图像类型字符串（当图像创建时，创建了图像类型参数）

width()

返回给定图像宽度的像素个数。

write(filename,options…)

把图像数据从图像写到名为 filename 的文件。下面的 options 可以给出：

选项	单位	说　明
format	字符串	给出写数据到文件时的图像文件格式句柄字符串。特别地，子命令查找名字匹配 format-name 的初始子串的第一个句柄，且该句柄有能力写一个图像文件。如果选项没有给定，此子命令使用第一个可写图像文件的句柄
from_coords	元组	指定图像 imageName 的一个矩形区域写到图像文件。如果只有 x1 和 y1 给定，区域从(x1,y1)到图像右底角。如果四个坐标都给定，则为对角矩形区域内部分。缺省情况下，如果选项没有给出，为整个图像

zoom(x,y=None)

在 x 方向和 y 方向分别按因子 x 和 y 放大。如果 y 没有给定，缺省值同 x。通过此选项，源图像中每个像素都将成一个 x×y 的新图像方块，所有点同色。x，y 必须大于 0。

Place geometry manager（放置几何管理器）

说明

放置器（Placer）是 Tk 的几何管理器。它提供对窗口的简单固定位置，其中你可以设置一个从窗口在主窗口中的精确大小和位置。同样它还提供了橡皮布局，其中你可以根据主窗口的尺寸来设置从窗口的大小和尺寸，因此从窗口会依据主窗口尺寸的变化来对其大小和位置做出相应的变化。最后，Placer 还允许你混合使用这些放置样式，举例来说，你可以使一个从窗口具有固定的宽度和高度，但是又在主窗口的中心位置。

继承关系

没有继承关系。

共享选项

无。

Place 特有选项

选项	说　明	单位	典型值	默认值
anchor	用来设定由 x，y，relx 和 rely 选项所定的（x，y）位置定位窗口中哪一个点。锚点的选取依据窗口的外部区域，也包括边界（如果有的话）。因此，如果 where 取值 SE，则窗口边框的右下角将位于主窗口给定的（x，y）处	常数	N SE	NW
bordermode	mode 决定了主窗口内放置从窗口时边界的利用程度。默认值和多数取值都是 INSIDE。这种情况下，Placer 认为主窗口的区域为该窗口的最里面的区域，在任何边框的里面；选项 x 取 0 对应于坐标 x 正好位于边框内，而选项 relwidth 取 1.0 表示窗口将占满主窗口边框内的区域。如果 mode 取 OUTSIDE，则 Placer 认为主窗口的区域包括其边框；这一模式通常是用来把窗口放置到主窗口的外面，这和选项 x 取 0、y 取 0、anchor 取 NE 的效果一样。最后，mode 还可以取 IGNORE，此时边框被忽略了；主窗口的区域被取为其正式的 X 区域，此时包括内部边界但没有外部边界。ignore 的 bordermode 也许并不十分有用	常数	OUTSIDE	INSIDE
height	该值以屏幕为单位设定了窗口的高度。height 为窗口的外部尺寸，如果有的话，也包括边框。如果没有设定 height 或者 relheight 选项，则窗口内部要求的高度值将被采用	整数	134	natural size

（续）

选项	说　明	单位	典型值	默认值
in_	取值设定了相对于哪个窗口放置的窗口标识。主窗口必须是父窗口或者父窗口的子代。此外，主窗口和窗口都必须是同一个顶层窗口的子代。这些限制是保证主窗口可视的情况下，窗口可视所必需的	widget	fred	parent
relheight	取值设定了窗口的高度。这种情况下，高度的取值为浮点数，表示与主窗口高度的相对值：0.5 表示窗口的高度为主窗口的一半，1.0 表示窗口的高度和主窗口的相同，如此等等。如果同时对一个从窗口设定了 height 和 reheight，它们的取值将被加起来。例如，relheight 取 1.0，height 取 2，则从窗口将比主窗口缩短 2 个像素	浮点数	0.45	1.0
relwidth	取值设定了窗口的宽度。这种情况下，宽度的取值为浮点数，表示与主窗口宽度的相对值：0.5 表示窗口的宽度为主窗口的一半，1.0 表示窗口的宽度和主窗口的相同，如此等等。如果同时对一个从窗口设定了 width 和 relwidth，它们的取值将被加起来。例如，relwidth 取 1.0，width 取 5，则从窗口将比主窗口宽 5 个像素	浮点数	0.5	1.0
relx	location 设定了在主窗口内，窗口的锚点的 x 坐标。这种情况下，位置的取值以一种相对的形式取为浮点数：0.0 对应于主窗口的左边界，1.0 对应于主窗口的右边界。location 并不一定要取在 0.0 到 1.0 以内。如果同时对一个从窗口设定了 x 和 relx，它们的取值将被累加起来。例如，relx 取 0.5，x 取 2，则从窗口的左边界将位于主窗口中心位置向左 2 个像素的位置	浮点数	0.66	0.0
rely	location 设定了在主窗口内，窗口的锚点的 y 坐标。这种情况下，位置的取值以一种相对的形式取为浮点数：0.0 对应于主窗口的顶边界，1.0 对应于主窗口的底边界。location 并不一定要取在 0.0 到 1.0 以内。如果同时对一个从窗口设定了 y 和 rely，它们的取值将被累加起来。例如，relx 取 0.5，x 取 3，则从窗口的顶边界将位于主窗口中心位置向下 3 个像素的位置	浮点数	0.34	0.0
x	location 设定了在主窗口内，窗口的锚点的 x 坐标。按屏幕单位取值，并且不一定要位于主窗口的范围内	整形数	105	0
width	size 设定了以屏幕为单位窗口的宽度（也就是说，可以是 Tk（Tk_GetPixels）所认可的任何一种形式）。width 将取为窗口的外围宽度，包括边界（如果有的话）。如果 size 为空，或者是没有对 width 或者 relwidth 选项设定，则窗口内部要求的宽度值将被采用	整形数	125	natural width
y	location 设定了在主窗口内，窗口的锚点的 y 坐标。其取值以屏幕为单位，并且不一定要位于主窗口的范围内	整形数	88	0

方法

place(option=value,…)

参数可以是一个或者多个选项-值对，它们用来设定自身几何管理的方式。如果 Placer

已经在管理 self 了，则选项-值对修改其窗口的设置。这种形式中，place 方法返回 None 作为结果。

place_forget()

place_forget 方法使得 Placer 停止管理窗口几何。该方法的副作用是 self 将不被映射，于是它就不出现在屏幕上。如果当前 Placer 并没有管理 self，则该方法不产生任何效果。

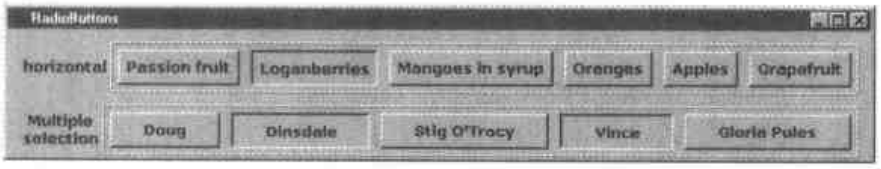
place_info()

place_info 方法返回一个字典，用来给出窗口的当前配置。该字典由选项-值对构成，并且和 place 方法所设定的形式完全相同。如果窗口的配置同 place_info 一起被检索，则该配置可以在下一次使用 place_forget 抹去窗口的现存信息，然后以保存下来的信息为参数调用 place 时恢复。

place_slaves()

place_slaves 方法返回一个以 self 为主窗口的所有从窗口的列表。如果 self 没有从窗口，则返回 None。

Radiobutton（单选按钮）



说明

Radiobutton 类定义了一个新的窗口，并且创建了一个单选按钮控件的事例。额外的选项（下面就会谈到）可以在方法调用中或者在选项数据库中设定，用来对单选按钮进行配置，例如颜色、字体、文本还有初始外形。Radiobutton 方法返回新控件的标识。在这条命令被调用的时候，单选按钮的父按钮必须存在。

单选按钮是一种控件，用来显示一个文本字符串、位图或者图像，以及一个菱形或者圈（称作指示器）。如果显示的是文本，则字体必须全都一样，但是可以在屏幕上占据多行（比如有换行，或者由于 wrapLength 选项而出现了回行，并且其中的一个字符可以通过 underline 选项加上下划线。

单选按钮具有简单按钮的全部功能：它可以显示为凸起、凹陷或者平铺；可以闪烁；并且它还可以在鼠标按钮 1 在点击按钮上点击时调用 Tcl 命令。此外，单选按钮是可以选择的。

如果单选按钮被选中，指示器通常绘制成所选的形式，并且与单选按钮相关的一个 Tkinter 变量被设置为某一特定的值（通常取为 1）。在 Unix 下，指示器绘制为一个凹陷形状，并且取一种特殊的颜色。在 Windows 下，指示器绘制为一个内部有标志的圈。

如果单选按钮没被选中，那么指示器就被绘制成内部默认的形式，相关变量被设置为某一特定的值（通常为 0）。在 Unix 下，指示器绘制为一个凸出来的形状，没有特殊的颜色。在 Windows 下，指示器内没有圆圈标志。

一般地说，几个单选按钮共用一个变量，并且变量的值表示哪一个单选按钮被选择。当一个单选按钮被选定时，它将设置变量的值来表明这一事实；每一个单选按钮都对变量的值进行监控，并且当变量改变时，自动地对自身选定或者解除选定。

配置选项也可以用来修改指示器显示的方式（或者是否需要显示）。默认的情况下，单选按钮被配置为在点击按钮时选择自身。

继承关系

继承自 Widget。

共享选项

选项	缺省值
activebackgroundr	SystemButtonFace
activeforeground	SystemWindowText
anchor	center
background(bg)	SystemButtonFace
bitmap	
borderwidth(bd)	2
command	
cursor	
disabledforeground	SystemDisabledText
font	(('MS', 'Sans', 'Serif'), '8')
foreground(fg)	SystemButtonText
height	0
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	1
image	
justify	center
padx	1
pady	1
relief	flat
state	normal
takefocus	
text	
textvariable	
underline	-1
width	0
wraplength	0

Rodiobutton 特有选项

选项	说　明	单位	典型值	默认值
indicatoron	设定指示器是否需要绘制。必须是一个正确的布尔数。如果取为 FALSE，则外形选项将被忽略，并且如果控件被选择时，控件的外形为凹陷；否则，为凸起	布尔数	0 TRUE	1
selectcolor	设定控件（通常是一个复选或者单选按钮）被选择时所选用的背景颜色。如果 indicatoron 取 TRUE，则该颜色对指示器生效。在 Windows 下，这个颜色将被作为指示器的背景，而与选取状态无关。如果 indicatoron 取 FALSE，则不论什么时候选择控件，这个颜色都将被作为整个控件的背景，其作用代替了 background 和 activeBackground。如果设定为空字符串，则在控件被选择时，不会显示任何特定的颜色	color	"red"	System -Window
selectimage	设定控件（通常是一个复选按钮）被选择时所显示的图像（代替了 image 选项）。除非 image 选项被设定了，否则该选项将被忽略	image	"redcross"	
value	设定一旦该按钮被选择时保存在与该按钮相连的 Tkinter 变量的值	string	0 "Power"	
variable	设定包含 content 的 Tkinter 变量名，并且设置控件的内容	variable	myVariable	selectedButton

方法

deselect()

取消对单选按钮的选择，并且设置相应的变量为空字符串。如果该单选按钮当前并没有被选择，则此方法无作用。

flash()

使单选按钮闪烁。这是通过重复显示单选按钮多次来完成的，并且在激活和正常颜色之间切换。在闪烁完毕之后，单选按钮回到与该方法调用之前相同的正常/激活状态。如果单选按钮的状态无效了，则此方法将被忽略。

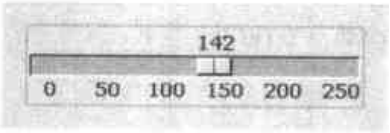
invoke()

一旦用户通过鼠标激活了单选按钮，则如同预期的那样操作：选择按钮并且调用与之相关的回调（如果有的话）。返回值是回调的返回值，如果没有与单选按钮相关的回调的话，则返回空字符串。如果单选按钮的状态无效了，则此方法将被忽略。

select()

选择单选按钮，并且设置关联变量为对应于该控件的值。

Scale（标尺）



说明

Scale 类定义了一个新的窗口，并且创建了一个标尺控件的实例。额外的选项（下面就会谈到）可以在方法调用中或者在选项数据库中设定，用来对标尺进行配置，例如颜色、取向还有外形。标尺方法返回新控件的标识。在这条命令被调用的时候，标尺的父体必须存在。

标尺是一种控件，显示为一个矩形槽和一个小的滑动块。矩形槽对应了一段实数值（由 from，to 和 resolution 选项决定），并且滑动块的位置选取一个特定的实数值。滑动块的位置（也就是标尺的值）可以由鼠标或者键盘来调整。一旦标尺的值改变了，则回调将被调用（使用 command 选项），来通知其他相关的控件做变化。此外，标尺的取值可以链接到一个 Tkinter 变量中（使用 variable 选项），这样其中任何一个的变化都会影响另一个。

在标尺控件中还可以显示三种批注：控件右上角的一个标签（对于水平标尺则在左上角），滑动块左边的一个数字，当前显示数字左边的一串数字刻度线（对于水平标尺则在矩形槽的下方）。通过配置选项可以对这三种批注的每一种启用或者禁止。

继承关系

Scale 继承自 Widget。

共享选项

选项（别名）	缺省值
activebackgroundr	SystemButtonFace
background(bg)	SystemButtonFace
borderwidth(bd)	2
command	
cursor	
font	(('MS', 'Sans', 'Serif'), '8')
foreground(fg)	SystemButtonText
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	2
relief	flat
state	normal
takefocus	
width	15

Scale 特有选项

选项	说　明	单位	典型值	默认值
bigincrement	一些与标尺控件的交互作用使得其值改变很“大”，该选项就是设定了这个大增加量的大小。如果设置为 0，则大增加量默认为标尺范围的 1/10	整形数	60	0
digits	用于设定标尺控件中的数值在转换为字符串时需要保留多少位有效数字的一个整数。如果该数小于或者等于 0，则标尺会选择最小的位数以保证每一个可能的滑动块的位置所显示的数字不同	整形数	2	0
from_	标尺控件的左端或者顶端所对应的数字	浮点数	0.0	0
label	作为标尺控件标签所显示的字符串。对于垂直标尺，标签将显示在标尺顶端的右边。对于水平标尺，标签将显示在标尺左端的上方。如果该选项设置为一个空字符串，则不显示任何标签	string	Power Level	
length	以屏幕为单位设置所需要的标尺长度（也就是说，winfo.pixels 所认可的任何一种形式）。对于垂直标尺，即为标尺的高度；对于水平标尺，即为标尺的宽度	distance	150 li	100
orient	对于那些既可以水平取向，又可以垂直取向的控件，例如滚动条，则该选项对采用哪一个取向进行设置。取值必须是 HORIZONTAL 和 VERTICAL 中的一个，或者它们之一的缩写	常数	VERTICAL "vertical"	vertical
repeatdelay	设置在自动重复之前按钮或者键盘所需保持的毫秒数。例如，用在滚动条中的向上和向下箭头	整形数	300	300
repeatinterval	与 repeatdelay 联合使用，一旦开始自动重复之后，此选项就用来决定两个自动重复之间的毫秒数	整形数	100	100
resolution	一个实数，用来设置标尺控件的分辨率。如果该值大于零，则标尺的取值将是该值偶数倍的四舍五入，标尺的标尺线和端点也一样。如果该值小于零，则不进行舍入。默认值为 1（也就是说取值为整数）	浮点数	2.0 10.0	1
showvalue	设置一个布尔数，用来指示当前标尺控件的取值是否需要显示	布尔数	TRUE 0	1
sliderlength	设置滑动块的尺寸，以屏幕为单位沿滑动块的长边测量。该值可以设置为 winfo.pixels 所认可的任何一种形式	distance	140 2i	30
sliderrelief	设置绘制滑动块时所用的外形，比如凸起还是凹陷			raised
tickinterval	必须取为实数。用来决定滑动块下边或者左边的数字标尺线的空间间隔。如果取为 0，则不显示标尺线	浮点数	0 5.0	0

（续）

选项	说　明	单位	典型值	默认值
to	设置对应于标尺右端或者下端的一个实数值。该值可以小于也可以大于 from 选项的值	浮点数	25.0 - 30.0	100
troughcolor	设置控件中矩形槽区域所采用的颜色，比如滚动调和标尺	color	'gray40'	System-Scroll-bar
variable	指定控件要设置内容的 Tkinter 变量名称，并设置控件内容	variable	myVariable	

方法

coords(value=None)

返回一个元组，其元素为矩形槽的中线上对应于 value 的点的 x 和 y 坐标。如果没有 value，则标尺的当前值将被采用。

get()

返回标尺的当前值。

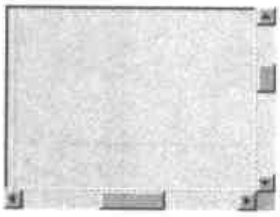
identify(x, y)

返回一个字符串，用来表明标尺的哪一个部分位于 x 和 y 给定的坐标下。返回 SLIDER 表示给定点位于滑动块上面；TROUGH1 表示给定点位于滑动块部分的上边或者左边；TROUGH2 表示给定点位于滑动块部分的下边或者右边。如果给定点不在这些区域上，则返回一个空字符串。

set(value)

调用此方法可以改变标尺的当前值，也就是滑动块显示的位置。value 给出标尺的新取值。如果标尺禁用，则该方法不产生任何效果。

Scrollbar（滚动条）



说明

Scrollbar 类定义了一个新的窗口，并且创建了一个滚动条控件的实例。额外的选项（下面就会谈到）可以在方法调用中或者在选项数据库中设定，用来对滚动条进行配置，例如颜色、取向、以及外形。scrollbar 方法返回新控件的标识。在这条命令被调用的时候，滚动条的父滚动条必须存在。

滚动条是一种控件，显示为两个箭头，一个在滚动条的一边，并且在滚动条的中间部分有一个滑动块。它提供了关联窗口中的显示信息，该窗口用于显示一些类型的文件（例如编辑文档或者是绘图）。滑动块的位置和大小表明了关联窗口中的哪一部分文档是可视的。

例如，在一个垂直滚动条中，如果滑动块占据了两个箭头之间上面的三分之一区域，那就是说关联窗口只显示文档上面的三分之一。滚动条可以通过鼠标的点击和拖拉来调整关联窗口中的视图。

继承关系

继承自 Widget。

共享选项

选项（别名）	缺省值
activebackground	SystemButtonFace
background(bg)	SystemButtonFace
borderwidth(bd)	0
command	
cursor	
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	0
relief	sunken
takefocus	
width	16

Scrollbar 特有选项

选　项	说　明	单位	典型值	默认值
activerelief	设置显示激活元素时采用的外形（如果有的话）。对于非激活元素总是显示凸起形状	常数	SUNKEN	raised
elementborderwidth	设置绘制滚动条内部元素（两个箭头和一个滑动块）边框时的宽度。取值可以是 winfo.pixels 所认可的任何一种形式。如果该值小于零，则使用选项 borderwidth 的取值	distance	10 1m	-1
jump	对于那些可以通过拖拉滚动条来调整取值的控件，例如滚动条，该选项决定什么时候对取值的改变做指示。其取值必须是一个布尔数，并且是 Tcl_GetBoolean 认可的形式。如果取值为 false，则在滑动块拖动的时候进行连续的刷新。如果取值为 true，则只在释放鼠标按钮来结束拖动时才刷新，此时做出一个简单的指示（取值是“跳动的”而不是连续的改变）	布尔数	TRUE NO	0
orient	对于那些既可以水平取向，又可以垂直取向的控件，例如滚动条，则该选项对采用哪一个取向进行设置。取值必须是 HORIZONTAL、VERTICAL 中的一个，或者它们之一的缩写	coastrant	VERTICAL "vertical"	vertical

（续）

选　项	说　明	单位	典型值	默认值
repeatdelay	设置在自动重复之前按钮或者键盘所需保持的毫秒数。例如，用在滚动条中的向上和向下箭头	integer	300	300
repeatinterval	与 repeatdelay 联合使用，一旦开始自动重复之后，此选项就用来决定两个自动重复之间的毫秒数	integer	100	100
troughcolor	设置控件中矩形槽区域所采用的颜色，比如滚动调和刻度	Color	'gray40'	System Scrollbar

方法

activate(element)

标记 element 所指示的元素为激活状态，这使得该元素以 activeBackground 和 activeRelief 选项所设置的方式来显示。该方法所唯一能识别的元素值只有 ARROW1，SLIDER 和 ARROW2。如果设置为其他值，则滚动条中没有任何一个元素激活。如果没有设置 element，则该方法返回当前处于激活状态的元素名，如果没有元素处于激活状态，则返回空字符串。

delta(deltaX, deltaY)

返回一个实数，来表示相对于滑动块位置给定的改变，滚动条设置改变的百分比数。例如，如果滚动条是水平的，则返回的结果表明滑动块向右移动 deltaX 个像素时，滚动条设置需要改变多少（这种情况下，deltaY 被忽略了）。如果滚动条是垂直的，则返回的结果表明滑动块向下移动时，滚动条设置需要改变的量。参数和返回值可以是零或者负数。

fraction(x, y)

fraction 为一个 0 到 1 之间的实数。控件必须调整其视图，使得 fraction 所给定的点出现在控件视窗的起始位置。fraction 取 0 表示文档的开头。1.0 表示文档的结尾处，0.333 表示整个文档的三分之一处，如此等等。

get()

以列表的格式返回滚动条设置，列表的元素即为最近的 set 控件方法的参数。

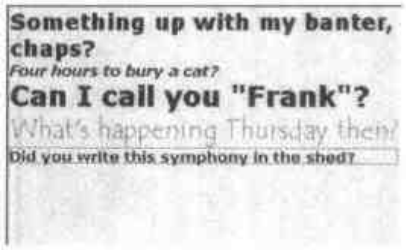
identify(x, y)

返回位于 x 和 y 所给定的点下方的元素名（例如 ARROW1），如果该点没有位于任何滚动条元素的上方，则返回空字符串。x 和 y 必须是和滚动条控件相连的像素坐标。

set(first, last)

该方法由滚动条的关联控件调用，用来通知滚动条该控件当前的视图。该方法具有两个参数，每一个都是 0 到 1 之间的小数。这两个小数表示在关联控件中文档的显示范围。例如，如果 first 是 0.2，last 是 0.4，则就是说在窗口中文档的显示是由文档的 20% 处开始到文档的 40%处结束。

Text（文本）



说明

Text 类定义了一个新的窗口，并且创建了一个文本控件的实例。其余选项（下面就会谈到）可以在方法调用中或者在选项数据库中设定，用来对文本进行配置，例如它默认的背景颜色外形。text 方法返回新窗口的路径名。

文本控件显示文本的一行或者多行，并且允许编辑文本。文本控件支持文本上的四种不同的批注：标签、标记、内嵌窗口以及内嵌图片。标签允许文本的不同部分显示为不同的字体和颜色。此外，Tcl 命令可以和标签相连，这样在一些特定的操作（例如按键盘键和鼠标按钮）在文本的特定的范围内发生时，就会调用脚本。

第二种批注由标记构成，它们是些文本中的浮动标号。标记用于在编辑文本时，跟踪不同的感兴趣位置。

第三种形式的批注使得任何窗口都可以嵌套在文本控件中。

第四种形式的批注则使得 Tk 图像可以嵌套在文本控件中。

许多针对文本的控件命令取一个或者多个指标（indices）作为参数。指标是一个字符串，用来指示文本内一个特定的位置，例如插入字符的位置，或者是一段要删除字符的一个端点位置。

继承关系

继承自 Widget。

共享选项

选项（别名）	缺省值
background(bg)	SystemWindow
borderwidth(bd)	2
cursor	xterm
disabledforeground	SystemDisabledText
font	(('MS', 'Sans', 'Serif'), '8')
foreground(fg)	SystemButtonText
height	24
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	0
padx	1
pady	1
relief	sunken
selectackground	SystemHighlight
selectborderwidth	0
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	0

（续）

选项（别名）	缺省值
padx	1
pady	1
relief	sunken
selectackground	SystemHighlight
selectborderwidth	0
selectforeground	systemHighlightText
state	normal
takefocus	
width	80
xscrollcommand	

Text 特有选项

选　项	说　明	单位	典型值	默认值
exportselection	设置控件中的一个选择是否也必须是 X 选择。取值可以是任何一种 Tcl_GetBoolean 所认可的形式，例如 true、false、0、1、yes 或者 no。如果选择被导出（exported），则控件内的选择将解除当前 X 选择的选择，控件外的选择将解除任何的控件选择，并且控件将在其具有一个选择时，对选择恢复请求做出回应。默认值通常是控件导出选择	boolean	0 YES	1
insertbackground	设置插入光标所在区域采用的背景颜色。该颜色通常会覆盖控件的正常背景颜色，或者选择的背景颜色（如果插入光标不小心到了选择区中）	color	yellow	SystemWindowText
insertborderwidth	设置一个非负数，用来表示在插入光标周围绘制的三维边界的宽度。取值可以是 Tkinter（Tk_GetPixels）所认可的任何一种形式	pixel	2	0
insertofftime	设置一个非负整数，用来表示一个闪烁循环中插入光标处于“灭”的毫秒数。如果该选项取为零，则插入光标将不再闪烁：所有的时候都是“亮”的	整形数	250	300
insertontime	设置一个非负整数，用来表示一个闪烁循环中插入光标处于“亮”的毫秒数	整形数	175	600
insertwidth	设置一个数值，用来表示在插入光标的宽度。取值可以是 Tkinter（Tk_GetPixels）所认可的任何一种形式。如果插入光标已经指定边界（通过 insertborderwidth），边界将画在 insertwidth 所给宽度内	pixel	2	2

（续）

选 项	说 明	单位	典型值	默认值
setgrid	指定一个布尔量，决定控件是否控制顶层窗口网格尺寸。此选项常用于文本控件中，这里控件中的信息有个自然尺寸（一个字符的大小），且它使窗口的大小为这个单元的整数倍，这些自然窗口尺寸组成网格，如果 setgGrid 选项设置为 true，则控件将与窗口管理器进行通信，以便在当用户交互地调整包含控件的顶层窗口时，窗口将按网格单位显示于用户，窗口尺寸为网格单位的整数倍。详细参看“网格几何管理器”部分	boolean	NO 1	0
spacing1	在控件的每行文本上面请求多余空间，使用屏幕距离的任何标准格式，如果一行发生了换行，仅用于第一行。此选项可以用标签中的 spacing1 选项取消	distance	2m 15	0
spacing2	对换行的那些文字行（这样屏幕上可以多行显示）此选项指定多余空间用在代表单行文本显示行之间,使用屏幕距离的任何标准格式，如果一行发生了换行，仅用于第一行。此选项可以用标签中的 spacing1 选项取消	distance	1m 3	0
spacing3	在每行控件的文本下面请求多余空间，使用屏幕距离的任何标准格式，如果一行发生了换行，仅用了最后一行。此选项可以用标签中的 spacing3 选项取消	distance	3m 14	0
tabs	为窗口指定一系列的制表点。选项值为一个屏幕距离列表，给出制表点位置。每个点能可选地跟着下面元素中的一个，这些值是 LEFT, RIGHT, CENTER 或 NUMERIC。这些值设定如何相对制表调整文本。LEFT 是缺省值，它使得制表符后的文本左边缘放在制表点。RIGHT 指制表符后的文本右边缘放在制表位置，CENTER 指文本居中，NUMERIC 指文本中的小数点放在制表点处。如果没有小数点，则数的最低有效位被放到制表点处。如果文本中没有数，则文本被右对齐到制表点处。如 tabs=(2c,left,4c,6c,center)按 2cm 间隔出创建 3 个制表点。前面两个使用左对齐，第三个居中。如果一个制表列表的没有足够的制表点对应下一个文本中的制表点，则 Tk 从最后一个制表点使用空格和对齐外插制表点。制表值选项可以通过标签中的-tabs 来取消。如果 tags 选项没有指定，或者指定为空串，则 Tk 使用缺省的制表符，间隔为 8 个空格（平均尺寸）	string		
wrap	设定如何控制单行文本窗口中太长的行的显示。值必须为 NONE 或 CHAR 或 WORD。换行模式为 NONE 表示每行文本恰恰显示在一行中，屏幕显示不下多余文字不显示。其他模式下，为使所有字符都能显示，每行都可以断为几行。CHAR 模式下，可以在任何字符处断开。WORD 模式下，仅仅在词边界上才能断开	constant	"char" NONE	char
yscrollcommand	指定一个与垂直滚动条通信的前缀，该选项与 xscrollcommand 的方法相同，只不过它用在垂直滚动条且由支持垂直滚动条的控件提供，选项详细情况参见 xscrollcommand 说明	function		

方法

bbox(index)

返回 index 给定字符屏幕区域的四个值的表。前两个为字符覆盖屏幕区域左上角的坐标（相对控件，采用像素单位），后两个为区域的宽和高，像素单位。如果字符在屏幕上只是部分可见，返回值只反映可见部分。如果字符在屏幕上不可见，返回值为空表。

compare(index1,op,index2)

根据关系 op 比较 index1 和 index2。如果满足关系 op，返回 true，否则 false。关系 op 必须是<,<=,==,>=,>,或!=。如果 op 是==，则当 index1 和 index2 为相同字符，返回真；如果 op 是<,则 index1 是 index2 前的字符时，返回 true。

debug(boolean=None)

如果 boolean 指定，它必须是 true 或 false 中的一个，按照 Tcl_GetBoolean 能接受的格式。如果 boolean 为 true，则对文本控件相关的 二义树代码的内部一致性检查将被开启。如果 boolean 为假，调试检查被关闭。两种情况下方法都返回布尔值，指示调试是否启用。如果 boolean 没有给出，方法返回 on 或 off 显示目前调试是否打开。所有文本控件有一个共享的调试开关：任何控件中开启或关闭它就对所有控件 on 或 off，有大量文本的控件，一致性检测可能导致速度大量降低。

delete(index1,index2 =None)

从文本中删除一定范围的字符。如果 index1 和 index2 都给定，从 index1 开始删除，在 index2 前一个字符停止（即不删除 index2）。如果 index2 指定的位置不在 index1 指定位置之后，则不删除字符。如果 index2 没有指定，则只删除一个在 index1 的字符。

对有换行的文本，不允许删除字符而使新行没有字符。返回 None。

dlineinfo(index)

返回一个五值元组，描述包含 index 的行所占的区域。前两个元素为所占区域左上角的 x 坐标和 y 坐标，第三个和第四个元素给出所占区域的宽和高，第五个元素给出行的基线位置，从上到下计算。所有这些信息都是按像素计算。

如果当前换行模式为 None 而行跨出窗口边界，返回区域为整个行的区域，如果行没有窗口宽，则仅返回字符和内嵌窗口所占区域。如果包含 index 的显示行不可见，返回空列表。

get(index1,index2=None)

从文本中返回一定范围的字符。返回值为文本中从 index1 开始，到 index2 前一个字符结束（即不包含 index2）的所有字符。如果 index2 没有指定则在 index1 的单个字符被返回。如果指定区域没有字符（如 index1 超出文件尾或 index2 小于或等于 index1），返回空串。如果给定范围包含内嵌窗口，返回字符串中不包含它们的信息。

image_cget(index,option)

返回内嵌图像的选项配置值。index 标识内嵌图像，option 标识特定选项。

image_configure(index,options…)

查询或者修改当前内嵌图像属性。如果没有一个 option 被指定，则返回一个描述在 index 的内嵌图像的所有选项的字典。如果 option 被指定没有值，则返回描述名为 option 的选项的字典（同没有选项时返回的子列表一样）。

如果有一个或多个选项-值对被指定，则此方法修改指定选项使具有指定的值。此时返回空串。

image_names()

返回当前所有内嵌窗口的名字列表。

index(index)

返回 line.char 表中 index 对应的位置。其中 line 是行号，char 是字符号。

insert(index,chars,tagList,chars,tagList…)

在 index 字符正前面加入所有 chars 参数。如果 index 指文本末尾（最后一个换行后的字符），则新字符插入到最后换行前。如果只有一个 chars 参数，没有 tagList，新文本将接收插入点前后两个字符上的标签。如果标签只出现在两个字符中的一个上，它将不能用于新文本。

如果 tagList 给定，它包含一个标签名字列表。新字符接收列表中所有标签，不接收别的，即使插入点周围有标签。如果多个 chars-tagList 给定，它们产生的效果，就像独立的插入控件方法依次对每个值对作用一样。最后一个 tagList 可以省去。

mark_gravity(markName,direction=None)

如果 direction 没有给定，返回 LEFT 或 RIGHT，指示邻近字符 markName 的哪一个被粘贴。如果 direction 给定，则必须是 LEFT 或 RIGHT，markName 的“重力”设为给定值。

mark_names()

返回当前所有内嵌窗口的名字列表。

mark_set(markName,index)

在 index 前面设置名为 markName 的标记。如果 markName 已经存在，则从旧位置去掉，如果不存在，产生新标记。

mark_unset(mark)

取消 mark 对应的标记。删除标记将不能在索引中使用，也不能被将来的调用为 mark_name 使用。返回 None。

scan_dragto(x,y)

该方法计算 x 和 y 坐标（通常是鼠标坐标）和上次 scan_mark 标记的地方的 x 和 y 的差距，然后调整视图为 10 倍距离大小。该方法通常与控件中的鼠标移动事件相关，产

生窗口中高速移动列表效果。返回值为空串。

scan_mark(x,y)

记录 x 和 y 及文本窗口当前视图，与上次 scan_dragto 调用相关。特别地，该命令把按下控件的鼠标按钮和鼠标的 x,y 联系起来。返回 None。

see(index)

调整窗口视图，以使 index 所给元素在窗口中可见。如果该元素已经可见，方法无作用，如果 index 略长于视图，该方法调整视图使 index 在窗口边界可见。如果 index 远超出视图，方法使 index 在窗口居中。

tag_add(tagName,index1,index2=None)

把标签 tagName 和所有以 index1 开头,index2 前一个字符（不包括 index2）结束的字符关联起来。单个命令中可以包含任何多个的 index1,index2 对，如果最后一个 index2 省去，单个字符 index1 被设标签。如果指定区域没有字符（如 index1 超出文件尾或 index2 小于或等于 index1），该方法无用。

tag_bind(tagName,sequence,function,add=None)

把 function 和 tagName 给定的标签关联起来，这样当 sequence 给定的事件序列对某个字符存在时，function 将被调用。

该方法同 bind 方法一样，只是它操作于文本中的字符，而不是整个控件。

如果所有参数给定，它产生一个新的捆绑，替代已经存在的任何 sequence 和 tagName 之间的捆绑（如果 function 的第一个符号是+，函数增加一个捆绑而不是替代它），此时，返回空串。如果 function 略去，方法返回 function 和 tagName 之间的捆绑（如果不存在这样的捆绑，出错）。

如果 function 和 sequence 都略去，方法返回所有 tagName 定义的捆绑的列表。

捆绑可以指定的所有事件是鼠标和键盘事件，比如 Enter,Leave,ButtonPress,Motion 和 Keypress，或虚拟事件。文本控件的事件捆绑使用 CURRENT 标记，当标签对当前字符首次出现时，激活 Enter 事件，当停止时，激活 Leave 事件。Enter 和 Leave 事件发生在要么 CURRENT 标记移动，要么该位置字符改变。

注意这些事件和窗口的 Enter 和 Leave 是不一样的。鼠标关联的事件定向到当前字符，如果有的话。键盘相关事件定位到激活字符。如果虚拟事件被用到捆绑中，该虚拟事件只有在被鼠标或键盘所定义的情况下才能用。可能当前字符有几个标签，当这存在时，每个标签调用一个捆绑。从最低优先级到最高优先级。

如果单个标签存在多个捆绑与之匹配，最特定的那个被调用。标签捆绑先被调用，接下来是一般捆绑。

tab_cget(tagName,option)

返回 tagName 给定标签相关的名为 option 的选项的当前值，option 可以是 tag_configure 接受的任何值。

tab_configure(tagName,options,…)

与 configure_widget 命令相同，只不过它是对标签 tagName 作用，而不是整个文本控件。如果没有一个 option 被指定，则返回一个描述 tagName 的所有选项和值的字典。如果 option 没有被指定值，则返回描述名为 option 的选项的字典(同没有选项时返回的子列表一样)。

如果有一个或多个选项-值对被指定，则此方法修改指定选项使具有指定的值。此时返回空串。

tagdelete(*tagNames)

删除每个 tagName 参数的标签信息。该方法把文件中的所有字符的标签除去，同时删除所有与标签相关的信息，如捆绑信息和显示信息。

tag_lower(tagName,belowThis=None)

改变 tagName 的优先级，让其正好低于 belowThis 标签。如果 belowThis 省去，方法使 tagName 的优先级低于所有标签。

tag_names(index=None)

返回在 index 给定字符处活跃的所有标签名字列表。如果省去 index，返回值描述所有文本的标签(包括在 tag 控件方法调用中命名但没有被 tag_delete 方法调用删去的标签，即使目前没有字符被标签标记)。标签名将按优先级从低到高排序。

tag_nextrange(tagName,index1,index2=None)

寻找标有 tagName 的字符范围，这些范围第一个字符不超前于在 index1 给定字符，又不在 index2 给定字符前一字符之后（index2 开始的范围不算在内）。如果有多个满足的范围存在，选择第一个。该方法返回值为一个二元素列表，为范围第一个字符索引和最后一个字符后一个索引。

如果没有满足的匹配范围，返回空串。如果 index2 没有给定，缺省为文本结尾。

Tag_prevrange(tagName,index1,index2=None)

寻找标有 tagName 的字符范围，这些范围第一个字符在 index1 给定字符之前，又不前于 index2 给定字符前一字符（index2 开始的范围不算在内）。如果有多个满足的范围存在，选择离 index1 最近的一个。该方法返回值为一个二元素列表，为范围第一个字符索引和最后一个字符后一个索引。

如果没有满足的匹配范围，返回空串。如果 index2 没有给定，缺省为文本开始。

tag_raise(tag Name,aboveThis=None)

改变 tagName 的优先级，让其正好高于 aboveThis 标签。如果 aboveThis 省去，方法使 tagName 的优先级高于所有标签。

Tag_range(tagName)

返回一个元组，元组值描述标有 tagName 的文本。前两个元素说明文本中的第一个标签范围，第二个元素说明第二个范围，等等。每对的第一个元素包含该范围第一个字符，第二个元素包含该范围最后一个字符之后的字符。如果没有字符有标签，返回空串。

tag_remove(tagName,index1,index2=None)

把所有以 index1 开头，index2 前一个字符（不包括 index2）结束的字符标签标签 Name 除去。单个命令中可以包含任何多个的 index1，index2 对，如果最后一个 index2 省去，单个字符 index1 被设标签。如果指定区域没有字符（如 index1 超出文件尾或 index2 小于或等于 index1），该方法无用。该方法返回 None。

tag_unbind(tagName,sequence,funcid=None)

除去 tagOrId 给定的所有项中事件 sequence 和事件句柄 funcid 之间的关联。如果提供 funcid，句柄将被破坏。

tk_textBackspage()

tk_textIndexCloser(a,b,c)

tk_textResetAnchor(index)

tk_textSelectTo(index)

这四个方法只有在你为文本写自己的事件处理程序时才有用。它们的功能是设置文本元素的状态如缺省行为发生过一样。它们在模拟用户与 GUI 交互时也有用。

window_cget(index,option)

返回内嵌窗口的配置选项值。index 标识内嵌窗口，option 指定特定配置选项。

window_configure(index,options…)

查询或者修改当前内嵌窗口属性。如果没有一个 option 被指定，则返回一个描述 index 内嵌窗口的所有选项和值的字典。如果某个 option 没有被指定值，则返回描述其选项的字典(同没有选项时返回的字典一样)。

如果有一个或多个选项-值对被指定，则此方法修改指定选项使具有指定的值。此时返回空串。

window_create(index,options…)

创建一个窗口注解，将在窗口中 index 位置显示，任何数量的选项-值对都可以用来配置注解。

windows_names()

返回当前窗口所有内嵌窗口名字列表。

xview_moveto(fraction)

调整窗口中的视图，使文本水平跨度的 fraction 部分处于窗口的左边界上。fraction 必须是 0 到 1 之间的一个分数。

xview_scroll(number,what)

根据 number 和 what 在窗口中左移或者右移视图。number 必须是一个整数。what 必须是 UNITS 或者 PAGES 或者两个中一个缩写。如果 what 为 UNITS，则视图将向左或者右调整 number 个平均宽度的字符；如果是 PAGES，则视图将调整 number 个全屏。如果 number 为负数，则左前方的字符就会被显示；如果是正数，则右后方的字符就会

被显示。

yview_moveto(fraction)

调整窗口中的视图，使 fraction 处字符出现在窗口的左边界上。fraction 必须是 0 到 1 之间的一个分数。0 代表第一个字符，0.33 代表三分之一处字符，等等。

yview_scroll(number,what)

根据 number 和 what 在窗口中上移或者下移视图。number 必须是一个整数。what 必须是 UNITS 或者 PAGES 或者两个中一个缩写。如果 what 为 UNITS，则视图将向上或者下调整 number 个平均宽度的字符；如果是 PAGES，则视图将调整 number 个全屏。如果 number 为负数，则上部字符就会被显示；如果是正数，则下部字符就会被显示。

yview_pickplace(index)

改变窗口中视图，使 index 可见。如果 pickplace 选项没有给定，则 index 显示在窗口顶部，如果 pickplace 给定，控件选择 index 在窗口中何处显示。如果 index 已经在窗口中某处可见，此方法无效。

如果 index 在窗口上方有几行跑到屏幕外边，那么 index 定位于窗口的顶部。如果 index 在窗口下方有几行跑到屏幕外边，那么 index 定位于窗口的底部。否则，index 定位于窗口的中心。

pickplace 选项因为 see 方法而过时了（see 方法方法处理 x 和 y 运动使一个位置可见，然而 pickplace 只处理 y 方向的运动）。

Toplevel（顶层）



说明

Toplevel 类定义了一个新的顶层控件（通过路径名参数给定）。其余选项（下面就会谈到）可以在方法调用中或者在选项数据库中设定，用来对文本进行配置，例如它默认的背景颜色，以及外形。toplevel 方法返回新窗口的路径名。

顶层和框架很相似，只不过它作为顶层窗口创建：它的 X 父体是屏幕的根窗口，而不是从它的路径名得到的逻辑父体。顶层的主要目的是作为对话框和其他一些控件集合的容器，顶层的唯一可见属性是背景颜色和可选的使顶层凸起或下沉的三维边界。

继承关系

Toplevel 继承自 Basewidget，Wm。

共享属性

选项（别名）	缺省值
background(bg)	SystemButtonFace
borderwidth(bd)	0
cursor	

（续）

选项（别名）	缺省值
height	0
highlightbackground	SystemButtonFace
highlightcolor	SystemWindowFrame
highlightthickness	0
relief	flat
takefocus	0
width	0

Toplevel 特有选项

选项	说　明	单位	典型值	缺省值
class	给窗口指定一个类。该类在查询窗口其他选项的选项数据库时使用，以后在其他如捆绑这样的功能上也会用到。采用 configure 方法不能修改 class 选项。注意，class 是一个保留字，tkinter 必须使用_class	class		Toplevel
colormap	指定窗口颜色映射。值可以为 NEW，此时为窗口和子窗口创建一个新颜色映射，或其他窗口名（必须在同一屏幕上和相同的路径画面），此时新窗口将从指定窗口中使用颜色映射。如果窗口使用和父窗口相同的颜色映射，此选项不能被配置方法改变	colormap	New mywindow	
container	该值必须是布尔值。如果为 TRUE，它意味着该窗口将用做容器，在这里一些应用为内嵌的（比如 Tkinter 顶层使用 use 选项内嵌）。窗口将对如几何请求之类的东西支持适当窗口管理协议。该应用中窗口不该拥有自己的子窗口。此选项不能被 configure 方法改变	boolean	TRUE 0	0
menu	指定与菜单按钮相关的菜单路径名。菜单必须是菜单按钮的子菜单	string	subMenuAction	
screen	指定放置新窗口的屏幕。任何有效屏幕名字都可以，甚至不同的显示中。缺省为父体的相同屏幕。此选项特殊之处在于它不能通过选项数据库指定，且不能通过配置方法修改	screen	"Default"	
use	指定当控件被选中时，存储到与控件相关的 Tkinter 变量中的值	string		
visual	按 winfo.visual 格式指定新窗口可视信息，如果选项没有指定，新窗口将使用与父体相同的视图，visual 选项不能通过 configure 方法修改	visual	monochrome	

方法

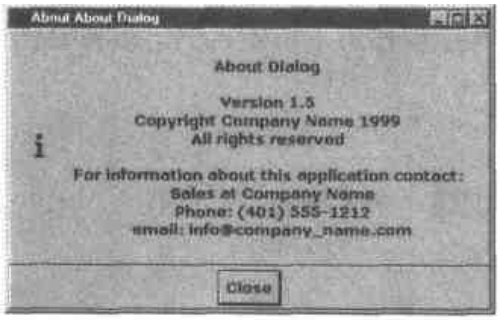
顶层有标准控件方法，如 configure。

附录 C　Pmw 参考：

Python megawidgets

本附录所述内容与附录 B 相比风格相近。利用 Python 语言运行 Pmw 包和 Pmw HTML 文件，用于生成大 ASCII 文件，其中包括头文件、文本和标签，用于导入 Frame Maker，本书由此方法生成。

AboutDialog（关于对话框）



说明

关于窗显示有应用，版权和联系的信息。

继承关系

AboutDialog 继承自 Pmw.MessageDialog。

AboutDialog 选项

选　项	说　明	单位	缺省值
activatecommand	如果该函数能被调用，那么只要当调用 activate()函数而激活了大控件时，就必须调用该函数	函数	无
applicationname	设置由 AboutDialog 显示应用的名称	字符串	无
borderx	表示消息区域从左边界到右边界的宽度	距离	20
bordery	表示消息区域从上边界到下边界的高度	距离	20
buttonboxpos	表示在对话框窗口的哪一侧放置按钮框。必须为 N,S,E,W 其中之一	锚点	S
buttons	表示按钮框中所有按钮的名称。必须为一元组或一列表	(字符串...)	'OK'
command	指向一调用函数，该函数在按钮框中某一按钮被击中时或在窗口管理器中删除该窗口时被调用。该函数被调用时，只有一个参数，其为被击中的按钮的名称，或在窗口管理器中删除该窗口时该参数为 None	函数	无
deactivatecommand	如果该函数能被调用，那么只要当调用 deactivate()函数而激活了大控件时，就必须调用该函数	函数	无

（续）

选　项	说　明	单位	缺省值
defaultbutton	表示按钮串中的缺省按钮。当当前高亮对话框中的 Return 键被击中时，缺省按钮将被调用。若 defaultbutton 为 None，则不存在缺省按钮，击中 Return 键无效	索引	0
iconmargin	如果图标存在，定义图标的左边位置	距离	20
iconpos	如果图标要显示，定出图标位置，值必是 E，S，W，N 或 None 之一	锚点	w
separatorwidth	若其大于 0，则依据指定宽度在按钮框和子位置间创建一被称作分隔线的窗口控件。如果按钮框的缺省边界和子位置被增大，则没有必要设置该选项，因为在按钮框与子位置间将存在一条可见的分隔线	距离	0
title	显示在窗口管理器的窗口标题栏上的标题	字符串	无

组件

buttonbox

包含对话框中的所有按钮。缺省时使用选项（hull_borderwidth=1,hull_relief='raised'）来创建。

dialogchildsite

表示对话框的子位置。可以通过在大控件中创建其他控件用于特定目的。缺省时使用选项（borderwidth=1,relief='raised'）来创建。

hull

表示整个大控件本体。其他控件可作为其子控件创建，来生成特定的控件。

icon

这是沿着 message 显示的图标。

message

显示在对话框的文本内容。

separator

如果 separatorwidth 不为 0，则分隔控件为分隔按钮框与子位置间的一条直线。

方法

除了从基类中继承的方法外，没有 About Dialog 方法。

函数

aboutversion(value)

设置由 AboutDialog 显示的版本为 value。

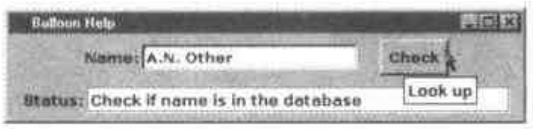
Aboutcopyright(value)

设置由 AboutDialog 显示的版权字符串为 value。

aboutcontact (value)

设置由 AbouDialog 显示的联系信息为 value。

Balloon（浮动图）



说明

这部分实现浮动图帮助系统，如果该系统存在，则可提供一机制，在状态位置提供相同（或不同）的信息。如果用户不需要该信息，可提供由用户关闭该信息的机制。

继承关系

Balloon 继承自 Pmw.MegaToplevel。

Ballon 选项

选　项	说　明	单位	缺省值
activatecommand	如果该函数能被调用，那么只要当调用 activate()函数而激活了大控件，就调用该函数	函数	无
deactivatecommand	如果该函数能被调用，那么只要当调用 deactivate()函数而激活了大控件，就调用该函数	函数	无
initwait	等待时间，以毫秒记，指示器点中控件后而浮动帮助出现之前的时无	微秒	500
state	决定出现浮动帮助还是状态信息，可为 none,ballon,status 之一或两个	常量	'both'
statuscommand	如果该函数能被调用，只要状态信息要被覆盖，就调用它，正常情况调用 Pmw.MessageBar's helpmess age 方法	函数	无
title	显示在窗口管理器的窗口标题栏上的标题	字符串	无
xoffset	浮动帮助控件的水平偏离，从相关的控件的边界箱的左下角开始	距离	20
yoffset	浮动帮助控件的垂直偏离，从相关的控件的边界箱的左下角开始	距离	1

组件

hull

表示整个大控件本体。可以创建作为其子体的其他特定控件。

label

如果 labelpos 为非 None，该控件将作为大控件的字符标签而被创建。参见 labelpos 说明。注意，例如在设置标签的字符选项时，用户必须使用 label_text 控件选项。

方法

bind (widget, balloonHelp, statusHelp=none)

增加 balloonHelp 到定义的小控件，如果 statusHelp 为空，balloonHelp 被用作状态信息。如果 statusHelp 被定义，则控件的状态位置被设为约束的信息。如果 balloonHelp 和 statusHelp 都空，则 bind(widget,None)等价于 unbind(widget)。

clearstatus()

移去存在的全部状态信息。

showstatus(statusHelp)

如果 statuscommend 被定义，则调用 statusHelp 作为它的内容。

tagbind (widget,tagOrItem, balloonHelp,statusHelp=None)

类似 bind，这种方法是把 balloonHelp 加到在 widget 中定义的 tagOrItem 项目。

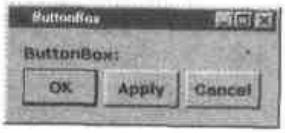
tagunbind(widget,tagOrItem)

移去全部存在于 widget 中的 tagOrId 约束。

unbind(widget)

移去 widget 中的所有的<Motion>、<Enter>、<Leave>和<ButtonPress>约束。

ButtonBox（按钮框）



说明

此类产生包含按钮的一个窗口管理器，其中的按钮可以由缺省定义，而缺省按钮被显示成平放的形式，按钮可纵排也可横排。

继承关系

ButtonBox 继承自 Pmw.MegaWidget。

ButtonBox 选项

选 项	说 明	单位	缺省值
labelmargin	若 labelpos 参数不是 None，该选项表示标签窗口控件与大控件其他控件间的距离	距离	0
labelpos	表示放置标签控件的位置。若不是 None，必须为 N,S,E 或 W 中一个或一个字母的组合。第一个字母表示在大控件的哪一侧放置标签。若第二个字母也被设置，则其表示在大控件该侧的何处放置标签。例如，若 labelpos 的值为 W，则表示标签放在左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端	锚点	无

（续）

选 项	说 明	单位	缺省值
orient	表示按钮放置的方向，可为 HORIZONTAL 或 VERTICAL	常量	HORIZONTAL
padx	表示每个按钮间，以及按钮与单选框外边界间在 x 方向上的间距	距离	3
pady	表示每个按钮间，以及按钮与单选框外边界间在 y 向上的间距	距离	3

组件

frame

若已创建了标签窗口控件（即 labelpos 不为 None），则也将创建一框架窗口，以作为通过 add()和 insert()创建的按钮的容器。如果不存在标签，则也不必创建框架，而用 hull 作为容器。

hull

表示整个大控件本体。其他窗口控件可作为其子控件创建，以用于特定目的。

label

如果 labelpos 为非 None，该控件将作为大控件的字符标签而被创建。参见 labelpos 说明。注意，例如在设置标签的字符选项时，用户必须使用 label_text 窗口控件选项。

方法

add(name,**kw)

在按钮框的末端插入一个控件名为 name 的按钮。生成该按钮时，任意的关键字输入都会传给构造函数。如果没给出 text 关键字，则该按钮的文本选项由缺省值 name 给出，该函数返回名为 name 的控件。

alignbuttons(when='later')

把所有按钮设为和最宽的按钮等宽。如果 when 为 later，则当编译器再次空闲时，做上述设置；否则立即重设。

delete(index)

删除按钮框中由 index 给出的按钮。index 接受 index()提供的任何形式。

index(index,forInsert=0)

返回与 index 有关的按钮的数值索引。有下列形式：

- Number 定义按钮个数，0 表示最左或最上按钮。
- End 表最右或最下按钮。
- Default 表当前缺省按钮。
- Name 定义按钮名，名为 name。

如果 forInsert 为真，end 返回按钮个数而不是最后按钮的索引。

insert(name,before=0,**kw)

在 before 定义的按钮之前增加一命名为 name 按钮，生成该按钮时，任意的关键字输入都会传给构造函数。befor 接受 index()提供的任何形式。为了在按钮框的最后增加一按钮，使用 add( )，该方法返回 name 命名的小控件。

invoke(index='default',noFlash=0)

执行与 index 定义的按钮有关的回调命令。除非 noFlash 是 ture，否则按钮的闪光提示用户有情况发生。index 可以有 index( )接受的任意形式。
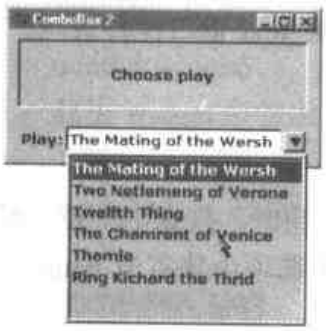
numbuttons()

返回按钮框中按钮个数。

setdefault（index）

设置缺省按钮到由 index 给出的按钮，这将引起已定义的按钮显示为缺省按钮的形式。如果 index 为空，不会有缺省按钮，index 可以有 index()方法接受的任意形式。

ComboBox（组合框）



说明

此类生成输入域和相关的滚动列表框。当选中列表框一项时，它会出现在列表框的输入域，用户也可以直接在输入域编辑。

对于简单的组合框，滚动列表框显示在输入域的下面。对于下拉组合框（缺省），当用户点击输入域右边的向下箭头按钮时，滚动列表框出现在输入域的下部，两种风格参见选项表。

继承关系

ComboBox 继承自 Pmw.MegaWidget。

ComboBox 选项

选　项	说　明	单位	缺省值
autoclear	如果 autoclear 和 history 是 true，增加值到历史列表中之后，按 Rreturn 键，则清空输入域	布尔	0
buttonaspect	表示箭头按钮的高宽比。其高度与输入域的高度相同	浮点	1.0
dropdown	定义组合框是下拉式还是简单的	布尔	1
fliparrow	如果选项为真，当列表框出现时，箭头按钮可被下拉。只能在下拉式组合窗控件中使用	布尔	0
history	如果 history 是 true 时，在输入域按 Return 键,输入域的当前值被附加到列表框中	布尔	1

（续）

选 项	说 明	单位	缺省值
labelmargin	若 labelpos 选项不是 None，该选项表示标签窗口控件与大控件其余控件的距离	距离	0
labelpos	表示放置标签控件的位置。若不是 None，必须为 N,S,E 或 W 中一个或二个字母的组合。第一个字母表示在大控件的哪一侧放置标签。若第二个字母也被设置，则其表示在大控件该侧的何处放置标签；例如，若 labelpos 的值为 W，则表示标签放在左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端	锚点	无
listheight	下拉列表框以像素表示的高度	高度	150
selectioncommand	当某项被选中时要调用的函数	函数	无
Pmw.unique	如果 unique 和 history 是 true，当该值已在列表框中存在时，输入域的当前值不再被加到列表框中	布尔	1

组件

arrowbutton
在一个下拉式组合框中，它是出现在列表框的上面的按钮。

entryfield
显示当前所选的输入范围。

hull
表示整个大控件本体。可以创建作为其子体的其他控件，用于特定目的。

label
如果 labelpos 为非 None，该部件将作为大控件的字符标签而被创建。参见 labelpos 说明。注意，例如在设置标签的字符选项时，用户必须使用 label_text 部件选项。

popup
在一下拉的组合框中，这是一个下拉窗。

scrolledllist
显示所选择项的滚动条。

方法

get（first=None, last=None）
如果 first 和 last 都没定义，则输入域的值被返回，否则该函数等同于使用滚动条的 get ()调用方法。

Invoke()
如果是下拉的组合框，显示下拉的列表框。而在简单的组合框中，选择当前在列表

框中已存在的项。调用 selectioncommand 并返回结果。

selectitem(index,setentry=1)

选择列表框中由 index 定义的项，该索引或者是列表框中选项之一，或者是列表框中的选项之一的整值索引。

size()

本方法是直接传递滚动条的 size()的函数。如果没有这一方法，则执行大控件的 size()函数（别名 grid_size()），程序倾向不用此函数。

ComboBoxDialog（组合框对话框）



说明

ComboBoxDialog 对简单的组合框是很方便的对话窗，常常只需要用户输入一个值或从组合框列表中选择一个值。

继承关系

ComboBoxDialog 继承自 Pmw.Dialog。

ComboBoxDialog 选项

选　项	说　明	单位	缺省值
activatecommand	如果该函数能被调用，那么只要调用 activate()函数激活了大控件，就调用该函数	函数	无
borderx	表示消息区域从左边界到右边界的宽度	距离	10
bordery	表示消息区域从上边界到下边界的高度	距离	10
buttonboxpos	表示在对话框窗口的哪一侧放置按钮框。必须为 N,S,E,W 其中之一	锚点	S
buttons	表示按钮框中所有按钮的名称。必须为一元组或一列表	(字符串...)	'OK'
command	指向一调用函数。该函数在按钮框中某一按钮被点击时或在窗口管理器中删除该窗口时被调用。该函数被调用时，只有一个选项，其为被点击的按钮的名称，或在窗口管理器中删除该窗口时该选项为 None	函数	无
deactivatecommand	如果该函数能被调用，那么只要调用 deactivate()函数激活了大控件时，就调用该函数	函数	无
defaultbutton	表示按钮框中的缺省按钮。当当前高亮对话框中的 Return 键被点击时，缺省按钮将被调用。若 defaultbutton 为 None，则不存在缺省按钮，点击 Return 键无效	索引	0

（续）

选 项	说 明	单位	缺省值
separatorwidth	若其大于 0，则依据指定宽度在按钮框和子位置间创建一被称作分隔线的窗口控件。如果按钮框的缺省边界和子位置被增大，则没有必要设置该选项。因为在按钮框与子位置间将存在一条可见的分隔线	距离	0
title	显示在窗口管理器的窗口标题栏上的标题	字符串	无

组件

buttonbox

包含对话框中所有的按钮。缺省时使用选项（hull_borderwidth=1,hull_relief='raised'）来创建。

combobox

该控件用作选取小控件，缺省时控件内容为 Pmw. ComboBox

dialogchildsite

表示对话框的子位置。可以通过在大控件中创建特定窗口。缺省时使用选项（borderwidth=1,relief='raised'）来创建。

hull

表示整个大控件本体。可以创建作为其子体的其他特定组件。

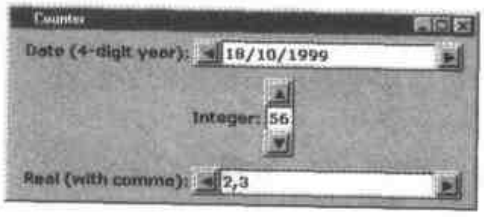
separator

如果 separatorwidth 的初始选项不为 0，则分隔控件为按钮框与子位置间的一条直线。

方法

该大控件没有自己的方法，它把来自 Pmw.Combobox 的方法传递到组合框的控件中。

Counter（计数器）



说明

此类包括箭头按钮的输入域，其输入范围中有增减量值。标准计数类型包括数值、时间、日期。用户可以定义计数函数用于特定的计数，计数可与输入域的有效性组合使用。计数控件可横排，也可纵排。

每按一次箭头按钮，则输入范围的值就按增量选项设定的值进行增减。如果新值无效（根据输入范围的 validate 选项，可能会出现溢出极大或极小的范围），则原值出现。

当按箭头按钮时，如果显示的值不是增量的严格的倍数时，它会自动向上或向下调整到最近的增量。

继承关系

Counter 继承自 Pmw.MegaWidget。

Counter 选项

选　项	说　明	单位	缺省值
autorepeat	若选项为 true，当按箭头按钮时，记数器将继续向上或向下计数	布尔型	1
buttonaspect	表示箭头按钮的高宽比。小于 1.0 则为瘦形箭头按钮，大于 1.0 为胖形	浮点型	1.0
datatype	计数器如何向上或向下计数，最常用的方法是定义数据类型选项作为字典。通过计数字典域定义计数类型，它可能是一函数，也可能是以下描述的标准计数之一的名称。字典中的任何其他域被作为关键字转给计数函数。如果数据类型不是字典，则它等价于把该数据类型定义为单 counter 域的字典，例：datatype=real 等价于 datatype={'counter' : 'real'}	常量	'numeric'
increment	当计数器增减时，定义多大的增量；如果当前显示值不是增量的倍数，则值变成比当前值大或小的下一个倍数	单位	1
initwait	在按住箭头按钮自动开始重复记数之前，定义初始延迟（μs）	微秒	300
labelmargin	若 labelpos 为非 None，则该选项表示标签控件与大控件其余控件的距离	距离	0
labelpos	表示放置标签控件的位置。若不是 None，必须为 N,S,E 或 W 中一个或二个字母的组合。第一个字母表示在大控件的哪一侧放置标签；若第二个字母也被设置，则其表示在大控件该侧的何处放置标签。例如，若 labelpos 的值为 W，则表示标签放在左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端	锚点	无
orient	表示按钮放置的方向，可为 HORIZONTAL 或 VERTICAL	常量	HORIZONTAL
padx	表示每个按钮间，以及按钮与单选框外边界间在 x 方向上的间距	距离	0
pady	表示每个按钮间，以及按钮与单选框外边界间在 y 向上的间距	距离	0
repeatrate	当箭头按钮被按住时定义自动计数的延迟(微秒)	微秒	50

标准计数器是：

- numeric　当 string.atoll()接受时，是整数。
- integer　同数值型。
- real　当 string.atoll()接受时，是实数。该计数器接受表示小数点的分割符，缺省分割符为‘.’。
- time　当 Pmw.timestringtoseconds()接受时，是时间型。该计数器接受用于分割时间域的分割符，缺省分割符是‘:’。
- data　接受 Pmw.datastringtojdn()时，是日期型。该计数器接受用于分割三个日期部分的分割符，缺省为‘/’。该计数器还接受不同的日期格式，格式用 Pmw.datastringtojdn()定义，得到想要的各部分的顺序，缺省为年月日（ymd）。

如果 counter 是一个函数，则当计数器被增减时，该函数被调用。调用该函数至少带有二部分内容，第一部分三块是（text,facfor,increment），其中 text 是当前输入域的内容。

当计数增时 facfor 是 1，减时为-1，increment 是大控件选项的增量值。

其他内容是关键字，它是组成日期类型字典的部分（不含 counter 域）。counter 函数将返回一个字符串表示增减量的值。如果文本无效，它将加一个 valueError。这时铃声响，输入文本不能修改。

组件

downarrow

箭头按钮用于计数递减，依赖于 orient 值，它将出现在输入域的左边或下边，控件组是 Arrow。

entryfield

用于文本的输入、显示和有效性检验的输入域控件。

frame

若已创建了标签组件（即 labelpos 不为 None），则也将创建一框架组件，以作为输入域和箭头按钮容器。如果不存在标签，则也不必创建框架，而用 hull 作为容器。不论何种情况，作为输入域和箭头按钮容器的边界框（不含标签）会加大。

hull

表示整个大控件本体。可以创建作为其子体的其他特定组件。

label

如果 labelpos 为非 None，该组件将作为大控件的字符标签而被创建。参见 labelpos 说明。注意，例如在设置标签的字符选项时，用户必须使用 label_text 组件选项。

uparrow

用于计数器增加的箭头按钮。依赖于 orient 值，它出现在输入域的右边或上边。控件组是 Arrow。

方法

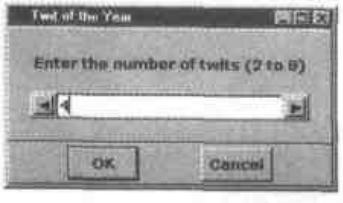
decrement()

当按向下箭头时，则计数下降。

Increment()

当按向上箭头时，则计数上升。

CounterDialog（计数对话框）



说明

计数对话框对简单计数器而言是很方便的对话窗口。常需要用户按向上或向下箭选择一个值。

继承关系

CounterDialog 继承自 Pmw.Dialog。

CounterDialog 选项

选　项	说　明	单位	缺省值
activatecommand	如果该函数能被调用，那么只要调用 activate()函数激活了大控件时，就调用该函数	函数	无
borderx	表示消息区域从左边界到右边界的宽度	距离	20
bordery	表示消息区域从上边界到下边界的高度	距离	20
buttonboxpos	表示在对话框窗口的哪一侧放置按钮框。必须为 N,S,E,W 其中之一	锚点	S
buttons	表示按钮框中所有按钮的名称。必须为一元组或一列表	(字符串...)	'OK'
command	指向一调用函数。该函数在按钮框中某一按钮被点击时或在窗口管理器中删除该窗口时被调用。该函数被调用时，只有一个选项，其为被点击的按钮的名称，或在窗口管理器中删除该窗口时该选项为 None	函数	无
deactivatecommand	如果该函数能被调用，那么只要调用 deactivate()函而激活了大控件时，就调用该函数	函数	无
defaultbutton	表示按钮框中的缺省按钮。当当前高亮对话框中的 Return 键被击中时，缺省按钮将被调用。若 defaultbutton 为 None，则不存在缺省按钮，击中 Return 键无效	索引	0
separatorwidth	若其大于 0，则依据指定宽度在按钮框和子位置间创建一被称作分隔线的部件。如果按钮框的缺省边界和子位置被增大，则没有必要设置该选项。因为在按钮框与子位置间将存在一条可见的分隔线	距离	0
title	显示在窗口管理器的窗口标题栏上的标题	字符串	无

组件

buttonbox

包含对话框中的所有按钮。缺省时使用选项（hull_borderwidth=1,hull_relief='raised'）来创建。

counter

缺省该组件是 Pmw.Counter。

dialogchildsite

表示对话框的子位置。可以用于在大控件中创建其他特定控件。缺省时使用选项（borderwidth=1,relief='raised'）来创建。

hull

表示整个大控件本体。可以创建作为其子体的其他特定组件。

separator

如果 separatorwidth 不为 0，则分隔组件为按钮框与子位置间的一条直线。

方法

deleteentry(first,last=None)

在 first 开始处和 last 的结束处删除输入内容。first 和 last 是整数索引，如果 last 是空，first 会被删除。

indexentry(index)

返回 index 有关项的数值索引。

insertentry(index,text)

在整数位置 index，输入 text。

Dialog（对话框）



说明

此类生成顶层窗口，由按钮框和其子域组成。子域常用来在大控件中生成其他特定控件，可以直接利用这部分做对话框，也可从这部分出发间接得到它。

继承关系

Dialog 继承自 Pmw.MegaTopLevel。

Dialog 选项

选　项	说　明	单位	缺省值
activatecommand	如果该函数能被调用，那么只要调用 activate()函数激活了大控件时，就调用该函数	函数	无
buttonboxpos	表示在对话框窗口的哪一侧放置按钮框。必须为 N,S,E,W 其中之一	锚点	S
buttons	表示按钮框中所有按钮的名称。必须为一元组或一列表	(字符串...)	'OK'
command	指向一调用函数。该函数在按钮框中某一按钮被点击或在窗口管理器中删除该窗口时被调用。该函数被调用时，只有一个选项，其为被点击的按钮的名称，或在窗口管理器中删除该窗口时该选项为 None	函数	无
deactivatecommand	如果该函数能被调用，那么只要调用 deactivate()函数激活了大控件时，就调用该函数	函数	无
defaultbutton	表示按钮框中的缺省按钮。当当前高亮对话框中的 Return 键被击中时，缺省按钮将被调用。若 defaultbutton 为 None，则不存在缺省按钮，击中 Return 键无效	索引	无

（续）

选　项	说　明	单位	缺省值
separatorwidth	若其大于 0，则依据指定宽度在按钮框和子位置间创建一被称作分隔线的窗口部件。如果按钮框的缺省边界和子位置被增大，则没有必要设置该选项。因为在按钮框与子位置间将存在一条可见的分隔线	距离	0
title	显示在窗口管理器的窗口标题栏上的标题	字符串	无

组件

buttonbox

包含对话框中所有按钮的按钮框。缺省时使用选项（hull_borderwidth=1,hull_relief='raised'）来创建。

dialogchildsite

表示对话框的子位置。可以用于在大控件中创建其他特定控件，缺省时使用选项（borderwidth=1,relief='raised'）来创建。

hull

表示整个大控件本体。可以创建作为其子体的其他特定组件。

separator

如果 separatorwidth 不为 0，则分隔组件为按钮框与子位置间的一条直线。

方法

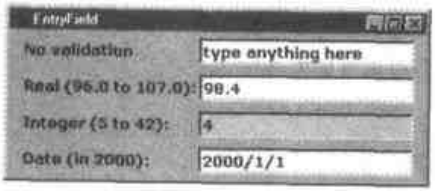
interior()

返回对话框的子域，与 component('dialogchildsite')相同。

invoke(index='default')

当 index 指向的按钮被按后，则由命令选项定义的命令被执行。index 可以为 Pmw .Buttonboxindes()接受的任意形式。

EntryField（输入域）



说明

此类由输入部件组成，部件含不同种类的确认选项。已有的确认函数有 integer、real、time 或 date，也可用外部提供的确认函数。如果输入有效文本，会在正常的背景显示。如果输入无效文本，则不显示，而出现上次出现的有效文本。如果输入部分有效文本，会显示一个不同的背景色以提示有出错的地方。例，部分有效的实文本为“-.”，它可能是有效的字符串中“-.5”的前两个。一些有效的部分，例 data 型，可以放宽部分有效要求。它允许用户可以较宽

松地输入文本。

每次按键前或修改文本前就已确认有效性。不论如何，如果允许部分有效，则输入的文本的有效性仅在其被使用之前被检查，这是进一步确定的有效性的形式。

可定义最小或最大值。某些有效性可以通过定义实现，比如，日期和时间的格式及分割符。

有效性确认函数返回值

有效性确认是通过一函数来实现的，它以输入文本为参数，返回三个标准值之一，显示文本是否有效：

Pmw.OK ： 文本有效。

Pmw .ERROR： 文本无效，不接受显示。这种情况下，输入域将保存以前值。

Pmw.PARTIAL：文本是部分有效,可接受显示，这时文本被显示为 errorbacdground 色。

继承关系

EntryField 继承自 Pmw.MegaWidget。

EnfryField 选项

选 项	说 明	单位	缺省值
command	当按 Return 键或调用 invoke()函数,指向一调用函数	函数	无
errorbackground	定义背景色，用以显示无效或部分有效的文本	颜色	'pink'
extravalidators	这是个外部有效符的字典，关键是有效符的名称，用在以后有效参数的函数调用中。字典中每个值都是两元组(validate_function,stringtovalue_function), validate_function 用于实现其有效性，stringtovalue_function 用于把输入值变换成能与最小和最大极限相比较的值。这些函数在有效性选项中描述 如果两者都不作为函数，它被描述成外部有效符和标准有效符之一的名称。当配置 validate 选项时，可进行别名查找，当 extravalidators 选项被配置或调用 validate 函数时，不能进行别名查找 如果外部有效符之一的名称与标准有效符之一的名称相同，则外部有效符优先	字典	{}
invalidcommand	输入无效文本，且保存以前文本值（即有效函数返回 Pmw.ERROR）时执行该函数。如果在调用 setentry()时试图设置无效文本，则也调用该函数	函数	self.bell
labelmargin	若 labelpos 选项不是 None，该选项表示标签部件与大控件其余部分的距离	距离	0
labelpos	表示放置标签部件的位置。若不是 None，必须为 N,S,E 或 W 中一个或二个字母的组合。第一个字母表示在大控件的哪一侧放置标签。若第二个字母也被设置，则其表示在大控件该侧的何处放置标签。例如：若 labelpos 的值为 W，则表示标签放在左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端。	锚点	无
modifiedcommand	当用户修改其输入内容或调用 setentry(),则调用该函数	函数	无
validate	定义输入文本的有效性的类型，详见下述	常量	无

（续）

选　项	说　明	单位	缺省值
value	定义输入的初始内容。如果该文本无效，出现 errorbackground 色且将调用 invalidcommand 函数。如果在构造函数中 value 和 entry_textvariable 选项都被定义，则 value 优先	字符串	''

Validators(有效符)

最常用的定义 validate 选项的方法是字典。其类型可以通过 validator 字典域进行定义，它可能是以下描述的标准的有效符之一的名称，也可能是由 extravalidators 选项提供的有效符的名称或是一个函数或是 None。

任何其他字典域用于定义输入值的其他限制。对全体有效符，下列域可能被定义：

- min：定义最小可接受的值，如果不检验最小值，则为空。缺省为空。
- max：定义最大可接受的值，如果不检验最大值，则为空。缺省为空。
- minstrict：如果为 true，则严格检验最小值。否则允许输入值小于最小值，但用 errorbackground 色显示输入部分。缺省为 true。
- maxstrict：如果为 ture，则严格检验最大值。否则，允许输入值大于最大值，但用 errorbackground 色显示输入部分。缺省为 ture。

如果字典包含 stringtovalue 域，它将不考虑有效符的正常的 stringtovalue 函数。stringtovalue 函数描述如下。

字典中其他域（除了上述的核心域）被传递给有效符，且把 stringtovalue 函数作为关键字。

如果 validate 不是一个字典，则等价于用单个 validator 域定义其为字典，例如，validate='real'等价于 validate={'validator': 'real'}，它定义实数没有最大或最小限制，用“.”作为小数点。

在 validator 域中标准的有效符有：

- numeric：大于等于 0 的整数，只针对数字，无符号。
- hexadecimal：十六进制数（可选前导符为 0x），被 string.atol(text,16) 接受。
- real：数，可有也可没有小数点和指数（e 或 E）选项，被 string.atof()接受。该有效符也接受 separator 参数，它指定表示小数点的字符。缺省分割符为“.”。
- alphabetic：由字母 a~z 和 A~E 组成。这种情况下，min 和 max 指向文本长度的极限。
- alphanumeric：由字母 a~z 和 A~Z 及数字 0~9 组成，这种情况下，min 和 max 指向文本长度的极限。
- time：时、分、秒，格式为 HH:MM:SS，被 Pmw.timestringtoseconds()接受。此有效符接受定义三个时间字段的分割符，缺省为“:”。时间可以为负。
- date：日、月、年，被 Pmw .datestringtojdn()接受。此运算符接受定义三个日期字段的分割符。缺省为“:”。该运算符接受 format 参数，它被传递给 Pmw.datastringtojdn()以定义想要的字段顺序，缺省为 ymd。

如果 validator 是函数，则当输入内容被用户改变或把它调用到 setentry()时，该函

数被调用。该函数在被调用时至少有一参数，第一个参数作为用户修改的新文本或setentry()修改的新文本。其他参数是构成有效字典的非核字段的关键字的内容。

有效符函数将如上述返回 Pmw.OK、Pmw.ERROR 或 Pmw.PARTIAL。它将不作最小和最大检验。如果返回 Pmw.OK，调用之后再做。

在字典中 stringtovalue 部分可能指定为标准有效符之一的名称，外部提供的有效符之一的名称，函数或 None。

stringtovalue 函数用于把输入变换成能与有效符定义的最小或最大值相比较的值。如果把 min 或 max 部分定义成字符串，则是利用 stringtovalue 函数对其进行变换。调用 stringtovalue 函数，其参数与 validator 函数的参数相同。对于标准数的有效符，stringtovalue 函数把字符串变换成数。而对标准的字符有效符，返回的是字符串的长度。而对于 time 和 date 有效符，分别返回秒数和日历天数。见 Pmw. Stringtoreal()、Pmw.timestringtoseconds() 和 Pmw.datestringtojdn()。

如果有效符被定义成函数而且没给出 stringtovalue 部分，则其缺省就是 Python len() 函数。

如果有效符为空，不做任何确认。不论如何，根据 stringtovalue 函数要检验最小和最大值。例如，限制输入文本到 5 个字符中的最大值：

```
Pmw.EntryField(validate={'max':5})
```

对每一标准有效符的 Validator 函数见下面：

```
Pmw.numericvalidator
Pmw.integervalidator
Pmw.hexadecimalvalidator
Pmw.realvalidator
Pmw.alphabeticvalidator
Pmw.alphanumericvalidator
Pmw.timevalidator
Pmw.datevalidator
```

无论何时 validate 选项被设置，则在输入控件中当前显示的文本会被确认。如果它是无效的，则设置 errorbackground 色，调用 invalidcommand 函数。不论如何，显示文本不修改。

组件

entry

用于用户输入文本的地方，长文本可以由鼠标的中间按钮拖着水平滚动。

hull

表示整个大控件本体。可以创建作为其子体的其他特定组件。

label

如果 labelpos 为非 None，该部件将作为大控件的字符标签而被创建。参见 labelpos 说明。注意，例如在设置标签的字符参数时，用户必须使用 label_text 组件选项。

方法

checkentry()

检验在当前输入部件中的内容的有效性。如果文本无效，设置 errorbackground 的背景并调用 invalidcommand 函数。如果一个变量由 entry_textvariable 的选项指定，这一方法在用 set()函数调用变量之后将调用该函数。如果这一步没出现，则输入部件的背景不能正确设置。

Clear()

从输入部件中移去全部文本。等价于 setentry（''）。

Invoke()

如果按 Return 键并且返回结果，则执行 command 选项定义的命令。

setentry(text)

设置输入窗部件的内容到 text，并且当用户输入文本时，做有效性检验。如果文本无效，则输入部件将不变，并且调用 invalidcommand 函数。

Valid()

如果输入窗部件内容是有效的，则返回真。

Group（组）



说明

此大控件由内部框架组成，该框架由外部的环形界及界上边显示的不同标题组成。程序员可以在内部的环行框架中生成其他控件。

继承关系

Group 继承自 Pmw.MegaWidget。

Group 选项

选　项	说　明	单位	缺省值
tagindentl	从环的左边到标题内容的左边的距离	距离	10

组件

groupchildsite

能包括分组的其他控件的框架。

hull

表示整个大控件本体。可以创建作为其子体的其他特定组件。

ring
这部分组成环绕 groupchildsite 的边界。缺省的边界宽为 2，缺省凹凸性是 groove

tag
不同的标题显示在环绕边界的上部。如果为空，则无标题显示。

方法

interior()
返回程序员在其中创建控件的框架，等同于 component('groupchildsite')。

LabeledWidget（标签控件）



说明

此大控件由内部框架组成。相应的标签放在框架的某边。程序员能在内部框架中生成其他部件。

继承关系

Labeledwidget 继承自 Pmw.MegaWidget。

Labeledwidget 选项

选 项	说 明	单位	缺省值
labelmargin	若 labelpos 选项不是 None，该选项表示标签窗口部件与大控件其他控件的距离	距离	0
labelpos	表示放置标签部件的位置。若不是 None，必须为 N,S,E 或 W 中一个或二个字母的组合。第一个字母表示在大控件的哪一侧放置标签。若第二个字母也被设置，则其表示在大控件该侧的何处放置标签。例如，若 labelpos 的值为 W，则表示标签放在窗口左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端	锚点	无

组件

hull
表示整个大控件本体。可以创建作为其子体的其他特定组件。

label
如果 labelpos 为非 None，该部件将作为大控件的字符标签而被创建。参见 labelpos 说明。注意，例如在设置标签的字符选项时，用户必须使用 label_text 组件选项。

labelchildsite

包括标注标签的其他控件的框架。

方法

interior()

返回程序员在其中创建的框架，等同于 component('groupchildsite')。

MegaArchetype

说明

此是所有的 Pmw 大控件的基础。它提供管理选项和小控件的方法。

此类常用于其他类的基类。如果定义了 hullClass 的内容，例如在 Pmw.MegaWidget 和 Pmw.MegaToplevel 部分中，则创建一个容器控件，把它作为所有其他控件的父类。源自子类的类生成其他控件及选项，用以实现大控件，满足实际使用。

如果没有 hullclass 作为构造函数，则不能产生容器控件，而只有选项配置函数有效。

继承关系

MegaArchetype 没有继承源。

方法

addoptions(optionDefs)

把附加选项增加到这个大控件中， optionDefs 参数与 defineoption()方法同样处理。派生类使用这种方法。如果一个大控件有条件地定义选项，可能会依赖其他选项的值，则这种方法才被用。通常情况下，大控件无条件地创建所有的选项，调用 defineoption()而不必使用 addoptions()。在调用 defineoptions()之后，而在调用 initialiseoptions()前，调用这种方法。

cget(option)

返回 option(在选项部分定义的格式)当前值。使用对象下标使得这种方法有效，例如 MyWidget['font'] 。不像 Tkinter 的 cget()总返回字符串，当 option 设置时，这种方法返回相同的值和类型（除了选项是部件选项且部件是 Tkinter 小控件的情况外，在这种情况下，它返回由 Tcl/Tk 返回的字符串）。

component(name)

返回名称为 name 小控件的内容。允许大控件的用户直接进入和配置小控件。

componentaliases()

返回部件内容的别名列表。列表的每一项均为两元组，其中第一项为别名，第二项为部件控件或其涉及的子控件的名称。

componentgroup(name)

返回名称为 name 的控件组，如果控件不在组内，则不返回。

Components()

返回大控件的控件名称经过分类后的列表。

configure(option=None,**kw)

查询或配置大控件的选项。如果没给出参数，返回由全部大控件选项和值组成的多元值。每个值是 5 元组（name,resoureName,resourceClass,default,value）。这与标准 Tkinter configure()方法返回值有相同的格式，但资源名称与选项名称相同，并且资源类是有首字母大写的选项名。

如果给出一个选项内容，则返回该选项的 5 元组。否则，设置由关键字内容指向的配置选项。

createcomponent(name,aliases,group,widgetClass,widgetArgs,**kw)

通过调用 widgetClass 创建一个小控件。其参数由 widgetArgs 和任意的关键字内容给出。Name 参数是该控件将出现的名称，且该控件不必包括下划线（_）。group 参数定义控件的组。aliases 选项是一 2 元组。其中第 1 元是别名，第 2 元是小控件或其涉及的子控件的名称。

createlabel（parent,childCols=1,childRows=1）

创建一个 label 控件，它在 parent 部件中命名为 label。这一方便的方法常用于一些需要选项标签的大控件。小控件必须有 labelpos 和 labelmargin 命名的选项。如果 labelpos 为空，不产生标签。否则产生标签，并且依据 labelpos 和 labelmargin 定位。利用 grid()函数把标签加到 parent 中，该方法由 childCols 和 childRows 显示标签的行列跨度是多少。注意所有 parent 部件的子件必须用 grid()函数加到 parent 部件中。在构造大控件过程中，派生类调用 createlabel()函数。

defineoptions(keywords，optionDefs)

创建大控件的选项。由 optionDefs 参数定义选项。这是一列 3 元组（name,default,callback），其中 name 是选项的名称，default 是缺省值，当把选项的值调入到 configure()函数时，调用回调函数。Keywords 参数是传到大控件构造函数中的关键字内容。用户可以通过提供一关键字到构造函数中的方法取代选项缺省值。

应在基类构造之前调用该函数，使得派生类中定义的缺省值取代基类中的值。

如果回调是 Pmw.INITOPT，则选项是一初始的选项。

destroycomponent(name)

删除名为 name 的大控件组件。派生类可以调用这种方法破坏一个大控件组件。该函数破坏该组件然后删除其所有记录。

Hulldestroyed()

如果与 hull 组件内容相关的 Tk 控件被损坏，则返回 true。

initialiseoptions(myclass)

检查关键字内容，并调用选项的回调函数。该函数必须在大控件的 myClass 设置为被定义的类参数后被调用。它确认赋予构造函数的所有关键字参数已使用。否则，给出关键字参数未被使用的出错提示。如果在一大控件构造过程中，关键字在对 defineoptions() 或 addoptions()(由大控件或其一基类定义)的调用中定义，表明关键字要使用；或它参考由名称区别的大控件的一个控件；或它参考由组区分的至少一个控件。

有时也针对所有配置选项调用配置回调函数。

这种函数只有当被叶类构造函数调用时才有效，即如果 myClass 等同于被构造的对象类时。当基类构造器调用该方法时，它立即返回。

Interior()

它返回构造内部空间的控件，这一空间是大控件创建全体子件的地方。默认情况下，如果创建一 hull 组件，则该函数返回该组件，否则返回空。一子类将使用由 interior()返回的控件，作为它创建的组件或任何子控件的父件。能被进一步子类化的大控件，例如 Pmw.Dialog，应重新指定义此方法以返回控件，其中子类须创建子件。以上包含的小部件作为 hull 组件总是有效的。

Isinitoption(option)

如果 option 是一个初始选项，返回为 true。否则返回 false(option 是配置选项)。option 必须是大控件的一个选项，而不是一个组件的选项；否则会出现异常情况。

Options()

返回大控件的选项的排序的列表。列表中每一项是 3 元组（option，default，isinit），其中 option 是选项名称，default 是缺省值，如果选项是初始选项，则 isinit 是 true。

MegaToplevel（大顶层）

说明

该类创建一包含在顶层窗口中的大控件，可直接用于生成顶层大控件，也可作为更特殊的顶层大控件的基类，例如 Pmw.Dialog。它创建名为 hull 的 Toplevel 组件，作为大控件的容器。Hull 控件的窗类名称设置成大控件最具体的类名。派生类通过创建作为 hull 控件的子控件的其他控件组件而使该控件具体化。

大控件可用于作为正常的顶层窗，或作为一模式对话框。对于正常用途，使用 show() 和 withdraw()函数，对于模式对话框，使用 activate()和 deactivate()函数。如果窗管理器删除一显示正常的窗，则缺省是损坏该窗。如果窗管理器删除处于激活态的窗（比如当作为一模式对话框时），则窗被禁用。使用 userdeletefunc()和 usermodaldeletefunc()函数将取消这些行为。如果用户希望把窗作为一模式对话框，则不能直接调用 protocol()函数，用于设置 WM_DELETE_WINDOW 窗管理器协议。

当前激活的窗形成一栈，栈的顶部是最近激活的窗。鼠标和键盘输入都进入顶层窗。当窗被禁用，栈中下一个窗开始接收事件。

继承关系

MegaToplevel 继承自 Pmw.MegaArchetype。

MegaToplevel 选项

选 项	说 明	单位	缺省值
activatecommand	如果该函数能被调用，那么只要调用 activate()激活了大控件，就调用该函数	函数	无
deactivatecommand	如果该函数能被调用，那么只要调用 deactivate()禁用了大控件，就调用该函数	函数	无
title	显示在窗口管理器的窗口标题栏上的标题	字符串	无

组件

hull

表示整个大控件本体。可以创建作为其子体的其他特定组件。

方法

activate(globalMode=0,master=None,geometry='centerscreenfirst')

显示窗作为一模式对话框，这就是指所有的鼠标和键盘输入都进入这个窗，其他窗不接受任何输入。如果不想把鼠标和键盘的输入限制在该窗，使用 show( )函数代替。当窗缩小时，释放开原窗，结果被 activate( )函数返回，直到调用 deactivate( )函数，否则 activate( )函数不返回值。

如果 globalMode 是 false，则该窗将获得鼠标指示器和键盘的控制权，而不会允许输入进入到其他在用的的最大化窗。如果 globalMode 是 true，则不允许输入进入到其他不在用的的最大化窗。Global 控制功能应少用。

当窗被显示时，根据在屏幕中的几何位置的定位有如下形式：

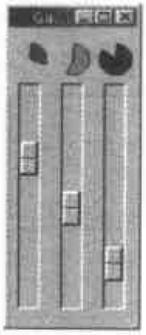
- centerscreenfirst 窗体首次激活时在屏幕的中央，尽管用户移动窗了，但以后的激活中，它总在上次显示的相同的位置。
- centerscreenalway 窗体在屏幕的中央（一半重叠，三分之一向下）。
- frist+spec 假定其余部分（首次之后）是标准的几何定义，窗将被设定在首次激活的位置。尽管用户移动窗了，但以后的激活中，它总在上次显示的相同的位置，例如 geometry=first+100+100，窗将第一次显示在位置（100，100）。activate( ) 调用其他的选项将不改变窗以前的位置。
- spec 是标准的几何定义，由此定义设置窗。

如果 BLT Tcl 扩展库是存在的，则时钟指针将显示到窗被禁用时为止。

如果 activate command 选项能被调用，则该函数在窗开始等待结果之前被调用。

如果 master 不是 None，则窗变成一主窗的瞬时窗，主窗将是另一个存在的顶层窗。

MegaWidget（大控件）



说明

该类创建一包含在框架窗口中的大控件。它是大控件的基类，这些大控件不包含在它们本身的顶层窗中，例如 Pmw.ButtonBox 和 Pmw.ComboBox。它创建一被称为 hull 的 Frame 组件，将其作为大控件的容器。Hull 控件的窗类名被设置为最具体的大控件的类名。派生类通过创建其他控件作为 hull 控件的子控件而生成。

继承关系

MegaWidget 继承自 Pmw.MegaArchetype。

组件

hull
表示整个大控件主体。可以创建作为其子体的其他特定组件。

方法

destroy()
破坏 hull 组件，包括它的全部子件。

MenuBar（菜单条）



说明

该类创建一包括菜单的管理器控件，提供有增加菜单按钮，增加菜单到菜单条及增加菜单项到菜单中的方法。菜单按钮可加到小控件的左边或右边。每个菜单按钮和菜单项有帮助信息，这些信息是由 Pmw.Balloon 控件显示的。

继承关系

MenuBar 继承自 Pmw.MegaWidget。

MenuBar 选项

选　项	说　明	单位	缺省值
balloon	定义 Pmw.Balloon 控件以显示菜单按钮和菜单项的帮助信息。如果空，无帮助显示	控件	无
hotkeys	如果菜单支持“热键”则定义，否则作为“加速键”定义，如果 true，用户可选择用菜单项文本中的下划线字母代替菜单项	布尔	1

（续）

选 项	说 明	单位	缺省值
padx	表示每个菜单按钮间，以及菜单按钮与菜单条外边界间在 x 方向上的间距	距离	0

组件

hull

表示整个大控件主体。可以创建作为其子体的其他特定组件。

方法

addcascademenu (menuName,submenu,help='',traverseSpec=None,**kw)

增加一层叠的称为 submenu-menu 的子菜单到 menuName 菜单中，submoun 不必总存在。如果定义 help，则 balloonHelp 用于帮助。如果定义 taverseSpec，则定义层叠菜单中的下划线字母用作盘键加速键。traverseSpev 能被定义为要么是字符，要么是有下划线的字符的整数索引。

addmenu(menuName,balloonHelp,statusHelp=None,zize='left',traverseSpec=None,**kw)

增加一菜单按钮和相关菜单到菜单条中，当生成菜单按钮时，任何关键字都被传到构造函数中。如果没给出文本关键字，则菜单按钮的文本选项缺省为 menuName。每个菜单按钮被加到菜单条中，可以在左边或右边。

如果定义了 balloon 选项，balloonHelp 和 statusHelp 被加到浮动图中作为菜单按钮的帮助字符串。至于这些帮助信息字符串如何显示，见 Pmw.Balloon 的 bind()函数。

创建的菜单按钮是名为 menuName-button 的部件。创建的菜单是名为 menuName-menu 的部件。该方法返回菜单按钮部件。

addmenuitem(menuName,type,hilp='',traverseSpec=None,**kw)

增加菜单项到 menuName 给出的菜单中，菜单项的种类由 type 给出，可能是 command，separator，checkbutton，radiobutton 或 cascade 之一。当创建菜单项时，任意关键字都会被传给菜单。对每一类型的有效选项见 Menu 函数。当鼠标移到菜单项时，由 help 给出的字符串由 statuscommand 浮标显示。

deletemenu(menuName)

删除称为 menuName 的菜单。在主菜单项被删除前，删除子层叠的菜单。

deletemenuitems(menuName,start='0',end=None)

删除称为 menuName 的菜单中的菜单项。如果 start 和 end 被定义，则删去由 start 索引开始到 end 索引结束的菜单项。如果 start 被定义而 end 未定义，则删去由 start 索引的菜单项；如果 start 和 end 都没定义，则删去 menuName 中的所有菜单项。

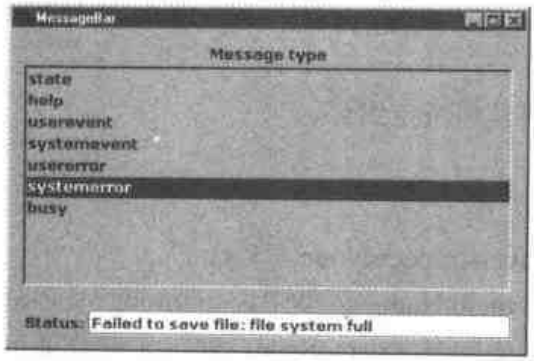
disableall()

禁用菜单条中的全部菜单项。

enableall()

激活菜单条中的全部菜单项。

MessageBar（消息条）



说明

该类创建单条消息显示区，可以显示几种不同类型的消息。各种消息对应的间隔过期后，则清除消息，每种消息都有一优先权以至于如果应用想同时显示多于一条消息时，则先显示最高优先权的消息。消息可以附加声音。

继承关系

MessageBar 从 pmw.MegaWidget 继承而来。

MessageBar 选项

选　项	说　明	单位	缺省值
labelmargin	如果 labelpos 非空，此选项指定 label 组件与大控件其他组件的距离	距离	0
labelpos	指定何处放置 label 组件。如果非空，它该是 N，S，E 或 W 中的一个或两个连起来组成。第一个字母指定在大控件哪一侧放置 label 组件，如果指定二个字母，则它给出在这侧的哪里放置 label 组件。例如，如果 labelpos 是 W，label 被放置在左侧中央；如果是 WN，label 被放置在左侧顶部，如果是 WS，label 被放置在左侧底部	锚点	None
messagetypes	选项指向由消息条和消息类型的字母提供的消息类型。它是一个字典，其关键之处在于一字符串定义一消息类型。选项值是由四个整数值构成的多元组（priority,showtime, bells, logmessage），其中 priority 是消息类型的级别，showtime 是显示该类型消息的秒数，bell 是可以听到的铃声，logmessage 是一布尔型变量用于定义这一消息是否被分割开供以后重新修复。高优先权的消息显示时要优先于低优先权的消息。如果一个高优先权的显示时间过期（由于显示有 showtime 秒数限制），则显示低优先权的消息。0 显示时间是指消息不会有时间溢出，有利于显示用于描述当前应用状态的消息，而不适用于显示描述事件的消息。分割不是当前完成的，缺省是 {'systemerror'：(5,10,2,1) 'usererror'：(4,5,1,0) 'busy'：(3,0,0,0) 'systemevent'：(2,5,0,0) 'userevent'：(2,5,0,0) 'help'：(1,5,0,0) 'state'：(0,0,0,0)}	字典	
silent	如果为 true，则不论 messagetypes 选项指向的铃的值是什么，铃声都不响	boolean	0

组件

entry
显示消息的控件，长消息可以通过鼠标中间按钮拖动水平滚动。

hull
表示整个大控件主体。其可以创建作为其子体的特定组件。

label
如果 labelpos 为非 None，该组件将作为大控件的字符标签而被创建。参见 labelpos 说明。注意，例如在设置标签的字符选项时，用户必须使用 label_text 组件选项。

方法

helpmessage(text)
根据帮助消息类型定义的字符特征，在消息框方便地显示 text 的方法。等价于 message('help',text)。

message(type,text)
根据帮助消息类型定义的字符特征，在消息框显示 text 的方法。

resetmessage(type)
清除 type 类型的消息并把所有消息类型设置到较低优先级，除非永久性消息，如 state。这对于清除繁忙消息和任何未完成事件及帮助信息很有用。

Messagedialog（消息对话框）



说明

消息对话框是一包含消息部件的方便对话窗。用于给用户在一临时的窗中显示多行文本。

继承关系

MessageDialog 从 pmwDialog 继承而来

MessageDialog 选项

选 项	说 明	单位	缺省值
activatecommand	如果该函数能被调用，那么只要调用 activate()函数激活了大控件时，就调用该函数	函数	无
borderx	表示消息区域从左边界到右边界的宽度	距离	20
bordery	表示消息区域从上边界到下边界的高度	距离	20
buttonboxpos	表示在对话框窗口的哪一侧放置按钮框，必须为 N,S,E,W 其中之一	锚点	S

（续）

选　项	说　明	单位	缺省值
buttons	表示按钮框中所有按钮的名称，必须为一元组或一列表	(字符串...)	'OK'
command	指向一调用函数。该函数在按钮框中某一按钮被击中或在窗口管理器中删除该窗口时被调用。该函数被调用时，只有一个选项，其为被击中的按钮的名称，或在窗口管理器中删除该窗口时该选项为 None	函数	无
deactivatecommand	如果该函数能被调用，那么只要调用 deactivate()函数激活了大控件时，就调用该函数	函数	无
defaultbutton	表示按钮框中的缺省按钮。当当前高亮对话框中的 RETURN 键被击中时，缺省按钮将被调用。若 defaultbutton 为 None，则不存在缺省按钮，击中 Return 键无效	索引	无
iconmargin	如果图标存在，定义图标的左边位置	距离	20
iconpos	如果图标要显示，定出图标位置，值必是 E，S，W，N 或 None 之	距离	无
separatorwidth	若其大于 0，则依据指定宽度在按钮框和子位置间创建一被称作分隔线的窗口控件。如果按钮框的缺省边界和子位置被增大，则没有必要设置该选项。因为在按钮框与子位置间将存在一条可见的分隔线	距离	0
title	显示在窗口管理器的窗口标题栏上的标题	字符串	无

组件

buttonbox

buttonbox 中包含对话框的所有按钮。缺省时使用选项（hull_borderwidth=1, hull_relief='raised'）来创建。

dialogchildsite

表示对话框的子位置。可以通过在主控件中创建其他控件来生成特定的窗口。缺省时使用选项（borderwidth=1,relief='raised'）来创建。

hull

表示整个大控件主体。可以创建作为其子体的其他特定组控件。

icon

缺省为标签。

message

缺省为标签。

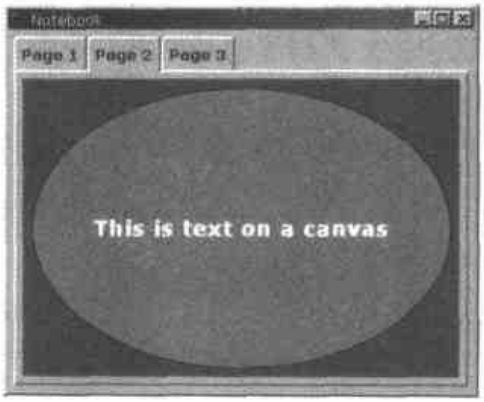
separator

如果 separatorwidth 不为 0，则分隔组件为按钮框与子位置间的一条直线。

方法

该窗口控件没有自己的方法。

NoteBook（记事本）



说明

该控件在版本 0_8_3 及以后版本中代替 NotebBookR 和 NoteBookS。

一个记事本里面包含几个工作页。任何时候，这些页中只有一个可见，其他页面则藏到下面（后面）。另一页可以通过按页面上的标签显示，标签沿顶端显示。

记事本可以选择不显示标签。此时，另一个选择控件，如 Pmw.OptionMenu，可以用来选择页面。

继承关系

NoteBook 从 Pmw.MegaArchetype 继承过来。

记事本选项

选　项	说　明	单位	缺省值
borderwidth	表和活动页面周围的边界宽度	integer	2
createcommand	当一页首次选中时，指定一个函数调用。函数为单选项调用，为选中页面名，且调用早于 raisecommand 函数。这使页面内容的创建迟于页面首次显示	函数	无
lowercommand	当一页面被一个新页面替代时，指定一个函数调用。函数为单选项调用，为前一个选中页面，且调用早于 createcommand 或 raisecommand 函数	函数	无
pagemargin	选中页面内部周围边框空白大小	pixels	4
raisecommand	当一页面被选中时，指定一个函数调用。函数为单选项调用，为选中页面	函数	无
tabpos	指定标签的位置。如果是 n，每个页面都产生标签，且置于记事本顶部。如果无，没有标签，此时选择控件可以用来通过调用 selectpage()方法选择页面	string	'n'

组件

hull

表示整个大控件主体。大控件内容创建为画布项，且使用画布坐标系置于 hull 中。缺省内容为 Tkinter.Canvas。

方法

add(pagename,**kw)
在记事本末加一页。详细参考 insert()。

delete(*pagename)
从记事本删除 pagename 给定的页面。每个页面名字是 index()方法接受的格式。如果当前选中页面被删除，按索引顺序的下一个页面将被选中。如果删除的是尾页，选中前一个页面。

getcurselection()
返回当前选中页面的名字。

index(index,forinsert=0)
返回与 index 相应页面的数字索引。可以按下列形式中的一种给出：

- name　指出页名
- number　按数字给出页码，0 代表第一页
- END　最后一页
- SELECT　当前选中页面。

insert(pageName,before=0,**kw)
在记事本中加入名为 pageName 的页面作为组件。该页加入到刚好在 before 页面的前面，before 可以是任何 index()可以接受的格式。如果 tabpos 非空，也建造一个名为 pagename_tab 的标签。当创建 page 或 tab 时，前缀为 page_或 tab_的关键字选项被传送到相关的指令去。如果 tab_text 关键字选项没有给定，文本选项缺省值为 pageName。如果一页被插入到空记事本中，该页被选中。加一页到记事本末尾，使用 add()。该方法返回 pageName 组件。

page(pageIndex)
返回 pageIndex 页的控件框架。pageIndex 可以是任何 index()方法接受的格式。

pagenames()
返回一个页列表，按照显示顺序。

recolorborders()
改变页面和边框颜色。该方法是必须的，因为边框是按画布多边形建造的，因此对正常颜色改变技术，如 pmw.color.Changecolor()无响应。

selectpage(page)
选择页面为当前页面。此页将上升，原先选中页面则下降。

Setnaturalpagesize(pagename=None)
设置记事本页面的大小为所有页面要求的最大宽度和高度。该方法在一个页面及其

内容创建之后调用，它调用 update_idletasks()来决定页面的高和宽。这使记事本在屏幕上从缺省大小转到自然大小。

tab(pageIndex)

返回 pageIndex 页面的标签组件。pageIndex 可以是任何 index()接受的格式。如果 tabpos 为空，返回空值。

NoteBookR



说明

NoteBookR 和 NoteBook 完成类似的功能。窗口通过标签安排成一系列重叠页面，其中标签实现把页面从栈中升到最高层。

继承关系

NoteBookR 从 Pmw.MegaWidget 继承而来。

NoteBookR 选项

选　项	说　明	单位	缺省值
balloon	指定控件的 balloon 帮助	字符串	无
ipadx	指定每个页面 x 方向的内衬大小	距离	4
ipady	指定每个页面 y 方向的内衬大小	距离	4

组件

hull

作为整个大控件的主体。其他特定组件创建为其子组件。

nbframe

缺省情况下，此为画布。

方法

add(pagename,**kw)

在记事本末加一页。详细参考 insert()。

initialise(e=None,w=1,h=1)

interior()

返回控件构成内部空间，所有的子控件都在其中产生。缺省情况下，此方法返回控件的 hull 组件，如果已经产生的话，不然返回空值。一个子类使用 interior()返回的控件

作为所有组件或其产生的子控件的父本。窗口组件还可以进一步产生子类，比如Pmw.Dialog，应该重新定义该方法返回子类会在其中产生子代的控件。包含的整体控件通常可以作为 hull 组件获得。

lift(pagenameOrIndex)

如果 pagenameOrIndex 为字符串，它把名为此的页面升起。如果是整数，它按索引升起页面。

pagecget(pagename,option)

返回 pagename 页面的 option 值。

pageconfigure(pagename,**kw)

配置 pagename 指定页面，这里名字是字符串，给出页面的名字，关键选项给出页面的选项值。

pages()

返回当前记事本中定义的页面列表。

raised()

返回当前在栈顶的页面。

tkdelete(pagename)

删去 pagename 页面的页及标签。

tkraise(pagenameOrIndex)

lift 的一个同名方法。

NoteBookS



说明

NoteBookS 和 NoteBook 完成类似的功能。窗口通过标签安排成一系列重叠页面，其中标签实现把页面从栈中升到最高层。NoteBookS 比 NoteBookR 在选项上有更多的精确控制。

继承关系

NoteBookS 从 Pmw.MegaWidget 继承而来。

NoteBookS 选项

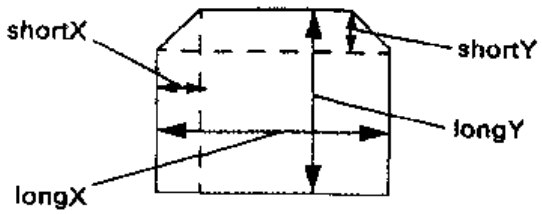
选　项	说　明	单位	缺省值
activeColor	活动标签及其相关页面的颜色	color	'red'

（续）

选 项	说 明	单位	缺省值
canvasColor	记事本页面的画布背景颜色（正常情况下如果至少有一个页面则不可见）	color	'white'
canvasHeight	基画布的整体高度	height	250
canvasWidth	基画布的整体宽度	width	400
deactiveColor	当前非活动页面的颜色	color	'grey'
longX	设置 X 方向长轴（参看下面的 diagram）	坐标	30
longY	设置 Y 方向长轴（参看下面的 diagram）	坐标	35
offsetY	设置页面顶部到画布顶部的偏移	坐标	30
shadeColor	每个页面背后的阴影颜色	color	'#666666'
shortX	设置 X 方向短轴（参看下面的 diagram）	坐标	7
shortY	设置 Y 方向短轴（参看下面的 diagram）	坐标	7
tabColor	标签后面画布的颜色	color	'blue'
tabHeight	标签所在区域的高度	height	40
textColor	标签文字颜色	color	'black'
textFont	标签文字字体	font	('Helvetica,10,normal')

组件

containerCanvas
缺省为画布。



hull
这作为整个控件的主体部分。其他组件创建为 hull 的子组件。

mainCanvas
缺省为画布。

tabCanvas
缺省为画布。

方法

addPage(name)
在记事本内产生名为 name 的 Frame。

delPage(name)
删除名为 name 的页面（Frame）。

getPage(name)
返回名为 name 的页面（Frame），并不把该页升到栈顶。

pageNames()
返回当前记事本中定义的页面的名字列表。

Pages()
返回当前记事本中定义的页面（Frame）列表。

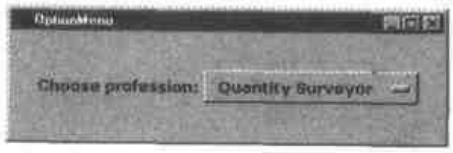
raisePage(name,select=1)
把名为 name 的页面升到栈顶。如果 select 为 false，不去掉当前选中页面。

raised()
返回当前活动页面名。

rebind()
允许通过点击标签选中页面。

unBind()
禁止通过点击标签选中页面。

OptionMenu（选项菜单）



说明

该类产生一个选项菜单，包含一个菜单按钮和一个与菜单相关的当菜单按下时出现的弹出项。菜单按钮中的文本在当菜单按钮选定时随之而变。当前选中值可以从 megaWidget 中获得。

继承关系

OptionMenu 从 Pmw.MegaWidget 继承过来。

OptionMenu 选项

选　项	说　明	单位	缺省值
command	当菜单项被选中或者调用 invoke()方法时，指定一个函数来调用。函数用其当前选中的值作为单选项调用	函数	无
initialitem	指定初始选定值，该选项如 setitem () 方法的 index 选项的一样。		无
items	一个包含初始的将在菜单组件中显示的项		()
labelmargin	如果 labelpos 选项非空，此项给出 label 组件与其他窗口控件之间的距离	距离	0

（续）

选 项	说 明	单位	缺省值
labelpos	指出何处放置 label 组件。如果非空，它该是 N,S ,E 和 W 三个字母中一二个字母连在一起的字串。第一个指出放在大控件的哪侧。如果第二个字母给定，它指出该侧中何处放置 label。比如，如果 labelpos 是 W，label 被放在左侧中部；如果是 WN，则放在左侧顶部；如果是 WS，放在左侧底部	anchor	无

组件

hull

这作为整个控件的主体。其他组件创建为 hull 的子组件。

label

如果 labelpos 非空，组件作为窗口控件的文本标签而被创建。详细参见 labelpos。注意，例如要设置标签的文本选项，你必须使用 label_text 组件选项。

menu

当菜单按下时显示的弹出菜单。

menubutton

显示当前选定值的菜单。

方法

get()

获得当前选定值。

index(index)

返回 index 相对应的菜单项数值索引。可以按如下方式指定：

- end 最后一个菜单项。
- nme 菜单项标签名。
- None 当前选定菜单。

invoke(index=None)

调用此方法如同选定 index 给定的菜单项，意思是 menubutton 组件显示的文本被更新，且 command 指定的函数被调用。index 可以是 index()接受的所有格式。

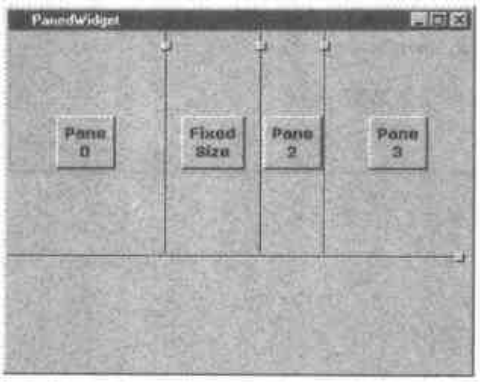
setitems(items,index=None)

用 items 序列中的项代替菜单组件中的项。如果 index 非空，它设置选中项为 index，它可以是任何 index()接受的格式。如果 index 为空，菜单组件的 textvariable 选项为空，它设置选定项为 items 中的第一项。如果 items 为空，选定值为空。

如果 index 为空且菜单按钮组件的 textvariable 选项为非空串，它不设置选定值。这

时假设变量已经（或将被）设到期望的值了。

PanedWidget（窗格控件）



说明

该类创建一个管理窗口，其中包含被称作窗格的大小可变的框架。每个窗格中可以包含其他窗口。用户可以通过拖拉小方形句柄或窗格间的分隔线来改变窗格大小。

继承关系

PanedWidget 继承自 Pmw.MegaWidget。

PanedWidget 选项

选　项	说　明	单位	缺省值
command	指向一函数。该函数在任一窗格大小发生改变时将被调用。该函数只有一个选项，该选项是依次描述窗格大小的列表。垂直方向上，窗格的大小即为该窗格的高度；水平方向上，窗格的大小即为该窗格的宽度	函数	无
orient	表示窗格化的窗口的位置方向。可以是 HORIZONTAL 或 VERTICAL。当为 VERTICAL 时，窗格上下依次排列；否则窗格前并肩排列	常量	VERTICAL
separatorrelief	表示窗格间的分隔线的凸凹状态	常量	SUNKEN

Pane 窗格选项

选　项	说　明	单位	缺省值
Size	表示窗格的起初大小	整型或实型	0
min	表示窗格的最小大小	整型或实型	0
max	表示窗格的最大大小	整型或实型	100000

组件

hull

表示整个大控件主体。可以创建作为其子体的其他特定组件。

方法

add(name,**kw)

在窗格化的窗口末端插入一个组件名为 name 的窗格控件。该函数也等价于使用 before 选项在当前多个窗格中调用 insert()函数。该函数返回名为 name 的子窗格。

configurepane(name,**kw)

设置名为 name 的窗格。其中 name 可以是一个表示该窗格索引的整型数，也可以是表示该窗格名称的一字符串。关键字 kw 选项用以表示该窗格选项的值。窗格选项的说明参见前面 Pane 选项一部分。

insert (name,before=0,**kw)

在名为 before 的窗格前面（即在其上或其左）插入一窗格。其中 before 可以是一个表示该窗格索引的整型数，也可以是表示该窗格名称的一字符串。关键字 kw 选项用以表示该窗格选项的起始值。窗格选项的说明参见前面 Pane 选项一部分。在窗格化窗口的末端加入一个窗格可用 add()函数。

pane(name)

返回由 name 指定的窗格的 Frame 窗格控件，其中 name 可以是一个表示该窗格索引的整型数，也可以是表示该窗格名称的一字符串。

panes(name)

按显示顺序返回所有窗格的名称列表。

remove(name)

删除名为 name 的窗格，其中 name 可以是一个表示该窗格索引的整型数，也可以是表示该窗格名称的一字符串。

PromptDialog（提示对话框）



说明

提示对话框是一个要求用户进行输入的方便的对话框。

继承关系

PromptDialog 继承自 Pmw.Dialog。

PromptDialog 选项

选　项	说　明	单位	缺省值
activatecommand	如果该函数能被调用，那么只要调用 activate()函数激活了大控件，就调用该函数	函数	None
Borderx	表示消息区域从左边界到右边界的宽度	距离	20
bordery	表示消息区域从上边界到下边界的高度	距离	20
buttonboxpos	表示在对话框窗口的哪一侧放置按钮框，必须为 N,S,E,W 其中之	锚点	S
buttons	表示按钮框中所有按钮的名称，必须为一元组或一列表。	(string,…)	('OK')

（续）

选　项	说　明	单位	缺省值
command	指向一调用函数。该函数在按钮框中某一按钮被击中时或在窗口管理器中删除该窗口时被调用。该函数被调用时，只有一个选项，其为被击中的按钮的名称，或在窗口管理器中删除该窗口时该选项为 None	函数	None
deactivatecommand	如果该函数能被调用，那么只要调用 deactivate()函数激活了大控件时，就调用该函数	函数	None
defaultbutton	表示按钮框中的缺省按钮。当当前高亮对话框中的 Return 键被击中时，缺省按钮将被调用。若 defaultbutton 为 None，则不存在缺省按钮，击中 Return 键无效	索引	None
separatorwidth	若其大于 0，则依据指定宽度在按钮框和子位置间创建一被称作分隔线的窗口部件。如果按钮框的缺省边界和子位置被增大，则没有必要设置该选项。因为在按钮框与子位置间将存在一条可见的分隔线	距离	0
title	显示在窗口管理器的窗口标题栏上的标题	字符串	None

组件

buttonbox

包含对话框中所有的按钮。缺省时使用选项（hull_borderwidth=1,hull_relief='raised'）来创建。

dialogchildsite

表示对话框的子位置。可以通过在大控件中创建其他控件来生成特定控件。缺省时使用选项（borderwidth=1,relief='raised'）来创建。

entryfield

缺省时该组件为 Pmw.EntryField

hull

表示整个大控件主体。可以创建作为其子体的其他特定组件。

separator

如果 separatorwidth 不为 0，则分隔线为按钮框与子位置间的一条直线。

方法

deleteentry(first,last=None)

从以 first 为始，last 为尾的 entryField 中删除字符。first 和 last 为整型索引。若 last 为 None，则 first 将被删除。

indexentry(index)

返回与 index 相对应的字符的数字式索引。

insertentry（index,text）
在整型数位置 index 处插入 text。

RadioSelect（单选框）

说明

该类创建一个含有按钮的管理器窗口。按钮可以按水平或垂直排列。在单选方式下，任何时刻只能选中一个按钮；在复选方式下，某个时刻可同时选择多个按钮，并且单击一个已被选中的按钮将会取消选中。

被显示的按钮可以是标准按钮、单选框或复选框。标准按钮被选中时，显示成凹状；单选和复选框被选中时，将显示相应的标示色或凸状。

继承关系

RadioSelect 继承自 Pmw.MegaWidget。

RadioSelect 选项

选　项	说　明	单位	缺省值
buttontype	表示用 add()函数创建的按钮的缺省类型。若创建的为按钮，则缺省类型为 button；若为单选框，则为 radiobutton；若为复选框，则为 checkbutton	常量	无
command	指向一调用函数，该函数在某个按钮被击中或激活 invoke()时被调用	函数	无
labelmargin	若 labelpos 选项不是 None，该选项表示标签窗口部件与主控件其他部分间的距离	距离	0
labelpos	表示放置标签部件的位置。若不是 None，必须为 N,S,E 或 W 中一个或二个字母的组合。第一个字母表示在大控件的哪一侧放置标签。若第二个字母也被设置，则其表示在大控件该侧的何处放置标签。例如：若 labelpos 的值为 W，则表示标签放在窗口左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端	锚点	无
orient	指定按钮放置方向，可以使 HORIZONTAL 或 VERTICAL	常量	HORIZONTAL
padx	表示每个按钮间，以及按钮与单选框外边界间在 x 方向上的间距	距离	5
pady	表示每个按钮间，以及按钮与单选框外边界间在 y 向上的间距	距离	5
selectmode	表示选择方式：即在某个时刻，能否选择单个按钮或多个按钮。若为 single 方式，则单击某个未被选中的按钮将选中该按钮，同时取消其他选择。若为 multiple 方式，则单击某个未被选中的按钮将选中该按钮，单击某个已被选中的按钮将取消选中该按钮。如果 buttontype 为 radiobutton 或 checkbutton，该选项可以忽略	常量	'Single'

组件

frame

若已创建了标签组件（即 labelpos 不为 None），则也将创建一框架组件，以作为通过 add()方法创建的按钮的容器。如果不存在标签，则也不必创建框架，而用 hull 作为容器。

hull

表示整个大控件主体。可以创建作为其子体的其他特定组件。

label

如果 labelpos 为非 None，该组件将作为大控件的字符标签而被创建。参见 labelpos 说明。注意，例如在设置标签的字符选项时，用户必须使用 label_tex 组件选项。

方法

add(name,**kw)

在单选框控件的末端加入一个按钮，作为名为 name 的组件。该按钮的缺省类型由 buttontype 设置。在创建该按钮时，当前所有关键字参数（除 command）都将传递给构建器。如果没有设置 text 关键字参数，则该按钮的 text 选项将缺省为 name。该函数返回 name 窗口组件。

deleteall()

删除所有按钮，并取消当前的选择。

Getcurselection()

在单选模式下，返回当前被选中的按钮名称。若当前没有按钮被选中，则返回 None；在复选模式下，返回当前所有被选中的按钮名称列表。

Index(index)

返回与 index 相对应的按钮的数字索引。可以为以下几种形式之一：

- number: 用数字来表示按钮。数字 0 表示最左或最顶端按钮。
- end: 指右或底端按钮。
- name: 表示名为 name 的按钮。

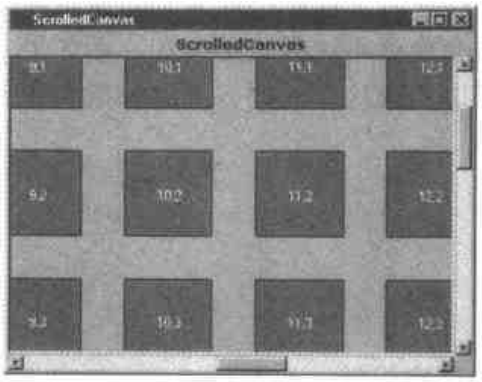
invoke(index)

调用该方法等同于单击名为 index 的按钮。所有按钮将依据 selectionmode 显示成被选中或未被选中状态，command 函数将被调用。index 可以是能被 index()函数接受的任何形式之一。

numbuttons()

返回单选窗口中的按钮数目。

ScrolledCanavas



说明

此大控件中包含一个标准画布控件和可选滚动条。该滚动条可用于滚动画布控件。滚动条是动态的，即只有在必须的时候才会显示。例如若画布的滚动区域比画布窗口大，则滚动条将显示。

继承关系

ScrolledCanavas 继承自 Pmw.MegaWidget。

ScrolledCanavas 选项

选　项	说　明	单位	缺省值
borderframe	一框架控件把画布窗口紧绕着，画布窗口没有边界，而框架控件有边界，从而使画布窗口看起来也像有边界一样	widget	frame
canvasmargin	画布四周的边缘。可用 resizescrollregion()来操作	距离	0
hscrollmode	水平滚动模式。若为 None，则总不显示水平滚动条；若为 static，则滚动条将总被显示；若为 dynamic，则滚动条视具体需要而显示	常量	dynamic
labelmargin	若 labelpos 非 None，则该选项表示标签组件与大控件其余部分间的距离	距离	0
labelpos	表示放置标签组件的位置。若不是 None，必须为 N,S,E 或 W 中一个或二个字母的组合。第一个字母表示在大控件的哪一侧放置标签。若第二个字母也被设置，则其表示在大控件该侧的何处放置标签。例如，若 labelpos 的值为 W，则表示标签放在左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端	锚点	无
scrollmargin	表示滚动条与画布控件窗口间的距离	距离	0
usehullsize	若为 true，则大控件的大小将只由主体组件的高度和宽度来决定。若为 false，则大控件的大小将只由画布控件窗口的高度，以及其他窗口部件，如标签、滚动条的大小和 scrollmargin 选项来决定。所有这些都将影响大控件的总体大小	布尔	0
vscrollmode	垂直滚动模式。若为 None，则总不显示垂直滚动条；若为 static，则滚动条将总被显示；若为 dynamic，则滚动条视具体需要而显示	常量	dynamic

组件

borderframe

一框架控件把画布窗口紧绕着，画布控件没有边界，而框架部件有边界，从而使画

布控件窗口看起来也像有边界一样。

canvas

画布控件可由滚动条滚动。在按位置放置小控件时存在一个问题：如果画布窗口内部的一个控件超出画布窗口框的某个边界时，该控件将会使画布边界重叠模糊。而如果 borderframe 为 true，并以 0 宽度来创建画布窗口，则可以克服这个问题。因此如果画布窗口没有边界，则就不可能产生重叠。

horizscrollbar

水平滚动条。其构件组为 Scrollbar。

hull

表示整个大控件主体。可以创建作为其子体的其他特定组件。

label

如果 labelpos 为非 None，该组件将作为大控件的字符标签而被创建。参见 labelpos 说明。注意，例如在设置标签的字符选项时，用户必须使用 label_text 组件选项。

vertscrollbar

垂直滚动条。其组件组为 Scrollbar。

方法

bbox(*args)

该方法将明确地被发送至列表框的 bbox()方法。若不明确发送，将激活大控件的 bbox()方法（别名 grid_bbox()），而且可能产生与用户想法不一致的结果。

interior()

返回一画布控件窗口。程序员在其内部创建图形部件和子窗口。类似 component('canvas')。

resizescrollregion()

按 canvasmargin 选项改变画布控件的滚动区域，使其将画布控件上所有组件及其四周的边缘都覆盖上。

ScrolledField（滚动域）



说明

该大控件中显示一单行文本。如果文本长度大于窗口，用户可以拖拉鼠标的中键来左右滚动文本内容，也可以单击或拖拉鼠标左键来选择文本。在显示例如应用程序状态等不知道长度的文本时可以代替 Label。

继承关系

ScrolledField 继承自 Pmw.MegaWidget。

ScrolledField 选项

选 项	说 明	单位	缺省值
labelmargin	若 labelpos 选项不是 None，该选项表示标签窗口组件与大控件其他部分间的距离	距离	0
labelpos	表示放置标签组件的位置。若不是 None，必须为 N,S,E 或 W 中一个或二个字母的组合。第一个字母表示在大控件哪一侧放置标签。若第二个字母也被设置，则其表示在该侧的何处放置标签。例如，若 labelpos 的值为 W，则表示标签放在窗口左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端	锚点	None
text	在滚动显示区域中显示的文本内容	字符串	None

组件

entry

用以显示文本，并且容许用户滚动、选择文本。当该组件状态设为 disabled 时，用户就不能改动文本。

hull

表示整个大控件主体。可以创建作为其子体的其他特定组件。

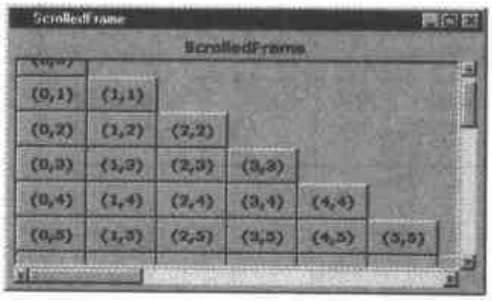
label

如果 labelpos 为非 None，该组件将作为大控件的字符标签而被创建。参见 labelpos 说明。注意，例如在设置标签的字符选项时，用户必须使用 label_text 组件选项。

方法

无。

ScrolledFrame（滚动框架）



说明

该大控件在一剪贴框架中包含一可滚动内部框架。程序员可以在该内部框架中创建其他控件。如果该内部框架比环绕在其周围的剪贴框架大，则用户可以使用水平和垂直滚动条来定位该框架。滚动条是动态的，即只有在必须的时候才会显示，例如若框架比环绕的剪贴框架小，则滚动条将不显示。

继承关系

ScrolledFrame 继承自 Pmw.MegaWidget。

ScrolledFrame 选项

选 项	说 明	单位	缺省值
borderframe	一框架窗口部件被剪贴框架紧绕着，剪贴框架没有边界，而框架窗口部件有边界，从而使剪贴框架看起来也像有边界一样	窗口部件	frame
horizflex	表示相对于剪贴框架如何改变可滚动内部框架的宽度。若宽度固定，则内部框架按该框架的组件所要求的自然宽度来设置。如果加大宽度或内部框架要求的自然宽度小于剪贴框架，则内部框架将拉大以与剪贴框架相符合。如果减小宽度或内部框架要求的自然宽度大于剪贴框架的宽度，则内部框架将减小以适合剪贴框架。若该选项值为 elastic，则表示内部框架的宽度永远设置成等于剪贴框架宽度	常量	fixed
horizfraction	表示在用户单击水平滚动箭头以滚动内部框架时，剪贴框架的宽度	距离	0.05
hscrollmode	水平滚动模式。若为 none，则总不显示水平滚动条；若为 static，则滚动条将总被显示；若为 dynamic，则滚动条视具体需要而显示	常量	dynamic
labelmargin	若 labelpos 为非 none，则该选项表示标签组件与大控件其他部分间的距离	距离	0
labelpos	表示放置标签部件的位置。若不是 None，必须为 N,S,E 或 W 中一个或二个字母的组合。第一个字母表示在大控件哪一侧放置标签。若第二个字母也被设置，则其表示在该侧的何处放置标签。例如，若 labelpos 的值为 W，则表示标签放在窗口左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端	锚点	无
scrollmargin	表示滚动条与剪贴框架间的距离	距离	2
usehullsize	若为 true，则大控件的大小将只由主体组件的高度和宽度来决定。若为 false，则大控件的大小将只由剪贴框架的高度，以及其他窗口部件，如标签、滚动条的大小和 scrollmargin 选项来决定。所有这些都将影响大控件的总体大小	布尔	0
vertflex	表示相对于剪贴框架，如何改变可滚动内部框架的高度。若高度固定，则内部框架按该框架的窗口部件所要求的自然高度来设置。如果加大高度或内部框架要求的自然高度小于剪贴框架，则内部框架将拉大以与剪贴框架相符合。如果减小高度或内部框架要求的自然高度大于剪贴框架的高度，则内部框架将减小以适合剪贴框架。若该选项值为 elastic，则表示内部框架的高度永远设置成等于剪贴框架高度	常量	fixed
verfraction	表示在用户单击垂直滚动箭头以滚动内部框架时，剪贴框架的高度	距离	0.05
vscrollmode	垂直滚动模式。若为 none，则总不显示垂直滚动条；若为 static，则滚动条将总被显示；若为 dynamic，则滚动条视具体需要而显示	常量	dynamic

组件

borderframe

一框架控件被剪贴框架紧绕着，剪贴框架没有边界，而框架控件有边界，从而使剪贴框架看起来也像有边界一样。

clipper

剪贴框架是为框架组件提供剪贴视图的。在按位置放置控件时存在一个问题：如果控件（当前即为框架组件）放置在框架（当前即为剪贴框架）内部，且前者超出框架的某个边界时，该控件将会使框架的边界重叠模糊。而如果 borderframe 为 true，并以 0 宽度来创建剪贴框架，则可以克服这个问题。因此如果剪贴框架没有边界，则就不可能产生重叠。

frame

在剪贴框架的内部包含可以滚动的窗口部件。

horizscrollbar

水平滚动条。其构件组为 Scrollbar。

hull

参见 scrolledField 的 hull 组件。

label

参见 scrolledField 的 label 组件。

vertscrollbar

垂直滚动条。其组件组为 Scrollbar。

方法

interior()

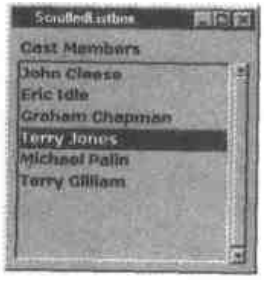
返回一框架。程序员在该框架内部创建可滚动的控件，与 component('frame')同。

reposition()

更新剪贴框架中框架组件的位置和滚动条的位置。

通常地，该方法不必特意地被调用，因为当框架组件或剪贴组件的大小发生改变时，或当用户点击滚动条时，框架组件和滚动条的位置会自动更新。然而，若 horizflex 或 vertflex 被增大，则大控件不能自动探测到被增大的框架已超出剪贴框架窗口的大小。因此在最初的大控件已创建好之后再在框架中加上新的控件时（或增大一个控件的大小时），必须调用该函数。

ScrolledListBox（滚动列表框）



说明

该大控件中包含一标准列表框控件。该窗口带有可选滚动条以用于滚动列表框。滚动条是动态的，即只有在必需的时候才显示。如列表框中记录数不够，则垂直滚动条将自动隐藏起来，并且若记录的长度比窗口小，则水平滚动条将自动地不显示。

继承关系

ScrolledListBox 继承自 Pmw.MegaWidget。

ScrolledListBox 选项

选　项	说　明	单位	缺省值
dbleclickcommand	指向一函数。该函数在鼠标 1 键双击列表框中某条记录时被调用	函数	无
hscrollmode	水平滚动模式。若为 none，则总不显示水平滚动条；若为 static，则滚动条将总被显示；若为 dynamic，则滚动条视具体需要而显示	常量	dynamic
items	一个包含了所有在列表框中显示的最初记录的元组	(string,…)	()
labelmargin	若 labelpos 为非 None，则该选项表示标签组件与大控件其余部分的距离	距离	0
labelpos	表示放置标签控件的位置。若不是 None，必须为 N,S,E 或 W 中一个或二个字母的组合。第一个字母表示在大控件哪一侧放置标签。若第二个字母也被设置，则其表示在该侧的何处放置标签。例如，若 labelpos 的值为 W，则表示标签放在窗口左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端	锚点	无
scrollmargin	表示滚动条与列表框窗口间的距离	距离	2
selectioncommand	指向一函数。该函数在鼠标 1 键单击列表框中某条记录时被调用	函数	无
usehullsize	若为 true，则大控件的大小将只由主体组件的高度和宽度来决定。若为 false，则大控件的大小将只由列表框的高度，以及其他组件，如标签、滚动条的大小和 scrollmargin 选项来决定。所有这些都将影响大控件的总体大小	布尔型	0
vscrollmode	垂直滚动模式。若为 None，则总不显示垂直滚动条；若为 static，则滚动条将总被显示；若为 dynamic，则滚动条视具体需要而显示	常量	dynamic

组件

horizscrollbar
水平滚动条。其组件组为 Scrollbar。

hull
参见 scrolledField 的 hull 组件。

label
参见 scrolledField 的 label 组件。

listbox
列表框可由滚动条滚动。

vertscrollbar
垂直滚动条。其组件组为 Scrollbar。

方法

bbox(index)

该方法将明确地被发送至列表框的 bbox()函数。若不明确发送，将激活大控件的 bbox()函数（别称 grid_bbox()），而且可能产生与用户想法不一致的结果。

get(first=None,last=None)

返回所有列表元素，除非 first 和 last 都未指定。该函数同于对列表框使用 get()函数。

getcurselection()

返回当前列表框中被选中的项目。该函数返回被选中的项目的文本，而不是如同 curselection()返回其索引。

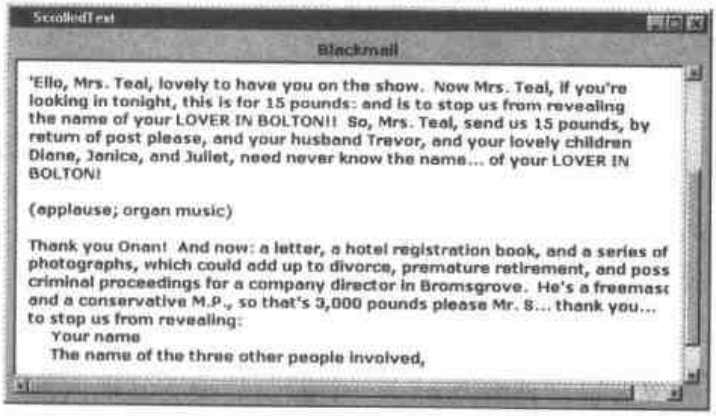
setlist(items)

用项目序列所指定的记录来代替列表框中所有的记录。

size()

该方法将明确地被发送至列表框的 size()函数。若不明确发送，将激活大控件的 size()函数（别称 grid_size()），而且可能产生与用户想法不一致的结果。

ScrolledText（滚动文本框）



说明

该大控件包含一个标准文本框。该文本框带有可选滚动条以用于滚动文本框。滚动条是动态的，即只有在必需的时候才显示。如文本框中的文本不是足够多（水平或垂直方向上），滚动条将自动隐藏起来。

继承关系

ScrolledText 继承自 Pmw.MegaWidget。

ScrolledText 选项

选 项	说 明	单位	缺省值
borderframe	框架控件把文本控件紧绕着，文本控件没有边界，而框架组件有边界，从而使文本窗口看起来也像边界一样	控件	frame
hscrollmode	水平滚动模式。若为 nne，则总不显示水平滚动条；若为 static，则滚动条将总被显示；若为 dynamic，则滚动条视具体需要而显示	常量	dynamic

（续）

选　项	说　明	单位	缺省值
labelmargin	若 labelpos 为非 None，则该选项表示标签组件与大控件其他部分的距离	距离	0
labelpos	表示放置标签组件的位置。若不是 None，必须为 N,S,E 或 W 中一个或二个字母的组合。第一个字母表示在大控件哪一侧放置标签。若第二个字母也被设置，则其表示在该侧的何处放置标签。例如，abelpos 的值为 W，则表示标签放在窗口左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端	锚点	None
scrollmargin	表示滚动条与文本窗口间的距离	距离	2
usehullsize	若为 true，则大控件的大小将只由大控件高度和宽度来决定。若为 false，则大控件的大小将只由文本窗口的高度，以及其他组件，如标签、滚动条的大小，和 scrollmargin 选项来决定。所有这些都将影响大控件的总体大小	布尔	0
vscrollmode	垂直滚动模式。若为 none，则总不显示垂直滚动条；若为 static，则滚动条将总被显示；若为 dynamic，则滚动条视具体需要而显示	常量	dynamic

组件

borderframe

一框架控件被剪贴框架紧绕着，剪贴框架没有边界，而框架控件有边界，从而使剪贴框架看起来也像有边界一样。

horizscrollbar

水平滚动条。其组件组为 Scrollbar。

hull

参见 scrolledField 的 hull 组件。

label

参见 scrolledField 的 label 组件。

text

文本窗口框可由滚动条滚动。在按位置放置控件时存在一个问题：如果文本框内部的一个组件超出文本框的某个边界，该组件将会使文本框的边界重叠模糊。而如果 borderframe 为 true，并以 0 宽度来创建文本框，则可以克服这个问题。因为如果文本框没有边界，则就不可能产生重叠。

vertscrollbar

垂直滚动条。其构件组为 Scrollbar。

方法

bbox(index)

该方法将明确地被发送至文本框的 bbox()函数。若不明确发送，将激活大控件的

bbox()函数（别称 grid_bbox()），而且可能产生与用户想法不一致的结果。

clear()
删除文本框中所有的字符。

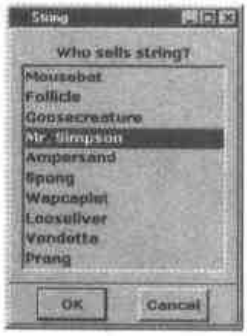
exportfile(fileName)
将文本框中的内容写到 fileName 文件中。

get(first=None,last=None)
返回文本框中所有的内容，除非 first 和 last 都未指定。该函数同于对文本框使用 get()函数。

importfile(fileName,where='end')
从 fileName 文件中读出内容，并在按 where 给定的位置处显示在文本框中。

settext(text)
用 text 代替文本框中所有的内容。

SelectionDialog（选择对话框）



说明

选择对话框是一个具有滚动列表框的简便对话框，它要求用户从滚动列表框中进行选择。

继承关系

SelectionDialog 继承自 Pmw.Dialog。

SelectionDialog 选项

选　项	说　明	单位	缺省值
activatecommand	如果该函数能被调用，那么只要调用 activate()函数激活大控件，就调用该函数	函数	None
borderx	表示消息区域从左边界到右边界的宽度	距离	10
bordery	表示消息区域从上边界到下边界的高度	距离	10
buttonboxpos	表示在对话框窗口的哪一侧放置按钮框。必须为 N,S,E,W 其中之一	锚点	S
buttons	表示按钮框中所有按钮的名称。必须为一元组或一列表	(String,…)	('OK')
command	指向一调用函数。该函数在按钮框中某一按钮被击中时或在窗口管理器中删除该窗口时被调用。该函数被调用时，只有一个选项，其为被击中的按钮的名称，或在窗口管理器中删除该窗口时该选项为 None	函数	None
deactivatecommand	如果该函数能被调用，那么只要调用 deactivate()函数激活大控件，就调用该函数	函数	None

（续）

选　项	说　明	单位	缺省值
defaultbutton	表示按钮框中的缺省按钮。当当前高亮对话框中的 Return 键被击中时，缺省按钮将被调用。若 defaultbutton 为 None，则不存在缺省按钮，击中 Return 键无效	索引	None
separatorwidth	若其大于 0，则依据指定宽度在按钮框和子位置间创建一被称作 separator 的组件。如果按钮框的缺省边界和子位置被增大，则没有必要设置该选项。因为在按钮框与子位置间将存在一条可见的分隔线	距离	0
title	显示在窗口管理器的窗口标题栏上的标题	字符串	None

组件

buttonbox

包含对话框中所有的按钮。缺省时使用选项（hull_borderwidth=1,hull_relief='raised'）来创建。

dialogchildsite

表示对话框的子位置。可以在大控件中创建其他特定控件。缺省时使用选项（borderwidth=1,relief='raised'）来创建。

hull

参见 scrolledField 的 hull 组件。

scrolledlist

缺省地，该组件是 Pmw. ScrolledlistBox。

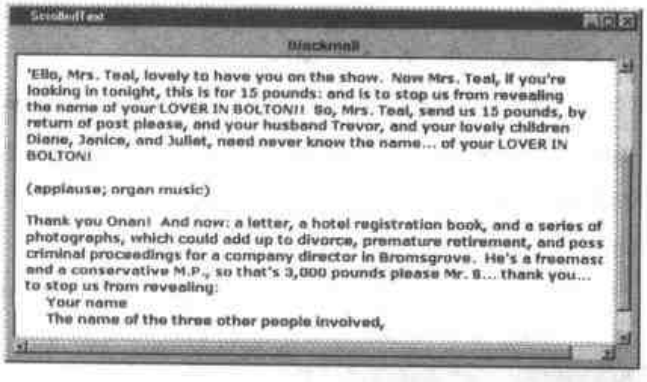
separator

如果 separatorwidth 不为 0，则分隔线为按钮框与子位置间的一条直线。

方法

无。

TextDialog（文本对话框）



说明

文本对话框是一个含有滚动文本框的对话窗口，它用于向用户显示多行文本。

继承关系

TextDialog 继承自 Pmw.Dialog。

TextDialog 选项

选 项	说 明	单位	缺省值
activatecommand	如果该函数能被调用，那么只要调用 activate()函数激活大控件时，就调用该函数	函数	None
borderx	表示消息区域从左边界到右边界的宽度	距离	10
bordery	表示消息区域从上边界到下边界的高度	距离	10
buttonboxpos	表示在对话框窗口的哪一侧放置按钮框。必须为 N,S,E,W 其中之一	锚点	S
buttons	表示按钮框中所有按钮的名称。必须为一元组或一列表	(String,…)	('OK')
command	指向一调用函数。该函数在按钮框中某一按钮被击中时或在窗口管理器中删除该窗口时被调用。该函数被调用时，只有一个选项，其为被击中的按钮的名称，或在窗口管理器中删除该窗口时该选项为 None	函数	None
deactivatecommand	如果该函数能被调用，那么只要调用 deactivate()函数激活大控件时，就调用该函数	函数	None
defaultbutton	表示按钮框中的缺省按钮。当当前高亮对话框中的 Return 键被击中时，缺省按钮将被调用。若 defaultbutton 为 None，则不存在缺省按钮，击中 Return 键无效	索引	None
separatorwidth	若其大于 0，则依据指定宽度在按钮框和子位置间创建一被称作 separator 的组件。如果按钮框的缺省边界和子位置被增大，则没有必要设置该选项。因为在按钮框与子位置间将存在一条可见的分隔线	距离	0
title	显示在窗口管理器的窗口标题栏上的标题	字符串	None

组件

buttonbox

包含对话框中所有的按钮。缺省时使用选项（hull_borderwidth=1,hull_relief='raised'）来创建。

dialogchildsite

表示对话框的子位置。可以在大控件中创建其他特定控件。缺省时使用选项（borderwidth=1,relief='raised'）来创建。

hull

表示整个大控件主体。可以创建作为其子体的其他特定组件。

scrolledtext

缺省地，该部件是 Pmw. ScrolledText。

Separator

如果 separatorwidth 不为 0，则分隔线为按钮框与子位置间的一条直线。

方法

无。

TimeCounter（计时器）



说明

计时器窗口有三个上/下计数器，一起用于容许用户输入时间。每增大 60 秒或 60 分钟，相应地就增大 1 分钟或 1 小时。

继承关系

TimeCounter 继承自 Pmw.MegaWidget。

TimeCounter 选项

选　项	说　明	单位	缺省值
autorepeat	若 autorepeat 为 true，在 initwait 中继之后，上下按钮将激活每个 repeatrate 毫秒	布尔型	1
buttonaspect	表示箭头按钮的高宽比。小于 1.0 则为瘦形箭头按钮，大于 1.0 为胖形	浮点型	1.0
initwait	若 autorepeat 为 true，则窗口部件在反复每次激活上下按钮之前将等待 initwait 毫秒	毫秒	300
labelmargin	若 labelpos 为非 None，则该选项表示标签组件与大控件其他部分间的距离	距离	0
labelpos	表示放置标签部件的位置。若不是 None，必须为 N,S,E 或 W 中一个或二个的组合。第一个字母表示在大控件哪一侧放置标签。若第二个字母也被设置，则其表示在该侧的何处放置标签。例如，若 labelpos 的值为 W，则表示标签放在窗口左侧中央；若为 WN，则表示标签放在左侧顶端；若为 WS，则表示标签放在左侧底端	锚点	无
max	窗口中显示的最大值。23:59:59 表示 24 小时计时	字符串	
min	窗口中显示的最小值，通常为 0:0:0。或作为缺省值保留		
padx	表示在 x 方向上，每个按钮间的间距	距离	0
pady	表示在 y 方向上，每个按钮间的间距	距离	0
repeatrate	若 autorepeat 为 true，则表示某个按钮被持续按中时被多次响应的间隔时间	毫秒	50
value	窗口中显示的最初值		

组件

downhourarrow

其组件组为 Arrow。

downminutearrow
其组件组为 Arrow。

downsecondarrow
其组件组为 Arrow。

frame
缺省时，该组件是一个 Frame。

hourentryfield
缺省时，该组件是一个 Pmw.EntryField

hull
表示整个大控件主体。可以创建作为其子体的其他特定组件。

label
如果 labelpos 为非 None，该组件将作为大控件的字符标签而被创建。参见 labelpos 说明。注意，例如在设置标签的字符选项时，用户必须使用 label_text 组件选项。

minuteentryfield
缺省时，该组件是一个 Pmw.EntryField。

secondentryfield
缺省时，该组件是一个 Pmw.EntryField。

uphourarrow
其组件组为 Arrow。

upminutearrow
其组件组为 Arrow。

upsecondarrow
其组件组为 Arrow。

方法

decrement()
将时间减少 1 秒。

getint()
将时间以整型数返回。

getstring()
将时间按 HH:MM:SS 格式以字符串返回。

increment()
将时间增加 1 秒。

invoke（）
若 command 能被调用，其将调用 command。

附录 D　创建和安装 Python,Tkinter

通常，你不必从源头创建 Python 或其组件，你可以轻易从 www.python.org 得到针对 Unix 的几个变种，Win32 和 MacOS 平台下的二进制发行版本。但是，如果你想扩展 Python，你必须获取到 Python 的源代码，有时甚至包括 Tcl 及 Tk。

如果你确实决定创建 Python,你可以一样创建 Tcl/Tk。这里提供的 1.5.2 版本的 Python 和 8.0.5 版本的 Tcl/Tk 信息，是稳定的发行版本。要升级版本，需要访问相关网页，获取升级信息。

我们按顺序说明 Unix、Win32 和 MacOS 下的创建。

Unix 下的创建

和其他操作系统相比，Unix 下创建 Python 和 Tkinter 是最直接的。对我个人而言，我已经有两个 Unix 系统，它给我带来了麻烦和问题，这归咎于它们是新系统。

在开始之前，你应该得到适当的源代码。

获取源代码

收集任何源代码之前，决定你将在什么地方安装源文件。你将有三个目录（Python、Tcl 和 Tk），每个以它们的版本为后缀。你要安排这些目录在磁盘相同层次的目录上(比如，所有的都在/python_source)。

其次，到 www.python.org 获取最新版本的 Python（你通常会在主题页面找到当前版本的参考手册）。这将指引你哪个文件是当前的 gzipped tar 文件。在此页上面，找出哪个版本是 Python 的二进制发布版本，这是你要用来创建 Tkinter 的版本。获取代码，放到你建的目录下，按如下解压源文件（把黑体数字替换为当前版本数）。

```
gunziip - c py152.tgz | tar xf -
```

Tcl/Tk 可以从 www.scriptics.com/procucts/tcltk 获取。从这页，你会发现当前版本的参考信息。如果你不想，可以不创建 Tcl/Tk，Tcl/Tk 的二进制版本包括库都可以得到。你很有可能会找到一个比发布 Python 更新的版本，通常，这样的软件包修改了一些 bug，问题会减少，然后，不要被引诱使用最新的版本（比如，若 Python 最初与 8.0.5 一起创建，有三个更新版本，8.0.6，8.1.0 和 8.2.0，选择 8.0.6）。获取两个压缩文件，放到你建的目录下，按如下解压源文件（把黑体数字替换为当前版本数）。

```
gunziip - c tcl8.0.5 tgz | tar xf -
gunziip - c tk8.0.5.tgz | tar xf -
```

你现在必须按照 Tcl、Tk 和 Python 的顺序来安装了。

创建 Tcl

改变目录到 Unix 下的 Tcl 目录，在那里你会发现一个 readme 文件，给出了创建 Tcl 的详细信息。下面是你需要阅读的部分。当然，某些 Unix 有特定的问题，你需要仔细阅读 readme 文件的全部或通过网络咨询。

1. 决定在什么地方安装可执行文件和库文件。我们假设安装目录为/usr/local（缺省）。

2. 运行配置脚本。它自动决定编译选项和创建使用的系统资源。

```
./configure
```

3. 运行实用程序创建 Tcl 库。

```
make
```

4. 安装可执行文件和库文件。

```
make install
```

假设你没有碰到问题，你可以继续创建 Tk 了。

创建 Tk

创建 Tk 与创建 Tcl 相似，改变 Unix 目录到到 Tk 目录之下。在此目录下，你会看到一个完整的 ReadMe 文件，给出创建 Tk 的完整详细信息，下面是我们需要阅读的大致内容。

1. 决定在什么地方安装可执行文件和库文件。我们假设安装目录为/usr/local（缺省）。

2. 运行配置脚本。它自动决定编译选项和创建使用的系统资源。

```
./configure
```

3. 运行实用程序创建 Tcl 库。

```
make
```

4. 安装可执行文件和库文件。

```
make install
```

我们现在已经准备好安装 Python 了。

创建 Python

创建 Python 与创建 Tcl 和 Tk 极为相似，需要多做的工作是配置创建程序来添加 Tkinter,它不是缺省安装的。你会在 Python 顶层目录发现 ReadMe 文件，它给出不同 Unix 版本下面区别的详细说明。再次，大致内容：

1. 我们假设安装目录为 usr/local/(缺省)，不一定要和 Tcl/Tk 同目录，但也没有必要不同。

2. 运行配置脚本。它自动决定编译选项和创建使用的系统资源。注意如果你打算使用线程，你必须添加－with－thread 选项来配置。与平台有关问题，请详细阅读 ReadMe 文件。

```
./configure
```

3. 把 Modules/Setup.in 拷贝到 Modules/Setup 目录下。此文件用来决定哪些内嵌模

块需要加到 Python 中去。此时，我们只关心创建 Tkinter。你会发现可以从创建程序添加或除去与平台有关的模块。

4. 编辑 Modules/Setup，定位注释为下面的行：

```
# The _tkinter module.
```

5. 按照文件中的指示，这里举的例子适合 Solaris 2.5 或 2.6（粗体部分不是说明）

```
# The TKPATH variable is always enabled,to save you the effort.
TKPATH=:lib-tk
# The command for_tkinter is long and site-specific. Please
# uncomment and/or edit those parts as indicated.If you don't have a
# specific extension(e.g. Tix or BLT),leave the corresponding line
# commented out. (Leave the trailing backslashes in! If you
# experience strange errors,you may want to join all uncommented
# lines and remove the backslashes—the backslash interpretation is
# done by the shell's "read"command and it may not be implemented on
# every system.

# ***Always uncomment this (leave the leading underscore in!):
_tkinter tkinter.c tkappinit.c -DWITH_APPINIT \
# ***Uncomment and edit to reflect where your Tcl/Tk headers are:
-I/usr/local/include \
# ***Uncomment and edit to reflect where your X11 header files are:
# -I/usr/X11R6/include \
# ***Or uncomment this for Solaris:
-I/usr/openwin/include \
# ***Uncomment and edit for Tix extension only:
# -DWITH_TIX -Itix4.1.8.0 \
# ***Uncomment and edit for BLT extension only:
# -DWITH_BLT -I/usr/local/blt/blt8.0-unoff/include -lBLT8.0 \
# ***Uncomment and edit for PIL (TkImaging) extension only:
# -DWITH_PIL -I../Extensions/Imaging/libImaging tkImaging.c \
# ***Uncomment and edit for TOGL extension only:
# -DWITH_TOGL togl.c \
# ***Uncomment and edit to reflect where your Tcl/Tk libraries are:
-L/usr/local/lib \
# ***Uncomment and edit to reflect your Tcl/Tk versions:
-ltk8.0 -ltcl8.0 \
# ***Uncomment and edit to reflect where your X11 libraries are:
# -L/usr/X11R6/lib \
# ***Or uncomment this for Solaris:
-L/usr/openwin/lib \
# ***Uncomment these for TOGL extension only:
# -lGL -lGLU -lXext -lXmu \
# ***Uncomment for AIX:
# -lld \
# ***Always uncomment this;X11 libraries to link with:
-lX11
```

6. 如果你想创建模块为共享对象，取消包含*shared*的行的注解。所有后面的模块都将创建为独立共享模块。

7. 保存 Modules/Setup。你可能想拷贝一个备份，这样你可以在将来更高版本时知道哪些是你选择的配置。

8. 运行 make 应用程序，创建 Python 可执行文件和库文件。

```
make
```

9. 安装可执行文件和库文件

```
make install
```

10. 定义环境变量来反映选择的安装位置。

```
PATH=……:/usr/local/bin:……
PYTHONPATH=/usr/local/lib/python1.5
TCL_LIBRARY=/usr/local/lib/tcl8.0
Tk_LIBRARY=/usr/local/lib/tk8.0
```

在 Windows 下创建

在 Windows 下创建 Python 和 Tkinter 是很直接的。但是，和 Unix 相比，它涉及到一些编辑工作，尤其当你需要添加多余模块到 Python 时。虽然可以使用 Borland C 编译器来创建 Tcl/Tk，但 Python 需要微软的 Visual C++版本 5 或 6。

获取源代码

收集任何源代码之前，决定你将在什么地方安装源文件。你将有三个目录（Python、Tcl 和 Tk）。每个以它们的版本为后缀，你要安排这些目录在磁盘相同层次的目录上(比如，所有的都在 C:\python_source)。

其次，到 www.python.org 获取最新版本的 Python（你通常会在主题页面找到当前版本的参考手册）。这将指引你哪个文件是当前的 gzipped tar 文件。在此页上面，找出哪个版本是 Python 的二进制发布版本，这是你要用来创建 Tkinter 的版本。Windows(及 Macinttosh)版本的和 Unix 版本的源代码一样，虽然你可能需要下载 zip 压缩格式。获取代码，放到你建的目录下，假设你已经取得压缩版本，在管理器点击压缩文件，解压到选定目录。

同样，Tcl/Tk 可以从 www.scriptics.com/procucts/tcltk 获取。从这页，你会发现当前版本的参考信息。如果你不想，可以不创建 Tcl/Tk，Tcl/Tk 的二进制版本包括库都可以得到。你很有可能会找到一个比发布 Python 更新的版本，通常，这样的软件包修改了一些 bug，问题会减少，然后，不要被引诱使用最新的版本（比如，若 Python 最初与 8.0.5 一起创建，有三个更新版本，8.0.6，8.1.0 和 8.2.0，选择 8.0.6）。 获取两个压缩文件，放到你建的目录下，使用 Winzip 解压到选定目录。

你现在必须按照 Tcl,Tk 和 Python 的顺序来安装了。

创建 Tcl

改变目录到 Windows 的 Tcl 目录，在那里你会发现一个 ReadMe 文件，给出了创建 Tcl 的详细信息。下面是你需要阅读的部分。

1. 决定在什么地方安装可执行文件和库文件。缺省安装目录为 C:\Program files ,但是

makefile 有一个 bug，导致安装失败（因为目录名中的内嵌空间）。建议你安装到 C:/Tcl。

2. 拷贝 MakeFile.vc 到 MakeFile 下，编辑 MakeFile，根据你安装的适当目录修改文件头的路径为：

```
ROOT        =..
TOLLS32     =c:\program files\devstudio\vc
TOOLS3_rc   =c:\program files\devstudio\sharedide
TOOLS16     =c:\msvc
INSTALLDIR  =c:\Tcl
```

3. 在 MS-DOS 窗口下运行实用程序 nmake 创建 Tcl 库。

```
nmake
```

4. 安装可执行文件和库文件。

```
nmake install
```

假设你没有碰到问题，你可以继续创建 Tk 了。

创建 Tk

创建 Tk 与创建 Tcl 相似， 在 Window 改变目录到 Tk 目录之下，在此目录下，你会看到一个完整的 ReadMe 文件，给出创建 Tk 的完整详细信息，下面是我们需要阅读的大致内容。

1. 你应该把 Tk 安装到 Tcl 对等的目录之下。

2. 拷贝 MakeFile.vc 到 MakeFile 下，编辑 MakeFile，根据你安装的适当目录修改文件头的路径为：

```
ROOT        =..
TOLLS32     =c:\program files\devstudio\vc
TOOLS3_rc   =c:\program files\devstudio\sharedide
TOOLS16     =..\..\tcl8.0.5
INSTALLDIR  =c:\Tcl
```

3. MS-DOS 窗口下运行实用程序 nmake 创建 Tk 库。

```
nmake
```

4. 安装可执行文件和库文件。

```
nmake install
```

我们现在已经准表好安装 Python 了。

创建 Python

当前的 Python 版本需要 Microsoft Visual C++ 5.x(或 6.x)。一旦 Tcl/Tk 已经被创建，Python 就很容易完成了。

1. 资源管理器中，到 PCbuild 目录。
2. 打开 Pcbuild.dsw 工作面。
3. 选择 Debug 或 Release 设置（在创建菜单中使用 Set Active Configure 设置）。
4. 从 Project 菜单的 Select Active Project 选择 python15。
5. 从 Build 菜单选择 Build python_15.dll。
6. 从 Project 菜单的 Select Active Project 选择 python。
7. 从 Build 菜单选择 Build python.exe。

8. 从 Project 菜单的 Select Active Project 选择 pythonw。

9. 从 Build 菜单选择 Build pythonw.exe。

10. 从 Project 菜单的 Select Active Project 选择_tkinter。

11. 从 Build 菜单选择 Build _tkinter.pyd。

12. 把 python.exe,pythonw.exe,python15.dll 和_tkinter.pyd 移到你想从那里运行 python 的目录下（详细参见第 19 章）。

在 MacOS 下创建

我必须承认我没有在 MacOS 下面创建过 Tcl/Tk 或者 Python。我已得知你不必创建 MacOS 下面的 Tcl/Tk，标准安装程序包含一个完整的安装软件。

访问 www.python.org，你会发现最新版本的 Python（你通常会在主题页面找到当前版本的参考手册）。这将指引你源文件页面，它在 www.cwi.nl/~jack/macpython.html。在这里你会发现 Stuffit 和 BinHex 源码。MacOS 的版本和 Unix 的几乎一样，除了你将得到的是 Stuffit 或 BinHex 版本。获取代码，放到你建的目录下。

获得源文件之后，按照里面的指示去进行安装。

附录 E　事件与键盘系统

此附录表为 Tkinter 认可（严格说是 Tk）的组合键、时间类型和键盘系统。键盘代码的解释比较依赖于实现，因此，应该认识到不是所有的键都能被多种不同系统结构所检测到的。

广义的事件格式如下：

```
< [modifier ['-modifier...]'-][[type[qualifier] |qualifier]>
```

组合键：

组合键	ALT-1	ALT-2	掩码
Control			ControlMask
Shift			ShiftMask
Lock			LockMask
Meta	M		META_MASK
Alt			ALT_MASK
B1	Button1		Button1Mask
B2	Button2		Button2Mask
B3	Button3		Button3Mask
B4	Button4		Button4Mask
B5	Button5		Button5Mask
Mod1	M1	Command	Mod1Mask
Mod2	M2	Option	Mod2Mask
Mod3	M3		Mod3Mask
Mod4	M4		Mod4Mask
Mod5	M5		Mod5Mask
Double			
Triple			

事件类型

事件名字	ALT-1	类型	掩码
Key	KeyPress	KeyPress	KeyPressMask
KeyRelease		KeyRelease	keyPressMask|KeyReleaseMask
Button	ButtonPress	ButtonPress	ButtonPressMask
ButtonRelease		ButtonRelease	buttonPressMask|ButtonReleaseMask

（续）

事件名字	ALT-1	类型	掩码
Motion		MotionNotify	buttonPressMask|PointerMotionMask
Enter		EnterNotify	EnterWindowMask
Leave		LeaveNotify	LeaveWindowMask
FocusIn		FocusIn	FocusChangeMask
FocusOut		FocusOut	FocusChangeMask
Expose		Expose	ExposureMask
Visibility		VisibilityNotify	visibilityChangeMask
Destroy		DestroyNotify	StructureNotifyMask
Unmap		UnmapNotify	StructureNotifyMask
Map		MapNotify	StructureNotifyMask
Reparent		ReparentNotify	StructureNotifyMask
Configure		ConfigureNotify	StructureNotifyMask
Gravity		GravityNotify	StructureNotifyMask
Circulate		CirculateNotify	StructureNotifyMask
Property		PeopertyNotify	PropertyChangeMask
Colormap		ColormapNotify	colormapChangeMask
Activate		ActivateNotify	ActivateMask
Deactivate		DeactivateNotify	ActivateMask
MouseWheel		MouseWheelEvent	MouseWheelMask

Qualifier

Qualifier	掩码
1	ButtonPressMask
2	ButtonPressMask
3	ButtonPressMask
4	ButtonPressMask
5	ButtonPressMask
Keysym	KkeyPressMask

键盘系统

不是所有的键都列在这里，事实上只有 Latin-1 集的。这通常是你要捆绑的键集。若要与其他键集（如 Cyrillic 或 Balkan）捆绑，你可在 Tk 源码中找到它的（Tk[Release]/*generic*/ks_namesh）。

E.5.1 Latin-1

键盘系统	键	键盘系统	键
Backspace	Backspace	Win_L	LeftWindow
Tab	Tab	Win_R	RightWindow
Linefeed	Linefeed	App	Application

（续）

键盘系统	键	键盘系统	键
Clear	Clear	Select	Select
Return	Return	Print	Print
Pause	Pause	Execute	Execute
Escape	Escape	Insert	Insert
Delete	Delete	Undo	Undo
Multi_key	Multi_key	Redo	Redo
Kanji	Kanji	Menu	Menu
Home	Home	Find	Find
Left	Left	Cancel	Cancel
Up	Up	Help	Help
Right	Right	Break	Break
Down	Down	Mode_switch	Mode_switch
Prior	Prior	script_switch	script_switch
Next	Next	Num_Lock	Num_Lock
End	End	KP_Space	Keypad Space
Begin	Begin	KP_Tab	Keypad Tab
KP_Enter	Keypad Enter	numbersign	numbersign
KP_F1	Keypad F1	dollar	dollar
KP_F2	Keypad F2	percent	percent
KP_F3	Keypad F3	ampersand	ampersand
KP_F4	Keypad F4	quoteright	quoteright
KP_Equal	Keypad Equal	parenleft	parenleft
KP_Multiply	Keypad Multiply	parenright	parenright
KP_Add	Keypad Add	asterisk	asterisk
KP_Separator	Keypad Separator	plus	plus
KP_Subtract	Keypad Subtract	comma	comma
KP_Decimal	Keypad Decimal	minus	minus
KP_Divide	Keypad Divide	period	period
KP_0…KP_9	Keypad 0…Keypad 9	slash	slash
F1…F35	Function1…Function 35	0…9	0…9
L1…L10	L1…L10	colon	colon
R1…R15	R1…R15	semicolon	semicolon
Shift_L	Left Shift	less	less
Shift_R	Right Shift	equal	equal
Control_L	Left Control	greater	greater
Control_R	Right Control	question	question
Caps_Lock	Caps_Lock	at	at
Shift_Lock	Shift_Lock	A…Z	A…Z
Meta_L	Left Meta	bracketleft	bracketleft
Meta_R	Right Meta	backslash	backslash
Alt_L	Left Alt	bracketright	bracketright

（续）

键盘系统	键	键盘系统	键
Alt_R	Right Alt	asciicircum	asciicircum
Super_L	Left Super	underscore	underscore
Super_R	Right Super	quoteleft	quoteleft
Hyper_L	Left Hyper	a...z	a...z
Hyper_R	Right Hyper	braceleft	braceleft
space	space	bar	bar
exclam	exclam	braceright	braceright
quotedbl	quotedbl	asciitilde	asciitilde

（续）

附录F 光　　标

此附录的表描述 Tkinter 应用中能用的光标，光标对所有平台都行。然而，某些操作系统中，特定的光标可能有特殊含义（比如 Win32 上的钟表光标使用了系统定义的当前钟表光标）。

光标名	光标	光标名	光标
x_cursor		crosshair	
arrow		diamond_cross	
based_arrow_down		dot	
based_arrow_up		dotbox	
boat		double_arrow	
bogosity		draft_large	
bottom_left_corner		draft_small	
bottom_right_corner		draped_box	
bottom_side		exchange	
bottom_tee		fleur	
box_spiral		gobbler	
center_ptr		gumby	
circle		hand1	
clock		hand2	
coffee_mug		heart	
cross		icon	
cross_reverse		iron_cross	

（续）

光标名	光标	光标名	光标
left_ptr		sb_right_arrow	
left_side		sb_up_arrow	
left_tee		sb_v_double_arrow	
leftbutton		shuttle	
ll_angle		sizing	
lr_angle		spider	
man		spraycan	
middlebutton		star	
mouse		target	
pencil		tcross	
pirate		top_left_arrow	
plus		top_left_corner	
question_arrow		top_right_corner	
right_ptr		top_side	
right_side		top_tee	
right_tee		trek	
rightbutton		ul_angle	
rtl_logo		umbrella	
sailboat		ur_angle	
sb_down_arrow		watch	
sb_h_double_arrow		xterm	
sb_left_arrow			

附录G 参考文献

此附录提供一些 Python 资源和一些参考书籍，这对你了解更详细的 Python 信息、X 视窗、Tcl/Tk、人力因素工程很有帮助，大部分资源来源于网上，我已经复制了原作者对资源的描述。

资源

Sio 模块

Sio.pyd 是一个 DLL，也是对另一个商业软件包 DLL 的打包软件。此软件包可以任意地使用，只要保留下面：

```
Serial Communications DLLs by MarshallSoft Computer, Inc.
POB 4543 Huntscville AL 35815.205-881-4630.
Email:mike@marshallsoft.com
Web:www.marshallsoft.com
```

SWIG

SWIG 是把 C,C+和 Objective-C 编写的程序与一些高级语言连接起来的软件开发工具。SWIG 主要使用普通脚本语言如 Perl、Python 和 Tcl/Tk 语言，但是也被扩展到包括 Java，Eiffel 和 Guile 等。SWIG 最常用来产生高级解释编程环境和系统集成，以及作为生成用户接口的工具。SWIG 可以免费用于商业或非商业的使用、发布或修改。

http://www.swig.org/

NumPy

NumPy 是 Python 数值扩展包的昵称。该扩展给 Python 增加了两个功能强大的新类型：一个新序列类型，有效实现多维数组；一个新函数类型，称为通用函数（universal functionu,ufunc），有效进行新数组和其他序列操作。这些新对象使 Python 具有像 MATLAB 和 IDL 这样的数值语言的数值处理功能，同时保持 Python 作为一门通用语言的优点。它如 Python 的其他部分一样是免费的。

ftp://ftp-icf.llnl.gov/pub/python/README.html

Python 新闻组

comp.lang.python

comp.lang.python.announce

Python 图像库(PIL)

Python 图像库在你的 Python 解释器中加入图像对象。你可以从很多文件格式加载图像，也可以对它们进行很多操作。

http://www.pythonware.com/downloads.htm

PythonWorks

PythonWorks 是一个 PythonWare/SecretLabsAB 开发的快速开发环境。

http://www.pythonware.com/products/works/index.htm

Python 书籍

1. Ascher, David and Mark Lutz, *Learning Python*. O'Reilly & Assiates, 1999, ISBN: 1-56592-464-9

2. Beazley,David.*Python Essential Reference*. New Riders. 1999. ISBN:0-7357-090-17.

3. Harms, Daryl and Kenneth McDonald, *The Quick Python Book*. Manning Publications,2000. ISBN:1-884777-74-0

4. Lundh, Fredrik.(the effbot guide to) *The Standard Python Library*. Electronic edtion at www.fatbrain.com

5. Lutz, Mark. *Programming Python*.O'Reilly & Assocites, 1996. ISBN:1-56592-197-6.

X Window 书籍

6. Nye, Adrian and Tim O'Reilly. *X Toolkit Intrinsics Programming Mannual for XII, Release 5 (Definitive Guides to the X Windows System, Vol 4)*. O'Reilly & Associates, 1992. ISBN: 1-56592-013-9

7. Young, Douglas. *The X Window System: Programming and Applications with Xt, OSFI Motif*, 2nd edition. Prentice-Hall, 1994, ISBN:0-13123-803-5

TCI/Tk 书籍

8. Flynt, Clifton. *Tcl/Tk for Real Programming*. Academic Press (AP Professional),1998 ISBN: 0-12261-205-1

9. Foster-Johnson, Eric . *Graphical Application with Tcl and Tk*,2nd edtion. M&T Books, 1997. ISBN:1-55851-569-0

10. Harrison, Mark and Michael J. McLennan .*Effictive Tcl/Tk Programming Writing Better Programs in Tcl and Tk*. Addison Wesley Longman,1997. ISBN: 0-20163-474-0

11. Ousterhout, John. *Tcl and the Tk Toolkit*.Addison-Wesley, 1994.ISBN; 0-20163-337-X.

12. Raines, Paul. *Tcl/Tk Pocket Reference*. O'Reilly & Associates , 1998. ISBN;

1-56592-498-3

Human factors engineering

13. Coe, Marlana. *Human Factors for Technical communicators.* John Wiley & Sons, 1996. ISBN; 0-47103-530-0

14. Copper, Alan *About Face:The Essentials of user Interface design.* IDG Books, 1995. ISBN; 1-56884-322-4

15. Olsen, Dan R.,Jr.,Dan E. Olsen and Dan R. Olsen. *Developing User Interfaces* .Morgan Kaufmann, 1998. ISBN; 1-55860-418-9

1-56592-498-3

Human factors engineering

13. Coe, Marlana. *Human Factors for Technical communicators.* John Wiley & Sons, 1996. ISBN; 0-47103-530-0

14. Copper, Alan *About Face:The Essentials of user Interface design.* IDG Books, 1995. ISBN; 1-56884-322-4

15. Olsen, Dan R.,Jr.,Dan E. Olsen and Dan R. Olsen. *Developing User Interfaces* .Morgan Kaufmann, 1998. ISBN; 1-55860-418-9

1-56592-498-3

Human factors engineering

13. Coe, Marlana. *Human Factors for Technical communicators.* John Wiley & Sons, 1996. ISBN; 0-47103-530-0

14. Copper, Alan *About Face:The Essentials of user Interface design.* IDG Books, 1995. ISBN; 1-56884-322-4

15. Olsen, Dan R.,Jr.,Dan E. Olsen and Dan R. Olsen. *Developing User Interfaces* .Morgan Kaufmann, 1998. ISBN; 1-55860-418-9