第十章

框架和应用

本章内容：

- 自动化客户支持系统
- 与 COM 的接口：廉价的公共关系
- 一个基于 Tkinter 的管理表格数据的编辑器
- 设计上的考虑
- JPython：Python 与 Java 的结合
- 其他的框架和应用
- 练习

目前为止这本书里的所有例子都很小，与实际的应用相比它们似乎像玩具。本章展示了一些框架，它们可供那些想在特定的领域创建真正应用程序的Python程序员使用。一个框架可以看作是某个特定领域的一组类与期望的类之间的交互模式。我们将介绍三个：COM 框架，也就是与微软的公共对象模型（Common Object Model，COM）的接口，Tkinter 图形用户界面，以及 Java 的 Swing 图形界面工具库。我们也将涉及到标准库里与 Web 相关的模块。

假设有一个小公司的Web站点，它需要收集、维护和响应客户通过表格对产品的反馈，我们将以此来说明框架的威力。我们将涉及三个程序，第一个是基于 Web 的数据录入表格，要求用户从浏览器输入信息，然后把信息存到磁盘里。第二个程序是使用这些数据让 Microsoft Word 软件打印出定制的基于那些数据的信件。最后是一个浏览已存数据的浏览器，使用的是 Tkinter 模块。Tkinter 是以一个强大的可移植的可以管理窗口、按钮、菜单等的图形界面工具集 Tk 为基础的。希望这些例子能让你认识到，当这些工具与Python的快速开发能力结合时，创建实际应用程序是多么的快。每一个程序都建筑在前一个的基础上，所以我们强烈建议你按顺序阅读每一个程序，即使你不能（或不想）让它们运行起来。

本章的最后一节谈到了 JPython，它是 Python 在 Java 上的移植。本章以一个中等

程度的 JPython 程序结束，借助于 Swing 工具库，用户可以图形化地操作数学函数。

自动化客户支持系统

我们这个例子的主角是一个新创公司 —— 乔氏牙膏公司，它销售最新型的牙膏。由于只有自己一个工作人员，乔无法应付太多的工作，所以创建了一个Web站点 www.toftoot.com 做推广和支持。网站上有各种漂亮的图片，也为顾客提供了一个输入抱怨或建议的地方。网页如图 10-1 所示。

图 10-1　顾客在 http://www.toftoot.com/comment.html 所看到的页

这一页关键的 HTML 代码显示在后面的选读部分“ 摘自 HTML 文件 ”中。由于本书不是讲 CGI、HTML 或其他技术的（注 1），我们假设你已经知道这些技术。HTML 代码里的重要部分用黑体字标出，这里是一个简短的描述：

摘自 HTML 文件

这是图 10-1 中网页的关键代码：

```
<FORM METHOD=POST ACTION="http://toftoot.com/cgi-bin/feedback.py">
  <UL><I>Please fill out the entire form:</I></UL>
  <CENTER><TABLE WIDTH="100%" >
  <TR><TD ALIGN=RIGHT WIDTH="20%">Name:</TD>
      <TD><INPUT TYPE=text NAME=name SIZE=50 VALUE=""></TD></TR>
  <TR><TD ALIGN=RIGHT>Email Address:</TD>
      <TD><INPUT TYPE=text NAME=email SIZE=50 VALUE=""></TD></TR>
  <TR><TD ALIGN=RIGHT>Mailing Address:</TD>
      <TD><INPUT TYPE=text NAME=address SIZE=50 VALUE=""></TD></TR>
  <TR><TD ALIGN=RIGHT>Type of Message:</TD>
  <TD><INPUT TYPE=radio NAME=type CHECKED VALUE=comment>comment 
      <INPUT TYPE=radio NAME=type VALUE=complaint>complaint</TD></TR>
  <TR><TD ALIGN=RIGHT VALIGN=TOP>Enter the text in here:</TD>
      <TD><TEXTAREA NAME=text ROWS=5, COLS=50  VALUE="">
          </TEXTAREA></TD></TR>
  <TR><TD></TD>
  <TD><INPUT type=submit name=send value="Send the feedback!"></TD></TR>
  </TABLE></CENTER>
  </FORM>
```

- FORM 行指定了当链接到表格时将激活哪一个 CGI 程序，URL 指向了我们将详细介绍的名为 *feedback.py* 的脚本。
- INPUT 标签标出了表格字段的名字（姓名、地址、电子邮件、文本和类型）。除了类型是由用户选择的“ 建议 ”或“ 抱怨 ”以外，这些字段的值都是用户输入的。
- 最后，INPUT TYPE=SUBMIT 标签是发送按钮，它将真正调用 CGI 脚本。

注 1： 如果你从来没有听说过这些缩写词，我可以解释一下：CGI 表示公用网关接口，它是一个由 Web 浏览器调用服务器上程序的协议。HTML 表示超文本标记语言，它是 Web 页的编码格式。

我们现在进入与Python相关的有趣部分：对请求的处理。这里是整个*feedback.py*程序：

```
import cgi, os, sys, string

def gush(data):
    print "Content-type: text/html\n"
    print "<h3>Thanks, %(name)s!</h3>" % vars(data)
    print "Our customer's comments are always appreciated."
    print "They drive our business directions, as well as"
    print "help us with our karma."
    print "<p>Thanks again for the feedback!<p>"
    print "And feel free to enter more comments if you wish."
    print "<p>"+10*" "+"--Joe."

def whimper(data):
    print "Content-type: text/html\n"
    print "<h3>Sorry, %(name)s!</h3>" % vars(data)
    print "We're very sorry to read that you had a complaint"
    print "regarding our product_ _We'll read your comments"
    print "carefully and will be in touch with you."
    print "<p>Nevertheless, thanks for the feedback.<p>"
    print "<p>"+10*" "+"--Joe."

def bail():
    print "<H3>Error filling out form</H3>"
    print "Please fill in all the fields in the form.<P>"
    print '<a href="http://localhost/comment.html">'
    print   o back to the form</a>'
    sys.exit()

class FormData:
    """A repository for information gleaned from a CGI form """
    def __init__(self, form):
        for fieldname in self.fieldnames:
            if not form.has_key(fieldname) or form[fieldname].value=="":
                bail()
            else:
                setattr(self, fieldname, form[fieldname].value)

class FeedbackData(FormData):
    """A FormData generated by the comment.html form. """
    fieldnames = ('name', 'address', 'email', 'type', 'text')
    def __repr__(self):
        return "%(type)s from %(name)s on %(time)s" % vars(self)
```

```
DIRECTORY = r'C:\complaintdir'

if __name__ == '__main__':
    sys.stderr = sys.stdout
    form = cgi.FieldStorage()
    data = FeedbackData(form)
    if data.type == 'comment':
        gush(data)
    else:
        whimper(data)

    # save the data to file
    import tempfile, pickle, time
    tempfile.tempdir = DIRECTORY
    data.time = time.asctime(time.localtime(time.time()))
    pickle.dump(data, open(tempfile.mktemp(), 'W'))
```

这个脚本的输出显然取决于输入，图10-1中的输入所产生的输出如图10-2所示。

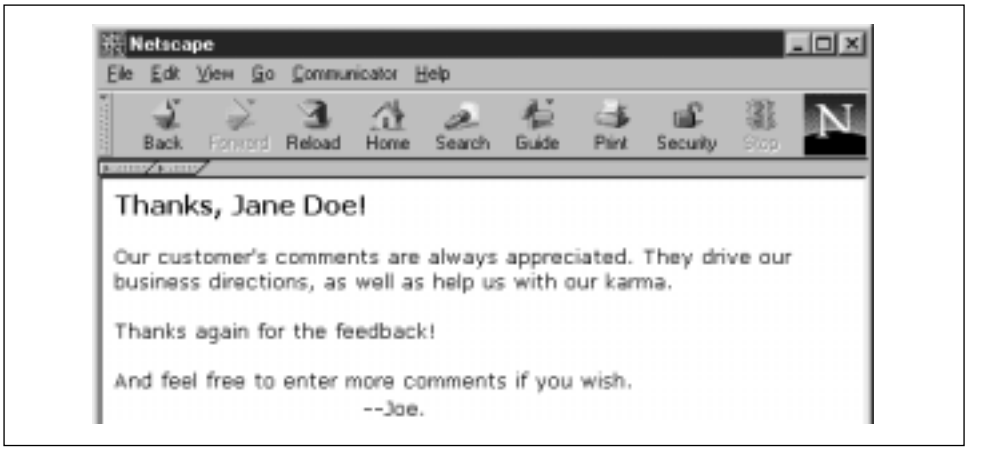

图10-2 用户按了“send the feedback（发送反馈）”按钮后所看到的页面

*feedback.py*是怎样工作的呢？这个脚本具有所有CGI程序常见的某些特点，都用黑体字标出。为了能启动，程序的第一行需要指向Python解释器。这是我们所用的Web服务器的要求，但也许你的环境要求不一样；即使是一样的，你的Python解释器的位置很可能不同。第二行导入包括`cgi`在内的模块，它负责处理CGI较难的部分，如解析环境变量和处理换义字符。（如果你从来没有手工的做这些事情，你是幸运的。）`cgi`模块的文档描述了它的直接而简单的使用方式。然而对于这个例子，脚本显得有点复杂。

让我们逐句地看 `if __name__ == '__main__'` 语句块的代码（注 2）。第一句把 `sys.stderr` 重定向到标准输出流。这是为了方便调试，因为 CGI 程序的标准输出流是 Web 浏览器，而 `stderr` 标准错误流是写到服务器的错误日志文件，阅读可能比看浏览器更容易一些。我们可以在浏览器上看到发生的运行错误，而不是去猜测发生了什么。第二行是重要的，它做了所有复杂的 CGI 的工作：它返回一个字典一样的对象（名为`FieldStorge`对象），键是表格里填写的变量名，可以通过它的 `value` 属性取到对应的值。听起来有些复杂，表格对象有键 `'name'`，`'type'`，`'email'`，`'address'` 和 `'text'`，通过 `form['name'].value` 可以找出用户在 Web 表格里填写的是什么。

第三行创建了一个用户定义的类`FeedbackData`的实例，把`form`对象作为参数传给它。如果你看 `FeedbackData` 的定义，你将发现它是另一个用户定义的 `FormData` 类的很简单的子类。我们在 `FeedbackData` 里定义了一个类属性 `fieldnames`，以及一个用于 `print` 语句的 `__repr__` 函数。显然`FormData`类的 `__init__` 方法一定会用到 `FieldStorge` 参数。实际上，它查看了实例里的`fieldnames`类属性的每一个字段名字，对应于每个字段名字，检查在`FieldStorge`对象里的对应非空键。如果有对应键，就在实例里设置同样名字的属性，把用户输入的文字作为属性的值；如果没有，就调用 `bail` 函数。

我们将很快就知道`bail`的作用，但首先让我们来看通常的情况：用户认真地输入了要求的所有数据。这时，`FieldStorge` 拥有所有 `FeedbackData` 所需的键（`'name'`，`'type'`等等）。`FormData`类的`__init__`方法也就在实例里设置属性。所以当 `data = FeedbackData(form)`调用返回时，将保证 `data` 是一个 `FeedbackData` 的实例，它具有 `name`，`type`，`email`等属性，以及有对应的用户输入的值。下面的代码也许有同样的效果：

```
form = cgi.FieldStorage()
form_ok = 1
if not form.has_key("name") or form["name"].value == "":
    form_ok = 0
else:
```

注 2： 你要记住只有当这个程序作为脚本运行，而不是被导入时这个`if`语句才为真。CGI程序是一个脚本，所以当服务器调用它时将执行 `if` 语句块。我们在后面将导入它，所以要注意。

```
    data_name = form["name"].value
if not form.has_key("email") or form["email"].value == "":
    form_ok = 0
else:
    data_email = form["email"].value
...
```

但这种写法是冗长乏味而容易出错的。我们的计划是，当乔氏公司改变网页的域名时，我们需要改变的就是FeedbackData类的fieldnaes属性。我们也可以在别的CGI脚本里重用同样的FormData类。

如果用户没有输入所有要求的字段会怎样呢？要么是FieldStorge字典里少一个键，要么是对应的值是空字符串。FormData.__init__方法就会调用bail函数，它将礼貌地显示错误信息并退出脚本。控制永远不会返回主程序，所以没有必要测试data变量的有效性。

我们可检查data实例来查看用户反馈的是否是我们要感谢的建议。如果反馈的类型是抱怨，我们将毫不吝惜地道歉，并承诺与他们联系。

我们现在有了一个基本的CGI结构。把数据保存到文件非常简单。首先我们在if测试的外面定义DIRECTORY变量，因为我们将在另一个脚本里使用它，所以我们希望这个脚本不是单独运行时也定义这个变量。

浏览 *feedback.py* 的最后几行：

- 导入tempfile，pickle和time模块。tempfile模块如前一章所讲提供了未使用的文件名，我们不需要担心任何的名字冲突。pickle模块用来保存任何的Python对象。time模块告诉我们当前的时间，乔用它来判断反馈的时间。
- 下一行设置tempfile模块的tempdir属性为DIRECTORY变量的值，也就是我们希望保存数据的地方。这是一个通过直接修改名字空间来定制模块的例子，就像前面修改sys模块的stderr属性一样。
- 下一行用了几个time模块的函数，提供了当前日期和时间的字符串（如'Sat Jul 04 18:09:00 1998'，这对乔已经足够精确了），给data实例创建了一个新的属性time，它也随data一起保存。

- 最后一行作了实际的保存，它以写模式打开一个由 `tempfile` 模块产生文件名的文件，把实例数据写到文件里。工作完成了！现在指定的文件里保存了实例。

与 COM 的接口：廉价的公共关系

我们用保存的数据做两件事。我们将写一个周期性运行的程序（比如每晚 2 点）（注 3）扫描保存的数据，找出哪些是客户的抱怨，并打印一份相应的信件。听起来很复杂，但你会惊讶于用恰当的工具来做是如此的简单。乔氏公司的网站运行在 Windows 上，所以我们假设这个程序也是在 Windows 上，但其他平台与此相似。

在我们讨论如何写这个程序前，先谈谈它使用的技术即微软的 COM（Common Object Model ）。COM 是两个程序间交互的标准（用术语说是对象请求代理服务），使得 COM 兼容的程序可以互相通信，交换数据，在另一个 COM 兼容程序上执行命令。调用程序称为 COM 客户，而执行命令的一方称为 COM 服务器。所有的微软产品都是能运行 COM 的，几乎都能作为 COM 服务器。我们这里用的微软 Word 8 就是其中之一。事实上 Word 很适合于写信。幸运的是，至少在 Windows 95、98 和 Windows NT 上 Python 也能感知 COM。Mark Hammond 和 Greg Stein 为 Python 的 Windows 版做了扩展，名为 *win32com*，这使得你的 Python 程序可以做 COM 相关的事情。你可以用 Python 写 COM 客户、服务器、ActiveX 脚本等等。我们只需要写 COM 客户就行了，这也是最简单的。我们的格式信件程序需要做以下几件事：

1. 在适当的目录里打开文件并提取数据。
2. 对每个实例文件测试，反馈是否是抱怨。如果是就找出填表人的名字和地址，进入第 3 步；否则跳过第 3 步。
3. 打开一个我们想发出的 Word 信件文档模板，用相应的信息填写。

注 3： 设置这种自动定时程序在许多平台上都很简单，例如在 Unix 上用 `cron`，在 Windows NT 上用 `AT`。

4. 打印信件并关闭文档。

这非常简单。这里是称为 *formletter.py* 的小程序：

```
from win32com.client import constants, Dispatch
WORD = 'Word.Application.8'
False, True = 0, -1
import string

class Word:
    def __init__(self):
        self.app = Dispatch(WORD)
    def open(self, doc):
        self.app.Documents.Open(FileName=doc)
    def replace(self, source, target):
        self.app.Selection.HomeKey(Unit=constants.wdLine)
        find = self.app.Selection.Find
        find.Text = "%"+source+"%"
        self.app.Selection.Find.Execute()
        self.app.Selection.TypeText(Text=target)
    def printdoc(self):
        self.app.Application.PrintOut()
    def close(self):
        self.app.ActiveDocument.Close(SaveChanges=False)

def print_formletter(data):
    word.open(r"h:\David\Book\tofutemplate.doc")
    word.replace("name", data.name)
    word.replace("address", data.address)
    word.replace("firstname", string.split(data.name)[0])
    word.printdoc()
    word.close()

if __name__ == '__main__':
    import os, pickle
    from feedback import DIRECTORY, FormData, FeedbackData
    word = Word()
    for filename in os.listdir(DIRECTORY):
        data = pickle.load(open(os.path.join(DIRECTORY, filename)))
        if data.type == 'complaint':
            print "Printing letter for %(name)s." % vars(data)
            print_formletter(data)
        else:
            print "Got comment from %(name)s, skipping printing." % vars(data)
```

主程序的前几行显示了一个设计良好的框架的威力。第一行是一个标准的import语句，*win32com*是一个包而不是一个模块。它实际上是一组子包、模块和函数。我们需要其中的两个东西：`client`模块里的`Dispatch`函数，它使得我们可以把函数分发到别的对象（这里是指COM服务器），另一个是同一个模块里的`constants`子模块，它定义了我们需要的常量。

第二行定义我们感兴趣的COM服务器的名称变量。名字是`Word.Application.8`，你可以通过使用COM浏览器或读Word的API（应用程序接口）来找出该名字（参见后面的选读部分“了解COM接口”）。

现在我们集中看`if __name__ == '__main__'`这一块，也就是类和函数定义的下一行。

第一个任务是读数据。显然地我们需要`os`和`pickle`模块，以及我们刚写的`feedback`模块中的三项：`DIRECTORY`是存贮数据的地方（如果在*feedback.py*我们改变了它，这个模块下次运行时也会相应改变），以及`FormData`和`FeedbackData`类。下一行创建了`Word`类的实例，它打开了与Word这个COM服务器的连接，如果需要的话就启动Word。

`for`循环简单的遍历目录里保存的所有文件。这个目录里只保存反馈的数据，我们就不作任何错误检查。通常我们应当让代码更健壮，但为了简单我们忽略了。

`for`循环的第一行是提取数据。它用了`pickle`模块里的`load`函数，该函数的唯一参数是被提取的文件。它将返回文件里的所有数据——我们这里只有一个。现在，`data`里存的就是`FeedbackData`实例。在文件里并没有保存类的定义，而只是保存了实例的值和对类的引用（注4）。

循环里的`if`语句是一目了然的。剩下要解释的就是打印信件的函数，以及`Word`类。`print_formletter`函数用提取的数据调用了`word`实例的不同方法。

注4： 这可以减少存储的对象的大小，而更重要的是它允许你解开以前版本的类，并自动地升级到新的类定义。

注意：在解开时间的时候，解开的实例会自动地把对应类所定义的模块导入。为什么我们需要导入它呢？第五章“模块”里我们说过当前运行的模块名是`__main__`。换句话说，包含定义的类的模块名是`__main__`（即使文件名是*feedback.py*），而且当我们解开实例时导入`__main__`也就导入了当前运行模块（*formletter.py*），它没有包含解开的实例的类定义。这就是为什么我们需要显式地从`feedback`模块里导入类定义。如果在调用`pickle.load`时没有这些定义，解开实例会失败。或者我们可以把类定义放在一个文件里，并在任何实例前导入它，或者甚至更简单，把类定义放在由*feedback.py*导入的单独模块里，在*formletter.py*解开实例的过程中隐含地导入。后一种是通常的情况，你不必显式地导入类的定义，解开实例时就完成了，很神吧（注5）。

注意我们用`string.split`函数提取了客户的名，这是为了信件更友好，但如果是非标准的姓名就会有出错的危险。

在`Word`类里，`__init__`方法看起来简单却隐藏了许多工作。它创建了与COM服务器的连接，并存放在一个实例变量`app`里。现在，有两种使用服务器的方式：动态分发和非动态分发。如果是动态分发，Python在运行这个程序时不知道COM服务器的接口。不过没关系，因为COM允许Python询问服务器并确定协议，例如函数需要的参数个数和类型，然而这种方式可能会慢。一个加速的办法是运行*makepy.py*程序，它对指定的COM服务器做一次询问操作，并把信息存放到磁盘上。当一个是用该服务器的程序运行时，分发就用预先算好的信息而不是动态分发。我们这个程序用两种方法都可以，如果曾对Word运行过*makepy.py*，就会用快速地分发，否则就用动态分发。关于这个问题的更多信息，参见*win32*扩展，网址在：http://www.python.org/windows/win32all/。

为了解释`Word`类的方法，我们先打开一个文档模板，看看我们需要做什么。如图10-3所示。

你可以看出这是一个很平常的文档，除了一些文字在`%`符号之间以外。我们用这个标记告诉程序哪一部分需要定制，但也可用别的方法。要使用这个模板，我们需要打开它，定制，打印，最后关闭它。`Word`类的`open`方法用来打开它。打印

注5： 存储顶级类是很微妙的，这超出了本书的范围。我们提到它只是因为我们需要解释我们用的代码。


JOE'S TOOTHPASTE, INC.

February 18, 1999

%name%
%address%

Dear %firstname%,

Thank you for filling out the feedback form at our website (www.toftoot.com). We're sorry to hear of your complaint, and we'll do our best to remedy it in the future. As a token of our deepest apologies, please accept a coupon for a free tube of our newest toothpaste, the "Firm Tofu Plus Toothpaste", enriched with Vitamin A, B and Z12.

Sincerely,

Joe Smith.

FREE COUPON

Coupon for 1 (one) free tube of Firm Tofu Plus Toothpaste (5oz)

Available at all WackyStores™

Expires: 12/99

SUITE 101, 22 MAIN ST • TOFUTOWN, CA • 04122
PHONE: 415-555-1212 • FAX: 415-555-1211

图 10-3　乔氏公司给抱怨者的信件模板

与关闭也类似。定制时我们用相应的字符串替代 `%name%`，`%firstname%` 以及 `%address%`，这是由 `replace` 方法完成的。

让这一切运转起来，产生的输出文本如下：

```
C:\Programs> python formletter.py
Printing letter for John Doe.
Got comment from Your Mom, skipping printing.
Printing letter for Susan B. Anthony.
```

打印了两份定制的准备邮寄的信件。注意 Word 程序在桌面上并不显示，缺省情况下 COM 服务器是不可见的，所以 Word 只是在后台行动。如果此时 Word 运行在桌面上，每一步用户都可以看到。

请留意：了解 COM 接口

你怎样找出COM对象的各种方法和属性呢？一般来说，COM对象与别的程序一样应该有文档。然而很可能我们有软件但没有文档。如果你想了解COM接口，有三种可能的方法：

- 寻找并购买文档，一些 COM 程序在 Web 上有文档，或者是印刷好的。
- 使用 COM 浏览器。Pythonwin（见附录二）就带有一个 COM 浏览器可用于探索 COM 对象的复杂等级体系。它也就是列出可用的对象与函数，有时这也就是你需要的。微软 Visual Studio 这样的开发工具也带有 COM 浏览器。
- 用另一个工具。在上面的例子中，我们用 Word 的“宏记录器”功能产生一个 VBA 脚本，它可以相当直接地翻译成 Python 脚本。宏可以找出文件菜单里的选择打印项的等效命令是 `ActiveDocument.PrintOut()`。

一个基于 Tkinter 的管理表格数据的编辑器

我们回顾一下：我们写了一个 CGI 程序（*feedback.py*），它从 Web 上获得输入并把信息保存在我们服务器的磁盘上。我们又写了一个程序（*formletter.py*），用这些信息产生了道歉的信件。下一个任务就是建造一个由人来查看建议和抱怨的程序，使用 Tkinter 工具库来建造一个反馈信息的浏览器。

Tkinter 工具库是一个 Tk 图形库的 Python 接口。Tk 事实上是大多数 Python 程序员的选择，因为它提供了职业的图形界面并且相当容易使用。它产生的界面与 Windows、Mac 或任何 Unix 都不完全一样，但看起来很接近，而且同样的 Python 程序在所有平台都可以工作，这对那些特定平台的工具来说是不可能的。另一个值得考虑的可移植工具包是 wxPython。它位于：http://www.alldunn.com/wxPython。

我们将在这个例子中用 Tk。它是由 John Ousterhout 开发的另一个脚本语言 Tcl 的工具包。Tk 已经被许多别的脚本语言采用，包括 Python 和 Perl。更多的关于 Perl 和 Tk 的信息请参见 O'Reilly 出版的《Learning Perl/Tk》。

这个程序的目标是简单的：在一个窗口中显示列出所有的反馈数据项，允许用户选择并查看细节。而且乔希望能删除处理过的项。实际运行的该程序的屏幕图如图 10-4 所示。

图 10-4　feedbackeditor.py 程序的屏幕图

我们将彻底地考察一种编程方式。我们的程序称为 *feedbackeditor.py*，如下：

```
from FormEditor import FormEditor
from feedback import FeedbackData, FormData
from Tkinter import mainloop
FormEditor("Feedback Editor", FeedbackData, r"c:\Complaintdir")
mainloop()
```

更多的细节都隐藏在 *FormEditor* 里。把这几行单独出来的意义在于，我们把与表格有关的部分独立出来，*FormEditor* 程序完全独立于特定的表格。另外，这个小程序展示了如何与 Tkinter 接口，你创建组件和窗口，然后调用 *mainloop* 函数启动 GUI 运行。接下来发生的就是 GUI 动作的结果。与 `formletter` 一样，它从 `feedback`模块导入了类，道理是一样的（解开实例）。然后创建了一个`FormEditor` 的实例，传递给初始化函数的参数是编辑器的名字，要解开的实例的类，以及要解开的实例的位置。

FormEditor的代码只有一个类的定义，我们将一次说明一个方法。首先是import语句和初始化方法：

```
from Tkinter import *
import string, os, pickle

class FormEditor:
    def _ _init_ _(self, name, dataclass, storagedir):
        self.storagedir = storagedir        # 保存一些引用
        self.dataclass = dataclass
        self.row = 0
        self.current = None

        self.root = root = Tk()             # 创建窗口并设置大小
        root.minsize(300,200)

        root.rowconfigure(0, weight=1)      # 定义当窗口调整尺寸时列和行的尺度
        root.columnconfigure(0, weight=1)
        root.columnconfigure(1, weight=2)
        # 创建标题标签
        Label(root, text=name, font='bold').grid(columnspan=2)
        self.row = self.row + 1
        # 创建主列表框并设置它
        self.listbox = Listbox(root, selectmode=SINGLE)
        self.listbox.grid(columnspan=2,sticky=E+W+N+S)
        self.listbox.bind('<ButtonRelease-1>', self.select)
        self.row = self.row + 1

    # 每个类的fieldnames变量里的变量调用一次self.add_variable
        for fieldname in dataclass.fieldnames:
            setattr(self, fieldname, self.add_variable(root, fieldname))

    # 创建几个按钮
        self.add_button(self.root, self.row, 0, 'Delete Entry', self.delentry)
        self.add_button(self.root, self.row, 1, 'Reload', self.load_data)

        self.load_data()
```

我们用了from ... import *的写法，我们曾警告过这有时是危险的。在Tkinter程序里，这通常是很安全的，因为Tkinter只输出显然与GUI相关的变量（标签、按钮等），而且都是以大写字母开头。

要理解__init__方法最好是把代码结构与屏幕结构对照。它们几乎是完全一致的。

前几行只是保存实例变量里的东西，并为一些变量设置缺省值。接下来的几行访问一个Toplevel组件（一个窗口，Tk()调用返回当前定义的顶层组件），设置它的最小值和一些别的属性。行和列的设置使得窗口里的组件随着窗口的放大而变化，并决定内部两列组件的相对宽度。

下一个调用创建了一个在Tkinter模块里定义的标签组件，你可以从屏幕看到它也就是一个文本标签。它横跨两列，也就是说它从窗口的最左边伸展到最右边。GUI 调用的主要职责就是安排图形元素的位置。

接下来创建的是列表框组件。它是一个文本行的列表，用户可以用方向键和鼠标做选择。这个列表框只允许一次选择一行（selectmode=SINGLE）并填充所有的空间（sticky 选项）。

for 循环是这个方法里最有趣的部分，通过遍历 dataclass 变量（也就是FeedbackData）的fieldnames属性，它找出哪一个变量在实例的数据里，并依次调用FormEditor类的add_variable方法，把返回值放到一个对应的实例变量里。这也等同于：

```
        ...
    self.name = self.add_variable(root, 'name')
    self.email = self.add_variable(root, 'email')
    self.address = self.add_variable(root, 'address')
    self.type = self.add_variable(root, 'type')
    self.text = self.add_variable(root, 'text')
```

不过代码例子里的版本更好些，因为字段名列表对程序是已知的，重复键入常常是不良设计的标志。而且对于我们的表格来说，FormData没有什么是特殊的。它可以用来浏览任何可变化的字段名的类。像这样的通用程序可以更好地重用在别的任务中。

在这个方法的结束部分，我们看到两个按钮，每一个都有一个被单击时要执行的命令。一个是删除当前项的 delentry 方法，另一个用于重读字典里数据的reloading 函数。

最后用load_data方法来装入数据。在对设置组件的add_varible和add_button方法的介绍结束后，我们再来介绍它。

add_varible创建了一个标签组件，显示字段的名字，同一行里还有一个标签对应于列表框里选项的值：

```
def add_variable(self, root, varname):
    Label(root, text=varname).grid(row=self.row, column=0, sticky=E)
    value = Label(root, text='', background='gray90',
                  relief=SUNKEN, anchor=W, justify=LEFT)
    value.grid(row=self.row, column=1, sticky=E+W)
    self.row = self.row + 1
    return value
```

add_button更简单些，它只需要创建一个组件：

```
def add_button(self, root, row, column, text, command):
    button = Button(root, text=text, command=command)
    button.grid(row=row, column=column, sticky=E+W, padx=5, pady=5)
```

load_data函数清除列表框的任何内容并重新设置item属性。循环与*printcomplaints.py*里用的很像，说明如下：

- 保存实例的文件名，被作为实例的一个属性（我们将很快知道为什么）
- 实例被加入item属性
- 项对应的字符串（注意反引号`的用法）被加入列表框
- 最后，选择列表框的第一项:

```
def load_data(self):
    self.listbox.delete(0,END)
    self.items = []
    for filename in os.listdir(self.storagedir):
        item = pickle.load(open(os.path.join(self.storagedir, filename)))
        item._filename = filename
        self.items.append(item)
        self.listbox.insert('end', `item`)
    self.listbox.select_set(0)
    self.select(None)
```

我们现在介绍前面提到的select方法。有两种情况会调用它，第一种就像上面的方法发生在数据装入完毕，第二种就是__init__方法里的捆绑操作，我们重新写在下面：

```
self.listbox.bind('<ButtonRelease-1>', self.select)
```

这个调用把一个特定组件的特定事件（'<ButtonRelease-1>'）与一个称为elf.select捆绑起来。换句话说，每当你的鼠标落在列表框的一项上时，就会调用编辑器的select方法。调用时的参数是事件类型，使我们知道是哪一个按钮按下了，但由于我们只需要知道事件发生了，所以我们将忽略这个参数。选择时应发生什么？必须识别出对应于被选择项的实例，然后对应的显示字段必须更新。这是通过遍历每个字段名（参考fieldname变量），找出选择的实例的对应值，并设置对应的标签（注 6）：

```
def select(self, event):
    selection = self.listbox.curselection()
    self.selection = self.items[int(selection[0])]
    for fieldname in self.dataclass.fieldnames:
        label = getattr(self, fieldname)              # GUI 字段
        labelstr = getattr(self.selection, fieldname)  # 实例的属性
        labelstr = string.replace(labelstr,'\r', '')
        label.config(text=labelstr)
```

load_data方法也就是我们需要的重新装入数据的功能，所以把它与reload（重新装入）按钮绑定。然而，删除一项是很简单的。我们曾提到，当我们从磁盘装入实例时做的第一件事就是记录对应的文件名，这里我们就用这个信息来删除文件，然后重新装入数据，列表框就自动地更新了：

```
def delentry(self):
    os.remove(os.path.join(self.storagedir,self.selection._filename))
    self.load_data()
```

这个程序很可能是本书里最难理解的一个，因为它大量地使用了复杂而强大的Tkinter 库。Tk 和 Tkinter 都有文档。

设计上的考虑

可以把CGI脚本*feedback.py*和GUI程序FormEditor看成操作同一个数据集（存

注 6： 需要 replace 操作是因为 Tk 把 Windows 的 \r\n 序列看作两个回车。

Tkinter 文档

Tkinter的文档正变得越来越好。Tkinter原来是Steen Lumholdt写的，当时他在用Python时觉得需要一个图形界面。不过他没有写太多的文档。Tkinter已经更新了多次，主要都是Guido van Rossum做的。Tkinter的文档还是不够完整。不过，还是有一些文档可用，到你读到这里时，也许会有更多的文档。

- 最完整的文档是Fredrik Lundh的，可在下面的Web地址找到：http://www.pythonware.com/library/tkinter/introduction/index.htm
- 一个较早的但仍然有用的称为“ Matt Conway的救命稻草 ”的文档的网址是：http://www.python.org/doc/life-preserver/index.html
- 《Programming Python》也有关于Tkinter的文档，特别是第十一、第十二和第十六章。
- 可能有更多的：参见python.org网站的Tkinter部分：http://www.python.org/topics/tkinter/。

在磁盘上的实例）的不同方式。什么时候应该用基于Web的接口？什么时候应使用一个普通的GUI接口？选择应基于几个因素：

- 用一个框架实现需要的功能是否容易？
- 你可以要求用户安装什么软件来访问或修改数据？

Web界面很适合于这样的情况，数据操作的复杂性较低，并且方便用户比功能齐全更重要。另一方面，用一个图形界面库建造的实际程序有最大的灵活性，代价是必须教用户安装使用专门的程序。Python在有经验的程序员中成功的原因之一，就是允许他们选择编程框架，而不是强迫他们使用某种语言设计者头脑里的框架。使用浏览器做界面开发一个全功能的应用也是可能的。Zope就是这样一个框架，可以免费从Digital Creations得到（遵循一种开源许可证）。如果你对开发基于Web的成熟程序感兴趣，试一试Zope（参见附录一“ Python资源 ”）。

JPython：Python 和 Java 的结合

JPython 是最近由 Jim Hugunin 用 Java 写的 Python。这对于 Python 社团和 Java 社团来说都是令人激动的进展。Python 的用户很高兴他们的 Python 知识可以转移到基于 Java 的开发环境，Java 程序员很高兴能够用 Python 脚本语言作为控制 Java 系统的一种方式，通过一个解释环境来测试和学习 Java 库。

JPython 可以从 http://www.jpython.org/ 得到，与 CPython（为区分 JPython 而使用对 Python 本身的称呼）的许可条款相似。

JPython 安装包括几个部分：

- `JPython`：等价于我们书中用的 Python。
- `JPythonc`：把 JPython 程序编译成 Java 类文件。结果类文件可用在任何 Java 类文件可用的地方，例如作为 applet，Servlet 或 bean。
- 标准库里面的大多数模块。
- 一些演示 JPython 编程的程序。

使用 JPython 与使用 Python 十分相似：

```
~/book> jpython
JPython 1.0.3 on Java1.2beta4
Copyright 1997-1998 Corporation for National Research Initiatives
>>> 2 + 3
5
```

实际上，JPython 与 CPython 看起来几乎完全一样。两者区别的最新列表可以访问 http://www.jpython.org/docs/differences.html，最重要的区别是：

- JPython 比 CPython 慢。慢多少依赖于测试的代码以及使用的 Java 虚拟机。但是，JPython 的作者承诺会进行优化，将来的版本会与 Python 一样快。
- JPython 里缺少一些内置成分和库模块。例如 `os.system()` 调用就没有实现，原因是在 Java 与操作系统的交互下这样做很困难。一些大的扩展模块如 Tkinter 图形界面框架没有实现，因为基础工具（如 Tk/Tcl）在 Java 里没有。

JPython 使 Python 程序员可以访问 Java 库

JPython 与 CPython 最重要的区别是，JPython 使 Python 程序员可以无缝地访问 Java 库。考虑下面的程序 *jpythondemo.py*，输出显示在图 10-5。

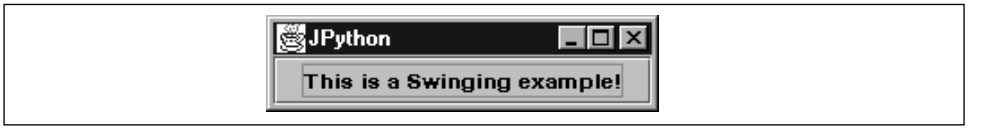

图 10-5　jpythondemo.py 的输出

```
from pawt import swing
import java

def exit(e): java.lang.System.exit(0)

frame = swing.JFrame('Swing Example', visible=1)
button = swing.JButton('This is a Swinging button!', actionPerformed=exit)
frame.contentPane.add(button)
frame.pack()
```

这个程序演示了一个使用 Java Swing GUI（注 7）框架的 Python 程序，它非常简单。第一行导入了 Java 的 `swing` 包（`pawt` 模块计算 swing 包的准确位置，可能是 `java.awt.swing`，或 `com.sun.java.swing`，或 `javax.swing`）。第二行导入另一个 Java 包，我们需要它的 `java.lang.System.exit()` 调用。第四行创建了一个 `JFrame`，把它的 `visible` 属性设为真。第五行创建了一个 `JButton`，指定了相应的标签和命令。最后两行把 `JButton` 放到 `JFrame` 里，并把它们都设为可见的。

有经验的 Java 程序员也许对 *jpythondemo.py* 里的部分代码感到意外，因为它与等价的 Java 程序有些不同。为了使 Python 程序员尽可能地容易使用 Java 库，JPython 在幕后做了许多工作。例如，当 JPython 导入一个 Java 包时，它积极地跟踪相应的包，然后使用 Java 的反射（reflection）API 找到包的内容，以及相应的类和方法。JPython 也会在 Python 和 Java 的数据类型之间作转换。例如在 jpythondemo.py 里，按钮的文字（`'This is a Swinging example!'`）是一个 Python 字符串。

注 7：　Swing 和 Java 基础类的文档可参考 http://java.sun.com/products/jfc/index.html。或者参考 O'Reilly 出版的《Java Swing》。

在调用 JButton 的构造函数前，JPython 查看可以用哪一个构造函数（比如不会使用第一个参数是 Icon 的），并自动地把 Python 字符串转换为 Java 字符串。更复杂的机制使得我们可在JButton的构造函数里方便地用actionPerformed=exit这个关键字参数。这个惯用法在Java里是不可能的，因为Java不能把函数（或方法）当作对象来操作。JPython 里就不必创建只有一个方法 actionPerformed 的 ActionListener 类，当然如果你喜欢烦琐的用法，这也是可以的。

JPython: Java 的脚本语言

JPython 正变得流行起来，因为它可以帮助程序员探索大量正在出现的 Java 库，而且是在交互式的、快捷的环境里。而且实践证明把Python作为脚本语言嵌入到Java 框架里也是有用的，可以让最终用户（不是开发人员）做定制、测试以及别的编程任务。在 JPython 的目录 *demo/embed* 里有一个把 Python 解释器嵌入 Java 程序的例子。

一个真正的 JPython/Swing 应用：grapher.py

*grapher.py*程序（输出显示在图10-6中）允许用户图形化地探索数学函数的行为。它也是基于 Swing GUI 工具库的。有两个输入 Python 代码的文本输入框。第一个是在画出函数前引用的任意的Python程序，通过它导入需要的模块，定义也许会用到的函数。第二个输入框（名为 Expression:）应该是一个 Python 的表达式（比如 sin(x)），而不是一个语句。依次用每个数据点调用它（变量 x 的值设为水平坐标）。

用户可以选择是画一个线图还是填充图，以及画的点数、颜色。最后，用户可以把设置保存到磁盘上，以便以后再装入使用（用 pickle 模块）。下面就是 *grapher.py* 程序：

```
from pawt import swing, awt, colors, GridBag
RIGHT = swing.JLabel.RIGHT
APPROVE_OPTION = swing.JFileChooser.APPROVE_OPTION
import java.io
import pickle, os

default_setup = """from math import *
```

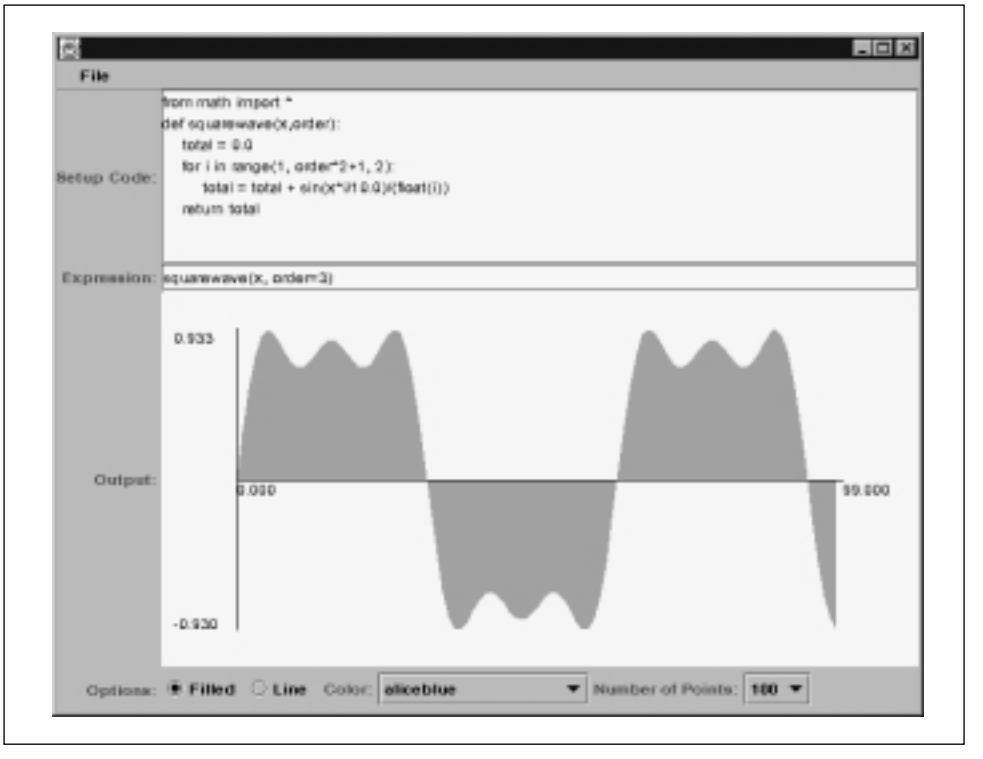

图 10-6　grapher.py 的输出

```
    def squarewave(x,order):
        total = 0.0
        for i in range(1, order*2+1, 2):
            total = total + sin(x*i/10.0)/(float(i))
        return total
"""
default_expression = "squarewave(x, order=3)"

class Chart(awt.Canvas):
    color = colors.darkturquoise
    style = 'Filled'

    def getPreferredSize(self):
        return awt.Dimension(600,300)

    def paint(self, graphics):
        clip = self.bounds
        graphics.color = colors.white
```

```
            graphics.fillRect(0, 0, clip.width, clip.height)

            width = int(clip.width * .8)
            height = int(clip.height * .8)
            x_offset = int(clip.width * .1)
            y_offset = clip.height - int(clip.height * .1)

            N = len(self.data); xs = [0]*N; ys = [0]*N

            xmin, xmax = 0, N-1
            ymax = max(self.data)
            ymin = min(self.data)

            zero_y = y_offset - int(-ymin/(ymax-ymin)*height)
            zero_x = x_offset + int(-xmin/(xmax-xmin)*width)

            for i in range(N):
                xs[i] = int(float(i)*width/N) + x_offset
                ys[i] = y_offset - int((self.data[i]-ymin)/(ymax-ymin)*height)
            graphics.color = self.color
            if self.style == "Line":
                graphics.drawPolyline(xs, ys, len(xs))
            else:
                xs.insert(0, xs[0]); ys.insert(0, zero_y)
                xs.append(xs[-1]); ys.append(zero_y)
                graphics.fillPolygon(xs, ys, len(xs))

            # draw axes
            graphics.color = colors.black
            graphics.drawLine(x_offset,zero_y, x_offset+width, zero_y)
            graphics.drawLine(zero_x, y_offset, zero_x, y_offset-height)

            # draw labels
            leading = graphics.font.size
            graphics.drawString("%.3f" % xmin, x_offset, zero_y+leading)
            graphics.drawString("%.3f" % xmax, x_offset+width, zero_y+leading)
            graphics.drawString("%.3f" % ymin, zero_x-50, y_offset)
            graphics.drawString("%.3f" % ymax, zero_x-50, y_offset-height+leading)

    class GUI:
        def __init__(self):
            self.numelements = 100
            self.frame = swing.JFrame(windowClosing=self.do_quit)

            # build menu bar
            menubar = swing.JMenuBar()
```

```
file = swing.JMenu("File")
file.add(swing.JMenuItem("Load", actionPerformed = self.do_load))
file.add(swing.JMenuItem("Save", actionPerformed = self.do_save))
file.add(swing.JMenuItem("Quit", actionPerformed = self.do_quit))
menubar.add(file)
self.frame.JMenuBar = menubar

# create widgets
self.chart = Chart(visible=1)
self.execentry = swing.JTextArea(default_setup, 8, 60)
self.evalentry = swing.JTextField(default_expression,
                                  actionPerformed = self.update)

# create options panel
optionsPanel = swing.JPanel(awt.FlowLayout(
    alignment=awt.FlowLayout.LEFT))

# whether the plot is a line graph or a filled graph
self.filled = swing.JRadioButton("Filled",
                                 actionPerformed=self.set_filled)
optionsPanel.add(self.filled)
self.line = swing.JRadioButton("Line",
                               actionPerformed=self.set_line)
optionsPanel.add(self.line)
styleGroup = swing.ButtonGroup()
styleGroup.add(self.filled)
styleGroup.add(self.line)

# color selection
optionsPanel.add(swing.JLabel("Color:", RIGHT))
colorlist = filter(lambda x: x[0] != '_', dir(colors))
self.colorname = swing.JComboBox(colorlist)
self.colorname.itemStateChanged = self.set_color
optionsPanel.add(self.colorname)

# number of points
optionsPanel.add(swing.JLabel("Number of Points:", RIGHT))
self.sizes = [50, 100, 200, 500]
self.numpoints = swing.JComboBox(self.sizes)
self.numpoints.selectedIndex = self.sizes.index(self.numelements)
self.numpoints.itemStateChanged = self.set_numpoints
optionsPanel.add(self.numpoints)

# do the rest of the layout in a GridBag
self.do_layout(optionsPanel)
```

```
    def do_layout(self, optionsPanel):
        bag = GridBag(self.frame.contentPane, fill='BOTH',
                      weightx=1.0, weighty=1.0)
        bag.add(swing.JLabel("Setup Code: ", RIGHT))
        bag.addRow(swing.JScrollPane(self.execentry), weighty=10.0)
        bag.add(swing.JLabel("Expression: ", RIGHT))
        bag.addRow(self.evalentry, weighty=2.0)
        bag.add(swing.JLabel("Output: ", RIGHT))
        bag.addRow(self.chart, weighty=20.0)
        bag.add(swing.JLabel("Options: ", RIGHT))
        bag.addRow(optionsPanel, weighty=2.0)
        self.update(None)
        self.frame.visible = 1
        self.frame.size = self.frame.getPreferredSize()

        self.chooser = swing.JFileChooser()
        self.chooser.currentDirectory = java.io.File(os.getcwd())

    def do_save(self, event=None):
        self.chooser.rescanCurrentDirectory()
        returnVal = self.chooser.showSaveDialog(self.frame)
        if returnVal == APPROVE_OPTION:
            object = (self.execentry.text,  self.evalentry.text,
                      self.chart.style,
                      self.chart.color.RGB,
                      self.colorname.selectedIndex,
                      self.numelements)
            file = open(os.path.join(self.chooser.currentDirectory.path,
                        self.chooser.selectedFile.name), 'w')
            pickle.dump(object, file)
            file.close()

    def do_load(self, event=None):
        self.chooser.rescanCurrentDirectory()
        returnVal = self.chooser.showOpenDialog(self.frame)
       if returnVal == APPROVE_OPTION:
            file = open(os.path.join(self.chooser.currentDirectory.path,
                     self.chooser.selectedFile.name))
            (setup, each, style, color,
             colorname, self.numelements) = pickle.load(file)
            file.close()
            self.chart.color = java.awt.Color(color)
            self.colorname.selectedIndex = colorname
            self.chart.style = style
            self.execentry.text = setup
```

```
            self.numpoints.selectedIndex = self.sizes.index(self.numelements)
            self.evalentry.text = each
            self.update(None)

    def do_quit(self, event=None):
        import sys
        sys.exit(0)

    def set_color(self, event):
        self.chart.color = getattr(colors, event.item)
        self.chart.repaint()

    def set_numpoints(self, event):
        self.numelements = event.item
        self.update(None)

    def set_filled(self, event):
        self.chart.style = 'Filled'
        self.chart.repaint()

    def set_line(self, event):
        self.chart.style = 'Line'
        self.chart.repaint()

    def update(self, event):
        context = {}
        exec self.execentry.text in context
        each = compile(self.evalentry.text, '<input>', 'eval')
        numbers = [0]*self.numelements
        for x in xrange(self.numelements):
            context['X'] = float(x)
            numbers[x] = eval(each, context)
        self.chart.data = numbers
        if self.chart.style == 'Line':
            self.line.setSelected(1)
        else:
            self.filled.setSelected(1)
        self.chart.repaint()
GUI()
```

这个程序的逻辑是相当简单直接的，类和方法的名字使我们很容易跟随控制流。这个程序的大部分可以用类似的Java代码来写（但要大得多）。粗体的部分显示了使用Python的威力：在模块的最前面定义了Setup和Expression这两个文本框的缺省值。前者从math模块里导入了函数，并定义了一个squarewave函数。

后者指定了一个对该函数的调用，有一个 order 参数（随着该参数的增加，画出的图越来越像一个方波，所以函数取名为 squarewave，即方波）。如果你安装了 Java、Swing 和 JPython，不妨用其他的 Setup 和 Expression 来玩一玩。

这个例子里用 Python 取代 Java 的关键是在 update 方法里：它简单地调用标准的 Python 的 exec 语句，以 Setup 里的代码作为参数，然后用编译好的 Expression 代码，循环地对每个坐标调用 eval 函数。用户可以在这个文本框里自由地使用其他的 Python 函数或表达式！

JPython 仍然是一项正在进行的工作：Jim Hugunin 不断地改进和优化 Python 和 Java 之间的接口。作为 Python 的另一个实现，JPython 也影响着 Guido van Rossum 对 Python 语言核心的抉择。

其他的框架和应用

由于篇幅有限，我们只能介绍与 Python 相关的几个最流行的框架。另外还有几个值得介绍的（也许会很有用）框架，我们简短地介绍如下：

Python 图像库（PIL）

Python 图像库是一个 Fredrik Lundh 写的大型框架，用于创建、操作、转换和保存各种格式的位图（如 GIF、JPEG、PNG）。它有 Tk 和 Pythonwin 的接口，使得我们可以用 Tk 或 Pythonwin 代码显示 PIL 产生的图像。PIL 的网址在：http://www.pythonware.com。

数字 Python（NumPy）

数字 Python 是 Python 的一组扩展，可快速而漂亮地操作较大的数字数组，它是由 Jim Hugunin（JPython 的作者）写的，得到了 Matrix-SIG（SIG 参见附录一）的订阅者的支持。由于 Jim 开始了 JPython 的工作，数字 Python 已经由 Lawrence Livermore 国家实验室的人接管了。NumPy 是科学家和工程师喜欢的一个非常强大的工具。更多的信息可以在 Python 主站点找到（http://www.python.org/topics/scicomp/）。

下面是一个典型的 NumPy 的例子 *numpytest.py*，以及它产生的数据：

```
from Numeric import *
coords = arange(-6, 6, .02)                 # 创建一组坐标
xs = sin(coords)                            # 取得 x 坐标的正弦值
ys = cos(coords)*exp(-coords*coords/18.0)
zx = xs * ys[:,NewAxis]                     # x 行乘以 y 列
```

如果你还记得你学过的数学，你也许算出 xs 是一个 -6 到 6 之间的数的正弦值数组，ys 是同一组数的余弦值（乘以一个指数函数）数组。zs 是这两个数组的外积。如果你想知道结果会是什么样，你可以把数组转换为一个图像（比如用上述的 PIL），图像可参考图 10-7。

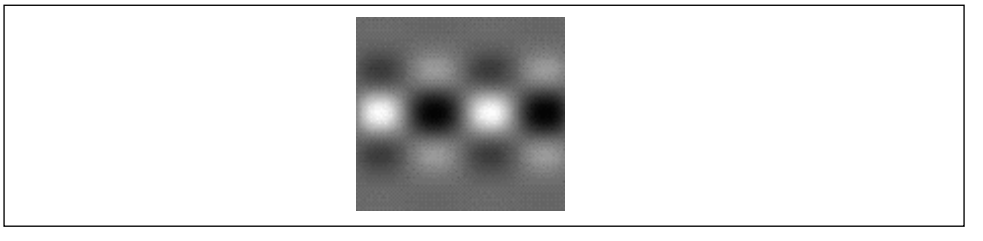

图 10-7　numpytest.py 的数组 zs 的图形表示

NumPy 使得你可以高效地操作大型的数组。代码运行时比用大的列表要快得多。许多的Python用户永远不会碰上这样的问题，但很多科学家和工程师每天都用得着。

SWIG

Python 的扩展模块可以用 C 或 C++ 来写，可以很容易地扩展新的函数和类型。写这种扩展模块的指南可以在库参考里找到，在《Programming Python》一书里也有介绍。扩展模块的一个常见的用法是写一个已存在的库的接口，库里面也许有成百上千的函数。在这种情况下，自动化工具就可以救命了。David Beazley 写的 SWIG（简单封装接口产生器，the Simple Wrapper Interface Generator）是同类工具中最流行的。在 http://www.swig.org 可找到它并且文档齐全。用 SWIG 你可以写简单的接口定义，然后由 SWIG 按照规则产生 C 程序。SWIG 有一个不错的特点，当用它来封装 C++ 类库时，它自动地产生 Python 的所谓的影子类，让

用户像操纵Python的类一样操纵C++类。SWIG也可以为其他语言（包括Perl,Tcl)创建扩展。

Python 复合组件（Pmw）

任何真打算用 Tkinter 做 GUI 工作的人应该看看 Pmw，它是一个 100% 的基于 Tkinter 的框架，用于创建复合组件。它是由 Greg McFarlane 写的，学会它肯定会有回报。它的主页在 http://www.dscpl.com.au/pmw/。

ILU 和 Fnorb

如果你对程序间的对话感兴趣，而且希望一个不需要 COM 支持的方案，那么有很多其他的类似功能的包。两个最常用的是 ILU 和 Fnorb。

ILU 意思是 Xerox PARC 的 Inter Language Unification project（语言间归一工程）。它是自由、支持良好、稳定而高效率的，并且除了 Python 外，还支持 C, C++, Java, Common Lisp, Modula-3 以及 Perl 5。网址在 ftp://ftp.parc.xerox.com/pub/ilu/ilu.html。

与 ILU 不同，Fnorb 是用 Python 写的，并且只支持 Python。它对学习了解 CORBA 系统特别有帮助，一旦你学会了 Python，它是很容易安装使用的。网址在 http://www.dstc.edu.au/Fnorb/。

练习

1. 欺骗 Web。你也许没有一个 Web 服务器可用来使用 *formletter.py* 和 *FormEditor.py*，因为它们要用到 CGI 脚本产生的数据。作为练习，写一个程序模拟产生与 GCI 脚本类似的文件。
2. 清除。*formletter.py* 程序有一个严重的问题：也就是当在晚上运行时，任何抱怨都会导致打印一封信。每个晚上都会发生，因为没有机制能标识已经处理了一封信，而不需要再处理它了。纠正这个错误。

3. 给 grapher.py 增加绘图参数。修改 *grapher.py*，让用户可指定返回 x 和 y 的表达式，而不只是 y。例如，用户可以写：`sin(x/3.1),cos(x/6.15)`，（注意逗号：这是一个元组！），得到的输出类似于图 10-8 所示。

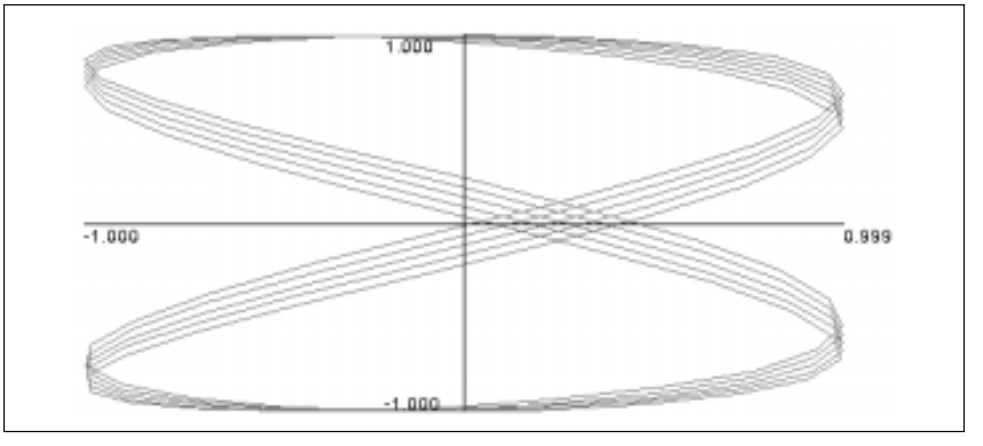

图 10-8　练习 3 的输出