

第 1 章

准备工作

Python 是一种对开发人员很有价值的语言。

不管你或你的客户使用的是什么操作系统，它都能够正常工作。除非使用平台相关的功能，或者使用了特定平台的程序库，否则就可以跨平台使用。例如，可以将 Linux 环境下开发的程序部署到安装了其他操作系统的平台上。不过，这并不是其独有的特性（Ruby、Java 等许多语言都能够做到）。综合考虑本书介绍的其他功能，Python 将是公司首选开发语言之一。

本章将介绍开始使用 Python 之前必须掌握的知识，不管使用的是哪种运行环境。主要包括：

- 如何安装 Python；
- 如何使用并增强其命令行；
- 如何通过安装 `setuptools` 扩展 Python；
- 如何配置开发环境，包括老款或新型方法。

如果你对 Python 很熟悉，并且安装了自己喜欢的代码编辑器，那么建议你跳过本章的 1.1 节，直接阅读其他章节，在那里可以找到一些增强编程环境的小技巧。不过，请一定不要跳过关于 `setuptools` 的小节，对于本书其他章节而言它是必须安装的工具。

如果使用的是 Windows 操作系统，那么必须确保安装了本章中介绍的所有软件，运行本书中提供的所有示例都将依赖于这些环境。

1.1 安装 Python

Python 语言能够在各种操作系统中使用，包括 Linux、Macintosh 和 Windows。在 Python

网站的主下载页面 <http://www.python.org/download> 中可以找到由 Python 核心开发团队提供的各种版本。针对其他平台的版本是由社区中的其他开发人员维护的，在“贡献（dedicated）”页面中有相关信息（参见 <http://www.python.org/download/other>）。在此，甚至可以找到早年所用操作系统的版本。



如果有一台电脑，就可以使用 Python，而不管这台电脑上安装的是什么操作系统。

否则，将其丢在一边吧。

在安装 Python 之前，先来看看各种现有的实现版本。

1.1.1 Python 实现版本

Python 的主实现版本是用 C 语言编写的，被称为 CPython。当人们提到 Python 时，通常指的就是它。随着该语言的不断演化，C 语言实现也随着改变。除了 C 语言实现之外，Python 也提供了其他的实现版本，以保持对主流的跟踪。这些实现版本通常比 CPython 要落后几个里程碑，但也为 Python 在特定环境中的使用和宣传提供了巨大的机会。

1. Jython

Jython 是 Python 语言的 Java 实现版本。它将其代码编译成 Java 字节码，因此开发人员可以在 Python 模块（在 Python 中，存储代码的文件被称为模块）中自由地使用 Java 类。Jython 让大家可以在诸如 J2EE 之类的复杂应用程序上，将 Python 作为顶层脚本语言使用。同时，也可以将 Java 应用程序引入到 Python 应用程序中。使 Apache Jackrabbit（一种基于 JCR 实现的文档仓库 API 能够应用于 Python 程序，就是 Jython 存在价值的一个良好示例。Jython 的当前版本是 2.2.1，Jython 开发团队正在研发 2.5 版本。现在许多诸如 Pylons 之类的 Python Web 框架正在通过 Jython 被移植到 Java 世界中。

2. IronPython

IronPython 将 Python 引入了 .NET 环境。该项目是由微软提供支持的，IronPython 的主开发人员供职于该公司。其最新的稳定版本是 1.1（发布于 2007 年 4 月），它是对 Python 2.4.3 版本的实现。它能够在 ASP.NET 环境中使用，开发人员在 .NET 应用程序中使用 Python 代码的方法和与在 Java 环境中使用 Jython 一样。它对于该语言的推广起着十分重要的作用。.NET 社区和 Java 社区一样，都是最大的开发人员社区之一。TIOBE 社区索引也显示了 .NET 语言

是一颗冉冉升起的新星。

3. PyPy

PyPy 或许是各种实现版本中最有趣的一款,其目标是用 Python 语言重写 Python。在 PyPy 中,Python 解释程序本身就是用 Python 编写的。对于 Python 实现版本之一的 CPython 而言,需要一个 C 代码层来承载具体的工作。但在 PyPy 实现版本中,这个 C 代码层将彻底用纯 Python 语言重写。这就意味着可以在运行态时改变 Python 解释程序的行为,以及实现一些在 CPython 实现版本中难以实现的代码模式(参见 <http://codespeak.net/pypy/dist/pypy/doc/objspace-proxies.html>)。PyPy 的运行速度远低于 CPython,不过在近几年中有了很大改善。随着诸如 JIT 编译器之类的技术的引入,其运行速度的提升是很有希望的。它当前的速度因子大概在 1.7~4 之间,针对的目标版本是 Python 2.4。PyPy 可以被看作汇编程序领域开发的领军项目,它在许多领域的创新都是先驱,其他主流的实现版本必将会从中受益。总体来看,PyPy 的开发更多是出于学术研究的动机,关注该项目的都是那些热衷于深入研究语言内部机制的人。因此,通常不会在具体的产品中使用它。

4. 其他实现版本

Python 还有许多其他实现版本和移植版本。例如, Nokia 就在其 S60 系列手机上实现了 Python 2.2.2。Michael Lauer 开发维护了一个应用于 ARM 芯片上的 Linux 环境中的 Python 移植版本,使其能够在诸如 Sharp Zaurus 之类的设备上使用。

这样的例子还有很多,不过本书关注的是在 Linux、Windows 和 Mac OS X 上安装 CPython。

1.1.2 在 Linux 环境下安装

如果使用的是 Linux 操作系统,那么或许已经安装了 Python。因此,可以试着在 shell 环境中调用它,如下所示。

```
tarek@dabox: ~$ python
Python 2.3.5 (#1, Jul 4 2007, 17:28:59)
[GCC 4.1.2 20061115 (prerelease) (Debian 4.1.1-21)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

如果找到该命令,那么将进入 Python 提供的交互式 shell 环境,其提示符为>>>。同时还将显示编译器相关的信息,包括编译 Python 的语言(在此是 GCC)以及目标系统环境(在此是 Linux)等信息。如果使用的是 Windows,那么可以用微软公司的 Visual Studio 作为编

译器。同时，Python 的版本号也将会显示出来。请确保运行的是最新的稳定版本（在本书上市时或许已经是 2.6 版本了）。

如果没有预装，也可以自己动手在系统上安装任意版本的 Python，所需的操作都是很普通的。Python 的版本可以从其文件全名中得知，也可以通过 Python 命令来了解。当然，是否能够执行该命令，取决于环境变量 `path` 的设置，如下所示。

```
tarek@dabox: ~$ which python
/usr/bin/python
tarek@dabox: ~$ python<tab>
python  python2.3  python2.5
python2.4
```

如果没有找到 `python` 命令（这在 Linux 系统中并不常见），还可以通过 Linux 系统中的软件包管理工具安装它，例如 Debian 中提供的 `apt`、Red Hat 中提供的 `rpm` 等。当然，也可以通过编译源代码的方式来安装。

虽然始终通过软件包管理工具安装是个好习惯，但在此我们仍然将对两种安装方法（通过软件包管理工具安装和通过编译源代码来安装）做进一步介绍。不过，在你的软件包管理工具中，不一定会提供 Python 的最新版本。

1. 通过软件包管理工具安装

安装 Python 最常见的方法是使用 Linux 中的软件包管理工具，这样还能使软件升级更加容易。根据使用的 Linux 发行版本的不同，可能使用以下 3 种不同的命令：

- `apt-get install python` 在基于 Debian 的发行版本中，诸如 Ubuntu；
- `urpmi python` 在基于 rpm 的发行版本中，诸如 Fedora 或 Red Hat 系列；
- `emerge python` 针对 Gentoo 发行版本。

如果在这些软件包安装工具中没有列出最新版本的 Python，就需要手动进行安装了。

当选择完全安装时，可能会同时安装一些其他的软件包。它们都是可选的，不一定会用到它们。不过，如果需要编写 C 扩展，或者对程序进行优化，那么将会用到它们。当选择完全安装时将同时安装以下软件包：

- `python-dev` 在编译 C 模块时所需的 Python 头文件；
- `python-profiler` 它包含了一些针对完全遵循 GPL 协议的 Linux 发行版本（诸如 Debian 或 Ubuntu）的非 GPL 协议模块（Hotshot 优化器）；
- `gcc` 在编译包含 C 代码的扩展时所需的软件包。

2. 通过编译源代码的方式安装

手动安装就是通过一个 `cmmi` 过程（顺序执行 `configure`、`make`、`make install` 命令）来完

成 Python 的编译并将其部署到系统中。在 <http://python.org/download> 中可以找到 Python 最新版本的源代码包。

使用 wget 下载



wget 程序是一个 Gnu 项目，是用来下载软件的命令行工具，它在所有平台中都能使用。在 <http://gnuwin32.sourceforge.net/packages/wget.htm> 中可以下载在 Windows 平台中使用的二进制版本。

在 Linux 或 Mac OS X 平台中，可以通过诸如 apt 或 MacPorts 之类的软件包管理工具来安装。

编译 Python 时需要使用 make 和 gcc。

- **make** 读取配置文件（通常名为 Makefile），检查编译该程序所需的各方面条件是否满足，同时将开始执行编译操作。它通过 configure 和 make 命令调用。
- **gcc** GNU C 编译器，是在构建程序时使用最广的开源编译器。

首先确保系统中已经安装了它们。在某些 Linux 发行版本中（如 Ubuntu），可以直接通过安装 build-essentials 软件包来安装这些构建工具。

在安装 Python 时，应执行以下命令序列。

```
cd /tmp
wget http://python.org/ftp/python/2.5.1/Python-2.5.1.tgz
tar -xzf tar -xzf Python-2.5.1.tgz
cd Python-2.5.1
./configure
make
sudo make install
```

采用这样的过程安装，会同时安装针对二进制安装版所需的头文件，它们通常被放在 python-dev 软件包。在源代码包中，也包含 Hotshot 优化器。当完成这些步骤之后，就可以在命令行使用 Python 了。

现在，你的系统已经支持 Python 了，让我们庆祝一下吧！

1.1.3 在 Windows 环境下安装

在 Windows 环境下编译 Python 和在 Linux 环境下类似。但这样做十分麻烦，因为还需要安装复杂的编译环境。python.org 的下载区中提供了标准的安装程序，它提供了向导式的安装过程（如图 1.1 所示），十分简单易用。

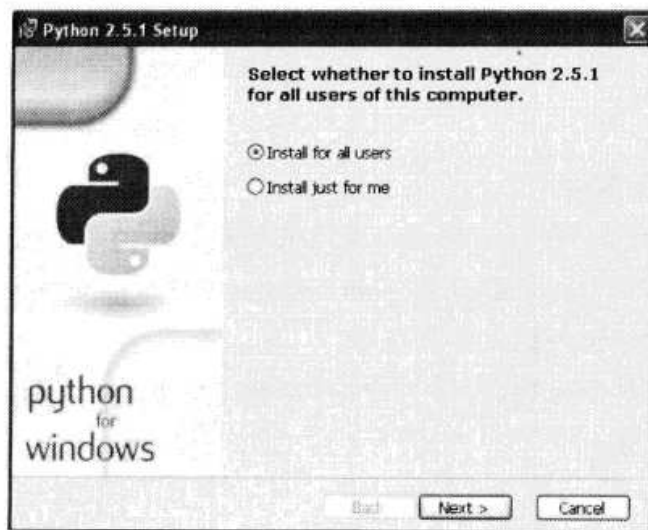


图 1.1

1. 安装 Python

如果一切都采用默认选项，Python 将被安装在 `c:\Python25` 目录下，而不是像其他软件那样被放在 Program Files 文件夹中。这样可以避免在环境变量 `path` 中出现空格。

最后一步是修改环境变量 `PATH`，以便能够在 DOS 命令行窗口中调用 Python。

对于绝大多数的 Windows 版本，都可以通过以下步骤完成：

- 在桌面或 start (开始) 菜单的 My Computer (我的电脑) 图标上单击右键，进入 System Properties (系统属性) 对话框；
- 切换到 Advanced (高级) 标签页；
- 单击 Environment Variables (环境变量) 按钮；
- 编辑系统变量 `PATH` 的值，添加上两个新的路径，并用 “; (分号)” 分隔。

要添加的搜索路径如下：

- `c:\Python25` 以便能够调用到 `python.exe`；
- `c:\Python25\Scripts` 以便调用通过扩展为 Python 添加的第三方脚本。

这样，就可以在命令行窗口中运行 Python 了。具体来说，就是单击 Start (开始) 菜单中的快捷方式 Run (运行)，输入命令 `cmd`，然后在弹出的命令行窗口中调用 `python`，如下所示。

```
C:\> python
Python 2.5.2 (#71, Oct 18 2006, 08:34:43) [MSC v.1310 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

这样就能够运行 Python 了。但这样的环境和 Linux 用户的环境相比还不够完整，要实现

本书中介绍的所有内容，还需要安装 MinGW。

2. 安装 MinGW

MinGW 是针对 Windows 平台开发的编译器。它实现了 gcc 编译器的各种功能，提供了相同的程序库和头文件。MinGW 可以彻底代替 Microsoft Visual C++。可以在系统中保留各种编译器，以便根据自己的需要选择不同的编译器。

首先，要下载 MinGW，找到指向 Sourceforge（参见 <http://sourceforge.net>——最大的、针对开源项目开发人员的网站）的链接。自动安装是最好的选择，它能够自动完成所有的操作。在此，可以下载安装程序并运行它。

和 Python 一样，也需要对环境变量 PATH 进行修改，增加 c:\MinGW\bin，以便能够调用它提供的命令。设置了 PATH 变量之后，就可以在命令行窗口中运行 MinGW 的各种命令，如下所示。

```
C:\>gcc -v
Reading specs from c:/MinGW/bin/../../lib/gcc-lib/mingw32/3.2.3/specs
Configured with: ../gcc/configure --with-gcc --with-gnu-ld --with-gnu-as
--host=
mingw32 --target=mingw32 --prefix=/mingw --enable-threads --disable-nls
--enable
-languages=c++,f77,objc --disable-win32-registry --disable-shared --
enable-sjlj-
exceptions
Thread model: win32
gcc version 3.2.3 (mingw special 20030504-1)
```

这些命令无法手动调用，但当 Python 编译器需要时会自动调用。

3. 安装 MSYS

在 Windows 平台中，还有一个需要安装的工具是 MSYS (Minimal SYStem)。它能在 Windows 平台上提供一个 Bourne Shell 命令行环境，在该环境中可以实现 Linux 或 Mac OS X 操作系统中常见的命令，如 cp、rm 等。

这听起来有点过分，毕竟 Windows 平台也提供了相应的工具，不仅在图形界面中提供了，在 MS-DOS 命令行窗口中也实现了。但该工具对于那些跨多个操作系统工作的开发人员而言，能够提供一个统一的命令格式。

下载 MSYS，然后将其安装到自己的系统。如果选择标准安装，那么 MSYS 将被安装在 c:\msys 目录下。因此，还需要像 MinGW 那样在 PATH 变量中增加 c:\msys\1.0\bin。

在本书中，将在所有示例中使用 Bourne Shell 命令。因此如果采用的是 Windows 平台，那么请安装 MSYS。



现在你已经安装了 MinGW 和 MSYS，再也不必羡慕那些安装了 Linux 操作系统的人，因为它们已经在 Windows 平台上模拟了 Linux 开发环境中所提供的各种主要功能。

1.1.4 在 Mac OS X 环境下安装

Mac OS X 是基于 Darwin 内核的，而 Darwin 则是基于 FreeBSD 的。这使得该平台与 Linux 很相似，也很兼容。Apple 在此之上添加了一个图形引擎（Quartz）及一个特殊的文件树。

从 Shell 角度看，主要的区别在于系统文件树的组织方式。例如，可能找不到一个名为 /home 的根文件夹，但可以找到 /Users 文件夹。应用程序通常会被安装在 /Library 目录下。虽然也有使用 /usr/bin 的，不过那是 Linux 中常用的。

和 Linux 与 Windows 平台一样，在 Mac OS X 上安装 Python 也有两种方法。可以通过软件包安装程序安装，也可以通过编译源程序的方式安装。通过软件包安装程序是最简单的方法，不过你可能会希望自己动手编译 Python。最新版本的 Python 也提供了相应的二进制安装版本。

1. 通过软件包安装程序安装

在 Mac OS X 的最新版本（本书写作时是 Leopard）中打包了 Python。要另外安装 Python，首先要从 <http://www.pythonmac.org/packages> 下载 Python 2.5.x 的二进制包。可下载到一个可以安装到系统中的 .dmg 文件，其中有一个 .pkg 的文件。安装界面如图 1.2 所示。

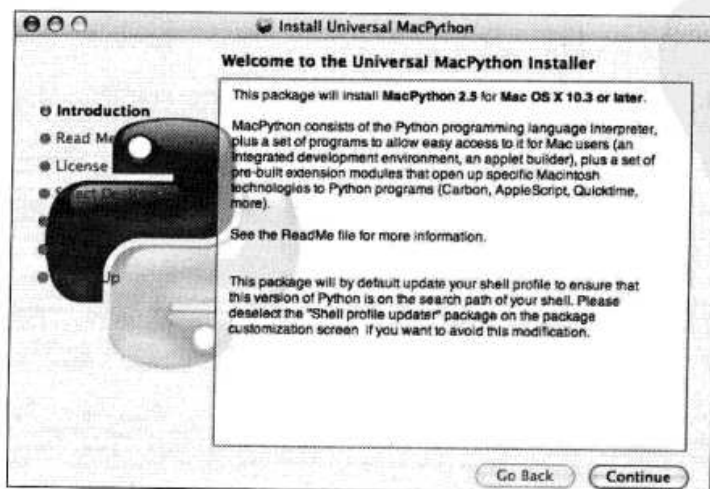


图 1.2

这样将 Python 安装到/Library 文件夹中，然后在系统中创建相应的链接，接下来就可以在 shell 中运行 Python 了。

2. 以编译源程序方式安装

如果想要编译 Python，则需要安装：

- gcc 编译器 Xcode 工具中包含该编译器，可以从安装光盘中找到它，也可以从 <http://developer.apple.com/tools/xcode> 下载；
- MacPorts 一个与 Debian 的软件包管理系统 apt 很类似的软件包管理系统，和在与 Linux 使用 apt 一样，它能够帮你安装相应的依赖包，参见 <http://www.macports.org>。

现在，可以采用与 Linux 中完全相同的步骤来完成编译工作了。

1.2 Python 命令行

执行 python 命令后，将出现 Python 命令行，通过它可以与 Python 解释程序进行交互。它很常用，例如，可以通过它来完成一些小型的计算，如下所示。

```
macziade:/home/tziade tziade$ python
Python 2.5 (r25:51918, Sep 19 2006, 08:49:13)
[GCC 4.0.1 (Apple Computer, Inc. build 5341)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>1 + 3
4
>>>5 * 8
40
```

当按下回车键时，Python 将解释这行程序并马上显示处理结果。该特性是从 ABC 语言中继承而来的，Python 的编程方式受它的影响很深。在代码文档中，所有的示例都是以一个命令行会话的形式展示的。

退出命令行



如果想退出命令行，在 Linux 或 Mac OS X 平台中可以按 Ctrl+D 组合键，在 Windows 平台中则应按 Ctrl+Z 组合键。

由于命令行交互模式在编程过程中扮演了十分重要的角色，因此需要使其变得更加简单易用。

1.2.1 定制交互式命令行

交互式命令行可通过启动文件来配置。当它启动时会查找环境变量 `PYTHONSTARTUP`，并且执行该变量中所指定文件里的程序代码。有些 Linux 发行版本提供了一个默认的启动脚本，它通常放在用户主目录下，文件名是 `.pythonstartup`。按 `Tab` 键时自动补全内容和命令历史。它们都是对命令行的有效增强，这些工具是基于 `readline` 模块实现的（需要 `readline` 程序库）。如果没有这个启动脚本文件，也可以自己创建一个。


下面就是一个最简单的启动脚本文件，它为 Python 命令行添加了按 `<Tab>` 键自动补全内容和命令历史功能。

```
# python startup file
import readline
import rlcompleter
import atexit
import os

# tab completion
readline.parse_and_bind('tab: complete')

# history file
histfile = os.path.join(os.environ['HOME'], '.pythonhistory')
try:
    readline.read_history_file(histfile)
except IOError:
    pass
atexit.register(readline.write_history_file, histfile)
del os, histfile, readline, rlcompleter
```

在用户主目录下创建该文件，并将其命名为 `.pythonstartup`。然后添加环境变量 `PYTHONSTARTUP`，并根据该文件的路径赋予相应的值。

 在 `pbp.script` 包中已经提供了 python 启动脚本，名为 `pythonstartup.py`。从 <http://pypi.python.org/pypi/pbp.scripts> 中下载该文件，然后将文件名改为 `.pythonstartup` 即可。

设置环境变量 PYTHONSTARTUP



如果使用的是 Linux 或 Mac OS X 操作系统,那么最简单的方法就是在自己的用户主目录下创建启动脚本。然后将其链接到环境变量 PYTHONSTARTUP 上,并将其放到系统的 shell 启动脚本中。例如, Bash 和 Korn Shell 使用的是 .profile 文件,可以在该文件中添加如下一行设置。

```
export PYTHONSTARTUP=~/.pythonstartup
```

如果使用的是 Windows 操作系统,那么就要以 administrator 角色登录,添加一个新的环境变量,然后将脚本保存在一个公共的位置上,而非放在特定的用户区域。

当启动交互式命令行时,将会执行 .pythonstartup 脚本,这些新功能将被启用。例如,按 Tab 键自动补全内容功能能够帮助回忆起模块的名称,如下所示。

```
>>> import md5
>>> md5.<tab>
md5.__class__      md5.__file__      md5.__name__
md5.__repr__      md5.digest_size
md5.__delattr__   md5.__getattr__   md5.__new__
md5.__setattr__   md5.md5
md5.__dict__      md5.__hash__      md5.__reduce__
md5.__str__       md5.new
md5.__doc__       md5.__init__      md5.__reduce_ex__ md5.
blocksize
```

还能通过修改该脚本来实现更多自动化功能,例如,让 Python 为模块提供一个进入点,以及提供基于类的解释程序模块(参见 <http://docs.python.org/lib/module-code.html> 中的 code module 部分)。但如果希望拥有一个更高级的交互式命令行,还有一个现成的工具——iPython 可供使用。

1.2.2 iPython: 增强型命令行

iPython 项目的目标是提供一个扩展的命令行。在它所提供的各种功能中,最有意思的包括:

- 动态对象反射;
- 在命令行中调用系统 shell 功能;
- 程序调优的直接支持;

- 调试工具。

要安装 iPython, 首先需要在 <http://ipython.scipy.org/moin/Download> 中下载安装程序, 然后根据针对所使用的操作系统的指南来完成安装。

运行 iPython 提供的 shell 后, 将出现如下提示。

```
tarek@luvdi: ~$ ipython
Python 2.4.4 (#2, Apr 5 2007, 20:11:18)
Type "copyright", "credits" or "license" for more information.
IPython 0.7.2 -- An enhanced Interactive Python.
?      -> Introduction to IPython's features.
%magic -> Information about IPython's 'magic' % functions.
help   -> Python's own help system.
object? -> Details about 'object'. ?object also works, ?? prints more.
In [1]:
```



iPython 和应用程序调试

当需要对程序进行调试时, iPython 将是一个很友好的命令行工具, 特别是针对那些以后台进程形式运行的服务器端代码而言。

1.3 安装 setuptools

Perl 拥有大量的第三程序库, 安装它们也很简单。Perl CPAN 系统使开发人员能够将一组简单的命令集以新程序库的形式发布。近几年中, 在 Python 领域也出现了类似的技术, 并且逐渐成了安装扩展的标准途径。它是基于:

- 一个存储在 Python 官方网站的集中式仓库, 它的名字是 Python Package Index(PyPI), 其前身就是 Cheeseshop (它引用了源于 BBC 的 Monty Python 的原型);
- 一个名为 setuptools 的包管理系统, 它是基于 distutils 开发的, 用来发布代码以及和 PyPI 交互。

在安装这些扩展之前, 需要对总体框架做些必要的解释。


1.3.1 工作原理

Python 附带提供了一个名为 distutils 的模块, 它提供了一系列用于发布 Python 应用程序

的工具。它提供的内容包括：

- 用来提供标准元数据字段（诸如作者名、版权类型等信息）的骨架；
- 一组用来将“包”（在 Python 中，包是一个由一个或多个模块组成的系统文件夹）中的代码构建软件安装包的辅助工具，不仅能够创建出一组预编译的 Python 文件，还能创建在 Windows 中可执行的安装程序。

但 distutils 工具仅适用于包，无法定义包之间的依赖关系。setuptools 通过添加一个基本的依赖系统以及许多相关功能，有效地弥补了该缺陷。它还提供了一个自动包查询程序，它可以自动获取包的依赖关系，并自动完成这些包的安装。换句话说，Python 中的 setuptools 相当于 Debian 中的 apt。

 在 Python 中准备一个 setuptools 封装器，是部署它的标准方法。第 5 章中对此将会有更详细的说明。

该工具现在十分流行，甚至当编写要发布的 Python 应用程序时，它几乎已经是必需的了。在近几年内，它很有希望被 Python 纳入自己的标准程序库中。在此之前，如果想拥有完整的 Python 系统，充分发挥 setuptools 的功能，还需要另外安装 setuptools，因为它还不是 Python 标准安装所涵盖的一部分。

1.3.2 使用 EasyInstall 安装 setuptools

要安装 setuptools，还需要安装 EasyInstall，它是一个软件包下载器和安装程序。该程序是对 setuptools 的有效补充，因为它知道如何获取相应的软件包以及如何安装它。安装它的同时也将完成 setuptools 的安装。

从 Peak 网站（<http://peak.telecommunity.com/DevCenter/EasyInstall>）中下载并运行 ez_setup.py 脚本程序，它的位置通常是 http://peak.telecommunity.com/dist/ez_setup.py，如下所示。

```
macziade:~ tziade$ wget http://peak.telecommunity.com/dist/ez_setup.py
08:31:40 (29.26 KB/s) - « ez_setup.py » saved [8960/8960]
macziade:~ tziade$ python ez_setup.py setuptools
Searching for setuptools
Reading http://pypi.python.org/simple/setuptools/
Best match: setuptools 0.6c7
...
```



```
Processing dependencies for setuptools
```

```
Finished processing dependencies for setuptools
```

如果之前安装了早期版本，将会看到一个出错信息。这时，需要使用升级选项（-U setuptools），如下所示。

```
macziade: ~ tziade$ python ez_setup.py
Setuptools version 0.6c7 or greater has been installed.
(Run "ez_setup.py -U setuptools" to reinstall or upgrade.)
macziade: ~ tziade$ python ez_setup.py -U setuptools
Searching for setuptools
Reading http://pypi.python.org/simple/setuptools/
Best match: setuptools 0.6c7
...
Processing dependencies for setuptools
Finished processing dependencies for setuptools
```

当一切都安装完之后，系统中就有一个名为 `easy_install` 的命令了。通过该命令可以完成任何扩展插件的安装和升级工作。例如，如果需要安装扩展插件 `py.test`（它是针对敏捷开发的一组工具，参见 <http://codespeak.net/py/dist>），那么只需执行以下命令。

```
tarek@luvdiit:/tmp$ sudo easy_install py
Searching for py
Reading http://cheeseshop.python.org/pypi/py/
Reading http://codespeak.net/py
Reading http://cheeseshop.python.org/pypi/py/0.9.0
Best match: py 0.9.0
Downloading http://codespeak.net/download/py/py-0.9.0.tar.gz
...
Installing pytest.cmd script to /usr/local/bin
Installed /usr/local/lib/python2.3/site-packages/py-0.9.0-py2.3.egg
Processing dependencies for py
Finished processing dependencies for py
```

如果使用 Windows 平台，那么该脚本的名称是 `easy_install.exe`，在 Python 安装路径的 `Scripts` 文件夹中可以找到它。由于它和 1.1.3 小节中在环境变量 `PATH` 中设置的路径相同，因此也可以执行 `easy_install` 命令（在 Linux 和 Mac OS X 平台上，无需在调用该命令时在前面添加 `sudo`，因为它本身就拥有 root 特权）。

有了该工具，扩展 Python 将更加容易，并且它们依赖的包都将自动完成安装。如果在

Windows 平台下安装扩展插件时需要编译，就会自动调用 MinGW 完成这一额外的步骤。

1.3.3 将 MinGW 整合到 distutils 中

当需要编译程序时，可以在 Python 的配置文件中指定。这在 Windows 平台中也十分简单。首先在 `python-installation-path\lib\distutils` 文件夹（在 Windows 平台下则应在 Lib 文件夹下，第一个字母是大写）下创建一个名为 `distutils.cfg` 文件，并在该文件中添加以下内容。

```
[build]
compiler = mingw32
```

这样就能够把 MinGW 链接到 Python 中，每次 Python 需要编译包含 C 程序代码的包时，就会自动调用 MinGW。



现在已经完成开始编码之前的准备工作了！

1.4 工作环境

花些时间对工作环境进行配置，对于提高生产率而言是十分重要的。当负责一个项目时，强制要求所有开发人员使用相同的工具集总不是个好主意。更好的方法是让每个人从推荐的一组公共工具集中挑选适合自己的部分。

在一个 Python 项目中，除了编写程序之外，还包括与数据文件、诸如源代码库之类的第三方服务交互之类的工作。



开发人员除了编写代码之外，还需要花费大量的时间来完成其他工作。

配置工作环境主要有两种方法：通过一组工具集构建（传统方法），或者使用集成工具完成（新方法）。当然，还有许多折衷的方案，每个开发人员都可以根据自己的爱好和习惯进行选择。

1.4.1 使用文本编辑器与辅助工具的组合

这种环境是历史最悠久的模式，不过或许也是效率最高的一种，因此这样做能够根据自

己的工作需要有机地组合它们。如果始终使用的是同一台电脑，安装和配置自己所需工具是十分简单的。但准备一个可移植的工作环境总是更好的选择。例如，将它们绑定在一起，存储在U盘中，这样就可以在不同电脑中使用它们。决定在不同平台使用相同的工具也是一种很好的实践，这样能够在任何地方有效地应用它们。

可移植的 Python 和类似项目

可移植的 Python 是一个针对诸如 Windows 平台提供 Python 系统的项目，它提供了一个开箱即用的、内嵌式的 Python 和代码编辑器。如果目标环境已经安装了 Python，那么可能不需要安装这样的工作环境。不过在该项目中还是能够发现一些有趣的东西的，详情参见 <http://www.portablepython.com>。



Damn Small Linux (DSL) 也是一个有趣的解决方案，它可以将一组工具存储在一个U盘中。大家可能知道，可以通过一个名为 Qemu 的系统模拟器（它可以在任何平台上运行）来运行一个嵌入式的 Linux。因此，可以在 DSL 中安装一个 Python 以实现所需的功能，详情参见 <http://www.damnsmalllinux.org/usb-qemu.html>。

Dragon 公司提供了一个能够构建可移植 Python 环境的 Ubuntu 系统。

现在，该工作环境将由以下几个部分组成：

- 在各种平台上都能够找到的源代码编辑器，可能是开源的、免费的；
- 一些扩展的二进制程序，为 Python 提供一些功能，但不能对其进行修改。

1. 源代码编辑器

和 Python 兼容的编辑器有很多。在一个由多个工具组成的工作环境中，最好是选择一个只关注于代码编写的编辑器。换句话说，在简单的代码编辑器和集成开发环境（IDE）之间，边界总是有些模糊的，甚至简单的编辑器也提供了与系统交互和扩展的机制。但一个配置齐全的源代码编辑器可以让你不用为多余的功能费心。

多年以来，最佳的选择是 Vim (<http://www.vim.org>) 或 Emacs (<http://www.gnu.org/software/emacs>)。乍一看，它们的界面并不友好，因为它们采用了不同的键盘快捷键标准，通常需要花费一些时间去熟悉它。不过，当熟悉了这些命令之后，会发现它们还是很有效率的工具。它提供了针对 Python 的模式，还有许多用来编辑其他类型文件的特定模式。

Vim 是一种对 Python 支持不错的编辑器，该社区对其给予了很大的兴趣。可以很方便地通过 Python 对其扩展。例如，可以看看在 PyCon 2007 大会的一场演讲的内容 (<http://www.tummy.com/Community/Presentations/vimpython-20070225/vim.html>)。



Vim 最大的优势是，多年来所有的 Linux 系统中都预装了它，因此在其他电脑甚至是服务器上都能找到它。

下一小节将介绍 Vim 的安装和配置方法。如果你更偏爱 Emacs，那么建议最好先浏览一下 <http://www.python.org/emacs>。

2. Vim 的安装和配置

Vim 的最新版本是 7.1，它提供了诸如与代码编译器绑定等优秀的功能。

如果使用 Linux 操作系统，那么肯定已经安装了某个版本的 Vim，但通常其版本是低于 7.1 的。可以通过 `vim -version` 命令来检查它的版本号。如果版本低于 7.0，可以通过 Linux 系统中的软件包管理工具或者用直接编译源代码的方法升级它。

在其他操作系统中，需要另外安装 Vim。Windows 用户能够找到一个可执行的安装程序，它不仅提供了 `gvim`（一个提供了图形化界面的版本），还提供了相应的控制台版本。如果 Mac OS X 的用户需要使用 7.1 版本，则需要采用编译源代码方式安装，因为还没有提供最新版本的二进制安装包。

在 <http://www.vim.org/download.php> 中获取相应版本的安装包，并视实际的需要对其进行编译。

如果要在多字节字符集（如带重读符号的法文）环境下编译 Vim，那么需要在调用 `configure` 命令时带上 `--enable-multibyte` 参数。其编译顺序如下所示。

```
./configure --enable-multibyte
make
sudo make install
```

这样，将在 `/usr/local` 目录下安装 Vim，其二进制命令位于：

```
/usr/local/bin/vim.
```

最后还有一点，如果使用的是 Linux 或 Mac OS X 操作系统，那么就在自己的用户主目录下创建一个名为 `.vimrc` 的文件；如果使用的是 Windows 操作系统，则应该将其命名为 `_vimrc`。将其放保存在 Vim 的安装文件夹下，同时添加一个名为 `VIM` 的环境变量（其值就是该路径），这样 Vim 就能够知道从哪里获取它。

`vimrc` 文件的内容如下所示。

```
set encoding=utf8
set paste
```



```

set expandtab
set textwidth=0
set tabstop=4
set softtabstop=4
set shiftwidth=4
set autoindent
set backspace=indent,eol,start
set incsearch
set ignorecase
set ruler
set wildmenu
set commentstring=\ #\ %s
set foldlevel=0
set clipboard+=unnamed
syntax on

```

例如，`tabstop` 选项就是用来将<Tab>键的用途定义为插入 4 个空白符。



在 Vim 中，对于每个选项都可以通过：`help` 命令来获得它的说明。

例如，使用：`help rule` 命令将会显示出一个与 `ruler` 选项相关的帮助页面，如图 1.3 所示。

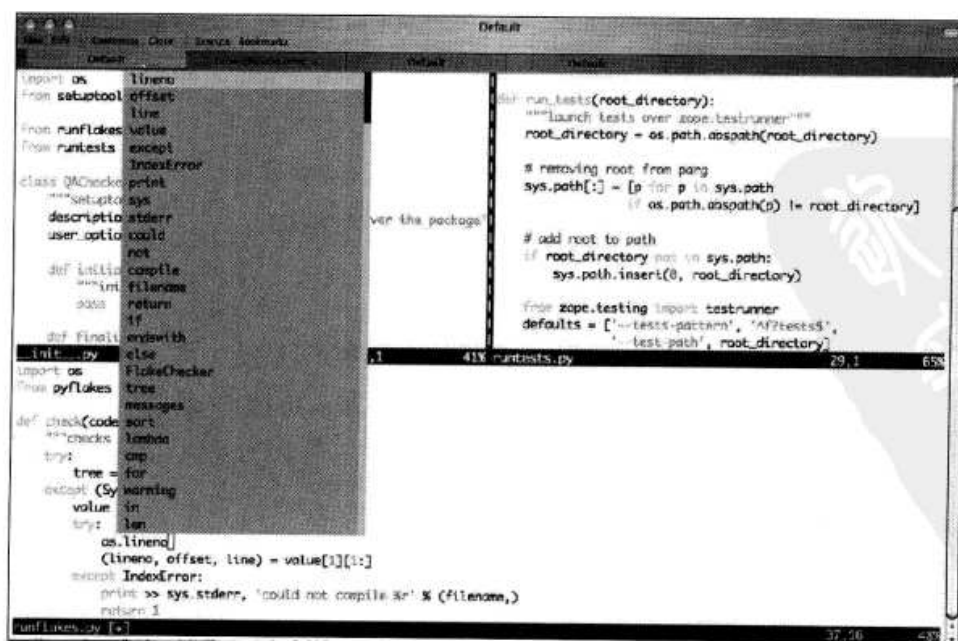


图 1.3

现在，运行 Vim 的准备工作就完成了。

3. 使用其他编辑器

如果不想使用 Vim 或 Emacs，而是想使用对鼠标操作提供更多支持的、具有可视化界面的编辑器，那么也可以选择其他编辑器。不过最好能够提供针对 Python 的模式，并满足以下标准。

- 可以将<Tab>键定义为插入 4 个空白符——这是一个十分重要的特性，在绝大多数编辑器都是这样处理的。如果编辑器不是这样定义的，那么建议放弃。否则，最终会对代码中的空白符和 tab 符产生混淆，从而导致编译器出错。
- 在保存文件时会去除多余的空白符。
- 在换行时，能够智能地定位鼠标指针的位置，以加快编码的效率。
- 提供标准的、基于颜色方案的强调显示模式。
- 提供简单的代码自动补全功能。

在比较不同的代码编辑器时，还有许多标准。不过有些是不太必要的，如代码折叠，而有些则是十分有用的，如 API 搜索。如果能够在编辑功能之外，提供 Python 交互命令行，则要比满足前面提及的 5 个标准还更有用。



如果真的不喜欢 Vim 或 Emacs，那么你可能属于新潮的开发人员。

4. 其他二进制程序

要使编辑器更加完善，还应该安装一些二进制程序，以满足以下一些常见的需求。

- diff，源于 GNU diffutils，它可以完成比较两个文件夹或文件的内容的功能。在所有 Linux 发行版本和 Mac OS X 中，都默认安装了该程序。在 Windows 平台中则需要另外安装，在 <http://gnuwin32.sourceforge.net/packages/diffutils.htm> 中可以找到安装程序。安装完成后，就可以在 DOS 窗口中使用 diff 命令了。
- grep，一个用来在文件中查找字符串的命令行工具。与系统工具相比，它的功能更强大，它在各种操作系统平台上的运行机制是一样的，在 Linux 和 Mac OS X 中也是默认提供的。在 Windows 平台需要另外安装，安装程序可以在 <http://gnuwin32.sourceforge.net/packages/grep.htm> 找到。


注意，MSYS 中已经提供了这些工具，包括 Windows 平台下的 grep 工具。

1.4.2 使用集成开发环境

IDE 中除了提供源代码编辑器之外，还集成了许多辅助工具。这使得其部署和使用都变

得非常快捷方便。

对于 Python 而言, 现在最优秀的开源 IDE 当属 Eclipse (<http://www.eclipse.org>) 与 PyDev (<http://pydev.org>) 插件的组合 (该插件不是免费的)。

 商业软件中还有一款名为 Wingware 的优秀 IDE, 详情可参见 <http://wingware.com>.

Eclipse 也是可移植的, 它使得能够在任何电脑中以相同的方式工作。PyDev 是针对 Eclipse 的一个插件, 用来添加一些 Python 特性:

- 代码补全;
- 语法格式的强调显示;
- 诸如 PyLint 和 Bicycle Repair Man 之类的质量保证 (QA) 工具;
- 代码覆盖;
- 集成的调试程序。

安装带 PyDev 插件的 Eclipse

Eclipse 是用 Java 语言编写的, 因此首先要安装 Java 运行环境 (JRE)。如果使用的是 Mac OS X, 那么 JRE 已经安装了。在 Sun 的网站 <http://java.sun.com/javase/downloads/index.jsp> 中可以下载到最新版本的 JRE。下载正常的安装程序, 然后根据其说明部署到系统中。

Eclipse 没有提供安装程序, 因为它只是由一个文件夹加上一些 Java 脚本构成的。因此要安装 Eclipse, 只需要下载一个压缩包, 然后在系统中将其解压出来。然后, 通过 Eclipse 界面中提供的优雅的软件包管理系统来完成插件的添加。但安装适当的插件集的确有些麻烦, 因为最新的 Eclipse 版本可能和这些插件不兼容。

由于这些插件也可以绑定在压缩包中, 因此最简单的方法是找一个自定义的 Eclipse 分发版本。现在还没有专门针对 Python 的分发版本, 不过可以根据需要在线创建一个。

位于 <http://ondemand.yoxos.com/geteclipse/W4TDelegate> 的 Yoxos 就借助了 AJAX 安装程序实现了该功能。在该网页中, 可以选择自己需要的插件, 它将根据选择生成一个可下载的压缩包。搜索名为 PyDev 的插件, 然后双击它, 以便添加到左边的插件列表中。添加该插件时, 其依赖的其他插件也会被同时加入。然后, 单击右上角的 Download (下载) 按钮获得这个自定义的压缩包, 如图 1.4 所示。

对这个压缩包进行解压, 例如, 在 Windows 平台中将其解压到 c:\Program Files\Eclipse 目录, 在 Linux 或 Mac OS X 系统中将其解压到用户主目录下。在这个文件夹中, 会发现一个能启动该应用程序的快捷链接, 如图 1.5 所示。至此, 就做好运行 Eclipse 的准备工

作了。

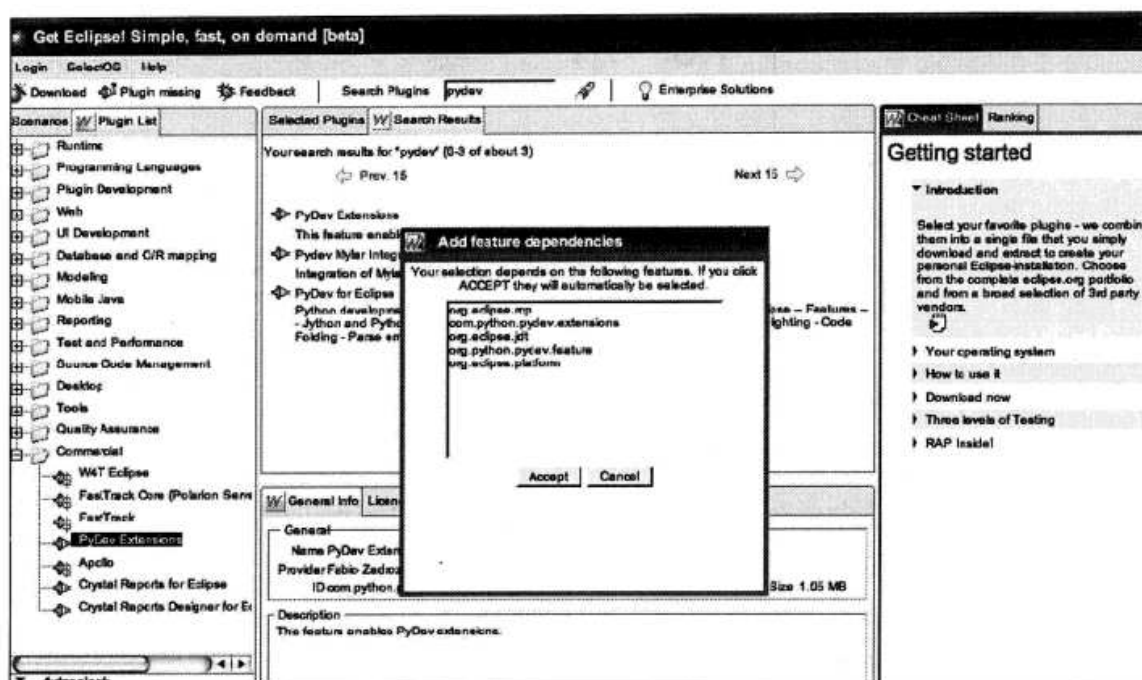


图 1.4

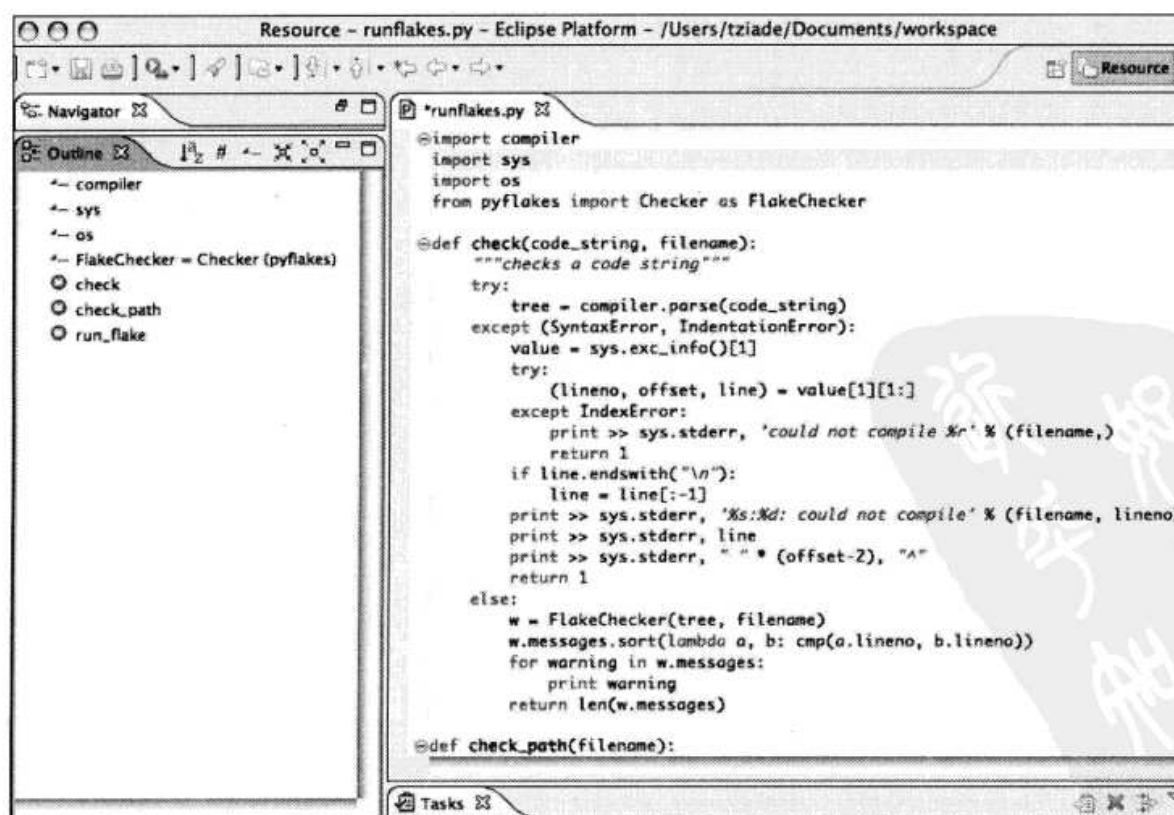


图 1.5

1.5 小 结

本章介绍了 4 个方面的内容，如下所示。

- **Python 的安装方法** Python 有多个版本，不过本书将主要关注于 CPython。它可以安装在 Linux、Mac OS X 和 Windows 平台上。可以通过编译源代码的方式来安装，但直接使用二进制安装文件会更简单些。
- **setuptools 的安装方法** 要完整安装 Python，还需要部署 setuptools。
- **定制命令行** Python 提供了一个交互式命令行，它可以通过启动脚本文件来定制。在编写程序时它将扮演十分重要的角色，因为通过它可以对小段代码进行测试。
- **工作环境** 当完成命令行的定制之后，开发人员可以使用下列工具：
 - 传统的源代码编辑器，如 Vim、Emacs，也能为 Python 代码提供友好模式的其他编辑器软件。该编辑器需要借助一组工具集来补充。
 - 融合了开发中所需使用的各种功能的集成开发环境。Eclipse 和 PyDev 的组合现在是佳的。

下一章将讲述低于类级的语法最佳实践。

