第5章 编写一个包

本章将聚焦于编写、发行 Python 包的可重复的过程，其目标是：

- 缩短开始真正的工作之前所需的设置时间，换句话说，就是提供样板化代码；
- 提供编写包的标准化方法；
- 简化测试驱动开发方法的使用；
- 为发行过程提供帮助。

本章内容由以下 4 个部分组成：

- 用于所有包的公用模式，描述所有 Python 包之间的相似之处，以及 distutils 和 setuptools 如何扮演核心角色；
- 产生式编程（generative programming，http://en.wikipedia.org/wiki/Generative_programming）如何通过基于模板的方法对此提供帮助；
- 包模板的创建，设置各种工作所需要的一切；
- 构建一个开发周期。

5.1 用于所有包的公共模式

在前一章中已经看到，组织一个应用程序代码最简单的方法是使用 egg 将其分解到多个包中。这使代码更加简单，而且更容易理解、维护和修改，还最大化了每个包的可复用性。它们就像组件一样工作。

对于指定公司的应用程序，可以有一组和主 egg 相结合的 egg。因此，所有的包都可以使用 egg 结构建立。

本节将介绍命名空间包（namespaced package）如何组织、发行并通过 distutils 和 setuptools

分发。

正如在前一章中所看到的那样，编写 egg 是通过对一个提供公用前缀命名空间的嵌套文件夹中的代码进行分层来完成的。例如，对于 Acme 公司，公共的命名空间可能是 acme。其结果是生成一个命名空间包。

例如，一个代码与 SQL 相关的包可以被称为 acme.sql。使用这样一个包的最佳方式是创建一个 acme.sql 文件夹，它包含 acme 文件夹，而 acme 文件夹下又包含 sql 文件夹（如图 5.1 所示）。

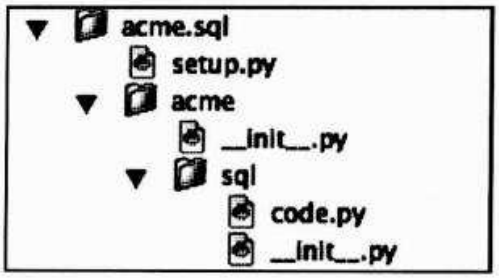

图 5.1

根目录中有一个 setup.py 脚本，它定义 distutils 模块中描述的所有元数据，并将其合并为一个标准的 setup 函数的参数。这个函数由提供大部分 egg 基础架构的第三方程序库 setuptools 扩展。

distutils 和 setuptools 之间的界线正在变得模糊，也许有一天将会合并。

因此，在这个文件中至少有以下内容。

```
from setuptools import setup
setup(name='acme.sql')
```

name 给出了 egg 的全名。由此，该脚本提供多个命令，可以使用--help-commands 选项列出这些命令，如下所示。

```
$ python setup.py --help-commands
Standard commands:
   build        build everything needed to install
   ...
   install      install everything from build directory
   sdist        create a source distribution
   register     register the distribution
   bdist        create a built (binary) distribution
Extra commands:
  develop   install package in 'development mode'
   ...
   test         run unit tests after in-place build
   alias        define a shortcut
```

```
    bdist_egg    create an "egg" distribution
```

最重要的命令已经在前面的清单中列出。Standard commands（标准命令）是 distutils 提供的内建命令，而 Extra commands（附加命令）是由诸如 setuptools 这样的第三方包或任何其他定义和注册新命令的包所创建的。

1．sdist

sdist 命令是最简单的命令。它用来创建一个发行树，运行一个包所需的一切内容都将复制到这里。然后，这棵树被归档到一个或多个档案文件中（它往往只创建一个 tar 档案）。这个档案基本上是一个源树的副本。

这个命令是从目标系统独立地分发一个包的最简单方式。它将创建一个 dist 文件夹，其中包含可被分发的档案。使用它之前，必须传递一个附加的参数给 setup，以提供版本号。如果不给它提供一个 version 值，那它将使用 version = 0.0.0，如下所示。

```
from setuptools import setup
setup(name='acme.sql', version='0.1.1')
```

这个版本号在升级时十分有用。每当发行包时都将提升版本号，这样目标系统就知道它已经被修改了。

使用这个附加参数来运行 sdist 命令，如下所示。

```
$ python setup.py sdist
running sdist
...
creating dist
tar -cf dist/acme.sql-0.1.1.tar acme.sql-0.1.1
gzip -f9 dist/acme.sql-0.1.1.tar
removing 'acme.sql-0.1.1' (and everything under it)
$ ls dist/
acme.sql-0.1.1.tar.gz
```

在 Windows 下，档案将是一个 ZIP 文件。

版本用来标记档案名称，这个档案可以被分发，并且安装到任何拥有 Python 的系统之上。在 sdist 分发中，如果包含有 C 程序库或扩展，目标系统将负责编译它们。这在基于 Linux 和 Mac OS 的系统上是很常见的，因为它们都提供编译器。但是，这在 Windows 下并不常见。这就是当一个包打算在多个平台下运行时，应该总是和一个预编译分发版本一起

分发的原因。

2．MANIFEST.in 文件

使用 sdist 构建一个分发版本时，distutils 将浏览包的目录，查找包含在档案中的文件。distutils 将包含：

- 所有 py_modules、packages 和 scripts 选项隐含的 Python 源文件；
- 所有在 ext_modules 选项中列出的 C 源文件；
- 符合 test/test*.py 模式的文件；
- README、README.txt、setup.py 和 setup.cfg 文件。

此外，如果包是由 Subversion 或 CVS 管理，那么 sdist 将浏览.svn 之类的文件夹以查找其包含的文件。sdist 将创建一个列出所有文件的 MANIFEST 文件，并将它们包含到档案文件中。

假设不使用这些版本控制系统，并且需要包含更多的文件，可以在与 setup.py 相同的目录下定义一个名为 MANIFEST.in 的 MANIFEST 文件模板，然后在这里指出 sdist 包含的文件。

这个模板中的每一行都将定义一个包含或排除规则，示例如下。

```
include HISTORY.txt
include README.txt
include CHANGES.txt
include CONTRIBUTORS.txt
include LICENSE
recursive-include *.txt *.py
```

命令的完整列表可在 http://docs.python.org/dist/中找到。

```
sdist-cmd.html    #sdist-cmd.
```

3．build 和 bdist

为了能够分发预编译的分发版本，distutils 提供了 build 命令，它分 4 个步骤来编译包：

- build_py　通过字节编译（byte-compiling）构建纯 Python 模块，并将其复制到构建文件夹中；
- build_clib　当包含有 C 程序库时，使用 Python 编译器编译并在构建文件夹中创建一个静态程序库；
- build_ext　编译 C 扩展，并将结果放到类似 build_clib 的构建文件夹中；
- build_scripts　编译标记为脚本的模块，当第一行被设置（!#）时修改解释程序路径并修复文件模式使其可执行。

每个步骤都是可以被单独调用的命令。编译过程的结果是一个包含安装包所需所有内容的构建文件夹。distutil 包中还没有提供交叉编译选项，这意味着执行该命令的结果只是针对构建时所用操作系统的。

最近有些人在 Python tracker 中提出了一些补丁，使 distutils 能够交叉编译 C 编写的部分。所以，将来也许能够用到这个功能。

当必须创建一些 C 扩展时，构建过程将使用系统编译器和 Python 头文件（Python.h）。这个引用文件在从源文件中编译出 Python 时就是可用的了。对于打包的分发版本，一个名为 python-dev 的附加包通常会包含它，所以这个包也必须安装。

使用的 C 编译器是当前系统中的编译器。对于基于 Linux 的系统或 Mac OS X 而言是 gcc，对于 Windows 而言可以使用 Microsoft Visual C++（有可用的免费命令行版本），也可以使用开源项目 MinGW。第 1 章中就已经讲解过，可以在 distutils 中进行相应的配置。

将使用 bdist 命令来创建一个二进制分发版本。它调用 build 和所有相关的命令，然后和 sdist 一样，创建一个档案文件。

下面在 Mac OS X 下为 acme.sql 创建一个二进制分发版本。

```
$ python setup.py bdist
running bdist
running bdist_dumb
running build
...
running install_scripts
tar -cf dist/acme.sql-0.1.1.macosx-10.3-fat.tar .
gzip -f9 acme.sql-0.1.1.macosx-10.3-fat.tar
removing 'build/bdist.macosx-10.3-fat/dumb' (and everything under it)
$ ls dist/
acme.sql-0.1.1.macosx-10.3-fat.tar.gz acme.sql-0.1.1.tar.gz
```

注意，新创建的档案文件名中包含了系统名及分发版本的名称（Mac OS X 10.3）。

在 Windows 下调用相同命令，也能创建一个特定的分发档案文件，如下所示。

```
C:\acme.sql> python.exe setup.py bdist
...
C:\acme.sql> dir dist
25/02/2008 08:18 <DIR> .
```

```
25/02/2008 08:18 <DIR> ..
25/02/2008 08:24              16 055 acme.sql-0.1.win32.zip
               1 File(s)         16 055 bytes
               2 Dir(s)   22 239 752 192 bytes free
```

如果这个包里包含 C 代码，那么除了提供源代码分发版本之外，应发行尽可能多的不同的二进制分发版本。至少，对于没有安装 C 编译器的人来说，一个 Windows 二进制分发版本就是很重要的。

在一个二进制发行版本中，包含一棵可以直接复制到 Python 树中的树。它主要包含复制到 Python 的 site-packages 文件夹中的一个文件夹。

4．bdist_egg

bdist_egg 命令是 setuptools 额外提供的一个命令。它基本上和 bdist 类似，用来创建一个二进制分发版本，不过它具有一棵与在源代码分发版本中相当的树。换句话说，这个档案文件可以被下载、解压，然后通过将文件夹添加到 Python 的搜索路径（sys.path）来使用。

近来，这种分发模式经常用来替代基于 bdist 生成的模式。

5．install

使用 install 命令可以将包安装到 Python 中。如果以前没有编译，那么它将尝试编译包然后将结果注入到 Python 树中。如果提供源代码分发版本，那么它可以被解压到一个临时文件夹中，然后使用这个命令安装。install 命令还将安装在 install_requires 元数据中定义的相关模块。

这是通过查看 Python 包索引（PyPI）来完成的。例如，为了和 acme.sql 一起安装 pysqlite 和 SQLAlchemy，setup 调用将被修改为：

```
from setuptools import setup
setup(name='acme.sql', version='0.1.1',
      install_requires=['pysqlite', 'SQLAlchemy'])
```

当运行该命令时，两个相关的模块都将被安装。

6．如何卸载一个包

用来卸载一个之前安装的包的命令，在 setup.py 中找不到。这个功能早先已经有人提议过了。因为安装程序可能修改系统其他元素使用的文件，所以这并不是很简单的。

最佳的方法应该是对所有被修改的元素建立一个快照，并且对自己新创建的文件和目录做一个记录。

install 中有一个 record 选项，用来将所有创建的文件记录到一个文本文件中，如下所示。

```
$ python setup.py install --record installation.txt
running install
...
writing list of installed files to 'installation.txt'
```

它不会对任何现有的文件创建备份，所以删除这些文件将会破坏系统。对这个问题，有许多特定于平台的解决方案。例如，distutils 允许以 RPM 包格式分发。但是现在还没有处理这一任务的通用方法。

目前删除一个包的最简单方法是删除该包所创建的文件，然后删除在 sitepackages 文件夹中 easy-install.pth 文件中列举的所有引用。

7．develop

setuptools 添加了一个有用的处理包的命令。develop 命令编译并且在适当的位置安装包，然后添加一个简单的链接到 Python site-packages 文件夹中。这使用户能够使用该代码的本地副本工作，即使它可在 Python 的 site-packages 文件夹中获得。在下一章中，在创建基于 egg 的应用程序时会发现，这是一个很好的特性。所有被创建的包都可以使用 develop 命令链接到解释程序。

当以这种形式安装一个包时，和常规的安装会有些不同，可以使用-u 选项显式删除，如下所示。

```
$ sudo python setup.py develop
running develop
...
Adding iw.recipe.fss 0.1.3dev-r7606 to easy-install.pth file
Installed /Users/repos/ingeniweb.sourceforge.net/iw.recipe.fss/trunk
Processing dependencies ...
$ sudo python setup.py develop -u
running develop
Removing
...
Removing iw.recipe.fss 0.1.3dev-r7606 from easy-install.pth file
```

注意，使用 develop 安装的包，总是要比用其他方式安装的包更好些。

8．test

另一个有用的命令是 test，它提供了一种执行包中所有测试的方法。它将扫描文件夹，并且收集寻找到的测试套件。这个测试运行程序尝试收集包中的测试，但是这相当有局限性。

一种更好的方法是与诸如 zope.testing 或 Nose 这样的提供更多选项的外部测试运行程序挂钩起来。

为了透明地将 Nose 与测试命令挂钩，可以将 test_suite 元数据设置为“'nose.collector”，并将 Nose 添加到 test_requires 列表中，如下所示。

```
setup(
...
test_suite='nose.collector',
test_requires=['Nose'],
...
)
```

第 11 章中将介绍一些测试运行程序，并说明 Nose 的使用方法。

9. register 和 upload

要分发一个包，有两个可用的命令：

- register 它将把所有的元数据上传到一个服务器；
- upload 它将前面建立在 dist 文件夹中的所有档案上传到服务器。

主 PyPI 服务器——之前称为 Cheeseshop（奶酪店），位于http://pypi.python.org/pypi，其中包含超过 3000 个来自开发社区的包。这是 distutils 包所使用的默认服务器，对 register 命令的第一个调用将在主目录中生成一个 C 文件。

因为 PyPI 服务器要进行身份验证，所以当修改一个包时，将要求创建一个用户。这可以在提示符下完成，如下所示。

```
$ python setup.py register
running register
...
We need to know who you are, so please choose either:
1. use your existing login,
2. register as a new user,
3. have the server generate a new password for you (and email it to
you), or
4. quit
Your selection [default 1]:
```

现在，主目录中将出现一个名为.pypirc 文件，它包含了必须输入的用户名和密码。这将

在每次 register 和 uploade 被调用时使用，如下所示。

```
[server-index]
username: tarek
password: secret
```

Windows 上有一个与 Python 2.4 和 2.5 有关的权限。这个主目录通过 distutils 无法找到，除非添加一个 HOME 环境变量。但是，这在 2.6 版本中已经修复了。如果要添加它，可以使用第 1 章中修改 PATH 命令时所描述的方法。然后为用户添加 HOME 变量，指向 os.path.expanduser('~')返回的目录。

当指定了 download_url 元数据或 url，并且指定的是一个有效的 URL 时，PyPI 服务器将使其对在项目网页上的用户也可用。

使用 upload 命令将直接使档案文件在 PyPI 上可用，所以 download_url 可以省略，如图 5.2 所示。

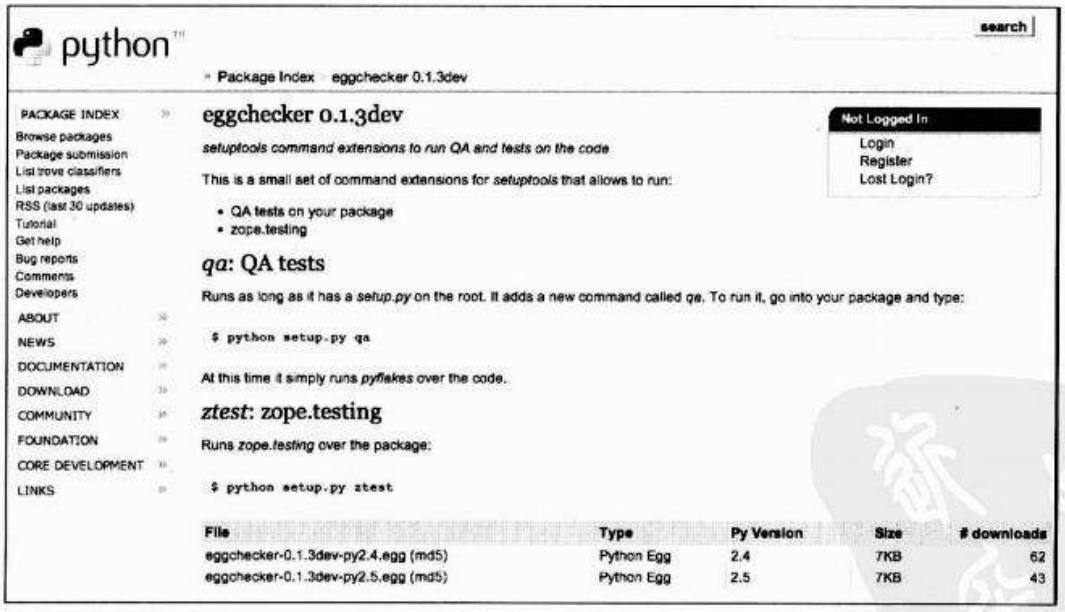

图 5.2

Distutils 定义了一个 Trove 分类（参见 PEP 301，http://www.python.org/dev/peps/pep-0301/#distutils-trove-classification ）来对包进行分类（如定义在 Sourceforge 上的包）。trove 是可以在 http://pypi.python.org/pypi?%3Aaction=list_classifiers 上找到的静态分类，它也不断地被新来者扩充。

每一行都是由“::”分隔的级别，如下所示。

```
...
Topic :: Terminals
Topic :: Terminals :: Serial
Topic :: Terminals :: Telnet
Topic :: Terminals :: Terminal Emulators/X Terminals
Topic :: Text Editors Topic :: Text Editors :: Documentation
Topic :: Text Editors :: Emacs
...
```

一个包可以被分在多个类别中，这些类别可以在分类符 meta-data 中列出。一个处理（例如）低级 Python 代码的 GPL 包可以使用如下所示的类别。

```
Programming Language :: Python
Topic :: Software Development :: Libraries :: Python Modules
License :: OSI Approved :: GNU General Public License (GPL)
```

10．Python 2.6 中.pypirc 的格式

.pypirc 文件在 Python 2.6 下已经有了一些改进，多个用户及其密码可以同多个类似 PyPI 的服务器一起管理。一个 Python 2.6 配置文件看起来如下所示。

```
[distutils]
index-servers =
    pypi
    alternative-server
    alternative-account-on-pypi
[pypi]
username:tarek
password:secret
[alternative-server]
username:tarek
password:secret
repository:http://example.com/pypi
```

register 和 upload 命令可以借助-r 选项协助选取服务器，可以使用仓库的完整 URL 或小节段，如下所示。

```
# 上传到 http://example.com/pypi
$ python setup.py sdist upload -r alternative-server
# 用默认账户（从 pypi 中获取的）注册
```

```
$ python setup.py register
# 注册到 http://example.com
$ python setup.py register -r http://example.com/pypi
```

通过该特性可以和PyPI之外的服务器进行交互。当要处理的包大部分都不是发布在PyPI上时，运行自己的类 PyPI 服务器是个良好的习惯。例如，Plone Software Center（参见http://plone.org/products/plonesoftwarecenter）就能够用来部署一个与 distutils upload 和 register 命令交互的 Web 服务器。

11．创建一个新命令

distutils 允许创建新的命令，如http://docs.python.org/dist/node84.html中描述的那样。一个新的命令可以使用一个入口（entry point）来注册。这是由 setuptools 引入的，是一种将包定义为插件的简单方法。

入口点是 setuptools 中的一个命名链接，它指向通过某些 API 可用的类或函数。任何应用程序都能够扫描所有已注册的包，并且将其链接的代码作为一个插件使用。

为了链接新的命令，可以在 setup 调用中使用 entry_points 元数据，如下所示。

```
setup(name="my.command",
        entry_points="""
           [distutils.commands]
           my_command = my.command.module.Class
        """)
```

所有命名链接都被集中在已命名的小节中。当 distutils 载入时，它将扫描登记在distutils.commands 之下的所有链接。

许多提供扩展性的 Python 应用程序都使用了这一机制。

12 setup.py 使用小结

setup.py 采用的 3 个主要操作：

- 创建一个包；
- 安装这个包，可能在开发模式下；
- 注册并上传到 PyPI 服务器。

所有命令都可以被组合到相同的调用中，以下是一些典型的使用模式。

```
# 使用 PyPI 注册这个包，创建一个源代码发布和一个 egg 发布，然后上传
$ python setup.py register sdist bdist_egg upload
# 就地安装，以便开发
$ python setup.py develop
```

```
# 安装它
$ python setup.py install
```

13. alias 命令

为了简化命令行的工作，setuptools 引入了一个新命令 alias。在 setup.cfg 文件中，它用来为指定的命令组合创建一个别名。例如，可以创建一个 release 命令来执行将一个源代码分发版本和一个二进制分发版本上传到 PyPI 服务器上所需的所有操作，如下所示。

```
$ python setup.py alias release register sdist bdist_egg upload
running alias
Writing setup.cfg
$ python setup.py release
...
```

14. 其他重要的元数据

除了要分发的包的名称和版本之外，setup 可能接受的最重要的参数包括：

- description 包的简单描述；
- long_description 包的完整的描述，可以是 reStructuredText 的形式；
- keywords 定义包的关键字列表；
- author 作者的姓名或组织；
- author_email 作者的邮件地址；
- url 项目的 URL；
- license 许可证（GPL、LGPL 等）；
- packages 包中所有名称的列表，setuptools 提供了一个名为 find_packages 的小函数来计算它；
- namespace_packages 命名空间包的一个列表。

针对 acme.sql 包而言，完整的 setup.py 将如下所示。

```
import os
from setuptools import setup, find_packages
version = '0.1.0'
README = os.path.join(os.path.dirname(__file__), 'README.txt')
long_description = open(README).read() + '\n\n'
setup(name='acme.sql',
      version=version,
      description=("A package that deals with SQL, "
                   "from ACME inc"),
```

```
    long_description=long_description,
    classifiers=[
      "Programming Language :: Python",
      ("Topic :: Software Development :: Libraries ::
      "Python Modules"),
      ],
    keywords='acme sql',
    author='Tarek',
    author_email='tarek@ziade.org',
    url='http://ziade.org',
    license='GPL',
    packages=find_packages(),
    namespace_packages=['acme'],
    install_requires=['pysqlite','SQLAchemy']
    )
```

应常备以下两个综合性指南。

- 在 http://docs.python.org/dist/dist.html 中可以找到的 distutils 指南；
- 在 http://peak.telecommunity.com/DevCenter/setuptools 中可以找到的 setuptools 指南。

5.2 基于模板的方法

acme.sql 中的样板化代码由一个创建根目录中少数文件命名空间的目录树组成。为了使所有包遵循相同的结构，可以通过一个代码生成工具来抽取和提供通用的代码模板。这个方法被称为产生式编程，它在组织级非常有用。它标准化了代码编写的方式，并使开发人员更多产，因为他们可以关注于真正需要创建的代码。这种方法也是为包准备少数被多个包公用的部分（如复杂的测试配置）的一个好机会。

在 Python 社区中，有许多可用的产生式工具，不过最常用的是 Python Paste（http://pythonpaste.org）。

5.2.1 Python Paste

Python Paste 项目是诸如 Pylons（http://pylonshq.com）这样的框架取得成功的部分原因。

开发人员被那些让他们在几分钟内创建出应用程序框架的大量模板集深深地触动。

从官方的教程中可以了解到，这是创建一个 web 应用程序并运行它的 3 个命令行，如下所示。

```
$ paster create -t pylons helloworld
$ cd helloworld
$ paster serve --reload development.ini
```

Plone 和 Zope 社区也遵循这一哲学，现在也为生成框架提供了 Python Paste 模板，ZopeSkel（http://pypi.python.org/pypi/ZopeSkel）就是其中之一。

Python Paste 中包含了多个工具，我们所感兴趣的模板引擎是 PasteScript，可以使用 easy_install 来安装它。它将从 Paste 项目中得到所有相关的模块，如下所示。

```
$ easy_install PasteScript
Searching for PasteScript
Reading http://pypi.python.org/simple/PasteScript/
Reading http://pythonpaste.org/script/
Best match: PasteScript 1.6.2
Downloading
...
Processing dependencies for PasteScript
Searching for PasteDeploy
...
Searching for Paste>=1.3
...
Finished processing dependencies for PasteScript
```

paster 命令将可用并带有几个默认的模板，可以使用 create 命令的 list-templates 选项将其列出，如下所示。

```
$ paster create --list-templates
Available templates:
  basic_package: A basic setuptools-enabled package
  paste_deploy: A web application deployed through paste.deploy
```

basic_package 几乎是 acme.sql 用 setup.py 文件创建一个命名空间包所需要的内容。运行时，该命令行将询问几个问题，相应的回答将被用来填充模板，如下所示。

```
$ paster create -t basic_package mypackage
Selected and implied templates:
```

```
  PasteScript#basic_package A basic setuptools-enabled package
...
Enter version (Version (like 0.1)) ['']: 0.1
Enter description ['']: My package
Enter long_description ['']: this is the package
Enter keywords ['']: package is mine
Enter author (Author name) ['']: Tarek
Enter author_email (Author email) ['']: tarek@ziade.org
Enter url (URL of homepage) ['']: http://ziade.org
Enter license_name (License name) ['']: GPL
Enter zip_safe [False]:
Creating template basic_package
...
```

其执行结果是一个有效的，与安装工具兼容的单级结构，如下所示。

```
$ find mypackage
mypackage
mypackage/mypackage
mypackage/mypackage/__init__.py
mypackage/setup.cfg
mypackage/setup.py
```

5.2.2 创建模板

Python Paste 可称为剪贴本（paster），举例来说，它可以使用 Cheetah 模板引擎（http://cheetahtemplate.org），并提供给它用户的输入信息。

为剪贴本创建一个新的模板，需要提供 3 个要素：

- 一个从 paste.script.templates.Template 中继承的类；
- 包含文件夹和文件（Cheetah 模板或静态文件）的结构；
- 一个指向 paste.paster_create_template 的 setuptools 入口点，用来注册类。

5.3 创建包模板

让我们来创建用于 acme.sql 包的模板。

本书中创建的所有模板包括收集在 PyPI 中的 pbp.skels 里的所有包，可以方便地使用。所以，如果不希望从头创建自己的模板，可以安装它，命令如下。

```
$ easy_install pbp.skels
```

本节将一步步地讲解 pbp.skels 的创建方法。

要创建 package 模板，第一件事就是创建新包的结构，命令如下。

```
$ mkdir -p pbp.skels/pbp/skels
$ find pbp.skels
pbp.skels
pbp.skels/pbp
pbp.skels/pbp/skels
```

然后，将创建一个带有以下代码的__init__.py 文件，并将其放在 pbp 文件夹中。它将告知 distutils 创建一个命名空间包，如下所示。

```
try:
    __import__('pkg_resources').declare_namespace(__name__)
except ImportError:
    from pkgutil import extend_path
    __path__ = extend_path(__path__, __name__)
```

接下来，使用正确的元数据在根目录（path_to_pbp_package/pbp.skels/__init__.py）中创建一个 setup.py 文件。正确的代码如下所示。

```
from setuptools import setup, find_packages
version = '0.1.0'
classifiers = [
    "Programming Language :: Python",
     ("Topic :: Software Development :: "
    "Libraries :: Python Modules")]
setup(name='pbp.skels',
    version=version,
    description=("PasteScript templates for the Expert "
                 "Python programming Book."),
    classifiers=classifiers,
    keywords='paste templates',
```

```
    author='Tarek Ziade',
    author_email='tarek@ziade.org',
    url='http://atomisator.ziade.org',
    license='GPL',
    packages=find_packages(exclude=['ez_setup']),
    namespace_packages=['pbp'],
    include_package_data=True,
    install_requires=['setuptools',
                      'PasteScript'],
    entry_points="""
    # -*- Entry points: -*-
     [paste.paster_create_template]
    pbp_package = pbp.skels.package:Package
    """)
```

该入口点添加了一个在剪贴本中可用的新模板。

下一步是在 pbp/skels 文件夹中的 package 模块里写入 Pacakge 类，如下所示。

```
from paste.script.templates import var
from paste.script.templates import Template
class Package(Template):
    """Package template"""
    _template_dir = 'tmpl/package'
    summary = "A namespaced package with a test environment"
    use_cheetah = True
    vars = [
        var('namespace_package', 'Namespace package',
            default='pbp'),
        var('package', 'The package contained',
            default='example'),
        var('version', 'Version', default='0.1.0'),
        var('description',
            'One-line description of the package'),
        var('author', 'Author name'),
        var('author_email', 'Author email'),
        var('keywords', 'Space-separated keywords/tags'),
        var('url', 'URL of homepage'),
        var('license_name', 'License name', default='GPL')
```

```
        ]
    def check_vars(self, vars, command):
        if not command.options.no_interactive and \
           not hasattr(command, '_deleted_once'):
           del vars['package']
           command._deleted_once = True
        return Template.check_vars(self, vars, command)
```

该类定义了：

- 包含模板结构的文件夹（_template_dir）；
- 将出现在剪贴本中的模板的摘要；
- 表明 Cheetah 是否用于模板结构的标志；
- 一个变量列表，每个变量由名称、标签和默认值（如果需要）组成，这些变量由剪贴本用于询问用户，提示输入用户所需要的数值；
- 在提示输入时所需的确定包变量的 check_vars 方法。

最后一步是创建 tmpl/package 目录，并将 acme.sql 目录的内容复制过来。所有包含被修改值（如命名空间）的文件必须带有_tmpl 后缀。这些值将被替换为${variable}，其中 variable 是 Package 类中列出的变量名称。例如，setup.py 文件将变成 setup.py_tmpl，并且其内容变为如下所示。

```
from setuptools import setup, find_packages
import os
version = ${repr($version) or "0.0"}
long_description = open("README.txt").read()
classifiers = [
   "Programming Language :: Python",
    ("Topic :: Software Development :: "
   "Libraries :: Python Modules")]
setup(name=${repr($project)},
   version=version,
   description=${repr($description) or $empty},
   long_description=long_description,
   classifiers=classifiers,
   keywords=${repr($keywords) or $empty},
   author=${repr($author) or $empty},
   author_email=${repr($author_email) or $empty},
   url=${repr($url) or $empty},
```

```
    license=${repr($license_name) or $empty},
    packages=find_packages(exclude=['ez_setup']),
    namespace_packages=[${repr($namespace_package)}],
    include_package_data=True,
    install_requires=[
        'setuptools',
       # -*- Extra requirements: -*-
    ],
    test_suite='nose.collector',
    test_requires=['Nose'],
    entry_points="""
    # -*- Entry points: -*-
     """,
    )
```

repr 函数将告知 Cheetah 在字符串值的外面添加一个引号。

可以对 acme.sql 中的所有文件使用相同的方法创建一个模板。例如，README.txt 文件将被复制为 README.txt_tmpl，然后所有指向 acme.sql 的引用都被替换成 Package 类中 var 列表里定义的值。

例如，可以使用以下命令获得完整的包名称。

```
${namespace_package}.${package}
```

最后，针对文件夹名称而使用的变量值，必须加上“+”前缀和后缀。例如，命名空间包文件夹名称将被命名为+namespace_package+，包文件夹则被命名为+package+。

在生成 acme.sql 之后，pbp.skeles 的最终结构将类似于如下所示。

```
$ cd pbp.skels
$ find .
setup.py
pbp
pbp/__init__.py
pbp/skels
pbp/skels/__init__.py
pbp/skels/package.py
pbp/skels/tmpl
pbp/skels/tmpl/package
pbp/skels/tmpl/package/README.txt_tmpl
pbp/skels/tmpl/package/setup.py_tmpl
```

```
pbp/skels/tmpl/package/+namespace_package+
pbp/skels/tmpl/package/+namespace_package+/__init__.py_tmpl
pbp/skels/tmpl/package/+namespace_package+/+package+
pbp/skels/tmpl/package/+namespace_package+/+package+/__init__.py
---
```

现在，该包可以使用 develop 命令符号连接到 Python 的 site-packages 目录，并且可用于剪贴本，如下所示。

```
$ python setup.py develop
...
Finished processing dependencies for pbp.skels==0.1.0dev
```

运行了 develop 命令之后，应该查找在 paster 中列出的模板，命令如下。

```
$ paster create --list-templates
Available templates:
   basic_package:   A basic setuptools-enabled package
   pbp_package:     A namespaced package with a test environment
   paste_deploy:    A web application ... paste.deploy
$ paster create -t pbp_package trying.it
Selected and implied templates:
   pbp.skels#package A namespaced package with a test environment
Variables:
   egg:      trying.it
   package: tryingit
   project:  trying.it
Enter namespace_package (Namespace package) ['pbp']: trying
Enter package (The package contained) ['example']: it
...
Creating template package
...
```

在此生成的树，之后将包含可以立刻投入工作的结构。

5.4 开发周期

包的开发周期由多个迭代组成。每个迭代中，代码将从初始状态转化到新的状态。这个

阶段通常会持续几周，最后得到一个发行版本。对于很简单的小型包不会发生这种情况，但是对于所有包含足够多模块的包都值得这么做。

在迭代结束时，将通过之前看过的命令创建一个发行版本。这时，包从开发状态转移到一个可发行状态，交付的代码可以被看作一个官方发行版本。

然后，一个新的循环将从包的一个增量版本开始。

1．应该使用什么版本号

对于包版本号的增量没有固定的约定，当开发人员感觉软件已经有了很大的发展时，他们通常会跳到更高的版本号，而不一定是和前一个版本保持连续。

大部分软件通常会从一个很小的数值开始，并且会使用 2 个或 3 个数字。当他们尝试完成一个版本时，有时会添加一个字母，例如，用 rc 后缀来表示它是一个预览版本（release candidate）。以下是一些处于测试阶段的、修改了一些问题的版本：

- 0.1、0.2、0.3
- 0.1.0、0.1.1、0.1.2a、0.1.2b
- 0.1、0.2rc1、0.2rc2

可以自己决定版本号的机制，只要一直沿用即可。在公司中，通常有所有应用程序都要遵循的标准，但是开放源码应用程序有自己的约定。

唯一应该被应用的规则是确保数字的数量始终相同，并且避免使用“-”符号，因为它被许多从包名称中提取版本号的工具用作一个分隔符。

例如，以下版本号是应该避免的：

- 0.1、0.1.1-alpha、0.1.1-b、0.2
- 0.1、0.1-a、0.1-b

2．每晚构建

如果包在迭代期间也仍然可以发行，那么可以建立开发版本，这也被称为每晚构建版本。这种连续的发行过程使开发人员能得到连续的反馈，并且节约 beta 测试的一些工作。例如，他们不需要从版本仓库中获得代码，并且可以和常规版本一样安装开发版本。

为了区分开发版本和常规的版本，用户必须在版本号后附加 dev 后缀。例如，如果 0.1.2 版本正在被开发并且还没有发行，将被称为 0.1.2dev 版本。

distutils 通过在 setup.cfg 中提供一种方法来做标志，那就是添加一个小节来告知 build 命令关于开发的状态，如下所示。

```
[egg_info]
tag_build = dev
```

这将在版本前自动添加 dev 前缀，如下所示。

```
$ python setup.py bdist_egg
running bdist_egg
running egg_info
...
creating 'dist/iw.selenium-0.1.0dev-py2.4.egg'
```

当包存在于 Subversion 仓库中时，另一个有用的标签可能就是修订号。它可以使用 tag_svn_revision 标志来添加，如下所示。

```
[egg_info]
tag_build = dev
tag_svn_revision = true
```

在这种情况下，修订号将出现在版本号中。

```
$ python setup.py bdist_egg
running bdist_egg
running egg_info
...
creating 'dist/iw.selenium-0.1.0dev_r38360-py2.4.egg'
```

最简单的方法是始终将这个文件保持在主干中，并且在建立常规发行版本之前删除它。换句话说，Subversion 的发行过程可以是：

- 建立主干的一个标签副本；
- 检查标签分支；
- 从这个分支中删除 setup.cfg（或 egg_info 专用的部分）并且提交修改；
- 由此构建发行版本；
- 提升主干中的版本号。

这看起来和下面的过程一样。

```
$ svn cp http://example.com/my.package/trunk http://example.com/
my.package/tags/0.1
$ svn co http://example.com/my.package/tags/0.1 0.1
$ cd 0.1
$ svn rm setup.cfg
$ svn ci -m "removing the dev flag""
$ python setup.py register sdist bdist_egg upload
```

第 8 章将介绍诸如 Subversion 之类的版本控制系统是什么，以及它们的工作机制。

5.5 小　结

本章介绍了以下内容。

- 如何创建一个命名空间包；
- setup.py 的主要任务，以及如何使用它建立和发行包；
- 基于模板生成包框架的方法；
- 剪贴本的工作原理以及创建包框架的方法；
- 如何发行包及提供每夜构建版本。

下一章将关注于相同的主题，不过将从应用程序级别上考虑。