第 7 章

使用 zc.buildout

前一章中介绍了编写基于多个 egg 的应用程序的方法。当分发这样的应用程序时，用户将包及其依赖模块安装到 Python 的 site-packages 文件夹中，并且获得一些入口点，如命令行实用程序。

但是对于比 Atomisator 更大的应用程序而言，这种方法就存在局限性：如果需要部署一些配置文件或者编写日志文件，将它们放在代码包内部就不现实了。

最好的办法是，创建一个专用的安装程序来将它们无缝地集成到目标系统上。以基于 Linux 的系统为例，日志文件应该被放在/var/log 中，而配置文件则应该被放在/etc。但是，创建这样的安装程序需要许多和系统相关的工作。

另一种方法，有点类似 virtualenv 所提供的，是建立一个具有运行应用程序所需的所有文件的自包含目录，然后分发它。这个目录也可以包含所需的包，以及在目标系统上启动所有程序的安装程序。

zc.buildout（参见 http://pypi.python.org/pypi/zc.buildout）是用来创建这样一个环境的工具，本章将介绍如何：

- 通过一种能够定义运行应用程序所需的所有包的描述性语言来组织一个应用程序；
- 将这样的应用程序作为一个源代码发行版本来部署。

zc.buildout 的替代工具是 Paver 和 AutomateIt（参见 http://www. blueskyonmars.com/projects/paver 和 http://automateit.org）。

本章由 3 个部分组成：

- zc.buildout 原理；
- 基于 zc.buildout 的应用程序的分发方法；
- 使用 paster 工具创建一个 zc.buildout 应用程序环境的应用程序模板。

7.1 zc.buildout 原理

virtualenv 对于隔离一个 Python 环境相当方便。正如前一章中所看到的，它是工作在本地的，但是仍然需要手工完成许多设置和维护项目环境的工作。

zc.buildout 提供了相同的隔离功能，并且还进一步提供了：

- 一个简单的、在一个配置文件中定义这些依赖性的描述性语言；
- 提供链接代码调用组合的输入点的插件系统；
- 一种部署和发行应用程序源代码及其执行环境的方法。

配置文件将描述环境中所需的 egg，它们的状态（正在本地开发，在 PyPI，或在其他地方可用）以及构建应用程序所需要的所有其他元素。

插件系统将注册这些包并按照执行的顺序将它们链接起来。

最后，整个环境是独立和隔离的，因而可以像发行和部署后那样使用。

zc.buildout 在其 PyPI 页面（http://pypi.python.org/pypi/zc.buildout）上提供了很好的文档。本节只是摘要地介绍为了构建和在应用程序级别工作时所需要知道的最重要的要素。这些要素包括：

- 配置文件结构；
- buildout 命令；
- Recipes。

7.1.1 配置文件结构

zc.buildout 依赖于一个结构与 ConfigParser 模块兼容的配置文件。这些类 INI 文件中有以[headers]分割的小节，小节中的行包含 name:value 或 name=value 这样的内容。

1. 最小的配置文件

最小的 buildout 配置文件中包含一个名为[buildout]的小节，其中有一个称作 parts 的变量。这个变量包含一个提供小节列表的多行值，如下所示。

```
[buildout]
parts =
   part1
   part2
[part1]
recipe = my.recipe1
[part2]
recipe = my.recipe2
```

在 parts 中指定的每个小节至少有一个提供包名称的 recipe 值。这个包可以是任何 Python 包，只要它定义一个 zc.buildout 入口点。

用这个文件，buildout 将执行以下工作序列：

- 检查 my.recipe1 包是否安装，如果没有安装，读取并完成本地安装；
- 执行 my.recipe1 入口点所指向的代码；
- 然后，对 parts 做同样的事情。

因而，buildout 是一个基于插件的脚本，将被称为 recipe 的独立包的执行链接起来。用这个工具构建的环境包括 recipe 正确顺序的定义。

2．[buildout]小节选项

除了 parts，[buildout]小节还有多个可用的选项，最重要的是：

- develop　它是一个多行值，列出使用 python setup.py develop 命令将安装的 egg。每个值是指向 setup.py 所在的包文件夹路径；
- find-links　它是一个多行值，提供了一个位置（URL 或文件）列表，easy_install 基于它查找在 eggs 中定义的 egg 或在安装 egg 时的依赖模块。

由此，一个 buildout 可以列出一系列将被安装在该环境中的 egg。

对于 develop 中指出的每个值，该工具运行 setuptools develop 命令并在定义了依赖性时读取 PyPI。

用于查找包的 Web 位置与 easy_install 所用的一样——http://pypi.python.org/simple。这是一个不打算供人们使用的，包含可被自动浏览的一个包链接列表的网页。

最后，find-links 选项提供了一个当包在其他位置可用时指向替代来源的方法。

示例如下。

```
[buildout]
parts =
   develop =
      /home/tarek/dev/atomisator.feed
find-links =
      http://acme.com/packages/index
```

使用这个配置，buildout 将和 python setup.py develop 命令一样安装 atomisator.feed 包。

可以使用 buildout 命令来构建这个环境。

7.1.2　buildout 命令

buildout 命令是 zc.buildout 安装的，通常会被 easy_install 调用，以对配置文件进行解释，如下所示。

```
$ easy_install zc.buildout
...
$ buildout
While:
    Initializing.
Error: Couldn't open /Users/tarek/buildout.cfg
```

一个空目录中的一个带有 init 选项的初始调用将创建一个默认的 buildout.cfg 文件和几个其他元素，如下所示。

```
$ cd /tmp
$ mkdir tests
$ cd tests
$ buildout init
Creating '/tmp/tests/buildout.cfg'.
Creating directory '/tmp/tests/bin'.
Creating directory '/tmp/tests/parts'.
Creating directory '/tmp/tests/eggs'.
Creating directory '/tmp/tests/develop-eggs'.
Generated script '/tmp/tests/bin/buildout'.
$ find .
.
./bin
./bin/buildout
./buildout.cfg
./develop-eggs
./eggs/setuptools-0.6c7-py2.5.egg
./eggs/zc.buildout-1.0.0b30-py2.5.egg
./parts
$ more buildout.cfg
[buildout]
parts =
```

bin 文件夹中包含一个本地 buildout 脚本，还创建了其他 3 个文件夹：

- parts　对应配置文件中定义的小节，它是每个被调用的 recipe 可以写入元素的标准位置；
- develop-eggs　保存将环境链接到 develop 中定义的包的信息；
- eggs　包含该环境使用的 egg，它已经包含了 zc.buildout 和 setuptools 两个 eggs。

接下来，在 cfg 中添加一个 develop 小节，如下所示。

```
[buildout]
parts =
develop =
    /home/tarek/dev/atomisator.feed
```

指定的文件夹将被 zc.buildout 再次调用 buildout 命令安装为 develop egg，如下所示。

```
$ bin/buildout
Develop: '/home/tarek/dev/atomisator.feed'
$ ls develop-eggs/
atomisator.feed.egg-link
$ more develop-eggs/atomisator.feed.egg-link
/home/tarek/dev/atomisator.feed
```

develop-eggs 文件夹现在包含了一个指向位于/home/tarek/dev/atomisator.feed 的 atomisator.feed 包的链接。当然，任何包文件夹都可以使用 develop 选项捆绑到 buildout 脚本上。

7.1.3　recipe

我们已经看到，每个小节将一个包指定为 recipe。例如，zc.recipe.egg 被用来指定一个或多个 buildout 中安装的 egg。这个 recipe 将像 easy_install 一样，通过调用 PyPI 拖动包，并且最终在 PyPI 没有包含它的情况下查找 find-links 中提供的链接。

例如，如果想将 Nose 安装到 buildout，可以在配置文件中配置一个专门的小节，并且在 buildout 小节中的 parts 变量中指向它，如下所示。

```
[buildout]
parts =
   test
develop =
     /home/tarek/dev/atomisator.feed
[test]
recipe = zc.recipe.egg
eggs =
   nose
```

再次运行 buildout 脚本，将执行 test 小节并和 easy_install 一样拖动 Nose egg，如下所示。

```
$ bin/buildout
Develop: '/home/tarek/dev/atomisator.feed'
Installing test.
Getting distribution for nose
Got nose 0.10.3.
```

nosetest 脚本将被安装到 bin 文件夹中，Nose egg 将在 eggs 文件夹中。再次使用 zc.recipe.egg 在 cfg 文件中添加一个名为 other 的新小节，如下所示。

```
[buildout]
parts =
    test
    other
develop =
    /home/tarek/dev/atomisator.feed
```

```
[test]
recipe = zc.recipe.egg
eggs =
   nose
[other]
recipe = zc.recipe.egg
eggs =
   elementtree
   PIL
...
```

这个新的小节定义了两个新的包。再次运行 buildout 脚本，如下所示。

```
$ bin/buildout
Develop: '/home/tarek/dev/atomisator.feed'
Updating test.
Installing other.
Getting distribution for elementtree
Got elementtree 1.2.7-20070827-preview.
Getting distribution for 'PIL'.
Got PIL 1.1.6.
```

在 parts 中指向的小节将按照所定义的顺序运行。再次运行时，zc.buildout 将检查已安装的部件，了解它们是否需要更新，如果需要则安装新的版本。在 other 小节中，eggs 文件夹增加了两个新的 egg。

Recipe 是简单的 Python 包，一般专门用于这个目的。它们也是嵌入的命名空间包，其中第一个部分是组织名，第二个部分是 recipe，第三个部分是 recipe 的名称。

目前已经用过的 recipe 是由 Zope 公司（zc）提供的，但是 PyPI 上还有很多 recipe 可以用来处理 buildout 环境中的许多需求。

因为 Zope 或 Plone 这样的框架依赖于这个工具，所以在http://pypi.python.org上搜索 buildout 或 recipe 将返回几百个可用于构成各种 buildout 的包。

1. 著名的 recipe

以下是在 PyPi 上找到的有用的 recipe 的简短列表。

- collective.recipe.ant　构建 Ant（Java）项目
- iw.recipe.cmd　执行一个命令行
- iw.recipe.fetcher　下载 URL 指向的一个文件
- iw.recipe.pound　编译和安装 Pound（一个负载平衡程序）
- iw.recipe.squid　配置和运行 Squid（一个缓存服务器）
- z3c.recipe.ldap　部署 OpenLDAP

2．创建 recipe

Recipe 是一个简单的类，具有 install 和 update 两个方法，它们将返回安装文件的列表。因此，编写新的 recipe 代码非常简单且可以用模板完成。

ZopeSkel 项目在 Zope 社区中被用于构建新的 recipe，安装它可以拥有一个新的 recipe 模板，如下所示。

```
$ easy_install ZopeSkel
Searching for ZopeSkel
Best match: ZopeSkel 2.1
...
Finished processing dependencies for ZopeSkel
$ paster create --list-templates
Available templates:
...
recipe:             A recipe project for zc.buildout
...
```

recipe 生成一个嵌套的命名空间包结构，以及必须被完成的一个 Recipe 类的骨架，如下所示。

```
$ paster create -t recipe atomisator.recipe.here
Selected and implied templates:
   ZopeSkel#recipe A recipe project for zc.buildout
   ...
Enter namespace_package ['plone']: atomisator
Enter namespace_package2 ['recipe']:
Enter package ['example']: here
Enter version (Version) ['1.0']:
Enter description ['']: description is here.
Enter long_description ['']:
Enter author (Author name) ['']: Tarek
Enter author_email (Author email) ['']: tarek@ziade.org
...
Creating template recipe
Creating directory ./atomisator.recipe.here
...
$ more atomisator.recipe.here/atomisator/recipe/here/__init__.py
# -*- coding: utf-8 -*-
"""Recipe here"""
class Recipe(object):
   """zc.buildout recipe"""
   def __init__(self, buildout, name, options):
```

```
        self.buildout, self.name, self.options = \
                buildout, name, options
    def install(self):
        """Installer"""
        # XXX 实现这里的 recipe 的功能

        # 返回 recipe 创建的文件
        # 当重新安装时，buildout 将删除所有返回的文件
        return tuple()
    def update(self):
        """Updater"""
        pass
```

7.1.4 Atomisator buildout 环境

Atomisator 项目可以通过创建一个和包在一起的专用的 buildout 配置来从 zc.buildout 获益，并且在配置中定义一个环境。

buildout 环境的构建可以分为两步：

（1）创建一个 buildout 文件夹结构；

（2）初始化 buildout。

buildout 文件夹结构

因为 buildout 允许将任何系统文件夹作为一个 develop 包来链接，所以应用程序环境可以与之分离。最清晰的设计是使用一个用于 buildout 和一个用于被开发的包的文件夹。

回顾一下前一章中创建的 Atomisator 文件夹。目前，它包含一个带有本地解释程序的 bin 文件夹和一个 packages 文件夹。在此，将添加一个 buildout 文件夹，如下所示。

```
$ cd Atomisator
$ mkdir buildout
```

然后，在 buidout 文件夹中构建一个新的 buildout 环境，如下所示。

```
$ cd buildout
$ buildout init
Creating 'Atomisator/buildout/buildout.cfg'.
Creating directory 'Atomisator/buildout/bin'.
Creating directory 'Atomisator/buildout/parts'.
Creating directory 'Atomisator/buildout/eggs'.
Creating directory 'Atomisator/buildout/develop-eggs'.
Generated script 'Atomisator/buildout/bin/buildout'.
```

修改 buildout.cfg 以便生成一个本地 nosetest 脚本，并将 Atomisator egg 作为 develop egg 安装，如下所示。

```
[buildout]
  develop =
     ../packages/atomisator.main
     ../packages/atomisator.db
     ../packages/atomisator.feed
     ../packages/atomisator.parser
  parts =
      test
  [test]
  recipe = pbp.recipe.noserunner
  eggs =
     atomisator.main
     atomisator.db
     atomisator.feed
     atomisator.parser
```

这个配置文件将在 buildout 文件夹中生成一个完整的 Atomisator 环境。

前一章中，在与开发包相同的本地解释程序中安装了 Nose，这多亏了 virtualenv。在 buildout 中工作时，要拥有相同的功能则需要更多的工作：将 Nose 作为一个 egg 安装到 buildout 中，但是不让其他 egg 直接看到这个测试运行程序。为了得到一个相似的环境，一个小的名为 pbp.recipe.noserunner 的 recipe 将生成一个使用特定环境的本地运行程序 nosetests。所有在这个运行程序的 eggs 变量中定义的 egg，都将被添加到测试运行程序执行环境中。

这个 recipe 使用小节名作为生成脚本的名称。所以在我们的案例中将有一个 test 脚本，它可以用来测试所有的 atomisator 包，如下所示。

```
$ bin/test atomisator
........
----------------------------------------------------------------------
Ran 8 tests in 0.015s
OK
```

7.1.5　更进一步

另一个可以执行的步骤是在 buildout 文件夹下的 etc 文件夹中创建和使用 atomisator.cfg 文件。这也需要创建一个新的 recipe，用它来读取 atomisator.cfg 中的值，并且生成 atomisator.cfg。

接下来创建的一个新小节，其内容类似于如下所示。

```
...
[atomisator-configuration]
recipe = atomisator.recipe.installer
sites =
```

```
        sample1.xml
        sample2.xml
    database = sqlite:///${buildout:directory}/var/atomisator.db
    title = My Feed
    description = The feed
    link = the link
    file = ${buildout:directory}/var/atomisator.xml
    ...
```

其中，${buildout:directory}将替换为 buildout 的路径。

7.2 发行与分发

在前一节中已经看到，buildout 是一个单独的文件夹，可以包含所有运行应用程序所需要的内容。所有需要的 egg 都被安装在这个文件夹中，控制台脚本则创建在 bin 文件夹中。

事实上，Atomisator 文件夹可以被做成一个档案文件，然后在其他拥有 Python 的电脑上解压。通过在新的目标系统上再次运行 buildout，可以正确地启动所有的部件，应用程序可以由此运行。

以这样的方法分发源代码，从整体上看可以与每个操作系统提供的包管理系统（如 apt 或 RPM）相比。所有部件都单独地存在于一个自包含的文件夹中，并且能够在每个系统上运行。因此，它不能无缝地与目标系统集成，并将使用自己的专有标准。这对于许多应用程序来说是不错的，但是纯粹主义者希望可以使用目标系统上的包管理系统来安装，从而简化系统的维护工作。

如果这是必需的，就需要一些额外的平台特定的集成工作。这是一个非常广泛的主题，其内容超出了本书的范围，所以在此不做介绍，但是这里介绍的源代码发行是针对特定目标发行的第一步。

让我们关注于 buildout 文件夹的分发。

但是，将 packages 文件夹和 buildout 文件夹及链接成 develop egg 的子文件夹一起交付并不是最佳的选择，因为我们希望发行每个 egg 的标记版本。buildout 能够解释任何配置文件。所以最好的办法是创建一个配置文件，这个文件不会将 develop 选项与刚刚创建的每个包构建的一组 egg 一起使用。

所以，发行一个 buildout 由以下 3 个步骤来完成：

（1）发行包；

（2）创建发行配置；

（3）构建和准备发行版本。

7.2.1　发行包

可以使用 sdist、bdist 或 bdist_egg 命令来将每个包发行为 egg。对于我们的应用程序而言，由于没有需要编译的代码，所以一个源代码分发版本就足以应付所有的平台。

对于每个包，按照前一章所看到的相同方式，建立一个源代码分发版本，如下所示。

```
$ python setup.py sdist
running sdist
...
Writing atomisator.db-0.1.0/setup.cfg
tar -cf dist/atomisator.db-0.1.0.tar atomisator.db-0.1.0
gzip -f9 dist/atomisator.db-0.1.0.tar
removing 'atomisator.db-0.1.0' (and everything under it)
$ ls dist/
atomisator.db-0.1.0.tar.gz
```

其结果是一个档案文件，它将被推送到 PyPI 或者保存在一个文件夹中。

7.2.2　添加一个发行配置文件

zc.buildout 提供了一个扩展机制，可以按照层次创建配置文件。使用指定另一个配置文件的 extends 选项，一个文件就可以继承所有值然后添加新的值，或者覆盖一些原有值。

一个专用于发行的新配置文件可以按以下的风格来创建，以用于特殊的设置：

- 需要指向 buildout 已发行的包；
- 需要去掉 develop 选项。

得到的结果如下。

```
 [buildout]
extends = buildout.cfg
develop =
parts =
   atomisator
   eggs
download-cache = downloads
[atomisator]
recipe = zc.recipe.eggs
eggs =
   atomisator.main
   atomisator.db
   atomisator.feed
   atomisator.parser
```

在这里，download-cache 是一个系统文件夹，buildout 将从 PyPI 上下载的 egg 保存在这里。downloads 文件夹最好是创建在 buildout 文件夹内，如下所示。

```
$ mkdir downloads
```

eggs 部分是从 buildout.cfg 继承的，并且不需要复制到新文件中。atomisator 部分将从 PyPI 拖动 egg 并将它们保存在 downloads 中。

7.2.3 构建和发行应用程序

接着，可以使用这个特殊的配置来构建 buildout。使用-c 选项指定一个特定的配置文件，使用-v 选项可以得到更多细节，如下所示。

```
$ bin/buildout -c release.cfg -v
Installing 'zc.buildout', 'setuptools'.
...
Installing atomisator.
Installing 'atomisator.db', 'atomisator.feed', 'atomisator.parser',
'atomisator.main'.
...
Picked: setuptools = 0.6c8
```

当这个步骤完成时，包将被下载并保存在 downloads 文件夹中，如下所示。

```
$ ls downloads/dist/
atomisator.feed-0.1.0.tar.gz
atomisator.main-0.1.0.tar.gz
atomisator.db-0.1.0.tar.gz
atomisator.parser-0.1.0.tar.gz
```

这意味着下一次运行时不需要从 PyPI 拖动包。换句话说，这时可以在脱机模式下构建 buildout。

发行版本已经可以通过分发 buildout 文件夹（例如，用一个归档的版本）来交付了。

最后要做的一件事情就是在文件夹中添加一个 bootstrap.py 文件，它可以自动化 zc.buildout 的安装，并和 buildout init 同样在目标系统上创建 bin/buildout 脚本，如下所示。

```
$ wget http://ziade.org/bootstrap.py
```

社区中有一些工具提供了一些脚本，可以使用附加的选项来准备这些归档版本，如 collective.releaser 和 zc.sourcerelease。

7.3　小　　结

本章中主要介绍了 zc.buildout：

- 可以用来构建基于 egg 的应用程序；
- 知道如何聚集 egg 以构建一个独立的环境；
- 链接 recipe（这是一些小的 Python 包）以构建一个脚本，这个脚本可以建立用于确定 Python 应用程序源代码分发版本的环境。

总而言之，使用 zc.buildout 的步骤包括：

- 使用一个 egg 列表创建一个 buildout，并用它来开发；
- 创建一个专用于发行的配置文件，并用它来构建一个可分发的 buildout 文件夹。

下一章中将进一步介绍这个工具，说明使用它和其他工具一起管理项目的方法。