

第 13 章

GitHub托管项目

GitHub 为当前最流行的开源项目托管平台，数以万计优秀的开源项目被托管在 GitHub 上面。随着越来越多的应用程序转移到云上，Github 已经成为管理软件开发以及发现已有代码的首选方法。对于普通开发人员来讲，在 GitHub 上托管的项目已经成为了他的一张个人名片。许多优秀的互联网公司在招聘开发人员时都希望对方能够提供个人的 GitHub 地址。当然，GitHub 作为一个开源软件的大宝库，学习和使用它，也会为我们的日常开发带来许多益处。

在使用 GitHub 之前首先需要了解 Git，Git 是一个开源的分布式版本控制系统，用以高效地处理从很小到非常大的项目版本管理。Git 是 Linux Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。相比 CVS、SVN 等版本控制工具，Git 无疑更加优秀，功能更加强大，在项目版本管理中被越来越多的人使用。但 Git 相对来说比较难学。

使用 Git 来管理项目有两种方式，一种是本地部署 Git 版本管理系统，另一种是通过在线代码托管。本地部署 Git 版本管理系统，需要自己搭建环境，但项目的提交与更新速度快，更适合较为封闭的项目；使用在线托管最大的好处是在有网络的情况下可以随时随地提交自己的代码，但项目是公开的，当然也可以创建私有项目，但需要支付一定的费用。

GitHub 就是基于 Git 的在线代码托管平台。

13.1 注册与安装

13.1.1 注册 GitHub

GitHub 官方地址：<https://github.com>。

在浏览器中打开 GitHub 网址，通过首页进行注册，如图 13.1 所示。

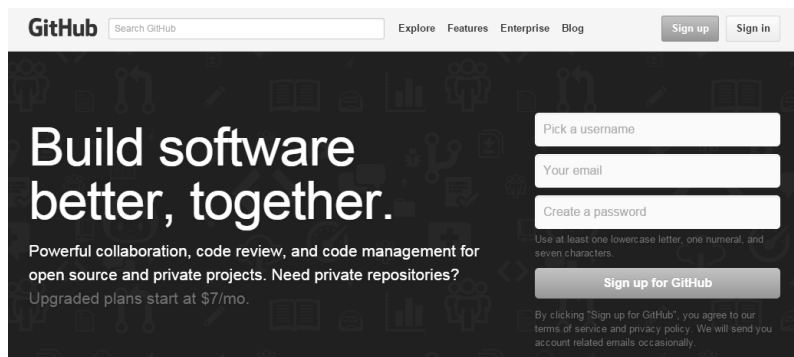


图 13.1 在 GitHub 首页

13.1.2 安装 Git

Git 官方下载地址：<http://git-scm.com/download/>。

Git 支持多平台（Mac OS X/Windows/Linux/Solaris），读者可根据自己的平台选择相应的版本下载。

Linux 各版本下安装 Git:

```
=====
Debian/Ubuntu $ apt-get install git-core
Fedora $ yum install git
Gentoo $ emerge --ask --verbose dev-vcs/git
Arch Linux $ pacman -S git
=====
```

下载并安装完成后，我们通常在 Mac OSX 及 Linux 平台下用终端工具（Terminal）来使用 Git，而在 Windows 平台下用 Git Bash 工具，如图 13.2 所示。



图 13.2 在 Windows 下安装 Git

13.1.3 建立连接

本地 Git 与 GitHub 服务器之间保持通信时，使用 SSH key 认证方式来保证通信安全，所以在使用 GitHub 前读者必须先创建自己的 SSH key。

我们后续的演示会在 Windows 环境下进行，打开 Git Bash，如图 13.3 所示。



图 13.3 Git Bash 界面

① 进入 SSH 目录。

Git Bash

```
fnngj@FNNGJ-PC ~
$ cd ~/.ssh

fnngj@FNNGJ-PC ~/.ssh
$ pwd
/c/Users/fnngj/.ssh
```

② 生成新的 SSH 密钥。

如果你已经有了一个密钥（默认密钥文件位置在 C:/Users/fnngj/.ssh/id_rsa）

Git Bash

```
fnngj@FNNGJ-PC ~/.ssh
$ ssh-keygen -t rsa -C "fnngj@126.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/fnngj/.ssh/id_rsa): --回车
Enter passphrase (empty for no passphrase): --回车
Enter same passphrase again: --回车
Your identification has been saved in /c/Users/fnngj/.ssh/id_rsa.
Your public key has been saved in /c/Users/fnngj/.ssh/id_rsa.pub.
The key fingerprint is:
78:51:9b:2c:6c:fb:74:0b:6b:b9:c4:23:8f:5e:10:6b fnngj@126.com
The key's randomart image is:
+--[ RSA 2048 ]-----+
|      .               |
|     . o o            |
|    * +               |
|   o *                |
|  . E o .             |
|   o = = .            |
|    . X .             |
|     B o              |
|    . o o             |
+-----+
fnngj@FNNGJ-PC ~/.ssh
$ ls
id_rsa id_rsa.pub
```

查看目录下会生成两个文件，`id_rsa` 是私钥，`id_rsa.pub` 是公钥。记住千万不要把私钥文件 `id_rsa` 透露给任何人。

③ 添加 SSH 公钥到 GitHub。

用文本工具打开公钥文件 `~/.ssh/id_rsa.pub`，复制里面的所有内容到剪贴板，如图 13.4 所示。

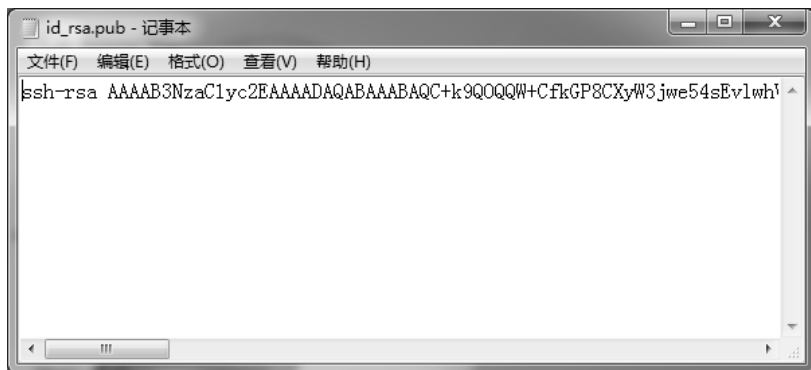


图 13.4 查看生成的公钥

登录 GitHub，单击右上角个人头像→Settings→SSH Keys→Add SSH Keys，在 Title 文本框中输入任意字符，在 Key 文本框粘贴刚才复制的公钥字符串，单击“Add key”按钮完成操作，如图 13.5 所示。

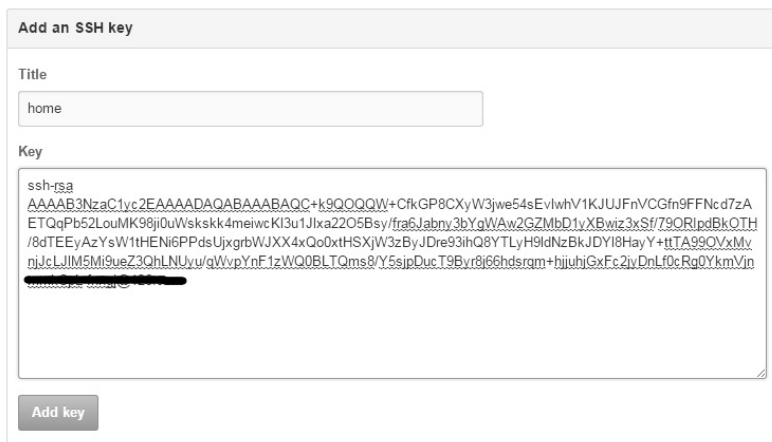


图 13.5 添加公钥到 GitHub

④ 测试连接。

以上步骤完成后，可以通过以下命令来测试是否可以连接 GitHub 服务器。

Git Bash

```
fnngj@FNNGJ-PC ~/.ssh
$ ssh -T git@github.com
The authenticity of host 'github.com (192.30.252.129)' can't be established.
RSA key fingerprint is 16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'github.com,192.30.252.129' (RSA) to the list of known hosts.
Hi defnngj! You've successfully authenticated, but GitHub does not provide shell access.
```

13.2 Git/GitHub 基本使用

13.2.1 GitHub 创建项目

在 GitHub 选择并创建一个项目。首先，登录 GitHub，单击页面右上角加号（+），选择“New repository”选项。

填写项目名称及描述，默认项目类型为“Public”，如果想创建“Private”项目，GitHub 需要收费。最后单击“Create repository”完成项目的创建，如图 13.6 所示。

创建完一个项目之后，GitHub 会提示我们如何提交项目到它上面。

Quick setup — if you've done this kind of thing before

HTTPS `https://github.com/defnngj/project-name.git`

SSH `git@github.com:defnngj/project-name.git`

...or create a new repository on the command line

```
echo # project-name >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:defnngj/project-name.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:defnngj/project-name.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

图 13.6 在 GitHub 上创建项目

13.2.2 本地创建项目

首先进行初始化配置：设置仓库人员的用户名和邮箱地址，这一步必不可少。

Git Bash

```
fnngj@FNNGJ-PC ~/.ssh
$ git config --global user.name "fnngj"

fnngj@FNNGJ-PC ~/.ssh
$ git config --global user.email "fnngj@126.com"
```

在本地创建一个“project-name”项目，与 GitHub 上创建的项目名保持一致。在项目

下创建一个 `test_case.py` 文件。

Git Bash

```
fnngj@FNNGJ-PC /d/project-name
$ ls
test_case.py
```

通过 “ls” 命令查看当前目录下的文件及文件夹。

Git Bash

```
fnngj@FNNGJ-PC /d/project-name
$ git init
Initialized empty Git repository in d:/project-name/.git/
```

“git init” 命令用于对当前目录进行初始化，将当前的 `project-name` 目录交由 Git 进行管理。

Git Bash

```
fnngj@FNNGJ-PC /d/project-name (master)
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    test_case.py

nothing added to commit but untracked files present (use "git add" to track)
```

“git status” 命令用于查看当前项目下所有文件的状态。

上面的信息显示当前处于 `master`（主）分支，罗列当前项目下的文件（`test_case.py`），并且提示使用 “git add <file>...” 命令对当前项目文件进行跟踪（跟踪文件增、删、改的状态）。

Git Bash

```
fnngj@FNNGJ-PC /d/project-name (master)
$ git add .

fnngj@FNNGJ-PC /d/project-name (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   test_case.py
```

“git add”命令将文件交由 git 进行跟踪。如果后面跟空格加点号“.”，则表示对当前项目下的所有文件进行跟踪。

再次通过“git status”命令查看当前 Git 仓库的信息。

Git Bash

```
fnngj@FNNGJ-PC /d/project-name (master)
$ git commit -m "first commit file"
[master (root-commit) 9b4b839] first commit file
1 file changed, 9 insertions(+)
create mode 100644 test_case.py
```

“git commit”命令将文件（由 git 跟踪的文件）提交到本地仓库。-m 参数对本次的提交加以描述，通常提交的描述必不可少，从而方便追溯每次提交都做了哪些修改。

准备工作已经完成，下面提交代码到 GitHub。这里 GitHub 提供了两种连接方式：HTTPS 和 SSH，提交的地址有所不同，可查看前面 GitHub 提示信息。

Git Bash

```
fnngj@FNNGJ-PC /d/project-name (master)
$ git remote add origin git@github.com:defnngj/project-name.git

fnngj@FNNGJ-PC /d/project-name (master)
```

```
$ git push -u origin master
Warning: Permanently added the RSA host key for IP address '192.30.252.128' to
t
he list of known hosts.
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 354 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:defnngj/project-name.git
* [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

“git remote add origin git@github.com:defnngj/project-name.git”

如果是第一次提交项目，通过这一行命令将本地的项目与远程的仓库建立连接。此处使用 SSH 方式进行连接：

“git push -u origin master ”

将本地的项目提交到远程仓库的主分支。

现在访问 GitHub 就可以看到我们提交的项目了，如图 13.7 所示。

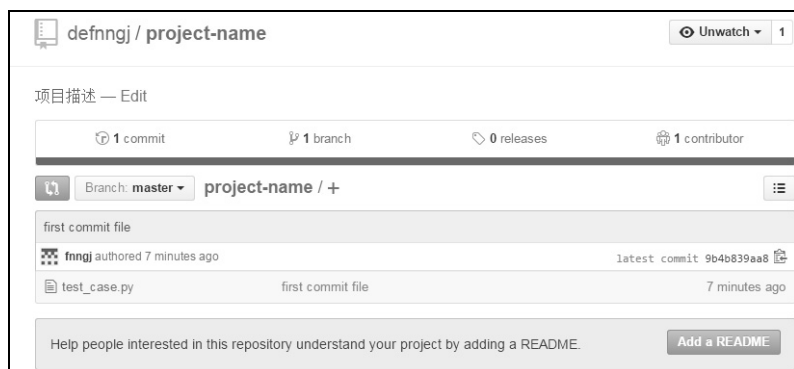


图 13.7 查看 GitHub 项目

13.2.3 克隆项目

我们除了可以向 GitHub 上提交项目之外，更多的时候是我们到上面克隆（下载）优秀

的开源项目来用，当然也可以将使用过程中发现的 bug，通过建立分支的方式提交给项目的原作者。

我们现在的场景是在家将项目提交到 GitHub 上，到公司之后，需要将 GitHub 上的项目克隆到公司电脑，那么对于公司的电脑来说，同样需要与 GitHub 建立连接。

首先，下载安装 Git。

其次，通过 Git 生成本地公钥，并且将公钥添加到 GitHub 中。

最后，设置仓库人员的用户名和邮箱地址。

具体过程请参考第 13.1 节。当一切都设置完成后，就可以从 GitHub 上克隆项目到本地了。我们同样以 Windows 系统为例，打开 Git Bash。

Git Bash

```
Meizu@MININT-IQVJFIT /d/my_test
$ git clone git@github.com:defnngj/project-name.git
Cloning into 'project-name'...
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
Checking connectivity... done.

Meizu@MININT-IQVJFIT /d/my_test
$ cd project-name/

Meizu@MININT-IQVJFIT /d/my_test/project-name (master)
$ ls
test_case.py
```

“git clone”命令用于克隆 GitHub 上的项目到本地，通过“cd”命令进入项目目录，查看项目文件。

13.2.4 更新项目

当在公司的电脑上对项目做了更新之后（删除了原有文件，创建文件与文件夹），需要再次提交项目到 GitHub。

Git Bash

```

Company@MININT-IQVJFIT /d/my_test/project-name (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    test_case.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        126login.py
        baidu.py
        package/
        po_login.py
        test_case/

no changes added to commit (use "git add" and/or "git commit -a")

```

通过“git status”命令查看当前变更。通过变更信息可以看出，删除了 test_case.py 文件。这个删除只是在项目目录下进行删除，Git 对此文件留有记忆，所以要通过“git rm”命令将其删除。

与此同时又重新创建一些测试目录与文件。提示通过“git add <file>...”命令交由 Git 跟踪管理。

Git Bash

```

Company@MININT-IQVJFIT /d/my_test/project-name (master)
$ git rm test_case.py
rm 'test_case.py'

```

如果删除的是文件夹同样用此命令，例如，git rm test_case/。

如果删除的文件名中包含空格，则需要通过双引号将文件名引起来，例如，git rm “test

case.py”。

Git Bash

```
Company@MININT-IQVJFIT /d/my_test/project-name (master)
$ git add .
```

```
Meizu@MININT-IQVJFIT /d/my_test/project-name (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
```

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
```

```

new file:   126login.py
new file:   baidu.py
new file:   package/__init__.py
new file:   package/location.py
new file:   po_login.py
deleted:    test_case.py
new file:   test_case/126login.py
new file:   test_case/allpage/__init__.py
new file:   test_case/allpage/base.py
new file:   test_case/allpage/login_page.py
```

```
Company@MININT-IQVJFIT /d/my_test/project-name (master)
$ git commit -m "update project"
[master 0e8eece] update project
10 files changed, 377 insertions(+), 9 deletions(-)
create mode 100644 126login.py
create mode 100644 baidu.py
create mode 100644 package/__init__.py
create mode 100644 package/location.py
create mode 100644 po_login.py
delete mode 100644 test_case.py
create mode 100644 test_case/126login.py
create mode 100644 test_case/allpage/__init__.py
create mode 100644 test_case/allpage/base.py
create mode 100644 test_case/allpage/login_page.py
```

```
Company@MININT-IQVJFIT /d/my_test/project-name (master)
```

```
$ git push origin master
Warning: Permanently added the RSA host key for IP address '192.30.252.128' to
t
he list of known hosts.
Counting objects: 14, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (13/13), 3.41 KiB | 0 bytes/s, done.
Total 13 (delta 2), reused 0 (delta 0)
To git@github.com:defnngj/project-name.git
  9b4b839..0e8eece master -> master
```

“git add”：对当前目录下的文件添加跟踪。

“git commit”：将更新的文件提交到本地仓库。

“git push”：将本地项目提交到远程仓库 GitHub。

除第一次下载项目需要通过“git clone”将项目克隆到本地外，后续则使用“git pull”命令将 GitHub 的更新拉取到本地。

Git Bash

```
fnngj@FNNGJ-PC /d/project-name
$ git pull origin master
Warning: Permanently added the RSA host key for IP address '192.30.252.131' to
t
he list of known hosts.
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 13 (delta 2), reused 13 (delta 2), pack-reused 0
Unpacking objects: 100% (13/13), done.
From github.com:defnngj/project-name
 * branch          master    -> FETCH_HEAD
   9b4b839..0e8eece master    -> origin/master
Updating 9b4b839..0e8eece
Fast-forward
 126login.py          | 28 ++++++++
  baidu.py            | 18 ++++++
  package/__init__.py |  0
  package/location.py | 81 +++++++++++++++++++++++++++++++++++++
  po_login.py         | 103 ++++++++++++++++++++++++++++++++++++++
```

```

test_case.py | 9 ----
test_case/l26login.py | 44 ++++++
test_case/allpage/__init__.py | 0
test_case/allpage/base.py | 47 ++++++
test_case/allpage/login_page.py | 56 ++++++
10 files changed, 377 insertions(+), 9 deletions(-)
create mode 100644 l26login.py
create mode 100644 baidu.py
create mode 100644 package/__init__.py
create mode 100644 package/location.py
create mode 100644 po_login.py
delete mode 100644 test_case.py
create mode 100644 test_case/l26login.py
create mode 100644 test_case/allpage/__init__.py
create mode 100644 test_case/allpage/base.py
create mode 100644 test_case/allpage/login_page.py

```

例如，我们再次回到家后，将在公司的更新通过“git pull”拉取到本地。为了避免冲突我们应该形成良好的习惯，在每次 push 代码之前先把服务器上最新的代码 pull 到本地。

本章小结

通过对本章的学习，我们可以自由地提交或拉取 GitHub 上的项目代码。在多人开发的项目中，会涉及更多的技术，例如 Git 的分支与同步等。代码版本控制不是本书重点，在这里更多的是起一抛砖引玉的作用，读者可以进一步地学习 Git。

国内主流的在线托管网站：

<https://gitcafe.com>

<http://git.oschina.net>

<http://code.csdn.net>

<https://coding.net>

第 14 章

持续集成Jenkins入门

持续集成（Continuous integration，简称 CI），随着近几年的发展，持续集成在项目中得到了广泛的推广和应用。本章将带领读者一起了解持续集成工具 Jenkins 的安装与使用。

1. 什么是持续集成？

软件集成就是用一种较好的方式，使多种软件的功能集成到一个软件里，或是把软件的各部分组合在一起。如果项目开发的规模较小，且对外部系统的依赖很小，那么软件集成不是问题，例如一个人的项目。但是随着软件项目复杂度的增加，会对集成和确保软件组件能够在一起工作提出了更多的要求-->要早集成、常集成。早集成、频繁的集成能够帮助项目开发者在早期发现项目风险和质量问题，越到后期发现的问题，解决的成本越高，从而有可能导致项目延期或者项目失败。

2. 定义

大师 Martin Fowler 对持续集成是这样定义的：持续集成是一种软件开发实践，即团队开发成员经常集成他们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译、发布、自动化测试）来验证，从而尽快地发现集成错误。许多团队发现这个过程可以大大减少集成的问题，让团队能够更快地开发内聚的软件。

3. 什么是 Jenkins？

提到 Jenkins 就不得不提另一个持续集成工具——Hudson，Hudson 由 Sun 公司开发，2010 年 Sun 公司被 Oracle 公司收购，oracle 公司声称对 hudson 拥有商标所有权。Jenkins

是从 Hudson 中分离出来的一个版本，并将继续走 Open Source 的道路。二者现在由不同的团队在维护。

Jenkins 主要用于监视执行重复工作，如建立一个软件项目或工作运行的计划任务。当前 Jenkins 关注以下两个工作。

不断地进行项目的构建/测试软件：就像 CruiseControl 或 DamageControl。概括地说，Jenkins 提供了一个易于使用的所谓的持续集成系统，使开发人员更容易修改整合到项目中，并使它更容易为用户获得一个新的版本。自动连续生成提高了生产效率。

监控外部运行的作业：如计划任务作业和 Qrocmail 的工作，即使是那些在远程机器上运行的计划任务。Jenkins 生成这些日志并且很容易让你注意到错误的出现。

14.1 环境搭建

Jenkins 是基于 Java 开发的一种持续集成工具，所以，Jenkins 的运行需要 Java 环境。关于 Java 环境的配置在本书第 9 章使用 Selenium Grid 时有过介绍，这里不再讲解。

1. 安装 Tomcat

Tomcat 是针对 Java 的一个开源中间件服务器（容器），基于 Java Web 的项目需要借助 Tomcat 才能运行起来。

Tomcat 官方网站：<http://tomcat.apache.org/>，打开后首页如图 14.1 所示

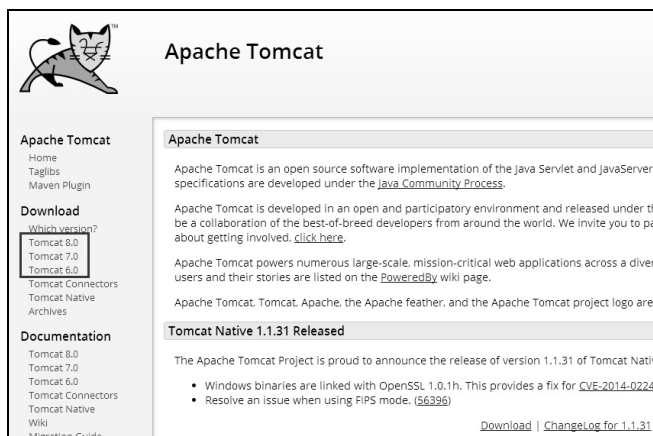
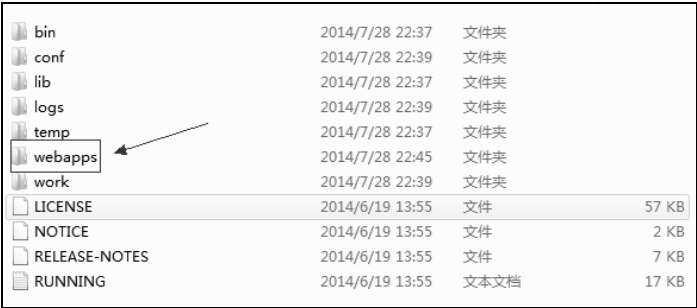


图 14.1 下载 Tomcat

单击页面左侧 Tomcat 版本进行下载，对下载的压缩包进行解压，目录结构如图 14.2 所示。



bin	2014/7/28 22:37	文件夹	
conf	2014/7/28 22:39	文件夹	
lib	2014/7/28 22:37	文件夹	
logs	2014/7/28 22:39	文件夹	
temp	2014/7/28 22:37	文件夹	
webapps	2014/7/28 22:45	文件夹	
work	2014/7/28 22:39	文件夹	
LICENSE	2014/6/19 13:55	文件	57 KB
NOTICE	2014/6/19 13:55	文件	2 KB
RELEASE-NOTES	2014/6/19 13:55	文件	7 KB
RUNNING	2014/6/19 13:55	文本文档	17 KB

图 14.2 webapps 目录用于 web 项目

通常将需要运行的应用放到 webapps/目录下，进入 bin/目录下，双击 startup.bat，启动 Tomcat 服务器。

2. 安装 Jenkins

Jenkins 官方网站：<http://jenkins-ci.org/>，打开后首页如图 14.3 所示。



图 14.3 下载 Jenkins

在页面右侧罗列了不同系统版本的 Jenkins，读者可根据自己的系统选择相应的版本下载。

下载完成，双击进行安装，如图 14.4 所示。



图 14.4 双击 Jenkins 安装

单击“next”按钮，在选择安装目录时，指定安装到 Tomcat 的 webapps\目录下，如图 14.5 所示。

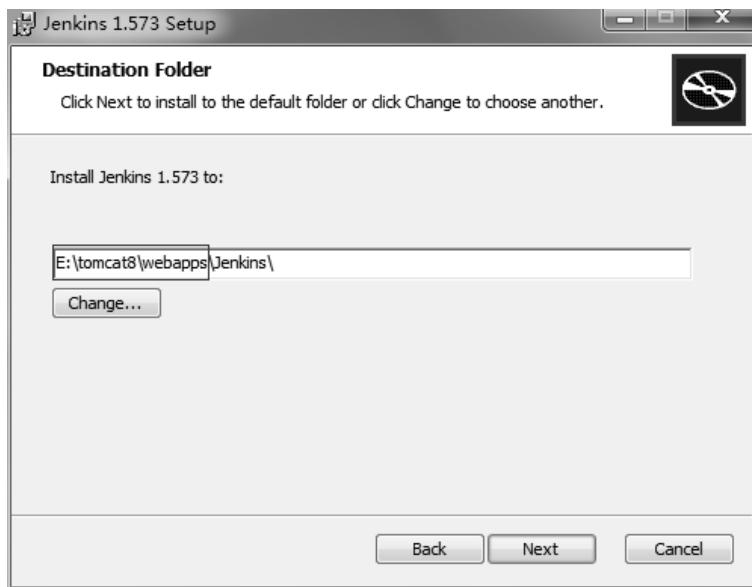


图 14.5 选择 Tomcat 的 webapps 目录

3. 运行 Jenkins

进入 Tomcat 的 bin/目录下启动 startup.bat ， 通过浏览器访问：<http://localhost:8080/>，如图 14.6 所示。



图 14.6 访问 Jenkins

Jenkins 已经安装成功，下面就跟着笔者来创建任务吧。

14.2 创建任务

单击首页“创建一个新任务”的链接，弹出如图 14.7 所示页面。

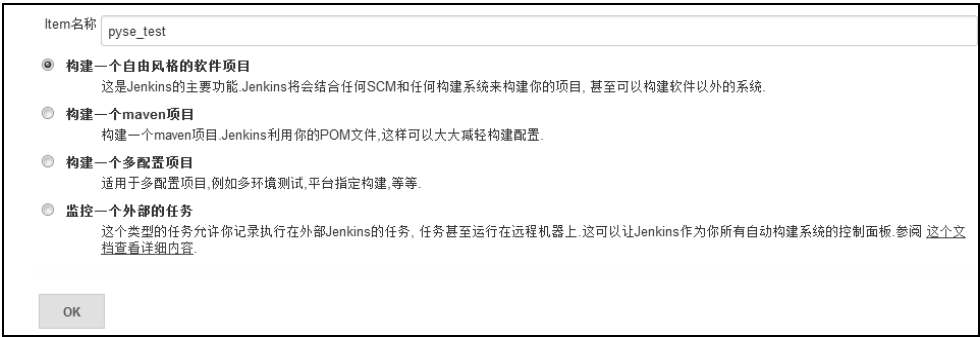


图 14.7 选择 Jenkins 任务类型

Jenkins 提供了四种类型的任务。

1) 构建一个自由风格的软件项目

这是 Jenkins 的主要功能。Jenkins 会结合任何 SCM 和任何构建系统来构建你的项目，甚至可以构建软件以外的系统。

2) 构建一个 maven 项目

构建一个 maven 项目。Jenkins 会利用你的 POM 文件，这样可以大大减轻构建配置。

3) 构建一个多配置项目

适用于多配置项目，例如多环境测试、平台指定构建等。

4) 监控一个外部的任务

这个类型的任务允许你记录执行在外部 Jenkins 的任务，任务可以运行在远程机器上。这可以让 Jenkins 作为你所有自动构建系统的控制面板。

如图 14.7 所示，输入项目的名称，选择第一项“构建一个自由风格的软件项目”，单击“OK”按钮，进入项目的详细配置页面，如图 14.8 所示。



The image shows the Jenkins project configuration page for a project named 'pyse_test'. The 'Project Name' field is filled with 'pyse_test'. The 'Description' field contains the text '这是一个python + selenium 自动化测试项目'. Below the description field, there is a '[Plain text] 预览' button. On the left side, there are four checkboxes, all of which are unchecked: '丢弃旧的构建' (Discard old builds), '参数化构建过程' (Parameterize build process), '关闭构建 (重新开启构建前不允许进行新的构建)' (Disable build (disallow new builds until build is restarted)), and '在必要的时候并发构建' (Concurrent builds when necessary). On the right side, there are four question mark icons, each corresponding to one of the checkboxes.

图 14.8 添加项目描述

1. 丢弃旧的构建

主要用来配置构建历史保存的几个版本，每次执行构建都会产生日志，日志的生成会占用一定的磁盘空间，通过勾选此选项就可以设置保持构建天数和构建次数。

高级选项如图 14.9 所示。

高级项目选项

☒ 安静期

安静期

5

秒

☒ 重试次数

SCM 签出重试次数

0

☐ 该项目的上游项目正在构建时阻止该项目构建

☐ 该项目的下游项目正在构建时阻止该项目构建

☒ 使用自定义的工作空间

目录

Custom workspace is empty.

显示名称

图 14.9 高级选项

2. 安静期

如果设置此选项，则一个计划中的构建在开始之前需要等待选项中设置的秒数，这对下面的情况非常有用：

- 合并多封 CVS 变更通知邮件为一封（当一次提交跨越多个目录时，一些 CVS 的变更日志邮件发生脚本会接二连三地生成多封通知邮件）。
- 如果你的编码风格导致你更改一个逻辑需要几次 CVS/SVN 操作，那么设置一个较长时间的等待会防止 Jenkins 因过早构建而失败。
- 节省构建。如果你的 Jenkins 有太多且高频率的构建，则设置长时间的等待会舒缓这些构建。

如果没有在项目级别设置此项，则会使用系统默认值。

3. 重试次数

如果从版本库签出代码失败，则 Jenkins 会按照这个指定的次数进行重试之后再放弃。

源码管理如图 14.10 所示。

源码管理

☒ None

☐ CVS

☐ CVS Projectset

☐ Subversion

图 14.10 源码管理

源码管理主要是为了与代码版本控制工具产生关联。当代码有更新时，自动触发构建，从而更早地获得代码的构建结果，从构建结果发现问题、修复问题，直到构建通过，这样将有效地保证代码的质量。

构建触发器如图 14.11 所示。

构建触发器

☒ Build after other projects are built

Projects to watch

☒ No project specified

☐ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

☒ Build periodically

日程表

No schedules so will never run

☒ Poll SCM

日程表

No schedules so will never run

Ignore post-commit hooks ☐

图 14.11 构建触发器

Build after other projects are built

在其他项目执行完成之后才触发执行构建。

Build periodically

设置定时构建。例如，可以设置周一到周五每天晚上 12 点处理构建。

Poll SCM

定时检查版本控制工具中的代码是否有变更（根据 SCM 软件的版本号），如果有更新就 checkout 最新代码下来，然后执行构建动作。

增加构建步骤如图 14.12 所示。

在增加构建步骤中提供了以下几个选项：

- Execute Windows batch command

执行 Windows 批处理命令。

- Execute shell

执行 shell 脚本。

- Invoke Ant

调用 Ant，即 Apache Ant。是一个将软件编译、测试、部署等步骤联系在一起加以自动化的一个工具，大多用于 Java 环境下的软件开发。

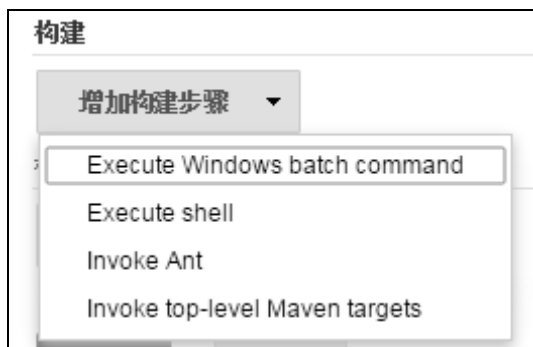


图 14.12 构建步骤

- Invoke top-level Maven targets

调用 Maven 对象。Maven 是基于项目的对象模型（POM），可以通过一小段描述信息来管理项目的构建、报告和文档的软件项目管理工具。

这里以选择“Execute Windows batch command”、通过 Windows 命令提示符执行一个 Python 脚本为例。在 D:\盘根目录下有一个 test.py 的脚本，在命令提示符下运行这个脚本，如图 14.13 所示。

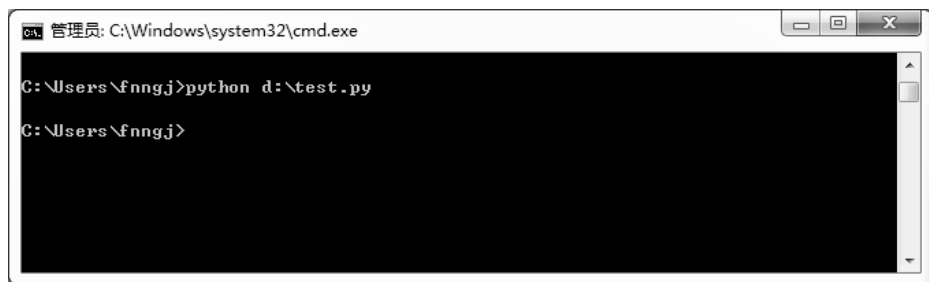


图 14.13 dos 命令

通过“python”命令执行 test.py 文件。如果执行的文件不在当前目录下，则需要指定

文件的路径。复制执行命令到 Jenkins 的命令输入框。如图 14.14 所示。

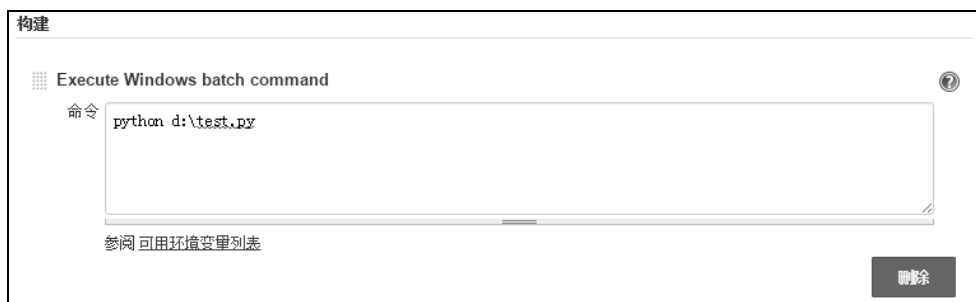


图 14.14 构建命令

创建完成，单击保存。

14.3 运行构建

项目首页如图 14.15 所示。

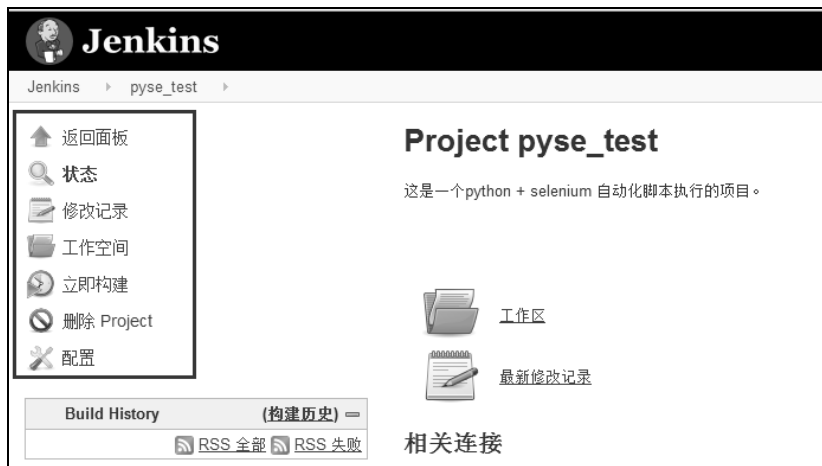


图 14.15 查看创建的项目

在创建的 `pyse_test` 项目左侧罗列了当前项目的操作。如果需要修改配置信息，可以单击“配置”链接重新进行修改。

单击“立即构建”选项，Build History 将显示项目的构建状态，如图 14.16 所示。



图 14.16 构建项目

单击构建历史时间（如图 14.16 中的 2014-7-29 23:00:15），将弹出如图 14.17 所示的构建版本。



图 14.17 构建版本

单击“Console Output”，即可查看构建日志，如图 14.18 所示。

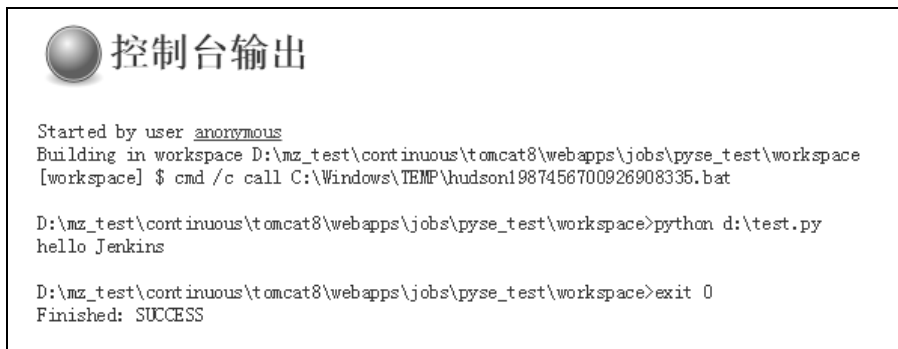


图 14.18 构建日志

图 14.18 所示的构建日志与我们在命令提示符下的操作一致，最后提示“SUCCESS”表示此次构建是成功的。

查看构建历史

单击图 14.17 中的“返回到工程”可以返回到项目首页，如图 14.15 所示。单击“构建历史”，如果进行了多次构建，则可以看到项目的构建历史图表，如图 14.19 所示。

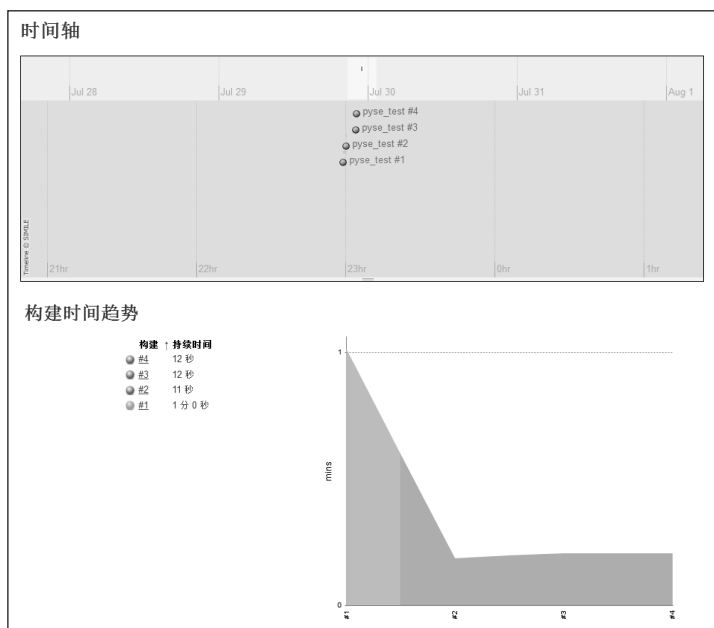


图 14.19 查看构建历史

14.4 定时执行构建

每次都手动的构建项目显然不够方便，有时候需要定时地执行自动化测试脚本。例如，每天晚上定时执行 `runtest.py` 文件来运行自动化测试项目，本书第 8 章已经介绍了如何集成 Python 的发邮件功能，这样第二天早上来到公司打开邮箱后就能看到自动化项目的执行情况。

定时执行 Python 脚本有很多方式，例如 Windows 系统自带的添加任务计划、Linux 系统的 `at` 命令和 `crontab` 命令都可以很灵活地设置定时任务。通过 Jenkins 实现定时任务的设

置只是其中的一种方法。

现在以已经创建的“pyse_test”项目为例，单击项目左侧的“配置”选项，修改项目的配置，如图 14.20 所示。



图 14.20 修改项目配置

在“构建触发器”中勾选“Build periodically”选项。

通过查看设置说明，此处定时任务的格式遵循 cron 的语法（可以与 cron 的语法有轻微的差异）。具体格式，每行包含五个字段，通过 Tab 或空格分隔，如表 14.1 所示。

表 14.1 cron 语法格式

字 段	说 明
MINUTE	Minutes within the hour (0–59)
HOURL	The hour of the day (0–23)
DOM	The day of the month (1–31)
MONTH	The month (1–12)
DOW	The day of the week (0–7)where 0 and 7 are Sunday.

在配置定时任务之前，我们先来了解 Linux 下 crontab 命令的格式，如图 14.21 所示。



图 14.21 crontab 格式说明

在以上各个字段中，还可以使用以下特殊字符。

星号 (*)：代表所有可能的值。例如，month 字段如果是星号，则表示在满足其他字段的制约条件后每月都执行该命令操作。

逗号 (,)：可以用逗号隔开的值指定一个列表范围。例如，“1,2,5,7,8,9”。

中杠 (-)：可以用整数之间的中杠表示一个整数范围。例如，“2-6”表示“2,3,4,5,6”。

正斜线 (/)：可以用正斜线指定时间的间隔频率。例如，“0-23/2”表示每两小时执行一次。同时正斜线可以和星号一起使用，例如，*/10，如果用在 minute 字段，表示每十分钟执行一次。

实例

假如，周一至周五每天早上 4 点跑自动化测试用例：

m	H	Dom	Mon	Dow	command
分钟	小时	天	月	星期	命令/脚本
*	4	*	*	1-5	python /home/pyse/test.py

假如，每周一和周三下午 6 点半运行自动化测试用例：

m	H	Dom	Mon	Dow	command
分钟	小时	天	月	星期	命令/脚本
30	18	*	*	1, 3	python /home/pyse/test.py

回到 Jenkins 的定时任务设置上，在“Build periodically”选项中只需设置脚本的执行日期和时间。如图 14.22 所示。



图 14.22 设置定时任务

—10 22 ** 1-5

表示每周一至周五的晚上 22:10 执行构建。

本章小结

本章只是带领读者对 Jenkins 有个初步的了解，它还有很多功能值得我们学习与探讨。比如与 CVS 和 SVN 等版本控制工具集合，使版本工具发生更新时进行项目的构建；再比如 Jenkins 分布式构建，类似于 Selenium Grid 的功能。Jenkins 同样可以配置 Master/Slave 架构，Master 提供 Web 接口让用户来管理 job 和 slave，job 可以运行在 master 本机或者被分配到 slave 上运行，一个 master 可以关联多个 slave 为不同的 job 或相同的 job 的不同配置服务。