



第 11 章

使用 Hudson 进行持续集成

本章内容

- ☐ 持续集成的作用、过程和优势
- ☐ Hudson 简介
- ☐ 安装 Hudson
- ☐ 准备 Subversion 仓库
- ☐ Hudson 的基本系统设置
- ☐ 创建 Hudson 任务
- ☐ 监视 Hudson 任务状态
- ☐ Hudson 用户管理
- ☐ 邮件反馈
- ☐ Hudson 工作目录
- ☐ 小结

作为最核心的敏捷实践之一——持续集成（Continuous Integration）越来越受到广大开发人员的喜爱和推崇。借助前文讲述的 Maven 所实现的自动化构建正是持续集成的一个必要前提，持续集成还要求开发人员使用版本控制工具和持续集成服务器。例如 Subversion 就是当前最流行的版本控制工具，而 Hudson 则是最流行的开源持续集成服务器软件。本章将简要介绍持续集成的概念和 Subversion 的基本使用，主要关注如何使用 Hudson，尤其是如何结合 Maven 与 Hudson 持续集成我们的项目。

11.1 持续集成的作用、过程和优势

简单地讲，持续集成就是快速且高频率地自动构建项目的所有源码，并为项目成员提供丰富的反馈信息。这句话有很多关键词：

- ❑ **快速：**集成的速度要尽可能地快，开发人员不希望自己的代码提交半天之后才得到反馈。
- ❑ **高频率：**频率越高越好，例如每隔一小时就是个不错的选择，这样问题才能尽早地被反映出来。
- ❑ **自动：**持续集成应该是自动触发并执行的，不应该有手工参与。
- ❑ **构建：**包括编译、测试、审查、打包、部署等工作。
- ❑ **所有源码：**所有团队成员提交到代码库里的最新的源代码。
- ❑ **反馈：**持续集成应该通过各种快捷的方式告诉团队成员最新的集成状态，当集成失败的时候，反馈报告应该尽可能地反映失败的具体细节。

一个典型的持续集成场景是这样的：开发人员对代码做了一些修改，在本地运行构建并确认无误之后，将更改提交到代码库。具有高配置硬件的持续集成服务器每隔 30 分钟查询代码库一次，发现更新之后，签出所有最新的源代码，然后调用自动化构建工具（如 Maven）构建项目，该过程包括编译、测试、审查、打包和部署等。然而不幸的是，另外一名开发人员在这一时间段也提交了代码更改，两处更改导致了某些测试的失败，持续集成服务器基于这些失败的测试创建一个报告，并自动发送给相关开发人员。开发人员收到报告后，立即着手调查原因，并尽快修复。

图 11-1 形象地展示了整个持续集成的过程。

通过图 11-1 可知，当持续集成服务器构建项目成功后，还可以自动将项目构件部署到 Nexus 私服中。

一次完整的集成往往会包括以下 6 个步骤：

1) **持续编译：**所有正式的源代码都应该提交到源码控制系统中（如 Subversion），持续集成服务器按一定频率检查源码控制系统，如果有新的代码，就触发一次集成，旧的已编译的字节码应当全部清除，然后服务器编译所有最新的源码。

2) **持续数据库集成：**在很多项目中，源代码不仅仅指 Java 代码，还包括了数据库 SQL 脚本，如果单独管理它们，很容易造成与项目其他代码的不一致，并造成混乱。持续集成也应该包括数据库的集成，每次发现新的 SQL 脚本，就应该清理集成环境的数据库，重新创建表结构，并填入预备的数据。这样就能随时发现脚本的错误，此外，基于这些脚本的

测试还能进一步发现其他相关的问题。

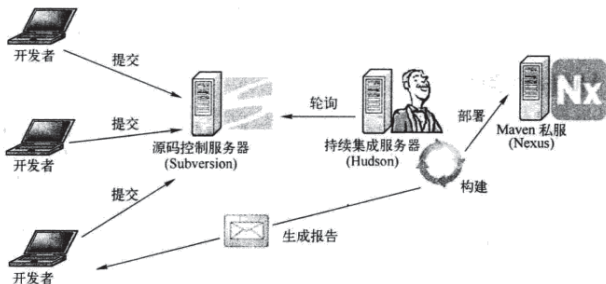


图 11-1 持续集成流程

3) **持续测试**：有了 JUnit 之类的框架，自动化测试就成了可能。编写优良的单元测试并不容易，好的单元测试必须是自动化的、可重复执行的、不依赖于环境的，并且能够自我检查的。除了单元测试，有些项目还会包含一些依赖外部环境的集成测试。所有这些测试都应该在每次集成的时候运行，并且在发生问题的时候能产生具体报告。

4) **持续审查**：诸如 Checkstyle 和 PMD 之类的工具能够帮我们发现代码中的坏味道 (Bad Smell)，持续集成可以使用这些工具来生成各类报告，如测试覆盖率报告、Checkstyle 报告、PMD 报告等。这些报告的生成频率可以低一些，如每日生成一次，当审查发现问题的时候，可以给开发人员反馈警告信息。

5) **持续部署**：有些错误只有在部署后才能被发现，它们往往是具体容器或者环境相关的，自动化部署能够帮助我们尽快发现这类问题。

6) **持续反馈**：持续集成的最后一步的反馈，通常是一封电子邮件。在重要的时候将正确的信息发送给正确的人。如果开发者一直受到与自己无关的持续集成报告，他慢慢地就会忽略这些报告。基本的规则是：将集成失败报告发送给这次集成相关的代码提交者，项目经理应该收到所有失败报告。

持续集成需要引入额外的硬件设置，特别是对于持续集成服务器来说，性能越高，集成的速度就越快，反馈的速度也就越快。持续集成还要求开发者使用各种工具，如源码控制工具、自动化构建工具、自动化测试工具、持续集成软件等。这一切无疑都增加了开发人员的负担，然而学习并适应这些工具及流程是完全值得的，因为持续集成有着很多好处：

- **尽早暴露问题**：越早地暴露问题，修复问题代码的成本就越低。持续集成高频率地编译、测试、审查、部署项目代码，能够快速地发现问题并及时反馈。
- **减少重复操作**：持续集成是完全自动化的，这就避免了大量重复的手工劳动，开发人员不再需要手动地去签出源码，一步步地编译、测试、审查、部署。
- **简化项目发布**：每日高频率的集成保证了项目随时都是可以部署运行的，如果没有持续集成，项目发布之前将不得不手动地集成，然后花大量精力修复集成问题。

□ 建立团队信心：一个优良的持续集成环境能让团队随时对项目的状态保持信心，因为项目的大部分问题区域已经由持续集成环境覆盖了。

既然持续集成有那么多优点，现在让我们开始动手架设自己的持续集成环境吧！

11.2 Hudson 简介

优秀的持续集成工具有很多，如老牌的开源工具 CruiseControl、商业的 Bamboo 和 Team-City 等。本书只介绍 Hudson，因为它是目前最流行的开源持续集成工具。该项目过去一直托管在 java.net 社区，不过现在已经迁移到 <http://hudson-ci.org/>。Hudson 主要是由 Kohsuke Kawaguchi 开发和维护的，Kohsuke Kawaguchi 自 2001 年就已经加入 Sun 公司（当然，现在已经是 Oracle 了），不过当笔者写下这些文字的时候，他刚宣布离开 Sun/Oracle 并开始基于 Hudson 自行创业。

Hudson 以其强大的功能和易用的界面征服了大量的用户，它与主流的构建工具、版本控制系统以及自动化测试框架都能进行很好的集成。因此，很多组织和公司选择它作为自己的持续集成工具，如 JBoss 的 <http://hudson.jboss.org/hudson/> 和 Sonatype 的 <https://grid.sonatype.org/ci/>。

Hudson 还有一个优秀之处就是它提供了灵活的插件扩展框架，大量开发者基于这种机制对 Hudson 进行了扩展。图 11-2 展示了 2006~2009 年 Hudson 插件数量的增长情况，其中

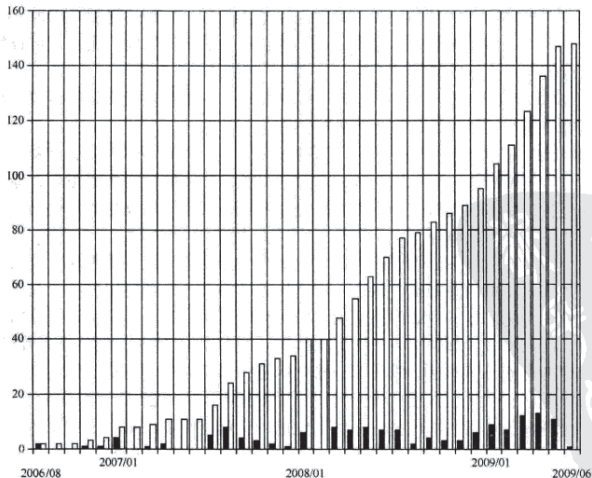


图 11-2 Hudson 插件数量的增长情况

黑柱表示当月新发布 Hudson 插件，白柱表示当月 Hudson 插件的总数量。该图十分显著地展现了 Hudson 插件生态系统的健康状况。

11.3 安装 Hudson

安装 Hudson 是十分简便的。需要注意的是，Hudson 必须运行在 JRE 1.5 或更高的版本上。可以从 <http://hudson-ci.org/> 下载最新版本的安装包，如图 11-3 所示。下载完成之后就能获得一个 hudson.war 文件。

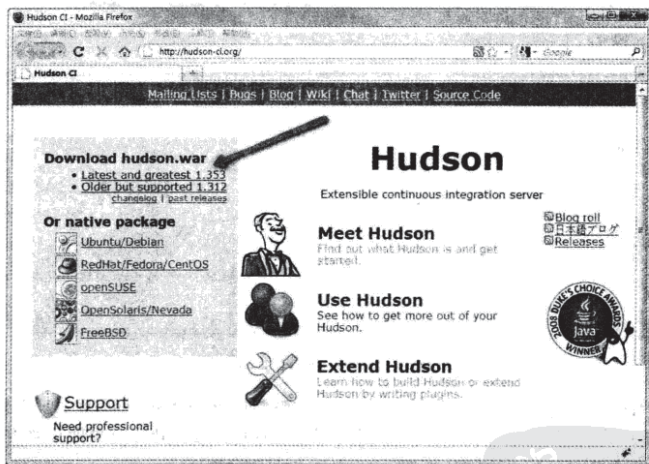


图 11-3 下载 Hudson 安装包

最简单的启动 Hudson 的方式是在命令行直接运行 hudson.war:

```
$ java -jar hudson.war
```

Hudson 的启动日志会直接输出到命令行，待启动完成之后，用户就可以打开浏览器输入地址 <http://localhost:8080/> 访问 Hudson 的界面了，如图 11-4 所示。

要停止 Hudson，可以在命令行按下 Ctrl + C 键。

默认情况下 Hudson 会在端口 8080 下运行，这可能会与用户已有的 Web 应用相冲突。这时，用户可以使用 `--httpPort` 选项指定 Hudson 的运行端口。例如：

```
$ java -jar hudson.war --httpPort=8082
```

既然安装包是一个 war 文件，Hudson 自然也就可以被部署到各种 Web 容器中，如 Tom-

cat、Glassfish、Jetty 及 JBoss 等。

这里以 Tomcat 6 为例，假设 Tomcat 的安装目录为 D:\bin\apache-tomcat-6.0.20\，那么只需要复制 hudson.war 至 Tomcat 的部署目录 D:\bin\apache-tomcat-6.0.20\webapps，然后转到 D:\bin\apache-tomcat-6.0.20\bin\目录，运行 startup.bat。这时可以从 Tomcat 的 console 输出中看到它部署 hudson.war。

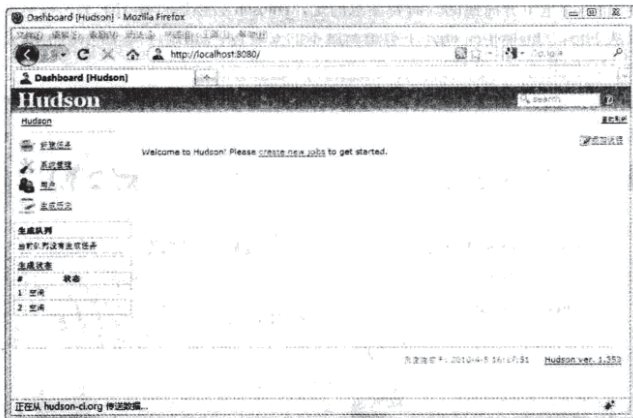


图 11-4 Hudson 的初始启动界面

待 Tomcat 启动完成之后，打开浏览器访问 <http://localhost:8080/hudson> 就能看到 Hudson 的界面了。用户可以将 Tomcat 作为一个系统服务运行，这样 Hudson 就能自动随操作系统一起启动了。

11.4 准备 Subversion 仓库

在正式创建 Hudson 持续集成任务之前，需要准备好版本控制系统。常见的版本控制工具有 CVS、Subversion、Git、Mercurial 等。由于 Subversion 可能是当前使用范围最广的版本控制工具，因此本书以它为例进行介绍。

首先需要安装 Subversion 服务器软件（本书仅讨论 svnserve）。对于大多数 Linux 发行版和 Mac OS X 来说，该工具应该已经被预先安装了。可以运行如下的命令查看，见代码清单 11-1。

代码清单 11-1 在 Linux/Mac OS X 中检查 svnserve 安装

```
$ svnserve --version
svnserve, version 1.5.5 (r34862)
```

compiled Dec 23 2008, 16:20:31

Copyright (C) 2000-2008 CollabNet.

Subversion is open source software, see <http://subversion.tigris.org/>

This product includes software developed by CollabNet (<http://www.Collab.Net/>).

The following repository back-end (FS) modules are available:

- * fs_base : Module for working with a Berkeley DB repository.
- * fs_fs : Module for working with a plain file (FSFS) repository.

Cyrus SASL authentication is available.

对于 Windows 用户来说, 可以安装 Slik Subversion (<http://www.sliksvn.com/en/download>)。需要注意的是, 在选择安装类型的时候, 需要选择 complete 安装, 否则默认的安装方式将不会安装 svnserve, 如图 11-5 所示。

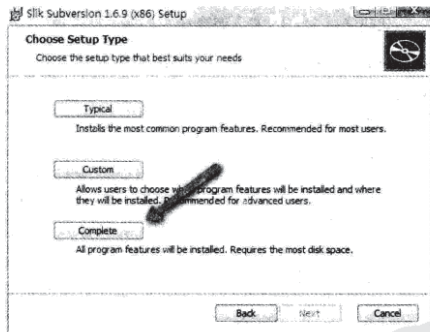


图 11-5 完整安装 Slik Subversion

安装完成之后, 可以运行如下命令进行验证, 见代码清单 11-2。

代码清单 11-2 在 Windows 中检查 svnserve 安装

```
D:\>svnserve -v -version
```

```
svnserve, 版本 1.6.2 (SlikSvn:tag/1.6.2@ 37679) WIN32
```

```
编译于 May 11 2009, 14:06:15
```

版权所有 (C) 2000-2009 CollabNet.

Subversion 是开放源代码软件, 请参阅 <http://subversion.tigris.org/> 站点。

此产品包含由 CollabNet (<http://www.Collab.Net/>) 开发的软件。

下列版本库后端 (FS) 模块可用:

- * fs_base : 模块只能操作 BDB 版本库。
- * fs_fs : 模块与文本文件 (FSFS) 版本库一起工作。

Cyrus SASL 认证可用。

接着需要创建一个 Subversion 仓库。运行命令如下：

```
D:\>mkdir svn-repos
```

```
D:\>svnadmin create svn-repos\account
```

svnadmin 是用来创建、维护、监测 Subversion 仓库的工具，在主流 Linux 和 Mac OS X 上一般都是预装的。在 Windows 上，它也被包含在 Slik Subversion 中。这里首先创建一个名为 svn-repos 的目录，然后在这个目录中创建一个 Subversion 仓库。

下一步是将本书背景案例现有的代码导入到这个 Subversion 仓库中。由于笔者的代码和 Subversion 仓库在一台机器上，因此直接使用 file 协议导入（导入之前应先使用 mvn clean 命令清除项目输出文件，这些文件是可以自动生成的，不该放入源码库中），见代码清单 11-3。

代码清单 11-3 导入源码至 Subversion 仓库

```
$ svn import -m "initial import" . file:///D:/svn-repos/account/trunk
增加      account-email
增加      account-email\src
增加      account-email\src\test
增加      account-email\src\test\java
增加      account-email\src\test\java\com
增加      account-email\src\test\java\com\juvenxu
增加      account-email\src\test\java\com\juvenxu\mvnbook
增加      account-email\src\test\java\com\juvenxu\mvnbook\account
增加      account-email\src\test\java\com\juvenxu\mvnbook\account\email
...
...
增加      account-captcha\pom.xml
```

提交后的版本为 1。

上述命令将当前目录的全部内容提交到 Subversion 仓库的 /account/trunk 路径下，-m 选项表示提交的注释。

仓库建立并初始化完毕，就可以启动 svnserve 服务了：

```
$ svnserve -d -r svn-repos --listen-host 0.0.0.0
```

选项 -d 表示将 svnserve 服务作为守护进程运行，-r 表示 Subversion 仓库的位置，而参数 --listen-host 是为了强制将 svnserve 绑定到 IP v4 地址（在有些系统上，svnserve 会默认绑定 IP v6 地址，当 Hudson 使用 IP v4 地址访问 Subversion 仓库的时候就会失败）。

最后，可以用简单的 svn 命令检查插件 svnserve 服务是否可用，见代码清单 11-4。

代码清单 11-4 检查 Subversion 仓库内容

```
$ svn list svn://192.168.1.101/account/trunk
.classpath
```



```
.project
.settings/
account-captcha/
account-email/
account-parent/
account-persist/
pom.xml
```

至此，Subversion 仓库就建立完成了，之后 Hudson 就可以基于这个仓库运行集成任务。

11.5 Hudson 的基本系统设置

在创建 Hudson 持续集成任务之前，用户需要对 Hudson 系统做一些基本的配置，包括 JDK 安装位置和 Maven 安装等在内的重要信息都必须首先配置正确。Hudson 会使用这些配置好的 JDK 及 Maven 进行持续集成任务。如果要使用 Ant 或者 Shell 来持续集成项目，Ant 或 Shell 的安装位置也应该预先设置正确。

用户应该单击 Hudson 登录页面左边的“系统管理”，然后单击页面右侧的“系统设置”以进入系统设置页面，如图 11-6 所示。



图 11-6 进入系统设置页面

在系统设置页面，首先要配置的是 Hudson 将使用的 JDK。在页面中找到对应的部分，然后单击 Add JDK 按钮，Hudson 就会提示用户进行安装。Hudson 默认会提示自动安装 JDK，用户可以看到一个 Install automatically 的复选框是被选上的，当单击“同意 JDK 许可证协议”并选择一个 JDK 版本后，Hudson 就会自动下载安装相应版本的 JDK。

虽然这种方式非常简单，但往往用户在本机已经有可用的 JDK，而且不想花时间去等待 Hudson 去再次下载 JDK。这时用户就可以取消选中 Install automatically 复选框，然后手动输入本机 JDK 的位置（往往就是 JAVA_HOME 环境变量的值）。

可以配置多个 JDK，当你的项目需要确保在多个不同版本 JDK 上都能正确集成的时候，

这一特性尤为有用。

JDK 的自动及手动配置方式如图 11-7 所示。

与 JDK 配置类似，用户也可以选择手动或者自动安装 Maven 供 Hudson 使用，还可以安装多个版本的 Maven 供 Hudson 集成任务使用。图 11-8 显示了手动方式指定 maven-3.0-beta-2 的安装位置。

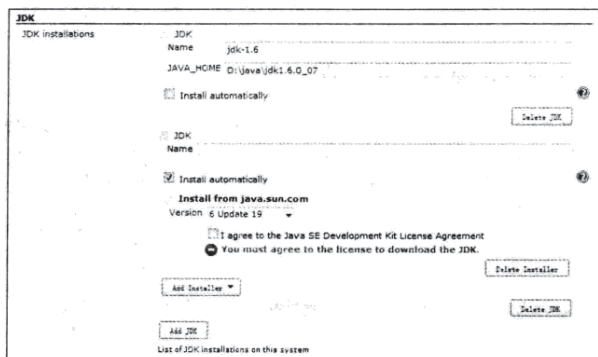


图 11-7 为 Hudson 配置 JDK

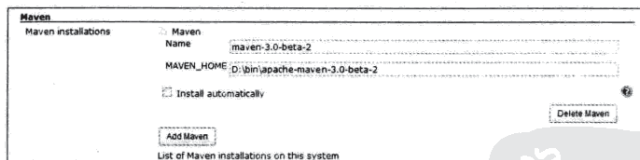


图 11-8 为 Hudson 配置 Maven

还可以在该页面配置 MAVEN_OPTS 环境变量，如图 11-9 所示。关于 MAVEN_OPTS 环境变量的具体解释可参看 2.7.1 节。

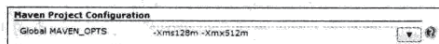


图 11-9 为 Hudson 配置 MAVEN_OPTS 环境变量

最后，别忘了单击页面下方的 Save 按钮保存系统设置。

11.6 创建 Hudson 任务

要创建一个 Hudson 任务来持续集成 Maven 项目，首先单击页面左边的新建任务，然后就需要在页面右边选择任务的名称及类型。对于一般的 Maven 项目来说，可选择类型有 Build a free-style software project 和 Build a maven2 project。前者不仅支持 Maven 项目，还支持其他类型的构建工具，如 Ant、Shell。对于 Maven 用户来说，两者最大的不同在于前者需要用户进行多一点的配置，而后者会使用 Hudson 自带的 Maven，且从项目的 POM 中获取足够的信息以免去一些配置。除非你已经十分熟悉 Hudson，笔者推荐选择 free-style 类型（见图 11-10）。因为这种方式更可控制，当任务出现问题的时候也更容易检查。

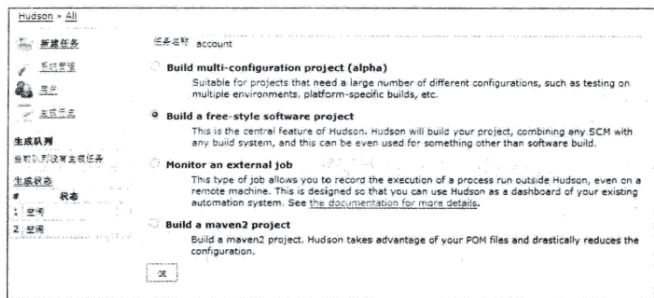


图 11-10 选择 free-style 类型的 Hudson 任务

如图 11-10 所示，输入任务名称，并选择 free-style 类型后，单击 OK 按钮即可进入详细的任务配置页面。

11.6.1 Hudson 任务的基本配置

下面依次介绍 free-style 任务的各种配置。首先是项目的名称和描述。当 Hudson 任务比较多的时候，简洁且有意义的名称及描述就十分重要。

接着是一个重要的选项 Discard Old Builds。该选项配置如何抛弃旧的构建。Hudson 每执行一次构建任务，就可以保存相应的源代码、构建输出、构建报告等文件。很显然，如果每次构建相关的文件都保存下来，将会渐渐消耗光磁盘空间。为此，Hudson 提供两种方式让用户选择保留哪些构建任务的相关文件，它们分别为：

- ❑ **Days to keep builds**：如果其值为非空的 N，就仅保留 N 天之内的构建文件。
- ❑ **Max # of builds to keep**：如果 # 非空，就仅保留最多 # 个最近构建的相关文件。

图 11-11 所示的配置表示最多保留 10 个最近的构建。

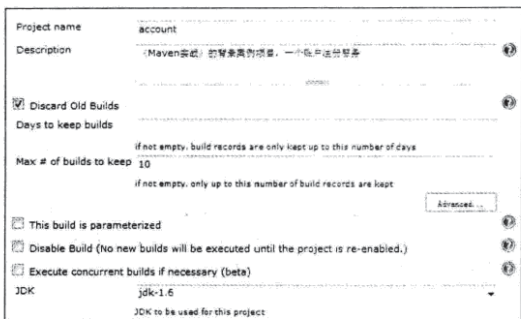


图 11-11 Hudson 任务的基本配置

图 11-11 中还有项目使用的 JDK 配置, 这里可供选择的 JDK 就是用户在系统设置中预先定义好的 JDK。

11.6.2 Hudson 任务的源码仓库配置

接着需要配置项目的源码控制系统。在项目配置页面的 Source Code Management 部分, 选择 Subversion 单选按钮, 然后在 Repository URL 文本框中输入项目的 Subversion 仓库地址。一般来说, 该部分的其他选项保留默认值即可, 如图 11-12 所示。

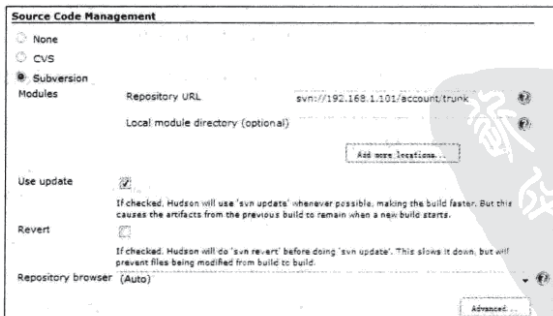


图 11-12 Hudson 任务的源代码管理配置

需要注意的是, 如果访问 Subversion 仓库需要认证, Hudson 会自动探测并提示用户输入认证信息, 如图 11-13 所示。

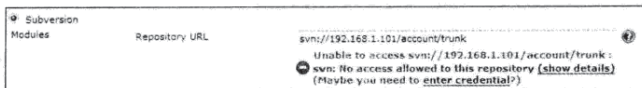


图 11-13 Hudson 提示用户输入源码仓库的认证信息

单击 enter credential 后, Hudson 会弹出一个页面让用户选择认证方式并输入认证信息。输入正确信息之后, Hudson 就能读取仓库源代码了。图 11-14 采用了用户名和密码的方式进行认证。

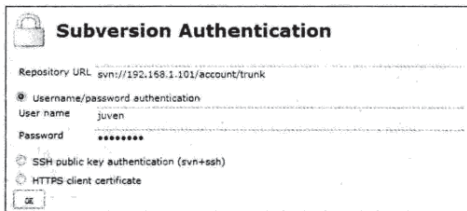


图 11-14 为 Subversion 仓库添加认证信息

11.6.3 Hudson 任务的构建触发配置

再往下的 Build Triggers 部分配置的是触发构建的方式。可选的三种方式分别为:

- ☐ Build after other projects are built: 在其他项目构建完成之后构建本项目。
- ☐ Build periodically: 周期性地构建本项目。
- ☐ Poll SCM: 周期性地轮询源码仓库, 发现有更新的时候构建本项目。

如无特殊高级的需要, 一般不会选择第一种方式; 而第二种方式显然会造成一些无谓的构建, 如果几次构建所基于的源代码没有任何区别, 构建的输出往往也就不会有变化; 第三种方式就没有这个问题, 它能避免无谓的构建, 节省持续集成服务器的资源。这种周期轮询源代码仓库的方式实际上也是最常用的构建触发方式。

既然是轮询, 就需要配置轮询的频率, Hudson 使用了著名的 UNIX 任务调度工具 Cron (<http://en.wikipedia.org/wiki/Cron>) 所使用的配置方式。这种配置方式使用 5 个字段表示不同的时间单位 (字段之间用空格或制表符分隔):

分 时 日 月 星期几

每个字段表示的意义及值范围分别为:

- ☐ 分: 一小时中的分钟 (0 ~ 59)。
- ☐ 时: 一天中的小时 (0 ~ 23)。
- ☐ 日: 一月中的日期 (1 ~ 31)。

- ☐ 月：月份（1~12）。
- ☐ 星期几：一周中的星期几（0~7，0 和 7 都表示星期天）。其中每个字段除了可以使用其范围内的值以外，还能使用一些特殊的字符：
- ☐ *：星号表示匹配范围内所有值。
- ☐ M-N：连字符表示匹配 M~N 范围内的所有值，如“1-5”。
- ☐ A, B, ..., Z：逗号表示匹配多个值，如“0, 15, 0”。
- ☐ */X 或 M-N/X：范围加上斜杠表示匹配范围内能被 X 整除的值，如“1-10/3”就等同于“3, 6, 9”。

下面一些例子可以帮助读者理解这种强大的配置方式：

- ☐ ****：每分钟。
 - ☐ 5****：每小时中的第 5 分钟。
 - ☐ */10****：每隔 10 分钟。
 - ☐ 45 10** 1-5：每周一到周五的上午 10:45。
 - ☐ 0,30 *13*5：每月 13 号的每半小时，或者每周五的每半小时。
- 对于一个健康的项目来说，常见的做法是：每隔 10 分钟轮询代码仓库，如图 11-15 所示。



图 11-15 Hudson 任务的代码仓库轮询配置

在配置轮询的时候，还可以使用“#”添加注释，此外空白的行会被忽略。例如：

```
# check if there is any subversion update every 15 minutes
*/15 ****
```

11.6.4 Hudson 任务的构建配置

接下来要告诉 Hudson 使用运行 Maven 命令构建项目。单击 Build 部分中的 Add build step 下三角按钮，然后选择 Invoke top-level Maven targets，如图 11-16 所示。

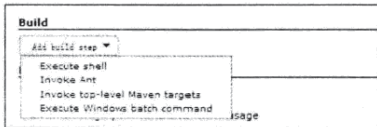


图 11-16 选择 Maven 作为 Hudson 任务的构建工具

再选择一个安装好的 Maven 版本, 输入 Maven 命令如 clean deploy 就可以了, 如图 11-17 所示。需要注意的是, 日常持续集成任务如果成功的话, 都会生成快照版的项目构件。如果维护了一个 Maven 私服, 那么持续集成任务就应当自动将构件部署到私服中, 供其他项目使用。这也就是这里的 Maven 命令应当为 clean deploy 的原因。

至此, 一个 Hudson 任务基本配置完成, 单击 Save 按钮保存后就能看到图 11-18 所示的页面。这时, 可以单击页面左边的“立即生成”来手动触发第一次集成。

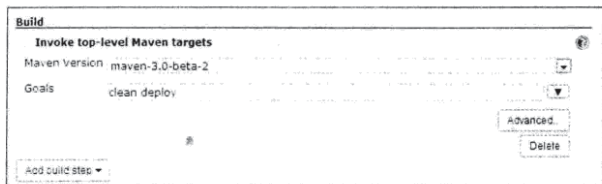


图 11-17 Hudson 任务的 Maven 构建命令配置



图 11-18 配置完成的 Hudson 任务

11.7 监视 Hudson 任务状态

Hudson 提供了丰富友好的图形化界面, 让用户从各方面了解各个任务的当前及历史状态, 这包括整体的列表显示、自定义视图、单个任务的具体信息, 如构建日志和测试报告等。用户应该基于 Hudson 提供的信息尽可能地将持续集成任务稳定在健康的状态。

11.7.1 全局任务状态

Hudson 的默认主页面显示了当前服务器上所有集成任务的状态, 如图 11-19 所示。

这个页面主要由四个部分组成：

- ❑ **导航菜单**：位于页面左上方，方便用户执行各类 Hudson 操作，如新建任务、系统管理等。
- ❑ **生成队列**：页面左边中间的部分，表示等待执行构建的任务，如图 11-19 中有一个 maven3 的构建任务在等待生成队列中。
- ❑ **生成状态**：页面左边下面的部分，表示正在执行构建的任务，如图 11-19 中有一个 account 的构建任务正在执行。
- ❑ **任务状态**：页面右边的部分，显示了所有任务的状态。

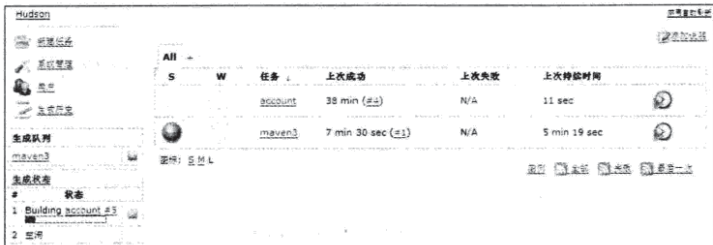


图 11-19 Hudson 的全局任务状态

下面重点介绍任务状态。在默认情况下，这里列出了 Hudson 中所有任务的状态，其中的每一列从左到右分别表示任务当前状态、天气，名称、上次成功的时间、上次失败的时间、上次持续的时间以及左右一个立即执行的按钮（方便用户手动触发执行任务）。

其中需要解释的是当前状态及图中第一列（S）下的球形图标。Hudson 使用各种颜色表示任务当前的状态：

- ❑ **蓝色**：任务最近一次的构建是成功的。
- ❑ **红色**：任务最近一次的构建是失败的。
- ❑ **黄色**：任务最近一次的构建表成功了，但不稳定（主要是因为有失败的测试）。
- ❑ **灰色**：任务从未被执行过或者被禁用了。

如果图标在闪烁，表示任务正在执行一次构建。

图中的第二列天气（W）也需要稍作解释。Hudson 使用一组天气的图标表示任务长期的一个状态，它们分别为：

☀️ 万里晴空，任务 80% 以上的集成都是成功的。

☁️ 稍有乌云，任务有 60% ~ 80% 的集成是成功的。

☁️ 乌云密布，任务只有 40% ~ 60% 的集成是成功的。



阴雨绵绵，任务的集成成功率只有 20% ~ 40%。



电闪雷鸣，任务的集成成功率不到 20%。

关于全局状态需要再次强调的是，当团队看到任务的集成状态不够健康时，应该尽快采取措施修复问题。

11.7.2 自定义任务视图

在一个稍有规模的公司或者组织下，持续集成服务器上往往会有很多的任务，Hudson 默认的视图会列出所有服务器上的任务，太多的任务就会造成寻找的不便。为此 Hudson 能让用户自定义视图，选择只列出感兴趣的任务，甚至还能自定义视图中显示的列。

用户可以单击默认视图 All 旁边的加号 (+) 以添加一个自定义视图，如图 11-20 所示。

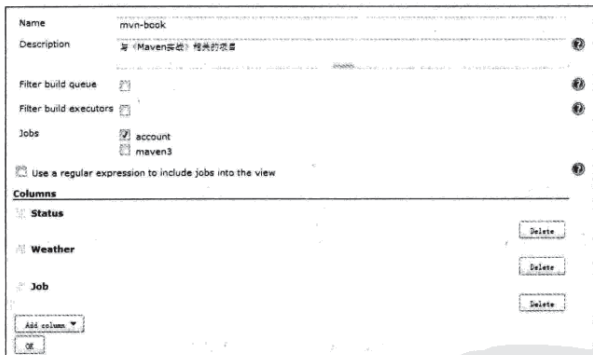


图 11-20 添加自定义 Hudson 任务视图

图 11-20 添加了一个名为 mvn-book 的任务视图，该视图仅包含 account 一个任务，并且只显示状态、天气、任务名三列。用户可以根据自己的需要，选择要包含的任务和要显示的列，甚至还能使用正则表达式来匹配要显示的任务名。上述配置保存后的效果如图 11-21 所示。



图 11-21 自定义 Hudson 任务视图效果

11.7.3 单个任务状态

在任务视图中，单击某个任务名称就能进一步查看该任务的状态。图 11-22 显示了 account 项目任务的一个整体状态。

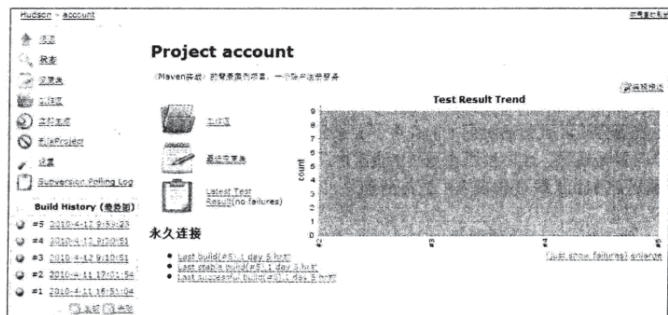


图 11-22 单个 Hudson 任务的状态

图 11-22 包含了丰富的信息。左下角是构建历史 (Build History)，该例中显示了最近 5 次全部成功的构建，包括每次构建的时间。图 11-22 下方还有 3 个永久连接，分别指向了最近一次构建、最近一次失败的构建以及最近一次成功的构建。无论构建历史还是永久连接，我们都能单击某一个构建以了解更具体的信息。例如，单击图 11-22 构建历史中的 #4 构建，就可以看到图 11-23 所示的内容。

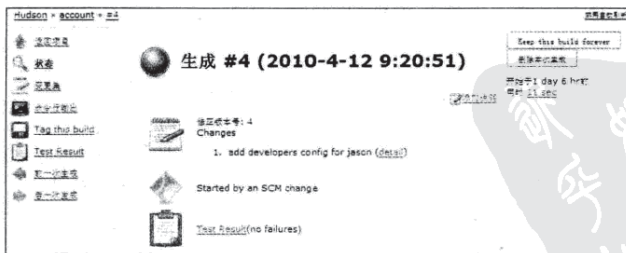


图 11-23 Hudson 任务的单次构建信息

请注意图 11-23 左上角的导航信息，Hudson > account > #4 表示当前的位置是 Hudson 服务器下 account 任务的第 4 次构建。从图 11-23 中可以了解到这次构建所发生的时间、相

关的代码变更等信息。

需要指出的是,在图 11-23 中左边的命令行输出链接。当构建失败的时候,了解这次构建的命令行输入至关重要。单击该链接后可以看到图 11-24 所示的页面。



图 11-24 Hudson 执行构建的命令行输出

在图 11-22 中还有一些链接包含了丰富的信息,例如最近变更集。单击该链接就能看到项目最近的代码变更,如图 11-25 所示。

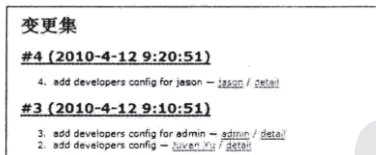


图 11-25 Hudson 任务的变更集

除了变更集,还可以单击工作区,以图形化的方式查看该 Hudson 从源代码库取得的源代码文件及构建输入文件,如图 11-26 所示。

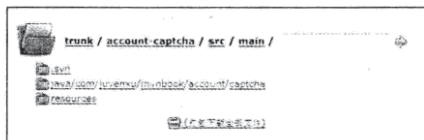


图 11-26 Hudson 任务的工作区

11.7.4 Maven 项目测试报告

图 11-22 还显示了项目的测试结果信息，为了获得这样的信息需要做一些额外的配置。在 11.6.1 节中，maven-surefire-plugin 会在项目的 target/surefire-reports 目录下生成与 JUnit 兼容的 XML 格式测试报告，Hudson 能够基于这种格式的文件生成图形化的测试报告。

用户可以配置一个 Hudson 任务，在配置页面的 Post-build Actions 部分选择 Publish JUnit test result report 选项，并且将 Test report XMLs 赋值为 `**/target/surefire-reports/TEST-*.xml`。

该表达式表示匹配任意目录下 target/surefire-reports/子目录中以 TEST-开头的 XML 文件，这也就是匹配所有 maven-surefire-plugin 生成的 XML 格式报告文件。配置如图 11-27 所示。

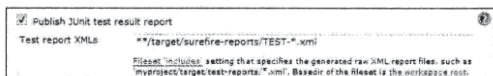


图 11-27 配置 Hudson 任务发布测试报告

有了上述配置之后，就能在任务状态页面中看到最新的测试结果与测试结果趋势，如图 11-28 所示。

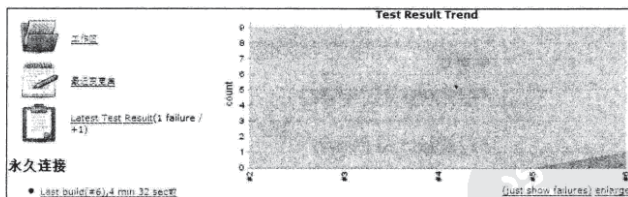


图 11-28 Hudson 任务测试的总体状态

单击 Latest Test Result 就能看到最近一次构建的测试报告。在测试结果趋势图中，用户也可以单击各个位置得到对应构建的测试报告，如图 11-29 所示。

用户还可以单击图中的链接得到更具体的测试输出，以方便定位并修复问题。

如果用户为一个 Hudson 任务配置了测试报告，就可以同时配置构建命令忽略测试。例如，图 11-17 中的 Maven 构建命令可以更改为 `clean deploy -Dmaven.test.failure.ignore`，这样失败的测试就不会导致构建失败。也就是说，构建的状态不会是红色，同时，由于 Hudson 能够解析测试报告并发现失败的测试，构建的状态也不会是健康的蓝色。用户最终会看到黄色的任务状态，表示构建不稳定。这种配置方式能够帮助用户区分失败的构建与不稳定的构建。

Test Result

1 failures (+1)

9 tests (#0)

Took 0.6 sec

All Failed Tests

Test Name	Duration	Age
>>> com.javenu.mybatis.account.catcha.AccountCatchaTest[com.javenu.mybatis.account.catcha.AccountCatchaTest]	0.947	

All Tests

Package	Duration	Fail	(diff)	Skip	(diff)	Total	(diff)
com.javenu.mybatis.account.catcha	2.7 sec	1	+1	0		4	
com.javenu.mybatis.account.email	0.47 sec	0		0		1	
com.javenu.mybatis.account.parent	0.48 sec	0		0		4	

图 11-29 一次构建的测试报告

11.8 Hudson 用户管理

与一般软件的用户管理方式不同的是，使用 Hudson 时，不需要主动创建用户，Hudson 能够在访问源码仓库的时候自动获取相关用户信息并存储起来。这大大简化了用户管理的步骤。

以 11.4 节建立的 Subversion 仓库为例，默认该仓库是匿名可读的，认证用户可写，不过我们并没有配置任何用户。现在要关闭匿名可读权限，同时添加一些用户。本书不涉及过多的配置细节，可以参考《Subversion 与版本控制》（<http://svnbook.red-bean.com/>）一书。

首先，编辑 Subversion 仓库下 conf/svnserve.conf 文件中的 [general] 小节如下：

```
[general]
anon-access = none
auth-access = write
password-db = passwd
```

这里的 anon-access = none 表示匿名用户没有任何权限，auth-access = write 表示经认证用户拥有读写权限，而 password-db = passwd 表示存储用户信息的数据位于同级目录下的 passwd 文件中。再编辑 conf/passwd 文件如下：

```
[users]
admin = admin123
juven = juven123
jason = jason123
```

这里为仓库配置了三个用户，等号左边是用户名，右边则是密码。

至此，就完成了个简单的 Subversion 仓库用户权限配置。像日常开发一样，接下来在 Subversion 客户端分别使用这几个用户名对代码进行更改后提交至 Subversion 仓库。例如，对 account-parent 模块的 pom.xml 加入 developers 配置后，再使用如下 svn 命令提交更改：

```
D:\svn\account>svn commit -m "add developers config" - -username juven - -pass-
word juven123
正在发送 account-parent\pom.xml
传输文件数据.
提交后的版本为 2.
```

然后使用另外两个用户 admin 与 jason 分别对代码进行更改并提交，Hudson 会很快轮到 Subversion 仓库内的更改，然后取得更改的代码信息，并了解到这些更改是由谁提交的。

待 Hudson 得到这些更改并触发集成任务之后，相关的 Subversion 用户信息就已经被 Hudson 存储起来了。单击 Hudson 页面左边的用户，然后就能在页面右边看到相关的用户信息，包括用户名、最近活动时间及相关的 Hudson 任务，如图 11-30 所示。

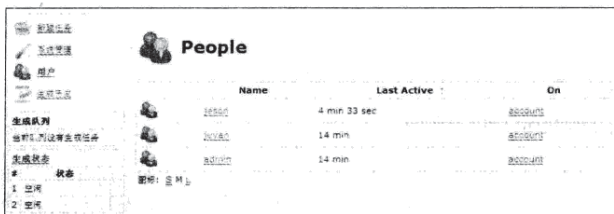


图 11-30 Hudson 自动获得的用户信息

当然，仅仅知道用户名是不够的，还需要为用户添加详细信息，其中最重要的就是 E-mail 地址，因为它将被用来发送邮件反馈（详见 11.9 节）。单击某个用户的名称（如 juven），然后再单击页面左边的设置，在右边的用户设置页面中，可以配置用户的名称（不同于 Subversion ID，该名称应该更容易识别人）、简要描述、个性化视图以及最重要的 E-mail 地址，如图 11-31 所示。

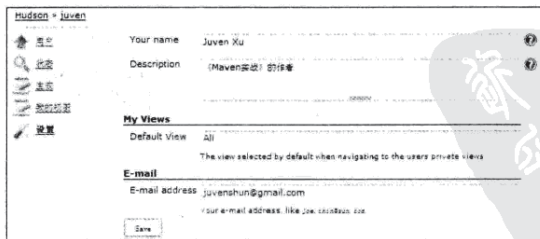


图 11-31 配置 Hudson 用户的详细信息

单击 Save 按钮后，一个 Hudson 用户的信息就完整了。

11.9 邮件反馈

持续集成中非常重要的一个步骤就是反馈。集成的状态信息（尤其是不健康的状态信息）必须及时地通知给相关团队成员，而最常见的反馈方式就是使用电子邮件。本小节介绍如何配置 Hudson 来及时地发送集成反馈邮件。

首先需要做的是为 Hudson 配置邮件服务器信息。进入 11.5 节提到的系统设置页面，找到 E-mail Notification 部分，然后输入以下信息：

- ❑ **SMTP server**：SMTP 邮件服务器地址。
- ❑ **Default user e-mail suffix**：默认用户邮件后缀。当用户没有配置邮件地址的时候，Hudson 会自动为其加上该邮件后缀。当用户数量很多，并且邮件地址都是一个域名的时候，该功能就显得尤其重要，例如配置后缀为@foo.com，且用户 mike 没有配置邮件地址，那么当 Hudson 需要发邮件给 mike 的时候就会发送到 mike@foo.com。
- ❑ **System Admin E-mail Address**：系统管理员邮件地址，即 Hudson 邮件提示所使用的发送地址。
- ❑ **Hudson URL**：Hudson 服务器的地址。该地址往往被包含在电子邮件中以方便用户访问 Hudson 取得进一步的信息，因此要确保该地址在用户机器上是可访问的。
- ❑ **SMTP Authentication**：SMTP 相关的认证配置。

完整的邮件服务器配置如图 11-32 所示。

图 11-32 Hudson 邮件服务配置

配置完成后，可以单击图 11-32 右下角的测试按钮，让 Hudson 发一封邮件至系统管理员邮件地址以确认配置成功。

接下来要做的是配置 Hudson 任务使用邮件反馈。进入任务的配置页面，然后找到最后 Post-build Actions 小节中的 E-mail Notification 复选框，将其选上。现在要关心的是两个问题：什么样的构建会触发邮件反馈？邮件会发送给谁？

关于第一个问题，答案是这样的：

- ☐ 失败的构建会触发邮件反馈。
- ☐ 成功构建后的一次不稳定构建会触发邮件反馈。不稳定往往是由失败的测试引起的，因此成功后的下一次不稳定往往表示有回归性测试失败。
- ☐ 失败或不稳定构建后的一次成功构建会触发邮件反馈，以通知用户集成恢复到了健康状态。
- ☐ 用户可以配置是否每次不稳定构建都触发邮件反馈。

关于第二个问题，首先可以在 Recipients 中配置一个邮件列表（用空格分离），列表中的用户会收到所有邮件反馈。一般来说，项目负责人应该在这个列表中。

其次，Hudson 还提供一选项：Send separate e-mails to individuals who broke the build。当用户选择该选项后，邮件会发送给所有与这次构建相关的成员，即那些提交了本地构建代码更新的成员。Hudson 无法精确地知道到底是谁的代码提交导致了构建失败，因此只能通知所有与代码更新相关的成员。

典型的邮件反馈配置如图 11-33 所示。

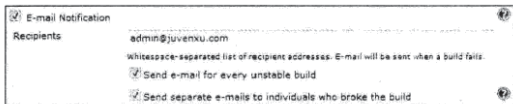


图 11-33 为 Hudson 任务配置邮件反馈

最后需要解释的是，图 11-33 中的 Send e-mail for every unstable build 选项表示是否为所有的不稳定构建触发邮件反馈，如果不将其选中，只有成功构建后的第一次不稳定构建才会触发邮件反馈。推荐的做法是将其选上。敏捷高效的团队不应该忽略持续集成中的任何不健康因素。

11.10 Hudson 工作目录

到目前为止，本章都是从用户界面的角度介绍 Hudson 的各种功能。用心的读者可以想象到，Hudson 的各种配置、任务、报告肯定是以文件的形式存储在磁盘中的。这就是 Hudson 的工作目录，了解该目录不仅能帮助读者理解 Hudson 用户界面中的各种特性，更重要的是，读者需要明白怎样为 Hudson 分配合理的磁盘空间，长期运行的持续集成服务往往会消耗大量的磁盘空间，理解哪些任务对应的哪些文件消耗了多少磁盘空间，对持续集成服务的维护来说至关重要。

默认情况下，Hudson 使用用户目录下的 .hudson/ 目录作为其工作目录。例如，在笔者的 Vista 系统上，该目录为 C:\Users\juven\.hudson\，而在 Linux 系统上，该目录为 /home/juven/.hudson/。由于该目录会渐渐消耗大量的磁盘空间，因此用户往往会希望自定义该工

作目录的位置,这时用户可以设置环境变量 HUDSON_HOME,例如将其设置为 D:\hudson-work。关于如何设置环境变量,请参考 2.1.3 节和 2.2.1 节。

一个典型的 Hudson 工作目录包含的内容如图 11-34 所示。

对这些文件、目录的解释如下:

- *.xml: 这些 XML 文件是 Hudson 核心及相关插件的配置,如 config.xml 配置了全局的 JDK、任务视图等信息, hudson.tasks.Maven.xml 配置了 Maven 安装信息, hudson.tasks.Mailer.xml 配置了邮件服务器信息,等等。

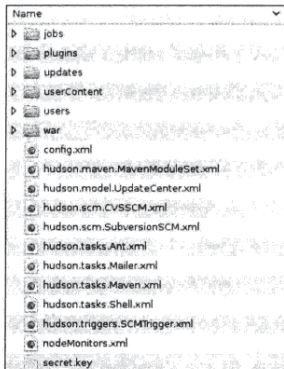


图 11-34 Hudson 工作目录的内容

- war: 如果用户独立运行 hudson.war, 那么其内容会被释放到该目录中后再启动。
- users: Hudson 所存储的用户信息。
- userContent: 用户可以将任意内容放到该目录下后通过 Hudson 服务页面的子路径访问, 如 <http://192.168.1.101:8080/userContent/>。
- updates: 这里存储了各类可更新的插件信息。
- plugins: 所有 Hudson 插件都被安装在该目录而不会影响到 Hudson 的核心。
- jobs: 该目录包含了所有 Hudson 任务的配置、存储的构建、归档的构建输出等内容。本节稍后会详细解释该目录。

上述目录中最重要的可能就是 jobs 子目录了, 这里包含了所有 Hudson 的任务配置、每个任务的工作区、构建历史等信息, 具体内容如图 11-35 所示。

图 11-35 中的 jobs 目录下有两个子目录 account 和 maven3, 它们分别对应了两个 Hudson 任务。每个任务都会包含如 config.xml、nextBuildNumber、scm-polling.log 等文件, 其中的 config.xml 包含了该任务的所有配置, 如 SCM 地址、轮询频率等。

每个任务目录下会包含一个 workspace 子目录, 这就是该任务的工作区。这里有最近一

次构建所包含的源代码及相关输出。

任务目录下还有一个 builds 子目录, 该目录包含了所有 Hudson 记录的历史构建, 每个构建对应了一个目录, 这些目录都是以构建所发生的时间命名的, 如 2010-04-15_14-56-08, 每个构建目录包含了一些文件记录其成功失败信息、构建日志、测试报告、变更记录等。如果用户为该任务配置了文件归档, 那么每次构建归档的内容都会存储在 archive 子目录下。

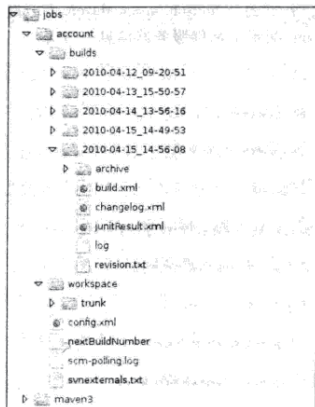


图 11-35 Hudson 工作目录的 jobs 子目录内容

可以想象, 如果用户没有如 11.6.1 节中介绍的那样抛弃旧的构建, 那么每次构建的记录都会保存在任务目录的 builds 子目录下。随着时间的推移, 这些记录会消耗大量的磁盘空间, 因此用户在使用 Hudson 的时候应该按照实际情况为其分配足够的磁盘空间, 同时合理地抛弃旧的构建记录。

11.11 小结

本章关注的是持续集成。首先介绍了持续集成相关的概念, 在此基础上, 再引入最流行的持续集成服务软件——Hudson。Hudson 非常易于安装, 它与主流的版本控制工具都集成得很好, 为了真实地体现持续集成场景, 本章简略介绍了如何架设简单的 Subversion 仓库。使用 Hudson 创建持续集成任务会涉及很多配置, 如 JDK 安装、Maven 安装、源码库、构建触发方式、构建命令等, 本章配以大量的图片以帮助读者使用和理解这些配置。任务创建完成后, 还能从各方面了解任务的状态, 包括成功与否、测试报告、源码变更记录等。Hudson 还能够智能地收集用户信息, 读者可以配置 Hudson 为用户提供邮件反馈。本章的最后介绍了 Hudson 的工作目录, 以帮助读者更好地维护持续集成服务。