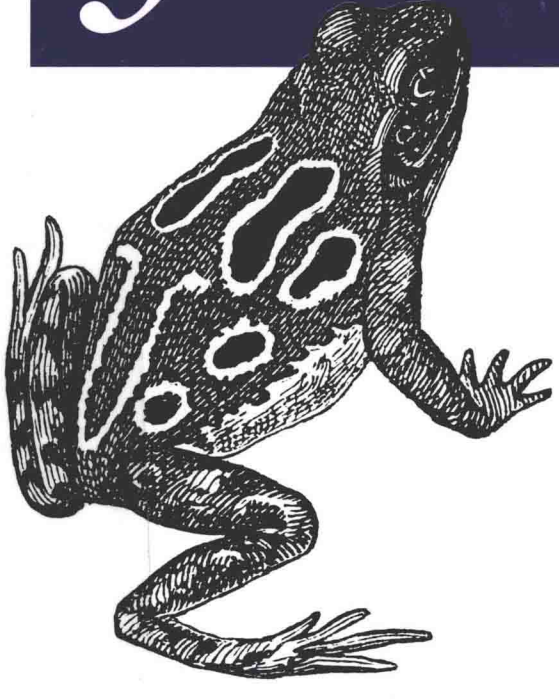
Jenkins: The Definitive Guide

涵盖Hudson



Jenkins

权威指南



Jenkins/Hudson创建者
Kohsuke Kawaguchi倾情作序

[美] *John Ferguson Smart* 著
郝树伟 于振苓 熊熠 译

O'REILLY®
中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Jenkins权威指南

使用Jenkins这个基于Java的开源工具进行流水线化软件开发，已经使团队考虑持续集成（CI）的方式发生了重大变革。这本完全指南向你展示了如何使用Jenkins自动化你的构建、集成、发布和开发过程——还论证了CI如何帮你节省时间、金钱并避免许多令人头疼的事。

对于开发者、软件架构师和项目管理者这些理想的读者群来说，本书不仅是一本CI指导材料，还是一本全面的Jenkins参考手册。通过学习使用Jenkins的最佳实践和一些实用技巧等宝贵财富，你将会发现使用Jenkins来建立一个CI服务是多么容易。

- 学习如何安装、配置和安全加固你的Jenkins服务器
- 组织管理和监控通用的构建作业
- 集成自动化测试来验证构建，并建立代码质量报告
- 建立有效的团队通知策略和技术
- 配置构建管道、参数化的作业、矩阵构建和其他高级作业
- 管理多个Jenkins服务器来运行分布式构建
- 完成自动化部署和持续交付

John Ferguson Smart，Wakaleo咨询的主管，帮助一些公司优化了他们的Java开发实践和基础架构。他提供一些在敏捷开发和测试实践上的培训和指导，比如持续集成、测试驱动开发、构建自动化和持续部署。

“我经常被问到是否有一本关于Jenkins的书，现在我可以肯定地回答这个问题了！但更重要的是，这本书反映了John的激情和他在长期教人们使用Jenkins时积累的经验……”

—— **Kohsuke Kawaguchi**
Jenkins项目的创建者以及CloudBees的架构师

图书分类：Java持续集成

策划编辑：付睿
责任编辑：李云静

O'REILLY®
oreilly.com.cn

Broadview®
www.broadview.com.cn

O'Reilly Media, Inc.授权电子工业出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale in the mainland of China (excluding Hong Kong, Macao SAR and Taiwan)

ISBN 978-7-121-29906-3

定价：89.00元

O'REILLY®

Jenkins权威指南

Jenkins: The Definitive Guide

【美】John Ferguson Smart 著
郝树伟 于振苓 熊熠 译

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书从实践者的角度，在讲解Jenkins基本原理的同时，深入到持续集成、持续交付领域，为读者梳理出如何使用Jenkins设计和实现一个自动化的、周期性的集成测试过程，帮助读者理解Jenkins适用于什么场景。书中首先介绍Jenkins以及持续集成、持续交付的基础知识；然后安装Jenkins和配置Jenkins服务器，使用Jenkins建立构建作业、进行自动化测试，介绍Jenkins支持的几种安全模式，如何使用Jenkins进行消息通知，如何确保代码质量；最后进阶展示如何建立高级的和分布式的构建，如何进行自动化的部署和交付以及对Jenkins的运维。

本书适合想要了解Jenkins工具以及持续集成、持续交付理念的开发和测试人员阅读，也非常适合基于Jenkins构建持续集成、持续交付平台的技术人员阅读。

© 2011 by John Ferguson Smart.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Publishing House of Electronics Industry, 2016. Authorized translation of the English edition, 2011 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

本书简体中文版专有出版权由O'Reilly Media, Inc. 授予电子工业出版社。未经许可，不得以任何方式复制或抄袭本书的任何部分。专有出版权受法律保护。

版权贸易合同登记号 图字：01-2015-8199

图书在版编目（CIP）数据

Jenkins权威指南 /（美）约翰·弗格森·斯马特（John Ferguson Smart）著；郝树伟，于振苓，熊熠译. —北京：电子工业出版社，2016.10

书名原文：Jenkins: The Definitive Guide

ISBN 978-7-121-29906-3

Ⅰ. ①J… Ⅱ. ①约… ②郝… ③于… ④熊… Ⅲ. ① 软件－指南 Ⅳ. ①TP31-62

中国版本图书馆CIP数据核字(2016)第219346号

策划编辑：付 睿

责任编辑：李云静

封面设计：Karen Montgomery 张 健

印 刷：三河市鑫金马印装有限公司

装 订：三河市鑫金马印装有限公司

出版发行：电子工业出版社

北京市海淀区万寿路173信箱 邮编：100036

开 本：787×980 1/16 印张：25.5 字数：545千字

版 次：2016年10月第1版

印 次：2016年10月第1次印刷

定 价：89.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819 faq@phei.com.cn。

O'Reilly Media, Inc.介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始,O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来,而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者,O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”;创建第一个商业网站(GNN);组织了影响深远的开放源代码峰会,以至于开源软件运动以此命名;创立了*Make*杂志,从而成为DIY革命的主要先锋;公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖,共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择,O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程,每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列(真希望当初我也想到了)非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人,他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了:‘如果你在路上遇到岔路口,走小路(岔路)。’回顾过去Tim似乎每一次都选择了小路,而且有几次都是转瞬即逝的机会,尽管大路也不错。”

——Linux Journal

译者序

2013 年末，由于工作需求，我们开始在团队内部全面实施持续集成、持续交付作业流程，持续集成和持续交付改变了我们从源码管理到产品发布的整个流程。要知道，我们曾经有过一段时间都在忙碌的打包、发包中度过，那可是非常痛苦的。

Jenkins 作为持续集成、持续交付领域内的优秀开源软件，自然而然地成为我们在项目中大量运用的工具。我们在 Jenkins 的优化配置以及运用上积累了丰富的经验，所以想花时间翻译一下这本书，其一是对我们所学知识的再学习和巩固，其二是想对自由与开源软件在国内的推广做一些文档汉化的事情。

Jenkins 已经成为持续集成、持续交付工具链中一个非常重要的部分。它是一款高效易用的工具，而且是开源的，提供了很多种类的插件供用户使用。但是，真正地掌握好 Jenkins 的使用并不是一件轻松的事情，如何更好地运用 Jenkins 创建出高效、稳定、灵活的作业流是从事持续集成、持续交付的人员所面临的问题。

本书就是一本能够让持续集成、持续交付人员轻松面对这一问题的书籍。本书基于作者优秀的实践经验，为读者梳理出如何使用 Jenkins 设计和实现一个高效、灵活的持续集成、持续交付作业流，帮助读者理解 Jenkins 适用于什么场景，让读者学会如何使用 Jenkins 去考虑和解决实际问题。

非常感谢电子工业出版社的付睿编辑对本书翻译工作的大力支持，还要感谢李云静编辑对本书校对做出的贡献。

全书由郝树伟、于振苓、熊熠三人共同翻译。由于水平有限，书中出现错误与不妥之处在所难免，恳请读者批评指正。

译者

2016 年 9 月

序

七年前我为 Jenkins 的项目写了第一行代码，它最初被称为 Hudson。我是那个曾经破坏过构建的人，所以我需要一个程序在我同事发现错误之前捕捉到错误。这只是一个简单的工具，并且只做了一件简单的事情。但它迅速发展，现在我想它是在整个市场上最占主导地位的 CI 服务器，包括广泛的插件系统、商业分布、Jenkins 即服务式的托管、用户组、分享会、培训等。

与我的其他项目一样，这个项目从成立以来就是开源的。它的生命，主要依赖于他人的帮助和关爱，没有这一点，项目就不会是当前的状态。在这段时间里，我也学到了一些关于运作开源项目的经验。从这一经验，我认为人们往往忽略有很多方法可以帮助一个开源项目，写代码仅是其中的一个。还有宣传、帮助其他用户、组织分享会，当然还有写文档。

从这个意义上来说，John 是 Jenkins 社区的重要组成部分，即使他没有贡献代码——相反，他让新用户更容易接受 Jenkins。例如，他有一个很多人关注的博客，在那里他经常谈论关于持续集成的做法和其他软件开发的主题。他善于解释事情，即使是刚接触到 Jenkins 的人们也可以理解它们，像我这样每天开发 Jenkins 的人通常是很难想象的。他的培训课程也是众所周知的，Jenkins 是其中的一部分。这是让更多的人接触到 Jenkins 的另一种方式。他显然有宣传新的思想和指导开发人员更加高效的激情。

这些天我把我的时间花在 CloudBees 开源 Jenkins 上，用 Jenkins CloudBees pro 版本，我们在 Jenkins 之上构建了插件，并且用 CloudBees DEV@cloud 服务把 Jenkins 部署到私有以及公有云。在这个角色中，我与 John 比以前有了更多的互动，我更加尊重他的激情。

所以他承担了写一本关于 Jenkins 的书的艰巨任务，我真的很高兴。这本书提供了关于持续集成的主要经典部分的伟大概述。而对我个人来说，我总是被问及是否有一本关于

Jenkins 的书，现在我终于可以肯定地回答这个问题了！但更重要的是，这本书反映了他的激情，以及他在教人们如何使用 Jenkins 方面的长期经验与其他东西。但是不要仅仅相信我的话。你需要自己去读，亲自去感受。

——Kohsuke Kawaguchi

Jenkins 项目创建者及 CloudBees 架构师

前言

读者对象

本书针对技术相关的读者（即使他们以往可能没有关于持续集成的经验）。你可能是个刚接触持续集成领域的新人，并想了解它能为你的开发团队带来哪些益处。又或者你可能正在使用 Jenkins 或 Hudson，并想深入了解如何能进一步地将自己的持续集成架构进行提升。

本书主要在 Java 以及 JVM 相关项目的范围内讲解 Jenkins。但是如果你正使用其他技术架构，本书将为你展示 Jenkins 持续集成的优秀生态。我们会讨论如何使用多种非 Java 技术进行项目构建，包括 Grails、Ruby on Rails 以及 .NET。除此之外，还有诸如综合配置、通知、分布式构建和安全等均良好地支持多种国际语言。

书籍编排

持续集成就像很多东西：你投入得越多，就能得到越多的价值。即使搭建一个基本的持续集成，也将能为你的团队流程改进起到积极的作用，它还有着逐步接收及实现一些更高级技术的显著优势。本书会逐渐深入 Jenkins 的持续集成世界，从易到难。在第 1 章中，我们从 Jenkins 的全面概述开始，从全局角度进行了解。之后，我们会介绍如何安装和配置 Jenkins 服务器以及如何搭建基本的构建作业。当我们掌握了基本知识后，将继续深入更高级的主题，包括自动化测试、安全性、高级通知技术和代码质量报告。接下来，我们在讨论如何利用 Jenkins 使用持续部署之前，将讨论更高级的构建技术，如矩阵构建，以及分布式构建和云端持续集成。最后，我们会介绍维护 Jenkins 服务器的一些技巧。

是 Jenkins，还是 Hudson

Jenkins 从开始直到最近都以 Hudson 而被熟知。在 2009 年，甲骨文收购了 Sun 并继承了 Hudson 代码库。在 2011 年年初，甲骨文和开源社区之间的关系破裂，该项目被分成两个独立的项目：Jenkins，由大部分原始 Hudson 开发人员组成；甲骨文管理的 Hudson。

正如书名所示，本书主要关注 Jenkins。本书的大部分内容写于分支拆分前，不过拆分后两种产品仍然非常相似。所以，尽管示例和插图通常来源于 Jenkins，但大部分讨论的内容却同样适用于 Hudson。

字体规范

本书遵循特定的字体使用规则约定。理解这些约定有助于更容易地学习本书。

斜体（*Italic*）
: 用于文件名、文件扩展名、URL、应用程序名称、强调和首次介绍的新名称。

等宽字体（`Constant width`）
: 用于 Java 类名、方法、变量、属性、数据类型、数据库元素以及出现在正文中的代码片段。

加粗的等宽字体（**`Constant width bold`**）
: 用于输入的命令行和高亮显示插入运行示例中的新代码。

等宽斜体（*`Constant width italic`*）
: 用于注释和输出。

命令行规范

本书常常会讨论到命令行指令。当我们在控制台操作时，将得到的输出结果（例如，命令提示或屏幕输出）以普通字符格式显示，命令（输入的）以**加粗**格式显示。例如：

```
$ ls -al
total 168
drwxr-xr-x  16 johnsmart staff   544 21 Jan 07:20 .
drwxr-xr-x+ 85 johnsmart staff  2890 21 Jan 07:10 ..
-rw-r--r--   1 johnsmart staff    30 26 May 2009 .owner
-rw-r--r--@  1 johnsmart staff  1813 16 Apr 2009 config.xml
drwxr-xr-x 181 johnsmart staff  6154 26 May 2009 fingerprints
drwxr-xr-x  17 johnsmart staff   578 16 Apr 2009 jobs
```

```
drwxr-xr-x   3 johnsmart staff   102 15 Apr 2009 log
drwxr-xr-x  63 johnsmart staff  2142 26 May 2009 plugins
-rw-r--r--   1 johnsmart staff    46 26 May 2009 queue.xml
-rw-r--r--@  1 johnsmart staff    64 13 Nov 2008 secret.key
-rw-r--r--   1 johnsmart staff 51568 26 May 2009 update-center.json
drwxr-xr-x   3 johnsmart staff   102 26 May 2009 updates
drwxr-xr-x   3 johnsmart staff   102 15 Apr 2009 userContent
drwxr-xr-x  12 johnsmart staff   408 17 Feb 2009 users
drwxr-xr-x  28 johnsmart staff   952 26 May 2009 war
```

在必要时，在一行结尾的反斜杠字符用于代表一行的结束：如果你愿意，可以在一行中全部输入（没有反斜杠）。不要忘记忽略换行后的起始字符“>”——它是 UNIX 提示字符。

```
$ wget -O - http://jenkins-ci.org/debian/jenkins-ci.org.key \
> | sudo apt-key add –
```

为了一致性，除非我们在讨论 Windows 的问题，否则都将采用 UNIX 风格的命令提示符（美元符号“$”），如下所示：

```
$ java -jar jenkins.war
```

或

```
$ svn list svn://localhost
```

然而，除非我们说明用另外的方式，否则 Windows 用户可以安全地在 Windows 命令控制台使用这些命令：

```
C:\Documents and Settings\Owner> java -jar jenkins.war
```

或

```
C:\Documents and Settings\Owner> svn list svn://localhost
```

贡献人

本书不是我独自写的。相反，它是由多人分工协作完成的。特别是以下人员慷慨地贡献了自己的宝贵时间、知识和协作技巧，使得本书更加完善：

- **Evgeny Goldin** 是一位出生于俄国而生活在以色列的软件工程师。他带领并负责汤森路透（Thomson Reuters）开发人员的诸多工作，其中一些直接关系到 Maven、Groovy 以及如 Artifactory、Jenkins 的构建工具。他有丰富的经验和技术，包括 Perl、Java、JavaScript 和 Groovy。构建工具和动态语言是 Evgeny 最喜欢的，他经常用到、展示并写进博客。最近他正在为 *GroovyMag* 杂志撰写 Methods&Tools，同时还运行维护了他自己的两个开源项目：Maven-plugins（*http://evgeny-goldin.com/*

wiki/Maven-plugins）和 GCommons（*http://evgeny-goldin.com/wiki/GCommons*）。他的博客地址是 *http://evgeny-goldin.com/blog*，还能用 *@evgeny_golin* 在 Twitter 上找到他。

Evgeny 贡献了第 10 章的自动生成 Maven 构建作业内容。

- **Matthew McCullough** 是一个精力充沛的人，拥有 15 年企业级软件开发及开源教育经验，他同时还是一家位于丹佛名为 Ambient Ideas 的咨询机构的联合创始人之一。Matthew 现在是 GitHub.com 的培训师，是 O'Reilly 出版的 *Git Master Class* 系列书的作者。其在 30 多个国家和国际会议上发表过演讲，他还是 DZone RefCards 排名前十的作者，也是丹佛开源用户组主席。他目前的研究课题是项目自动化：构建工具（Maven，Leiningen，Gradle）、分布式版本控制（Git）、持续集成（Jenkins）以及代码质量度量（Sonar）。Matthew 和他美丽的妻子以及两个年幼的女儿一起居住在科罗拉多州的丹佛，他几乎积极参与科罗拉多州的任何一项室外活动。

  Matthew 撰写了第 5 章的 Jenkins 集成 Git 部分的内容。

- **Juven Xu** 是一位来自中国的在 Sonatype 工作的软件工程师。他是一个活跃的开源社区成员、Maven 认证专家。Juven 还负责 *Maven:The Definitive Guide* 的中文翻译，这本书被当作 Maven 项目上最原始的中文参考手册。他当前也在致力于本书的中文翻译工作。

  Juven 撰写了第 8 章中的 IRC 通知部分。

- Rene Groeschke 是 Cassidian Systems（原名 EADS Deutschland GmbH）公司的一名软件工程师，也是位开源爱好者。他拥有七年多个 Java 企业级项目开发工作经验的认证 ScrumMaster，他特别关注于敏捷方法，如持续集成和测试驱动开发。除了日常工作，腓特烈港的 Corporate Education 大学还聘请他为信息技术专业本科学生进行 Scrum 及其相关课题的讲座。

  Rene 撰写了第 5 章 Gradle 构建项目部分。

评审小组

这本书的技术评审过程采取的方法与其他大部分图书有些不同。不是让一两个技术评审人在本书撰写工作接近尾声时通读全书，而是让一队 Jenkins 社区的志愿者，包括很多核心 Jenkins 开发人员在本书撰写的时候就可以阅读一些章节的内容。这个评审小组由以下人员组成：Alan Harder，Andrew Bayer，Carlo Bonamico，Chris Graham，Eric Smalling，Gregory Boissinot，Harald Soevik，Julien Simpson，Juven Xu，Kohsuke Kawaguchi，Martijn Verberg，Ross Rowe，Tyler Ballance。

图书赞助商

因为众多组织给予了撰写图书过程的资金帮助，所以本书得以顺利完成。

Wakaleo 咨询

Wakaleo（*http://www.wakaleo.com*）是一家咨询公司，帮助企业优化他们的软件开发流程。由 John Ferguson 领导，他是本书和 *Java Power Tools*（*http://oreilly.com/catalog/9780596527938*）的作者，Wakaleo 咨询提供 Java 敏捷开发和测试实践、软件开发生命周期优化以及敏捷方法论的咨询、培训和顾问服务。

Wakaleo 在一些如持续集成、构建自动化、测试驱动开发、自动化 Web 测试和代码整洁领域提供帮助并培训企业，使用的开源工具包括 Maven、Jenkins、Selenium 2 和 Nexus。Wakaleo 咨询也提供公共和现场关于持续集成和持续部署、构建自动化、代码整洁实践、测试驱动开发和行为驱动开发的培训，以及 Scrum 开发（CSD）认证课程。

CloudBees

CloudBees（*http://www.cloudbees.com*）是唯一一家专注于云端完整的 Java Web 应用程序开发 - 部署生命周期服务的云公司。这个公司也是 Jenkins/Hudson 持续集成工具世界上首屈一指的专家。

Jenkins/Hudson 创建者 Kohsuke Kawaguchi 领导了全球 CloudBees 专家小组。他们创造了 Nectar，一个订阅预置且支持增强的 Jenkins 版本。如果你依赖 Jenkins 处理关键软件流程，Nectar 提供高度测试、稳定且全方位支持的 Jenkins 版本。它还包括 Nectar 独有的功能，如自动调整 VMware 虚拟机。

如果你准备探索云端持续集成，CloudBees 将 Jenkins/Hudson 集成进它的 DEV@cloud 构建平台的一部分。从 Jenkins 开始并不需要前期在构建服务器方面花费大量投资，且没有性能限制和维护麻烦。当一个应用已经准备上线，你只需轻点几次鼠标就能将其作为服务部署到 CloudBees 的 RUN@cloud 平台。

使用 CloudBees 的 DEV@cloud 和 RUN@cloud 服务，你无须担心服务器、虚拟机以及 IT 人员。和 Nectar 一起用，你会喜欢对 Jenkins 最强大的、稳定的支持。

Odd-e

Odd-e（*http://www.odd-e.com*）是个亚洲公司，以创新的方法构建产品并帮助别人实现相同的目标。其团队由有经验的教练和产品开发人员组成，他们在工作中一致追求迭代、

敏捷、简洁、精致的价值观，公司的组织架构也是如此。例如，Odd-e 没有一个层次性组织，也没有管理人员为他人做决定。相反，个人自我组织并且使用所有的技能来提高其竞争力。公司提供培训和后续督导帮助他人寻求更好的工作方式来合作。

不是工作而是价值观让 Odd-e 凝聚在一起。它的成员喜欢构建软件以及价值观的学习和贡献，而不是利润最大化，并致力于支持亚洲开源的发展。

使用代码示例

本书是一本开放源代码的书，在创作共用许可证（Creative Commons License）下出版。本书的英文原版书用 DocBook 编写，使用 XmlMind。这本书的源代码可以在 *https://github.com/wakaleo/jenkins-the-definitive-guide-book* 的 GitHub 找到。

本书中的 Jenkins 项目使用的示例是开源的并且可以在网上免费获得——在 *http://www.wakaleo.com/books/jenkins-the-definitive-guide* 提供了更多细节性的指导。

本书的目的是帮助你完成工作。一般情况下，你可以在自己的程序和文档中使用本书的代码。除非你复制大量的代码，否则不必与我们联系许可。例如，使用本书的几块代码编写程序不需要许可。销售或分销 O'Reilly 图书光盘的例子需要许可。引用这本书和引用示例代码来回答问题不需要许可。把来自本书的大量示例代码应用到你的产品文档确实需要许可。

我们对注明版权归属的行为表示赞赏，但不强制要求。一个归属通常包括作品名称、作者、出版商和 ISBN。例如，“*Jenkins: The Definitive Guide* by John Ferguson Smart (O'Reilly). Copyright 2011 John Ferguson Smart,978-1-449-30535-2.”

如果你觉得自己的示例代码使用不在正当使用或上面给出的许可范围内，请随时与我们联系：*permissions@oreilly.com*。

Safari® Books Online



Safari Books Online 是一个按需点阅的数字图书馆，可让你轻松地搜索超过 7500 种技术和创造性的参考书籍和视频，并快速找到所需的答案。

有了订阅，你可以阅读任何页面，并从我们的在线图书馆观看任何视频。阅读你的手机和移动设备上的书。访问新的标题之前，它们可供打印，并获得独家访问的开发手稿和作者的反馈。复制和粘贴示例代码，整理你最喜欢的书籍，下载章节，标记关键部分，创建注释，打印页面，并从其他节省时间的特性上受益。

O'Reilly媒体把这本书上传到了Safari Books Online服务。为了获取该书的全部数字访问权限以及O'Reilly和其他出版商的其他类似主题，可免费注册*http://my.safaribooksonline.com*。

联系我们

请将对本书的评价和发现的问题通过如下地址告知出版者。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街2号成铭大厦C座807室（100035）
奥莱利技术咨询（北京）有限公司

我们有这本书的网页，在那里我们列出了勘误表、实例和任何附加的信息。你可以访问此页：

http://www.oreilly.com/catalog/9781449305352

评论或询问这本书的技术问题，请发送电子邮件至：

bookquestions@oreilly.com

关于书籍、课程、会议和新闻的更多信息，请访问我们的网站：*http://www.oreilly.com*。

在Facebook找到我们：*http://facebook.com/oreilly*

在Twitter跟随我们：*http://twitter.com/oreillymedia*

在YouTube上观看我们：*http://www.youtube.com/oreillymedia*

致谢

首先，要感谢我的好妻子Chantal以及孩子们James和William，没有你们的爱、支持和宽容，这本书是不可能完成的。

我要感谢Mike Loukides再次和我一起从事这本书以及整个O'Reilly团队高标准的工作。

谢谢 Kohsuke Kawaguchi 创建了 Jenkins，并一直是这个伟大产品背后的驱动力。还要感谢 Francois Dechery、SachaLabourey、Harpreet Singh 以及 CloudBees 团队其他成员的帮助和支持。

我也很感谢那些花了时间和精力为本书做出贡献的人：Evgeny Goldin、Matthew McCullough、Juven Xu 以及 Rene Groeschke。

非常感谢下面的审稿人，他们在整个写作过程中，提供了宝贵的反馈意见：Alan Harder，Andrew Bayer，Carlo Bonamico，Chris Graham，Eric Smalling，Gregory Boissinot，Harald Soevik，Julien Simpson，Juven Xu，Kohsuke Kawaguchi，Martijn Verberg，Ross Rowe，Tyler Balance。

感谢 Andrew Bayer、Martijn Verburg、Matthew McCullough、Rob Purcell、Ray King、Andrew Walker 等人，他们的讨论和反馈给了我此书的灵感和想法。

还有许多人对本书的顺利完成提供了各种帮助，这使本书比原来更加丰富和完整：Geoff and Alex Bullen，Pete Thomas，Gordon Weir，Jay Zimmerman，Tim O'Brien，Russ Miles，Richard Paul，Julien Simpson，John Stevenson，Michael Neale，Arnaud Héritier，Manfred Moser。

最后十分感谢 Hudson/Jenkins 开发人员和用户社区持续的鼓励和支持。

目录

序 xvii

前言 xix

第 1 章 Jenkins 简介 1

简介 1
持续集成基础 1
Jenkins 简介（née Hudson） 3
从 Hudson 到 Jenkins——一个简短的故事 3
应该使用 Jenkins 还是 Hudson 4
引进持续集成到你的公司 5
阶段 1——无构建服务器 5
阶段 2——夜间构建 6
阶段 3——夜间构建加自动化测试 6
阶段 4——加入度量指标 6
阶段 5——更认真地对待测试 6
阶段 6——自动化验收测试和自动化部署 7
阶段 7——持续部署 7
接下来做什么呢 7

第 2 章 迈入 Jenkins 的第一步 9
简介 9
准备环境 9
安装 Java 10
安装 Git 11
创建 GitHub 账号 11
配置 SSH 密钥 11
创建代码库分支 12
启动 Jenkins 13
配置工具 17
配置你的 Maven 设置 18
配置 JDK 19
通知 20
设置 Git 20
你的第一个 Jenkins 构建作业 21
你的第一个构建作业 26
更多的报告——显示 Javadoc 33
添加代码覆盖率和其他度量 34
小结 40
第 3 章 安装 Jenkins 41
简介 41
下载和安装 Jenkins 41
为 Jenkins 准备好构建服务器 44
Jenkins 主目录 45
在 Debian 或者 Ubuntu 系统上安装 Jenkins 47
在 Red Hat、Fedora 或者 CentOS 上安装 Jenkins 48
在 SUSE 或者 OpenSUSE 上安装 Jenkins 49
以单机应用形式运行 Jenkins 49
基于 Apache 服务器运行 Jenkins 53
在应用服务器上运行 Jenkins 54
内存方面的考虑 54
安装 Jenkins 为一个 Windows 服务 55

Jenkins 主目录里有什么 59
备份 Jenkins 数据 62
升级 Jenkins 安装 63
小结 64

第 4 章 配置 Jenkins 服务器 65

简介 65
配置面板——Manage Jenkins 界面 65
配置系统环境 68
配置全局属性 69
配置 JDK 70
配置构建工具 72
Maven 73
Ant 74
Shell 脚本 75
配置版本控制工具 75
配置 Subversion 75
配置 CVS 75
配置邮件服务器 76
配置代理 77
小结 78

第 5 章 设置构建作业 79

简介 79
Jenkins 构建作业 79
创建自由式构建作业 80
通用选项 81
高级项目选项 82
配置源代码管理 83
用 Subversion 工作 84
使用 Git 86
Build Triggers 96
构建作业完成后触发下一个 97

计划构建作业 97
轮询 SCM 98
远程触发构建 99
手工构建作业 101
构建步骤 101
Maven 构建步骤 102
Ant 构建步骤 103
执行 Shell 或 Windows 批处理命令 104
在构建中使用 Jenkins 环境变量 105
运行 Groovy 脚本 108
用其他语言构建项目 110
构建后的操作 110
报告测试结果 110
归档构建产物 110
通知 114
构建其他的项目 114
运行新的构建作业 115
使用 Maven 构建作业 115
每当 SNAPSHOT（快照）依赖被建立时都要进行构建 116
配置 Maven 构建 116
构建后期操作 118
部署到企业仓库管理器 118
部署到商业企业仓库管理器 122
管理模块 122
Maven 构建作业中额外的构建步骤 122
用其他语言使用 Jenkins 124
用 Grails 构建项目 124
用 Gradle 构建项目 126
用 Visual Studio MSBuild 构建项目 129
用 NAnt 构建项目 130
使用 Ruby 和 Ruby on Rails 构建项目 130
小结 132

第 6 章 自动化测试 133
简介 133
自动化单元和集成测试 134
在 Jenkins 中配置测试报告 135
显示测试结果 137
忽略测试 140
代码覆盖率 142
使用 Cobertura 测量代码覆盖率 143
使用 Clover 测量代码覆盖率 152
自动化验收测试 154
JMeter 自动化性能测试 157
请求帮助！我的测试运行得太慢了 165
添加更多的硬件 166
运行更少的集成 / 功能测试 166
并行运行测试 167
小结 167

第 7 章 Jenkins 安全 169
简介 169
激活 Jenkins 安全 169
Jenkins 安全实例 170
安全域——定义 Jenkins 用户 171
使用 Jenkins 内置用户数据库 171
使用 LDAP 仓库 174
使用微软活动目录 176
使用 UNIX 用户和用户组 177
Servlet 容器授权 177
使用 Atlassian Crowd 178
与其他系统集成 179
授权——谁可以做什么 181
安全矩阵 181
基于项目的安全 185
基于角色的安全 188

审计——跟踪用户行为 190
小结 193

第 8 章 通知 195

简介 195
电子邮件通知 195
更高级的电子邮件通知 197
声明构建 200
RSS 订阅 202
构建分发器 202
即时消息 204
用 Jabber 进行 IM 通知 204
使用 IRC 的 IM 通知 209
IRC 通知 209
桌面通知器 212
通过 Notifo 通知 213
移动通知 216
短信通知 216
制造噪声 219
极端反馈设备 221
小结 222

第 9 章 代码质量 225

简介 225
构建过程中的代码质量 226
受欢迎的 Java 和 Groovy 代码质量分析工具 227
Checkstyle 227
PMD/CPD 230
FindBugs 235
CodeNarc 237
使用 Violations 插件报告代码质量问题 238
与自由风格下的构建作业一起工作 239
使用 Maven 构建作业 242

使用 Checkstyle、PMD 和 FindBugs 报告 244
报告代码复杂度 247
报告未完成的任务 248
集成 Sonar 249
小结 253

第 10 章 高级构建 255

简介 255
参数化构建作业 255
创建一个参数化构建作业 256
为构建适配参数化构建脚本 257
高级参数类型 259
使用 Subversion 标签构建 260
使用 Git 标签构建 261
远程启动参数化构建作业 262
参数化构建作业历史 263
参数化触发 263
多重结构的构建作业 266
搭建多重结构构建 266
配置从节点轴 267
配置 JDK 轴 268
自定义轴 268
执行多重结构构建 268
自动生成 Maven 构建作业 271
配置作业 271
使用继承复用作业配置 273
插件支持 275
自由式作业 278
构建协作 278
在 Jenkins 中并行构建 279
依赖关系图 279
连接 280
Locks and Latches 插件 281

构建管道及优化 282
通过 M2Release 插件管理 Maven 版本发布 283
复制构建产物 286
构建进阶 289
汇总测试结果 295
构建管道 296
小结 299

第 11 章 分布式构建 301

简介 301
Jenkins 分布式构建架构 301
Jenkins 主 / 从策略 302
主节点使用 SSH 启动从节点代理 303
使用 Java Web Start 手工启动从代理 307
把 Jenkins 从节点安装为 Windows 服务 309
无主节点模式启动从节点 310
以远程服务方式启动 Windows 从节点 310
把构建作业与一个或一组从节点关联 311
节点监控 313
云计算 314
使用 Amazon EC2 314
使用 CloudBees DEV@cloud 服务 318
小结 319

第 12 章 自动化部署和持续交付 321

简介 321
实现自动化和持续部署 322
部署脚本 322
数据库更新 322
冒烟测试 325
回滚更改 326
部署到应用程序服务器上 326
部署一个 Java 应用 327

部署像 Ruby 和 PHP 这样基于脚本的应用程序 336
小结 339

第 13 章 Jenkins 的维护 341

简介 341
监控磁盘空间 341
使用 Disk Usage 插件 343
磁盘使用及 Jenkins Maven 项目类型 344
监控服务器负载 345
备份配置 346
Jenkins 备份基础 346
使用备份插件 348
更为轻量的自动备份 350
构建作业归档 350
构建迁移 351
小结 355

附录 A 自动化你的单元和集成测试 357

索引 367

第 1 章 1

Jenkins简介

简介

持续集成，也就是通常所说的 CI（Continuous Integration），可以说是现代软件开发技术的基础。事实上，它是真正的游戏规则改变者——引进持续集成到公司，能从根本上改变一个团队对于整个软件开发过程的思考方式。它绝对有潜力去启用和触发一系列的增量式的过程改进，从简单的、有计划的、自动化的构建逐渐改进为可以持续交付投入生产中。一个好的持续集成基础设施可以简化开发过程直到部署，可以帮助开发人员更快地检测和修复 bug，为所有开发和非开发人员提供一个有用的项目仪表盘，最后帮助团队交付更真实的业务价值给最终用户。每一个专业的开发团队，无论多小，都应该且有必要学习和实践持续集成。

持续集成基础

回想在使用持续集成之前的那些充满瀑布项目和甘特图的日子，开发团队的时间和精力都耗费在发布一个版本之前的那段时期里，也就是所谓的集成阶段。在这个阶段，某些开发人员或者小团队做出的代码改动，都会零零散散地整合进最终产品。这可是件有难度的工作，有时甚至仅集成一些冲突性的变更都会消耗数月之久的时间。而且不仅很难去预期会突然出现什么类型的问题，更难的是去修复它们，因为它可能会使你不得不去重写之前已经写了几周或者几个月的代码。这个过程不仅痛苦，而且充满了风险和危险，且往往会导致延迟重要交付、额外开销的后果，最终引来客户的不满。持续集成的出现就是为了定位和解决这些问题。

持续集成，就其最简单的形式来讲，就是一个能监控你版本控制系统变化的工具。无论

任何时候，只要检测到有变化，这个工具就会自动编译和测试你的应用程序。如果出现问题，它就马上通知开发人员，以便他们可以立即着手解决这个问题。

2 持续集成可以做的事情当然还能更多。持续集成能帮你密切监视代码库的健康，自动监控代码质量和代码覆盖率度量，还能帮助你降低技术债务和减少维护成本。这些公开可见的代码质量度量可以使得开发人员为自己开发的高质量代码而自豪，并且激励他们继续努力去改善。结合自动化的端到端的验收测试，持续集成也可以作为一种沟通工具，清晰地发布和展现总体开发工作的当前状态。通过构建自动化部署过程，持续集成能大大简化和加速你的交付过程，自动化和一键部署应用程序的最新版本。

从本质上讲，持续集成是通过提供更快的反馈来降低风险的。首先，它被设计成用来更快地识别和修复集成以及回归相关的问题，目的是更平滑、更快的交付和更少的bug。通过更好地为技术和非技术团队成员提供项目状态的可视化，持续集成打开并改善了团队成员间的沟通渠道，同时鼓励他们协作解决问题和过程改进。通过把部署过程自动化，持续集成可以帮助你更快、更可靠、更轻松地把软件交付到测试人员和最终用户的手上。

自动化部署的想法设计是很重要的。事实上，理论上来讲自动化部署的过程可以使你能够推送每一个带有必要的自动化测试的构建到生产当中去。这种直接自动化部署每个成功的构建到生产当中的实践，就是所谓的*持续部署*。

单一的持续部署方式并不总是适合每个人。比如，很多用户并不喜欢一周内接二连三地部署新的版本，他们更喜欢一个可预见的（透明的）发布周期。除此之外，在真正部署一个新版本的时候，商业和市场营销也是需要考虑的重要方面。

上述这些持续部署需要考虑的东西，基本也是*持续交付*所需要考虑的，只是略有不同而已。使用持续交付，任何通过了相关自动化测试和质量关的构建，都能通过完全自动化且一键部署的方式被部署到生产中，并且在几分钟内交付给最终用户。当然，这整个过程并不是自动进行的：它是业务，而不是信息技术，需要决定一个最佳时机来交付最新的变化。

3 因此，持续集成技术，特别是持续部署和持续交付，就是为了更快地给最终用户提供价值。你的团队从一个小的代码改动到投入生产需要多长时间？这个过程会涉及多少可以提早修复的问题？你了解任何一个团队成员做出的代码修改吗？会让劳动密集型手工测试的QA团队花费多少时间？会涉及多少手动部署的步骤？大概只有少数人能真正了解上述全部问题。持续集成不是无所不能的，不过它确实能帮助你简化许多这样的问题。

持续集成可以说是一种思维工具集。想要充分地利用好持续集成，你的团队必须要先进入持续集成的思维方式中。例如，你的项目必须有一个可靠的、可重复的、自动化的构建过程，并且不涉及人工干预。能保证优先修复有问题的构建，而不是停滞不前、不管

不理。这个部署过程应该是自动化的，不应该有任何的人工干预。由于持续集成服务器可信度的好坏在很大程度上取决于你测试的质量，所以团队需要设计强健的、高质量的测试和测试实践。

在本书中，我们将探讨如何使用 Jenkins 或者 Hudson 来实现一个健壮的和全面的持续集成解决方案。

Jenkins 简介（née Hudson）

Jenkins，最开始被称作 Hudson，是一个 Java 语言编写的开源的持续集成工具。Jenkins 在持续集成领域的市场份额居于主导地位，其被各种规模的团队用于各种语言和技术的项目中，比如 .NET、Ruby、Groovy、Grails、PHP 等，当然还有 Java。是什么使 Jenkins 如此成功呢？又为什么你的持续集成基础设施中要使用 Jenkins 呢？

首先，Jenkins 是易于使用的。用户界面非常简单、直观，增加了视觉上的吸引力，而且 Jenkins 作为一个整体，具有平滑的学习曲线。在第 2 章中我们会发现，你可以在几分钟之内开始使用 Jenkins。

其次，Jenkins 拥有良好的扩展性，能够极其灵活和方便地迎合你的想法。它有数以百计的开源插件可供使用，而且每周会有更多的开源插件贡献进来。这些开源插件覆盖系统版本控制、构建工具、代码质量度量、构建通知、外部系统集成、用户界面定制化、游戏等。而且这些插件的安装都非常快捷和简单。

最后，Jenkins 之所以受大众喜欢，得益于其开源社区的规模和活跃度。Jenkins 社区包含一个有规模的、流动的、响应式且开放态的讨论群，活跃的邮件列表，IRC 频道，有知名度的博客区和 Twitter 账户。Jenkins 社区的发展速度非常快，每周都会有新功能、新特性，以及 bug 修复和插件更新发布出来。

当然，Jenkins 也满足那些不想每周都进行产品升级的用户的需求。对于那些更喜欢尽量减少版本改动的需求，Jenkins 提供一个长期支持的版本，也就是所谓的 LTS，这个版本落后于最新发布的 Jenkins 版本，却提供更加稳定的功能和较慢的更新变化。LTS 版本每三个月发布一次新版本，新版本主要包含重要的 bug 修复和关键补丁。这个概念类似于 Ubuntu LTS 版本。

从 Hudson 到 Jenkins——一个简短的故事

Jenkins 始于 2004 年 Sun 公司员工 Kohsuke Kawaguchi 基于个人爱好开发的一个名字叫 Hudson 的项目。经过几年的发展，Sun 公司内部越来越多的团队开始在各种各样的项目

中使用 Hudson。2008 年年初，Sun 公司认可了这个工具的质量和价值，要求 Kohsuke 把全部时间都投入到 Hudson 上，并且为 Hudson 提供专业的服务和技术支持。到 2010 年，Hudson 已经成为持续集成解决方案领域内的领头羊并拥有超过 70% 的市场份额。

2009 年，Oracle 收购了 Sun 公司。直到 2010 年年底，Hudson 开发者社区和 Oracle 之间出现了紧张关系，最初是由于 Java.net 的基础设施问题引起的，后来 Hudson 的商标问题加剧了相互之间的不愉快。这些紧张的关系也反映出大家关于把项目放在 Oracle 公司的管理之下这种方式的根本分歧。事实上，Oracle 是想要 Hudson 走向一个版本发布更慢的、更严格控制的开发过程，而以 Kohsuke 为首的大多数核心开发人员更希望像过去 Hudson 稳定工作的那样，继续维持开放、灵活、快节奏的社区主导型模式。

2011 年 1 月，Hudson 开发者社区投票通过，正式将这个项目更名为 Jenkins。随后他们把原始的 Hudson 代码迁移为一个新的 GitHub 项目（*https://github.com/jenkinsci*），并在 GitHub 上继续他们的工作。绝大多数的核心开发人员和插件开发人员都跟随 Kohsuke Kawaguchi 以及其他核心贡献者到了 Jenkins 阵营，也就是我们今天能看到大量的开发活动的地方。

经过这次事件，大部分用户也跟着 Jenkins 开发者社区转向 Jenkins。在撰写本书时，有调查显示，75% 的 Hudson 用户已经转向 Jenkins；13% 的用户依旧在使用 Hudson；另外 12% 的用户既使用 Hudson 又使用 Jenkins，或者正在迁移到 Jenkins 的过程中。

不过，Oracle 和 Sonatype（开发 Maven 和 Nexus 的公司）仍在继续开发和维护 Hudson 代码库（现在也放在 GitHub 上：*https://github.com/hudson*），但是关注的重点跟之前相比有很大不同。事实上，Sonatype 的开发人员主要把精力集中在底层基础设施的变化和管理依赖关系注入框架以及插件体系架构的 Maven 集成上。

5 应该使用 Jenkins 还是 Hudson

应该使用 Jenkins 还是 Hudson 呢？因为这是一本关于 Jenkins 的书，下面列出一些你可能会选择 Jenkins 的原因。

- Jenkins 就是新版的 Hudson。事实上，Jenkins 只是简单地给以前的 Hudson 换了个新名字，所以如果你喜欢 Hudson，那你就应该也喜欢 Jenkins！ Jenkins 使用 Hudson 的代码库，开发团队和项目管理保持不变。简而言之，最开始编写了 Hudson 核心部分的绝大部分开发人员，只是重新像以前那样在 Jenkins 上恢复了他们的工作。
- Jenkins 社区。就像很多其他成功的开源项目一样，Hudson 的力量来源于它有一个规模庞大的、有流动性的社区以及大规模的应用。bug 的认定（修复）非常快，

并且如果你遇到一些问题，别人也可能会遇到和你同样的问题。如果你遇到了自己解决不了的问题，你可以把这个问题发到邮件列表里或者 IRC 频道上——一定会有人能帮助你。

- *开发速度快。*和Hudson一样，Jenkins延续使用众多开发人员喜欢的快速发布周期。每周都会发布新功能、新特性，以及新插件和 bug 修复；bug 修复的周转期实际上是非常短的。如果你喜欢更稳定，可以使用 LTS 版本。

为了公平起见，下面列出一些你可能更喜欢坚持使用 Hudson 的原因。

- *除非它坏了，否则你根本不用去修复它。*你已经安装部署了一套令你满意的 Hudson，并且觉得没有必要升级到最新的版本。
- *企业集成以及 Sonatype 产品自带工具。*Hudson 可能非常看重跟企业级工具的集成，比如 LDAP/Active Directory，还有 Sonatype 公司的产品，比如 Maven 3、Nexus 和 M2Ecipse；而 Jenkins 相对于那些与其有竞争的工具来说却更加开放，比如 Artifactory 和 Gradle。
- *插件体系架构。*假如你打算编写自己的 Jenkins/Hudson 插件，你要意识到 Sonatype 公司正在为 Hudson 插件提供 JSR-330 依赖项注入。尽管这将来会在 Jenkins 和 Hudson 之间产生插件兼容性的一系列问题，但对于新的开发人员来讲却是非常易用的。

好消息是，无论你正在使用 Jenkins 还是 Hudson，它们仍然非常类似，绝大多数在这本书中讨论的技术和技巧都适用于两者。事实上，为了说明这一点，本书中的很多截图引用的是 Hudson 而不是 Jenkins。

引进持续集成到你的公司

6

持续集成并不是一个一蹴而就的事物。事实上，把持续集成引进到一个公司需要通过几个不同的发展阶段。这些阶段涉及对技术基础设施的不断改善，或者说更重要的是对这个开发团队本身的相关实践和团队文化的改善。在接下来的段落里，我们将逐一介绍每个阶段。

阶段 1——无构建服务器

最初，团队没有任何形式的中央构建服务器。软件是在开发人员的机器上手工触发构建的，尽管这个构建可能使用了 Ant 脚本或者其他类似的东西。源代码可能存储在一个中央源代码仓库中，但开发人员却不需要定期提交其代码变更。在计划发布一个新版本之前的那段时间里，开发人员需要手动集成他的代码改动，这个过程是非常痛苦的。

阶段 2——夜间构建

在这个阶段，团队有一个构建服务器，并且会按计划定期（通常是夜间）触发自动化构建过程。这种构建只是简单的代码编译，没有可靠的或可重复的单元测试。事实上，如果写了自动化测试的话，它也不是强制性地要作为构建过程的一部分，并很可能运行得不正确。在这个阶段，开发人员至少会在每天结束之前定期提交他们的更改。假如开发人员提交的代码改动与另外一个开发人员的代码改动发生冲突，构建服务器会在第二天早上通过发送邮件的形式向团队报警。不过，在这个阶段，团队仍然倾向于只使用构建服务器的消息通知机制——他们不觉得有义务去立即修复一个构建服务器出现的问题，所以有些构建可能一直坏在构建服务器上没人处理。

阶段 3——夜间构建加自动化测试

在这个阶段，团队开始更认真地对待持续集成和自动化测试了。无论任何时候，只要有代码更新提交到版本控制系统中，构建服务器就会触发一个构建，团队成员可以轻易地看到是哪些源代码变化触发了这个特定的构建，以及这些变化是想要处理哪些问题的。此外，构建运行脚本去编译应用程序并且运行一系列的自动化单元和 / 或集成测试。除了发送邮件以外，构建服务器也会使用更积极主动的渠道（比如即时通信）来提醒团队成员在集成的过程中出现了哪些问题。运行失败的构建在这个阶段一般都可以很快地得到修复。

阶段 4——加入度量指标

在这个阶段，构建中会运行自动化的代码质量和代码覆盖率检查，来帮助评估代码库的质量和（至少在一定程度上）测试的相关性以及有效性。检查代码质量的构建也会自动为应用程序生成 API 文档。所有这些都会帮助团队保持一个高质量的代码库，也会对测试质量的下降提出警报。团队一般还会建立“构建展示器”，它是一个项目状态的仪表盘视图，可以放在屏幕的一个突出位置上让所有团队成员看到。

阶段 5——更认真地对待测试

持续集成的好处是可以和可靠的测试实践紧密结合。现在，驱动测试开发模式被广泛应用，以至于自动化构建的结果越来越值得信赖。应用程序不再只是简单地被编译和测试，如果测试通过了的话，它会被自动部署到一个应用服务器上来进行更全面的端到端测试和性能测试。

阶段 6——自动化验收测试和自动化部署

这个阶段的开发工作是由验收测试驱动的，能更好地指导开发工作和提供高水准的项目状态报告。这些自动化测试采用行为驱动开发和验收测试驱动开发工具作为交流和文档工具以及文档测试工具，最后用非开发人员都可以理解的业务术语来发布测试结果报告。在整个开发过程的早期阶段就使用了这些高级的自动化测试，它们清晰地展示出哪些功能特性已经完成了，哪些还有待完成。不管是有改动提交，还是定期在夜间触发构建，应用程序都会被自动部署到测试环境当中以便 QA 团队进行测试。当测试人员认为可以的时候，就可以手动触发构建来部署一个新的版本去进行用户验收测试或者直接交付到生产环境。如果当前版本中出现了严重的问题，团队还可以使用构建服务器把当前版本回滚到上一个版本。

阶段 7——持续部署

在这个阶段，团队对于自动化的单元、集成和验收测试已经非常有信心了，在上个阶段开发的自动化部署技术现在可以用来将一些新的变化直接推送到生产中去。

上述可能与现实情况并不匹配。比如，在集成了代码质量和代码覆盖率报告之前，你可能已经很了解自动化 Web 测试。然而，应该给出一个常用的思路，就是怎么做才能在现实世界里的公司中实现持续集成策略）。

接下来做什么呢

在这本书的剩余部分，我们会学习 Jenkins 提供的各种各样的功能特性，以及学习并实践如何充分利用这些功能特性，然后再一起看看如何使用 Jenkins 去做一些改善和提高。记住，本书中使用过的所有例子都可以在线上找到（详情参见 *http://www.wakaleo.com/books/jenkins-the-definitive-guide*，所以你可以亲自动手实践）。

第 2 章 迈入Jenkins的第一步

简介

在本章中，我们会简要介绍 Jenkins 的一些主要特征。你会看到安装 Jenkins 以及设置第一个 Jenkins 自动化构建作业是多么容易。在此我们不会探讨细节，接下来的章节会详细探讨具体细节，本书的最后一章（第 13 章）也会详细地探讨如何管理 Jenkins。在本章的最后，你可以观察测试结果，生成 Javadoc 并且发布代码覆盖率报告。本章还会涉及很多内容，现在就让我们开始吧！

准备环境

可以采用两种方式阅读本章。你可以只是阅读而不动手操作，仅仅对 Jenkins 有个大概的了解；你也可以一边阅读一边动手操作。

如果你想在家里学习，则可能需要在本地的机器上设置一些软件。任何持续集成工具最基本的功能是监控版本控制系统中的源代码，并在任何更改被提交时获取和构建源代码的最新版本。所以，你需要一个版本控制系统。在我们的例子中，我们将使用 Git（*http://git-scm.com*）。我们的简单项目的中心源代码库存储在 GitHub（*http://github.com*）上。不用担心代码的变化会搞乱了这个库；如果你愿意，也可以创建自己的库分支。如果你还没有使用 Git 或者没有在 GitHub 上创建账号，也不用担心，我们将讲解基本概念，GitHub 网站有完整的安装文档。我们会进一步详细地解释如何配置。

在本章中，我们将使用 Jenkins 构建基于 Maven 的 Java 应用程序。Maven 是 Java 里面广泛使用的构建工具，它具有许多强大的功能，比如声明依赖管理，配置约定，以及大量插件的支持。我们将使用最新版本的 JDK 和 Maven，但是你如果没有在自己的机器上安装这些包也没有关系，Jenkins 能够帮你安装。

安装 Java

你需要做的第一件事情就是在你的机器上安装 Java。Jenkins 是一个基于 Java 的 Web 应用程序，所以你至少需要安装 JRE。在本章的例子中，你需要最新的 Java 6 版本（这些例子基于 Java 6 update 17。在本书写作之际，最新的版本是 Java 6 update 19）。如果你不能确定版本的话，则可以从命令行（在 Windows 中打开一个 DOS 控制台）查看，并运行 `java-version`。如果你的机器安装了 Java，则应该得到类似这样的内容：

```
$ java -version
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b04-248-10M3025)
Java HotSpot(TM) 64-Bit Server VM (build 14.3-b01-101, mixed mode)
```

如果你还没有安装，或者你安装的版本较旧，请到 Java 网站（*http://java.sun.com/javase/downloads/index.jsp*）下载和安装最新的 JRE，如图 2-1 所示。

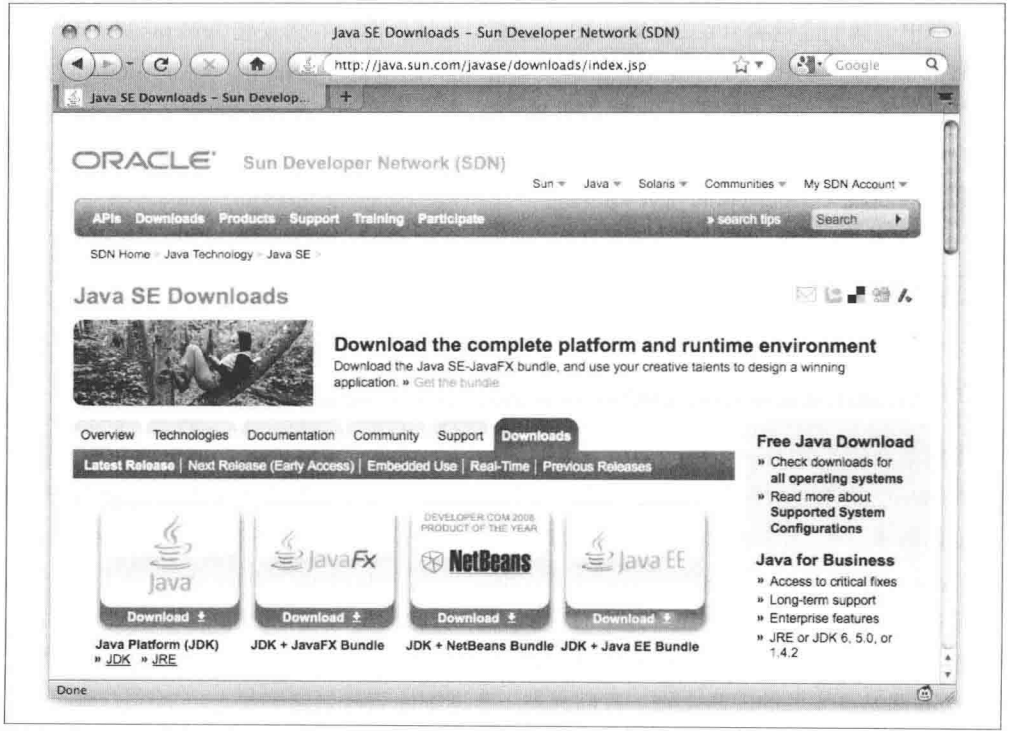


图2-1　安装Java

安装 Git

由于我们要使用 Git，因此需要在你的机器上安装和配置 Git。如果你是 Git 新手，则可能想要在 Git 参考网站（*http://gitref.org*）学习基础知识。如果你需要帮助，则整个安装过程在 GitHub 帮助页面（*http://help.github.com*）上是有据可查的。

首先需要在你的机器上安装 Git。这包括从 Git 网站（*http://git-scm.com*）根据你的操作系统下载合适的安装包。可以下载 Windows 和 Mac OS X 的安装包。如果你在使用 Linux，Git 支持大多数 Linux 发行版。如果你使用 Ubuntu 或者 Debian，你可以运行：

```
$ sudo apt-get install git-core
```

在 Fedora 或者其他基于 RPM 的发行版，你可以通过 yum 安装：

```
$ sudo yum install git-core
```

在 Linux 平台，你也可以选择以源码的方式安装。具体的操作方法请参考 Git 网站。

一旦完成，可以通过下面的命令行检查 Git 是否安装成功：


```
$ git –version
git version 1.7.1
```

创建 GitHub 账号

其次，如果你没有 GitHub 账号，则需要创建一个。这很容易（至少对于我们的目的是如此）并且是免费的。打开 GitHub 的注册页面（*https://github.com/plans*），选择“Create a free account”选项。你只需要提供用户名、密码和电子邮件地址（见图 2-2）。

配置 SSH 密钥

GitHub 使用 SSH 密钥建立计算机和 GitHub 服务器之间的安全连接。这些设置并不难，但仍需做一些工作，幸运的是在 GitHub 网站（*http://help.github.com/set-up-git-redirect*）上有每个操作系统明确和详细的说明。

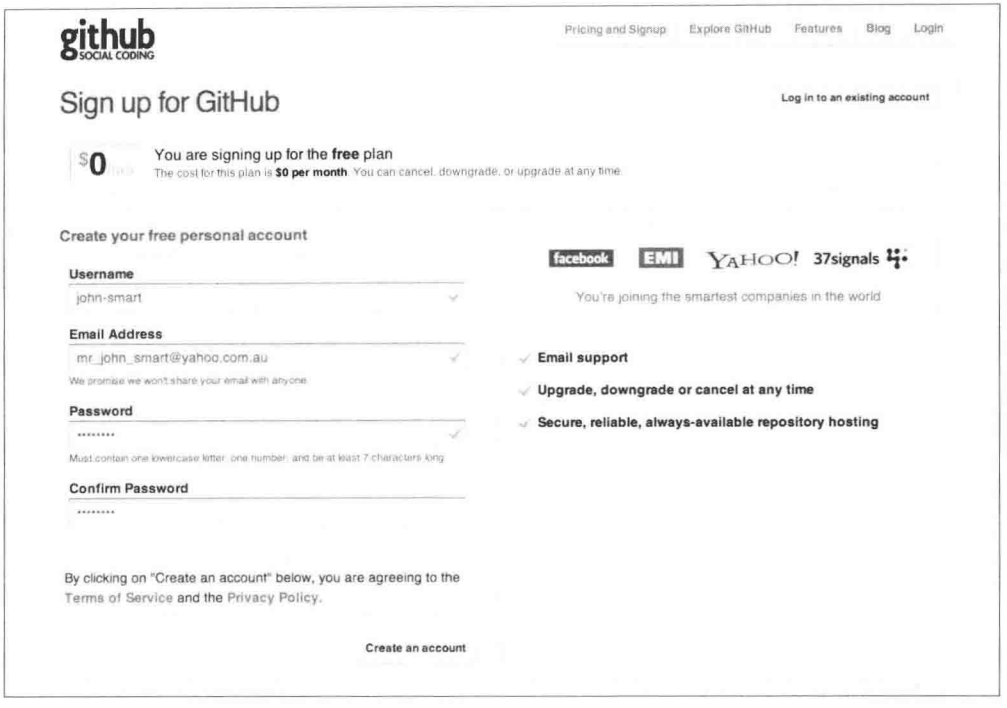


图2-2　创建GitHub账号

创建代码库分支

正如我们前面所提到的，本书的所有示例代码存放在 GitHub 上，该链接为 *https://github.com/wakaleo/game-of-life*。这是一个公共代码库，你可以在线上查看源代码，并复制自己的工作副本。但是，如果你想修改源代码的话，需要创建自己的分支。一个分支是代码库的个人副本。要创建一个分支，首先登录到你的 GitHub 账号并跳转到 repository URL。之后单击 Fork 按钮（见图 2-3）。这会创建你自己的代码库副本。

一旦你已经创建了代码库分支，则应该复制一个本地副本以确保一切配置正确。在命令行运行以下命令（用你自己的 GitHub 用户名替换 ***<username>*** ：

```
$ git clone git@github.com:<username>/game-of-life.git
```

这会在本地硬盘克隆一个项目副本（或者用 Subversion 术语，称为检出）：

```
git clone git@github.com:john-smart/game-of-life.git
Initialized empty Git repository in /Users/johnsmart/.../game-of-life/.git/
remote: Counting objects: 1783, done.
remote: Compressing objects: 100% (589/589), done.
```

```
remote: Total 1783 (delta 1116), reused 1783 (delta 1116)
Receiving objects: 100% (1783/1783), 14.83 MiB | 119 KiB/s, done.
Resolving deltas: 100% (1116/1116), done.
```

13

你现在应该有一个可以构建和执行的项目的本地副本。我们以后将使用这个项目来触发代码库中的更改。

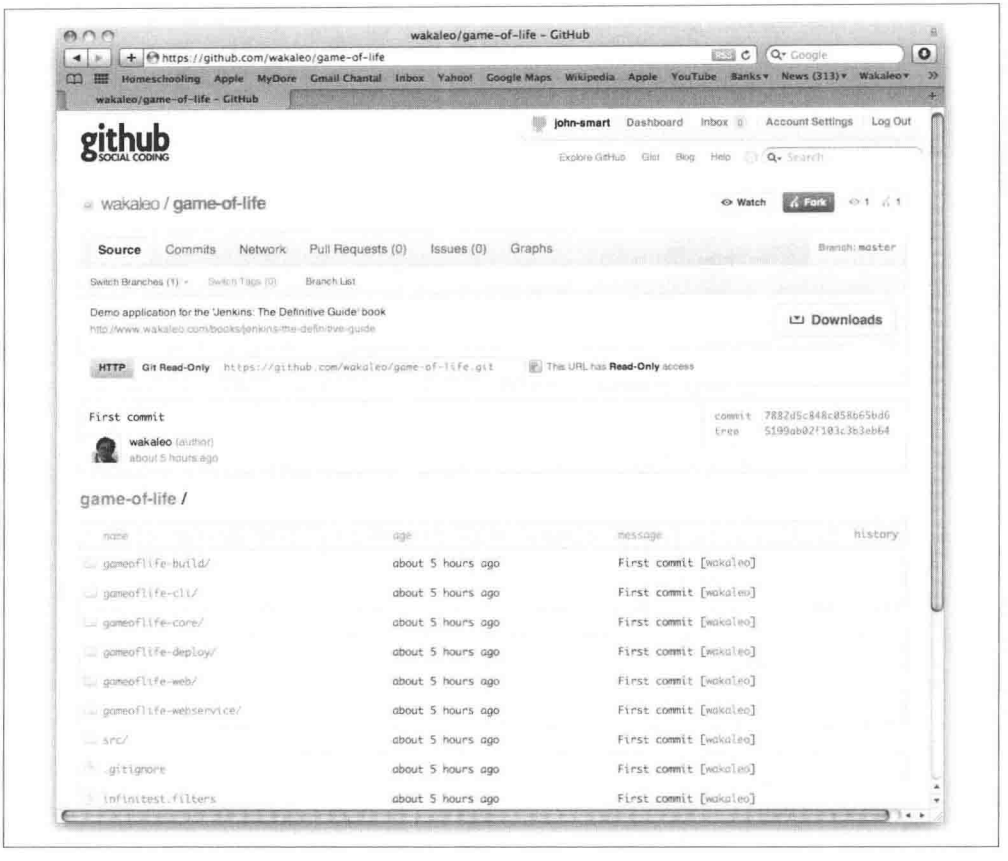


图2-3　创建源代码库分支

启动 Jenkins

有几种方法可以在你的机器上运行 Jenkins。第一次运行 Jenkins 的最简单的方法之一是使用 Java Web Start 方式。Java Web Start 是一种绑定了 Java JRE 的技术，你可以通过 Web 页面的 URL 在你的机器上启动 Java 应用程序。在我们的例子中，会在你的机器上启动 Jenkins 服务，并且你感觉像从本地安装一样。所要做的就是需要我们在前面章节安装的 Java Runtime Environment（JRE）的最新（Java 6 或更高）版本。

14

为了方便，在本书资源页（*http://www.wakaleo.com/books/jenkins-the-definitive-guide*）有个可以启动 Jenkins Java Web Start 实例的链接。在 Book Resources 部分可以看到一个大的橙色 Launch 按钮（见图 2-4）。你还可以在 Jenkins 网站（*http://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins*）的 Meet Jenkins 页面找到这个链接，向下滚动到 Test Drive，你会看到一个类似的启动链接。

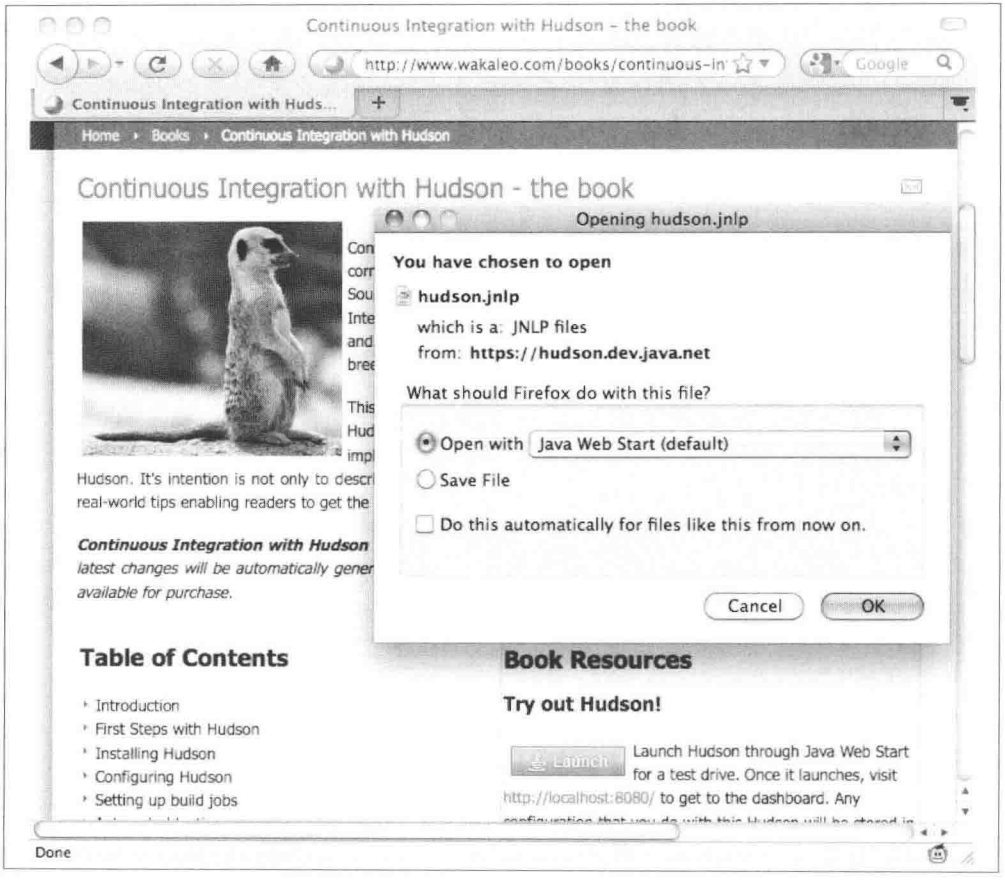


图2-4　从网站使用Java Web Start启动Jenkins

Java Web Start 在 Firefox 上支持得最好，在 Firefox 中打开网站页面，然后单击 Launch 按钮，浏览器会询问是否要使用 Java Web Start 打开 *jenkins.jnlp* 文件。单击 OK 按钮，会下载 Jenkins 并且在你的机器上启动服务（见图 2-5）。

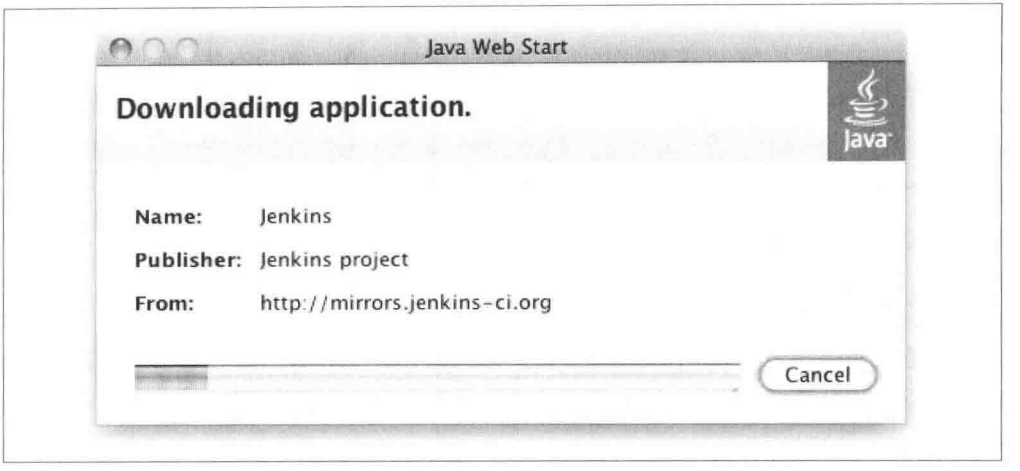


图2-5 Java Web Start下载并且运行最新版本的Jenkins

使用其他的浏览器，单击此按钮可能仅仅下载 JNLP 文件。在 Internet Explorer（IE）中，你可能需要右击链接并选择“目标另存为”来保存 JNLP 文件，然后从 Windows 资源管理器运行。然而，不管怎样，当你打开 JNLP 文件时，Java Web Start 会下载并启动 Jenkins。

Java Web Start 只需要下载一次某个版本的 Jenkins。以后，当你再次单击 Launch 按钮时，Java Web Start 使用已经下载的 Jenkins 副本（直到下个版本发布）。你可以忽略操作系统或反病毒软件提示的任何信息，在你的机器上运行 Jenkins 是相当安全的。

一旦完成了下载，它将在你的机器上启动 Jenkins。你能够看到它运行在一个被称为“Jenkins Console”的小窗口（见图 2-6）。在任何时间想要停止 Jenkins，只要关闭此窗口即可。

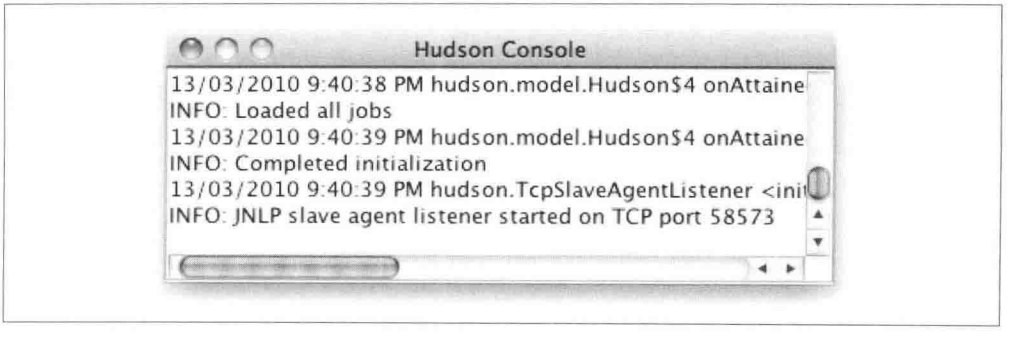


图 2-6 Java Web Start运行Jenkins

16 在 Jenkins 网站（*http://jenkins-ci.org*）有主流操作系统的安装包。如果你是一个有经验的 Java 用户并且精通 WAR 文件，你可能喜欢下载 Jenkins 的最新版本并从命令行运行它。你可以在 Jenkins 网站主页（*http://http://jenkins-ci.org*）下载到最新的可执行的 Jenkins 的 WAR 文件。为了便于参考，在本书网站（*http://www.wakaleo.com/books/jenkins-the-definitive-guide*）Resources 部分列出了 Jenkins 的新版本链接。

一旦完成下载，你可以从命令行启动 Jenkins：

```
$ java -jar jenkins.war
```

无论你是从 Java Web Start 还是从命令行启动 Jenkins，Jenkins 现在均应该运行在你的本地机器上。默认情况下 Jenkins 运行在 8080 端口，所以你可以在你的浏览器上通过 *http://localhost:8080* 访问 Jenkins。

另外，如果你熟悉 Java 应用服务器，比如 Tomcat，则可以把 Jenkins WAR 文件部署到应用程序服务器——Tomcat 上，只要把 *jenkins.war* 文件放在 Tomcat 的 *webapps* 目录下。如果你是在一个应用程序服务器上运行 Jenkins，访问 Jenkins 的网址将略有不同。在默认的 Tomcat 安装配置下，在 Web 浏览器中使用 *http://localhost:8080 /Jenkins* 访问 Jenkins。

当你在浏览器中打开 Jenkins 时，会看到一个如图 2-7 所示的屏幕。现在你已经准备好了迈向 Jenkins 的第一步！



图 2-7 Jenkins启动页面

配置工具

在开始之前，我们需要做一些配置。更确切地说，我们需要告诉 Jenkins 构建要用到的构建工具和 JDK 版本。

在主页面单击 Manage Jenkins 链接（见图 2-7），这将把你带到 Manage Jenkins 页面，为你的 Jenkins 配置提供中央一站式服务。从这个屏幕，你可以配置 Jenkins 服务器，安装和升级各种插件，跟踪系统负荷，管理分布式构建服务器，等等。现在我们先单击列表顶部的 Configure System 链接（见图 2-8）。

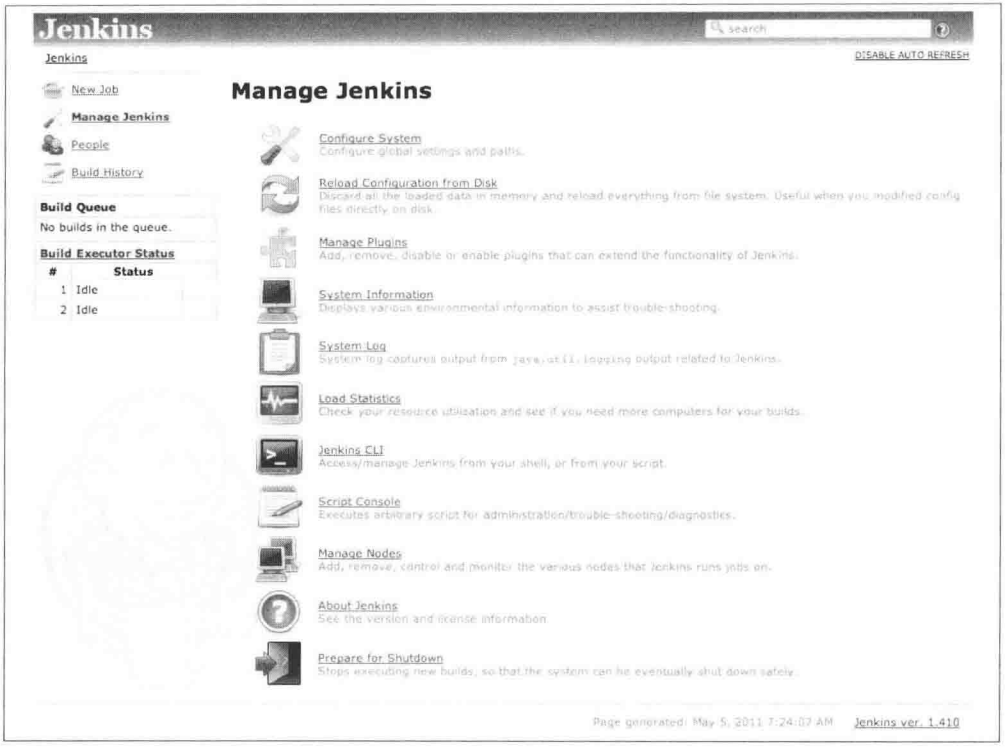


图2-8　Manage Jenkins屏幕

这会把你带到 Jenkins 的主配置屏幕（见图 2-9）。在这里你可以配置所有的东西，从安全配置、构建工具，到电子邮件服务器、版本控制系统和第三方的集成软件。该屏幕包含了很多信息，但大多数字段设置了合理的默认值，所以现在你可以安全地忽略它们。

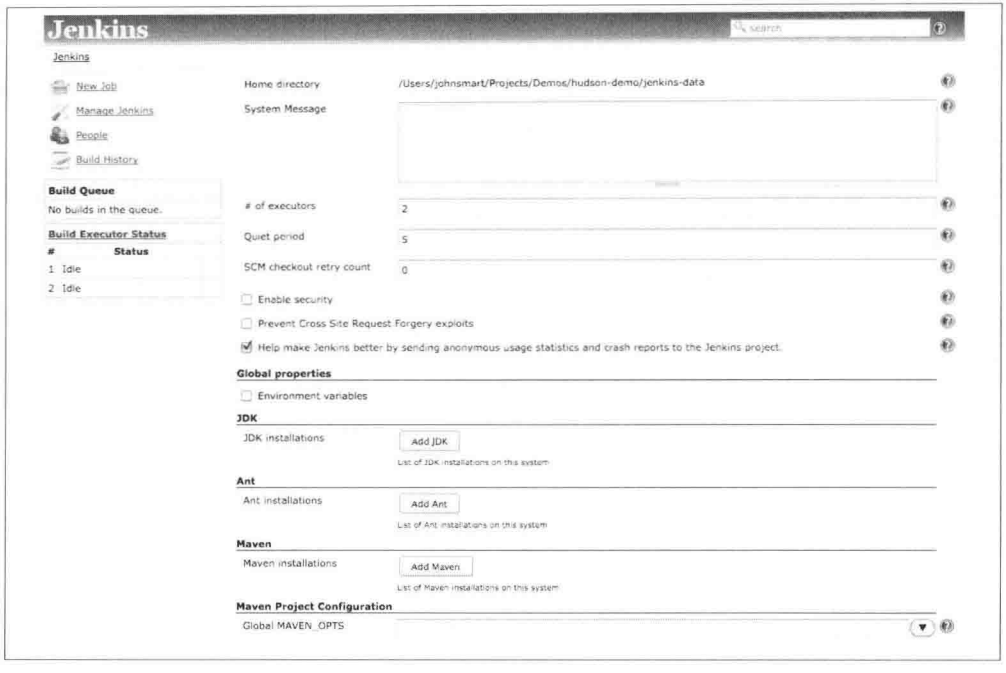


图2-9　配置Jenkins屏幕

现在，你需要配置必要的工具来构建我们的示例项目。我们将使用 Maven 构建一个 Java 应用程序。在这种情况下，我们要做的是设置最近的 JDK 和 Maven 安装。

然而在开始之前，我们先看一下屏幕右边的蓝色问号图标。这些都是 Jenkins 的上下文帮助按钮。如果你对某个特定字段感到好奇，单击旁边的 Help 图标，Jenkins 会详细描述它是什么，以及它是如何工作的。

配置你的 Maven 设置

我们的示例项目使用 Maven，所以我们需要首先安装和配置 Maven。Jenkins 对 Maven 提供了开箱支持。向下滚动，直到达到 Configure System（配置系统）屏幕的 Maven 部分（参见图 2-10）。

Jenkins 提供了几个选项来配置 Maven。如果你已经在自己的机器上安装了 Maven，则在 MAVEN_HOME 字段提供路径即可。你可以通过解压共享目录下的 ZIP 文件来安装 Maven 发行版，或者执行主目录（home）安装脚本。或者你可以让 Jenkins 完成这些工作并为你下载 Maven。勾选 Install automatically 复选框，这样 Jenkins 会在第一次构建作业需要时从 Apache 网站下载并安装 Maven。选择要安装的 Maven 的版本号，Jenkins 会

做余下的工作。你还需要给选定的 Maven 版本起个名字（在本例中叫“Maven 2.2.1”），这样你就可以在构建作业中引用它了。



图2-10　配置Maven安装

要使其工作，你需要有互联网连接。如果你使用代理服务器，则需要提供代理信息，我们将在第 4 章的“配置代理”一节讨论如何设置。

Jenkins Maven 安装过程的优点之一是，它能与远程构建代理一起工作。在本书的后面，我们将看到 Jenkins 还可以在远程构建服务器运行构建。你可以为所有的构建服务器定义安装 Maven 的标准安装方式（从互联网下载，在共享服务器解压发布包等）。当你使用此 Jenkins 配置添加新的远程构建代理或搭建新的构建服务器时，所有这些选项都能正常工作。

配置 JDK

一旦配置了 Maven 安装，你还需要配置 JDK 安装（见图 2-11）。如果你已经有 Java JDK（而不是 JRE，JDK 包含额外的开发工具，如 Java 编译器）安装在自己的工作站上，则可以在 JAVA_HOME 提供 JDK 路径。否则，你可在第一次构建作业需要它时，让 Jenkins 从 Oracle 网站（*http://www.oracle.com/technetwork/java/index.html*）下载 JDK。这是类似于自动的 Maven 安装功能——只需要选择你需要的 JDK 版本，Jenkins 做其他的事情。然而，由于协议的原因，你还需要选择一个复选框表明你同意 Java SDK 许可协议。

现在在屏幕底部，单击 Save 按钮。



图2-11　配置JDK安装

通知

你通常会设置的另一个重要方面是通知。当 Jenkins 构建中断并再次工作时，它可以发送电子邮件到整个团队来传播消息。使用插件，你还可以发送即时消息或 SMS 消息，在 Twitter 上发布帖子，或以其他方式通知大家。这一切都取决于怎样的工作方式最适合你的组织文化。如果你知道自己的本地 SMTP 服务器地址，电子邮件通知很容易设置——只要在页面配置底部的 Email Notification 部分提供此值即可。然而，为了保持事情简单，我们不打算考虑通知。

设置 Git

我们需要为此演示配置的最后一件事是让 Jenkins 使用 Git。Jenkins 开箱即用地支持 Subversion 和 CVS，但是你需要安装 Jenkins Git 插件来完成本教程的其余部分。别担心，这个过程非常简单。首先，单击屏幕左侧的 Manage Jenkins 链接回到主要配置页面（见图 2-8），然后单击 Manage Plugins。这将打开插件配置屏幕，在这里，你想管理的额外功能安装在 Jenkins 服务器上。你应该看到四个选项卡：Updates、Available、Installed 以及 Advanced（见图 2-12）。

现在，只需要单击 Available 选项卡。在这里，你会看到一个很长的可用插件列表。在此列表中查找 Git 插件条目，并勾选复选框（见图 2-13），然后向下滚动到屏幕底部，单击 Install 按钮。这将下载并安装 Jenkins Git 插件到你的本地 Jenkins 实例。

一旦完成，你将需要重新启动 Jenkins 确保所做的更改生效。要做到这一点，你可以单击显示在安装屏幕上的“Restart Jenkins when no jobs are running ”按钮，或者也可以手工关闭并重启 Jenkins。

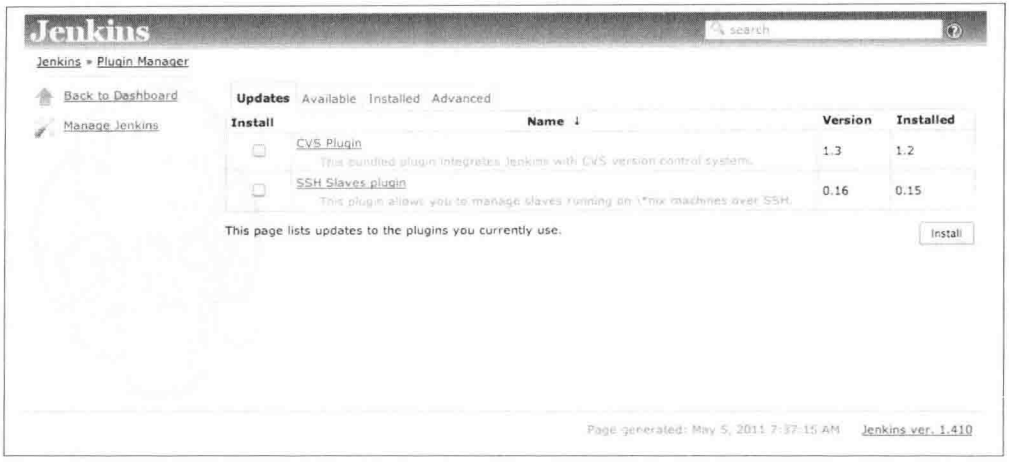


图2-12　在Jenkins中管理插件

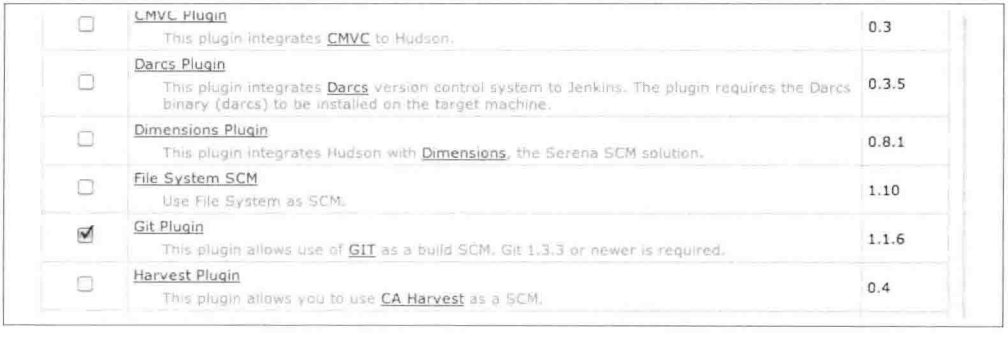


图2-13　安装Git插件

这是我们在这个阶段需要的所有配置。现在你已经准备好设置自己的第一个 Jenkins 作业了！

你的第一个 Jenkins 构建作业

构建作业是 Jenkins 构建过程的核心。简单地说，你可以认为 Jenkins 构建作业是你的构建过程的一项特定任务或步骤。这可能涉及编译你的源代码和运行单元测试。或者你可能希望构建作业来做其他相关的任务，例如运行集成测试、测量代码覆盖率或代码质量度量，生成技术文档，或者甚至将你的应用程序部署到 Web 服务器。一个真实的项目通常需要许多独立但是相关的构建作业。

22 我们的示例应用程序是“Game of Life”，这是John Conway开发的一个简单的Java实现。[注1]Game of Life是一个发生在二维网格细胞（我们称之为宇宙）的数字游戏。每一个细胞要么活着，要么死了。细胞与其直接邻居进行交互，以确定其在下一代细胞中存活还是死去。对于每个新一代的细胞，应用以下规则：

- 任何其邻居活细胞数量低于两个的活细胞都将死于不可复制。
- 任何其邻居活细胞数量多于三个的活细胞都将死于拥挤。
- 任何其邻居活细胞数量为二或者三的活细胞都将会活到下一代。
- 任何其邻居活细胞数量为三的死细胞都会变为活细胞。

我们的应用程序是一个使用Maven构建的Java模块，实现Game of Life的核心业务逻辑。我们稍后再考虑用户接口。现在，让我们看看如何在Jenkins中实现构建自动化。如果你还不熟悉Maven，或者更喜欢Ant或其他构建框架，也不必担心！示例不需要太多的Maven知识，我们会在本书的后面看到使用其他构建工具的大量实例。

对于我们的第一个构建作业，我们将保持简单：只要编译和测试示例应用程序。单击New Job链接，你应该到达一个类似于图2-14的屏幕。Jenkins支持几种不同类型的构建作业。最常用的两种是自由式和Maven 2 / 3。自由式项目允许你配置任何类型的构建作业：它们是高度灵活和很容易配置的。Maven 2 / 3构建理解Maven项目结构，可以让你花费较少的工作量来建立起Maven构建作业，同时还能支持一些额外的功能。也有为其他类型的构建作业提供支持的插件。然而，我们的项目不使用Maven，我们打算使用自由式构建作业，这只是为了让事情简单、容易上手。所以选择“Build a free-style software project”，如图2-14所示。

23

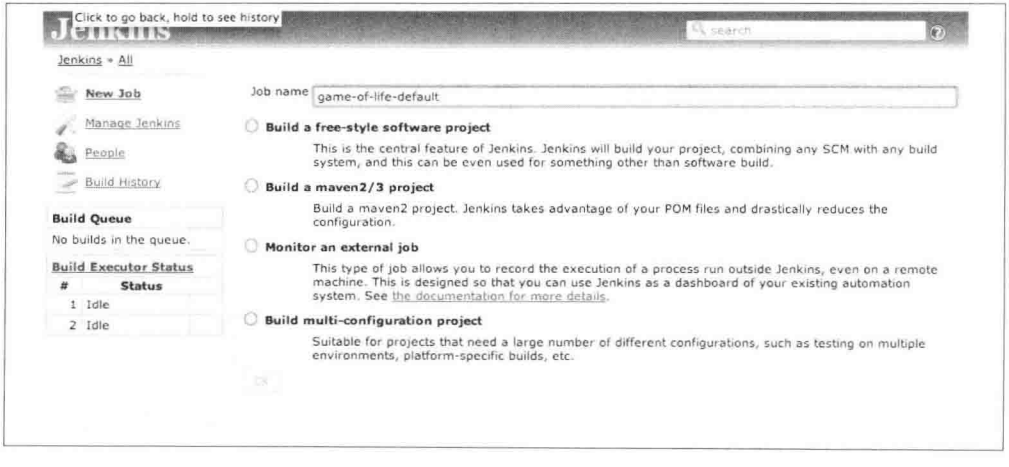


图2-14 在Jenkins里设置你的第一个构建作业

注1： 参见 *http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life*。

你还需要给构建作业起个合理的名字。在这里，我们称之为 *game-of-life-default*，它将成为我们的 Game of Life 项目默认的 CI 构建。

一旦你单击 OK 按钮，Jenkins 将显示项目配置屏幕（见图 2-15）。

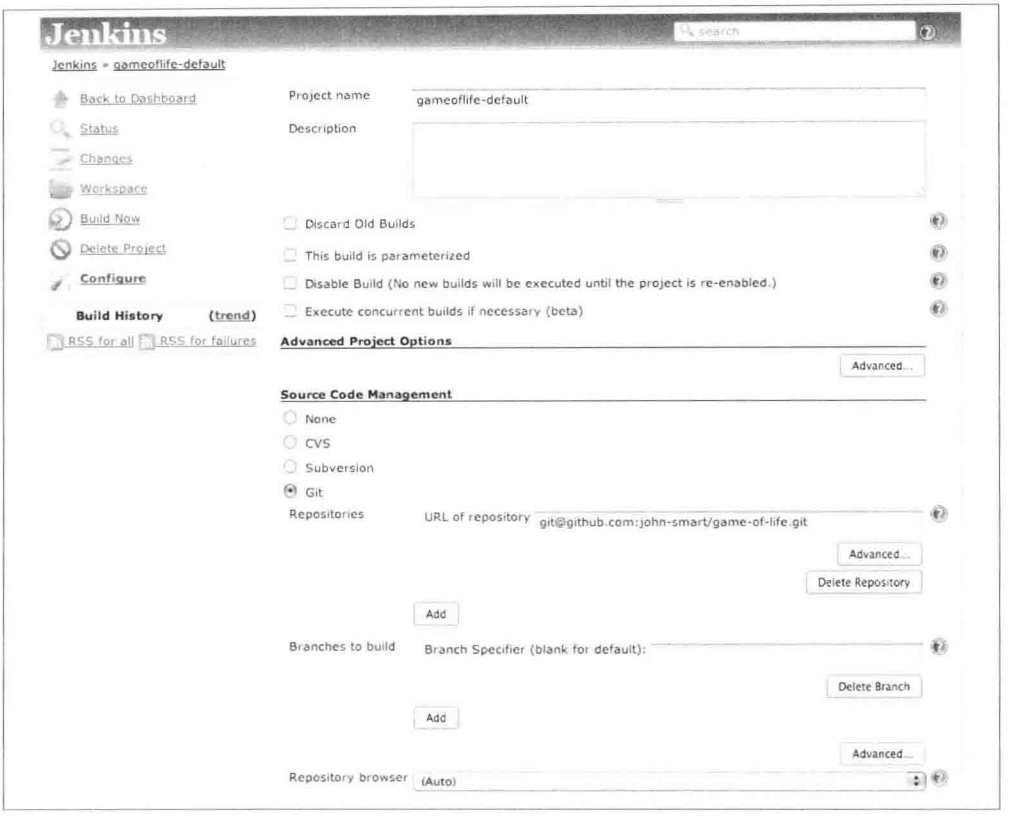


图2-15　告诉Jenkins在哪里可以找到源代码

简单地说，Jenkins 通过检查你的项目源代码并在它自己的工作区进行构建。所以你需要做的下一件事就是告诉 Jenkins，在哪里可以找到你的项目的源代码。你在源代码管理（Source Code Management）部分做必要的配置（见图 2-15）。Jenkins 支持 CVS 和 Subversion，还可以通过插件支持 Git、Mercurial、ClearCase、Perforce 等。

对于这个项目，我们会从自己之前设置的 GitHub 仓库获取源代码。在 Jenkins 屏幕上，选择“Git”并输入我们在本章前面“创建代码库分支”一节中定义的资源库 URL（见图 2-15）。请确保这是你自己分支的 URL，而不是原始资源库的 URL：它应该使用这种格式，即 *git@github.com:<username>/game-of-life.git*，在这里用你自己的 GitHub 账号用户名替换 *<username>*。你可以保留所有其他的选项都用它们的默认值。

一旦我们告诉了 Jenkins 在哪里可以找到应用程序的源代码，还需要告诉它应该多久检查一次更新。我们想要 Jenkins 监控仓库，而且任何更改被提交时就会开始构建。这是在持续集成环境中建立构建作业的常见方式，如果构建失败则它能够提供快速反馈。其他的方法包括在固定时间间隔构建（例如每天一次），或者用户手动触发，也可以在 SCM 使用“提交后”（post-commit）钩子远程触发构建。

24 我们在 Build Triggers 部分配置所有这一切（见图 2-16）。选择 Poll SCM 选项并在 Schedule 框中输入“* * * * *”（这是由空格分隔的五个星号）。Jenkins 时间表使用 UNIX 知名的 `cron` 语法。`cron` 语法包括以空格分隔的五个字段，分别指的是分钟（0 ~ 59），小时（0 ~ 23），月的一天（1 ~ 31），月（1 ~ 12），周的一天（0 ~ 7，0 和 7 是星期日）。星号是一个通配符字符，接受任何有效的值。所以五个星号意味着“哪一天当中的哪一小时的哪一分钟”。你还可以提供值的范围：“* 9-17 * * *”意味着“每天上午 9 点和下午 5 点之间的每一分钟”。你也可以使用间隔的时间空间：“*/5 * * * *”意味着“每隔 5 分钟”。最后，还有一些其他方便的快捷方式，如“@daily”和“@hourly”。

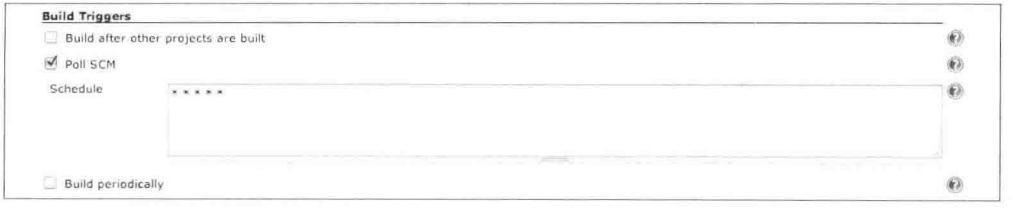


图2-16　调度构建作业

不要担心你的 UNIX 技能有点生疏，如果你单击 Schedule 框中的蓝色问号图标，Jenkins 将带来非常完整的帮助。

下一步就是配置实际的构建本身。在自由式构建作业中，你可以把构建作业分解成若干步骤。这使它更容易以清晰、独立的阶段来组织构建。比如，一个构建可能在一个步骤
25 中运行一系列的功能测试，如果所有的功能测试都成功，则在第二步为构建添加标签。用专业术语来讲的话就是，构建步骤可能涉及调用 Ant 作业或 Maven 的目标，或者运行一个 Shell 脚本。还有 Jenkins 的插件，让你使用更多类型的构建步骤：Gant、Grails、Gradle、Rake、Ruby、MSBuild 和其他构建工具。

现在，我们只想运行一个简单的 Maven 构建。向下滚动到 Build 部分，单击“Add build step”并选择“Invoke top-level Maven targets”（参见图 2-17）。然后在 Goals 字段中输入“clean package”。如果你不熟悉 Maven，这将会删除之前所有的构建产物，编译我们的代码，运行单元测试并生成 JAR 文件。

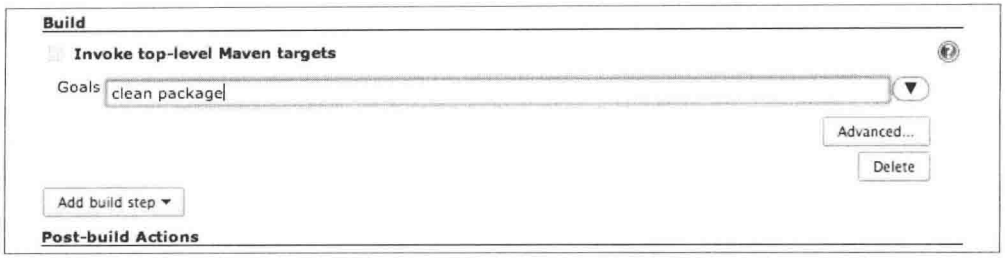


图2-17　添加构建步骤

默认情况下，如果代码没有编译或者任何单元测试失败，则此构建作业将失败。这是你对于任何一个构建服务器所期望的最基本的东西。但 Jenkins 也做了很好的工作，能够帮助你显示测试结果和测试结果的趋势。

在 Java 世界中测试报告的标准是 JUnit 采用的 XML 格式。许多其他测试工具也使用这种格式，如 TestNG、Spock 和 Easyb。Jenkins 了解这种格式，所以如果你的构建将产生 JUnit XML 测试结果，Jenkins 可以随着时间的推移生成漂亮的图形测试报告和测试结果的统计数据，还可以让你查看任何测试失败的详细信息。Jenkins 还能从整体上或者基于每个测试跟踪你的测试运行多长时间——如果你需要跟踪性能问题，这可以派上用场。

所以我们要做的下一件事是让 Jenkins 密切关注单元测试。

来到 Post-build Actions 部分（见图 2-18），勾选“Publish JUnit test result report”复选框。当 Maven 在项目中运行单元测试时，它会在目标（*target*）目录的 *surefire-reports* 目录下自动生成 XML 测试报告，所以在“Test report XMLs”字段输入“**/target/surefire-reports/*.xml”。路径开头的两个通配符（“*”）是一种最佳做法，以使配置更加健壮：无论我们如何配置 Jenkins 检出源代码，它们允许 Jenkins 总能找到目标目录。

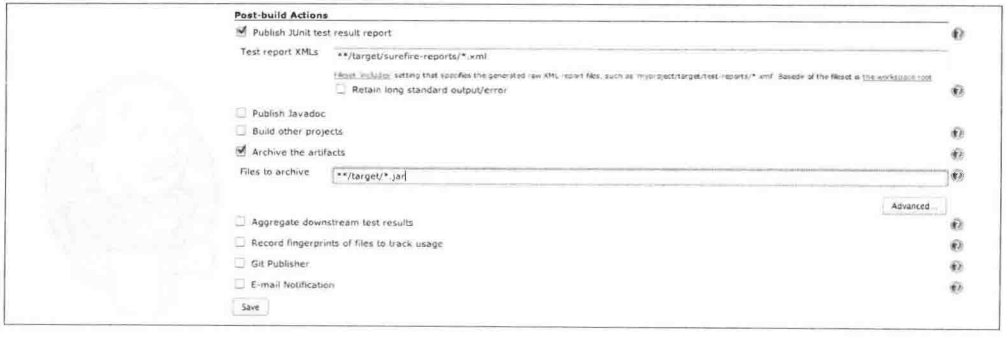


图2-18　配置JUnit测试报告和构建产物归档

你常常想做的另一件事就是把你的构建产物归档。Jenkins 可以存储构建生成的二进制构

建产物的副本，允许你从构建产物页面直接下载构建生成的二进制文件。它还在项目主页发布最新的二进制文件，这是分发你最新最好版本的应用程序的一个简便方式。你可以通过勾选“Archive the artifacts”复选框来激活这个选项并且标明想要 Jenkins 归档的二进制文件。例如在图 2-18 中，我们已经配置了 Jenkins 来存储所有由该构建作业生成的 JAR 文件。

现在已经完成——只需单击屏幕底部的 Save 按钮即可。我们的构建作业现在应该准备好运行了。因此，让我们看到它在行动！

你的第一个构建作业

一旦你保存新的构建作业，Jenkins 就会显示该作业的主页（见图 2-19）。这里 Jenkins 将显示最新构建产物和构建历史的详细信息。

如果你等待 1 分钟左右，构建就会自动触发，可以在图 2-19 左下角的 Build History 部分看到条状的进度条。或者，如果你不想等待，你还可以用 Build Now 按钮手动触发构建。

构建现在显示在 Jenkins 服务器的主页上（见图 2-20）。此页显示了你所有构建的作业，包括当前构建的状态以及每个构建的总体健康状态。它告诉你：每一个构建的最后一次成功运行时间、最后一次失败时间以及最后一次构建的结果。

27

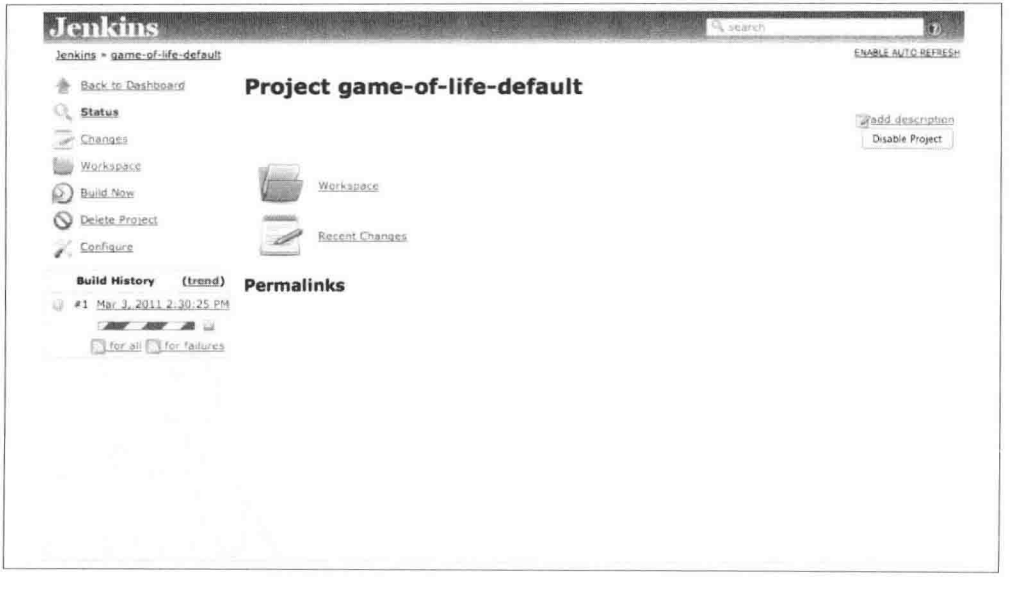


图2-19　你的第一个正在运行的构建作业

Jenkins的一个特性是它能让你获得构建随着时间而推移的情况。例如，Jenkins用天气的比喻来帮助你了解构建的稳定性。从本质上讲，你的构建失败得越多，天气变得越坏。这可以帮助你了解：某个中断的构建是一个孤立的事件；或如果该构建定期会中断，在这种情况下，它可能需要一些特别的关注。



图2-20　Jenkins仪表盘

在这里，你也可以使用构建调度按钮（就是那个看上去有点像位于时钟顶部的一个绿色播放按钮）手工触发一个构建作业。

当构建完成后，构建历史框中的球变成纯蓝色，这意味着构建成功。构建失败通常被标识为红色的球。对于某些类型的项目，你也可以区分构建错误（红色的球表示的错误，如编译器错误）和黄色的球表示的其他类型的构建失败（如单元测试失败或代码覆盖不足）。最新的测试结果还有一些其他详细信息，如最后一次构建运行时间，等等。但在查看细节之前，让我们回到持续集成服务器的核心商业模式——任何人更改代码，都会触发构建！

28

我们要使用从本章前面“创建代码库分支”一节中检出的源代码，向GitHub提交代码，然后看看会发生什么。我们已经配置好了Jenkins来监视GitHub分支；我们所做的任何更改，Jenkins应该都可以检测到。

现在，让我们做些更改，这个想法是引入会导致单元测试失败的代码更改。如果你对Java有点生疏，也不用担心，你不需要知道任何Java语法就能够中断构建——只要按照提示做就行！

现在在正常的开发流程中，你首先会修改描述这种行为的单元测试。然后，你将验证使用现有的代码会导致测试失败，修改代码以确保测试通过。之后将你的更改提交到版本控制系统，让Jenkins构建作业。然而，这并不是展示Jenkins如何处理单元测试失败的

好做法。所以在此实例中，我们并没有遵守所有的最佳实践，只是简单地直接修改应用程序代码。

首先，在 *gameoflife-core/src/main/java/com/wakaleo/gameoflife/domain* 目录，打开 *Cell.java* 文件。用你喜欢的文本编辑器打开此文件。你应该看到类似这样的内容：

```
package com.wakaleo.gameoflife.domain;

public enum Cell {
    LIVE_CELL("*"), DEAD_CELL(".");

    private String symbol;

    private Cell(String symbol) {
       this.symbol = symbol;
    }

    @Override
    public String toString() {
        return symbol;
    }

    static Cell fromSymbol(String symbol) {
       Cell cellRepresentedBySymbol = null;
       for (Cell cell : Cell.values()) {
          if (cell.symbol.equals(symbol)) {
            cellRepresentedBySymbol = cell;
            break;
          }
       }
       return cellRepresentedBySymbol;
    }
    public String getSymbol() {
        return symbol;
    }
}
```

该应用程序可以以文本数组打印网格的状态。目前，该应用程序把我们的活细胞打印为星号（*），死细胞打印为减号（-）。所以，在一个 5×5 网格中心包含单个活细胞如下所示：

```
-----
--*--
-----
```

现在用户要求对应用程序做修改——他们想要加号（+）而不是星号（*）！所以我们要稍微修改 Cell 类方法，重写为如下所示（所做的修改以**粗体**显示）：

```
package com.wakaleo.gameoflife.domain;

public enum Cell {
    LIVE_CELL("+"), DEAD_CELL(".");

    private String symbol;

    private Cell(String symbol) {
       this.symbol = symbol;
    }

    @Override
    public String toString() {
        return symbol;
    }

    static Cell fromSymbol(String symbol) {
       Cell cellRepresentedBySymbol = null;
       for (Cell cell : Cell.values()) {
          if (cell.symbol.equals(symbol)) {
            cellRepresentedBySymbol = cell;
            break;
          }
       }
       return cellRepresentedBySymbol;
    }

    public String getSymbol() {
        return symbol;
    }
}
```

保存此更改，然后通过运行 `git commit` 将其提交到本地 Git 仓库。

```
$ git commit -a -m "Changes stars to pluses"
 [master 61ce946] Changes stars to pluses
 1 files changed, 1 insertions(+), 1 deletions(-)
```

这会把更改提交到本地，但因为 Git 是分布式仓库，所以现在必须将这些更改推到你的 GitHub 上的分支。你可以通过运行 `git push` 来做到这一点：

30

```
$ git push
Counting objects: 21, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (11/11), 754 bytes, done.
Total 11 (delta 4), reused 0 (delta 0)
To git@github.com:john-smart/game-of-life.git
   7882d5c..61ce946 master -> master
```

现在回到 Jenkins 页面。一个新的构建 1 分钟左右自动触发并且运行失败。事实上，这个修改会影响几个其他的地方，和这些功能相关的回归测试失败了。在构建作业主页上，在构建历史那里你会看到第二次构建用红色的球显示（见图 2-21）——这就告诉你最新版本的构建已经失败。

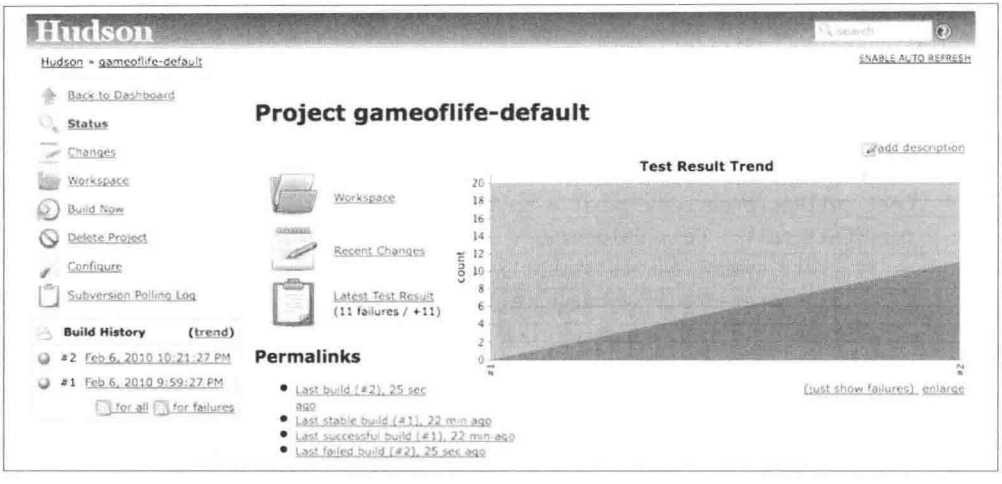


图2-21　失败的构建

你可能还注意到 Build History 标题旁边的云团——这和我们在主页看到的“天气”图标相同，并且是同样的目的——可以让你大致了解随着时间的推移构建的稳定性 。

如果你单击新的构建历史记录条目，Jenkins 会呈现更加详细的错误信息（见图 2-22）。Jenkins 告诉我们在这个构建中，有 11 个新的测试失败，这在测试结果趋势图（Test Result Trend）中更加一目了然——红色表示测试失败。你甚至可以看到哪些测试是失败的，以及它们已经中断了多久。

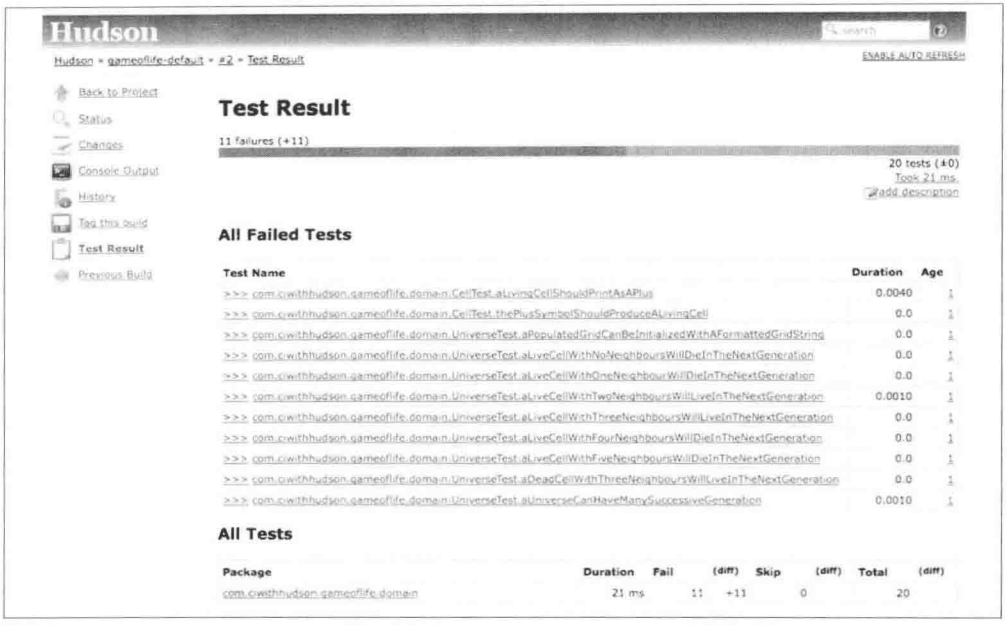


图2-22　所有中断测试的列表

如果你想要知道到底出了什么错，这也很容易。如果你单击失败的测试类，Jenkins 会显示测试失败的实际细节（见图 2-23），这对于我们重现和修复问题是很有帮助的。



图2-23　失败测试的细节

Jenkins 以非常可读的形式显示了一系列测试失败的信息，包括测试产生的错误信息、堆栈跟踪、测试已被中断多长时间，以及测试运行时间。通常情况下，这些就足以帮助开发人员正确地跟踪和修复问题。

现在让我们来修复该构建。为了让事情简单一些，我们只是回滚自己的更改，重新提交代码到它的原始状态（最终用户只需更改其在星号处的考虑）。因此，只要撤销你对 `Cell` 类所做的更改即可（再次强调，该更改将以**粗体**突出显示）：

```
package com.wakaleo.gameoflife.domain;

public enum Cell {
    LIVE_CELL("*"), DEAD_CELL(".");

    private String symbol;

    private Cell(String symbol) {
       this.symbol = symbol;
    }

    @Override
    public String toString() {
        return symbol;
    }

    static Cell fromSymbol(String symbol) {
       Cell cellRepresentedBySymbol = null;
       for (Cell cell : Cell.values()) {
          if (cell.symbol.equals(symbol)) {
            cellRepresentedBySymbol = cell;
            break;
          }
       }
       return cellRepresentedBySymbol;
    }

    public String getSymbol() {
        return symbol;
    }
}
```

当你修改完之后，再次提交自己的更改：

```
$ git commit -a -m "Restored the star"
 [master bc924be] Restored the star
1 files changed, 1 insertions(+), 1 deletions(-)
$ git push
Counting objects: 21, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
```

```
Writing objects: 100% (11/11), 752 bytes, done.
Total 11 (delta 4), reused 6 (delta 0)
To git@github.com:john-smart/game-of-life.git
   61ce946..bc924be master -> master
```

一旦你提交这些更改，Jenkins 就会发现它们并且开始构建。一旦执行完毕，你将能够在构建作业主页看到自己的工作成果（见图 2-24）——构建状态又变成了蓝色，一切均正常。你也会注意到我们正在建立的趋势图显示了随着时间的推移成功的单元测试数量——这种报告真的是 Jenkins 的强项之一。

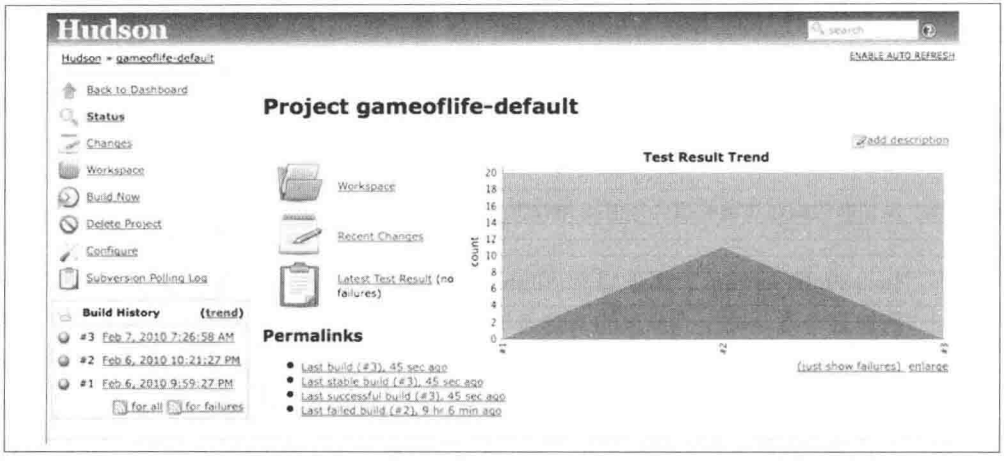


图2-24　现在构建恢复到正常状态

更多的报告——显示 Javadoc

对于许多 Java 项目而言，Javadoc 注释是低级技术文档资料的重要来源。甚至还有工具比如 UMLGraph，让你用嵌入式的 UML 图来生成 Javadoc，这样你就可以对应用程序类如何协同工作有更好的了解。这种类型的技术文档的优点是：生产成本低廉，准确并且始终保持最新。

Jenkins 能把 Javadoc API 文档直接集成到 Jenkins 网站。这样，每个人都能在众所周知的地方很容易地找到最新的 Javadoc。通常情况下，这种任务在一个独立的构建作业执行，但是为了简单起见，我们打算向 *game-of-life-default* 构建作业添加额外的构建步骤来生成和显示 Game of Life API 的 Javadoc 文档。

再次回到“game-of-life-default”配置屏幕。单击 Add build step 按钮，并添加一个新的构建步骤“Invoke top-level Maven targets”（见图 2-25）。在 Goals 字段中输入 `javadoc:javadoc`——这将告诉 Maven 生成 Javadoc 文档。

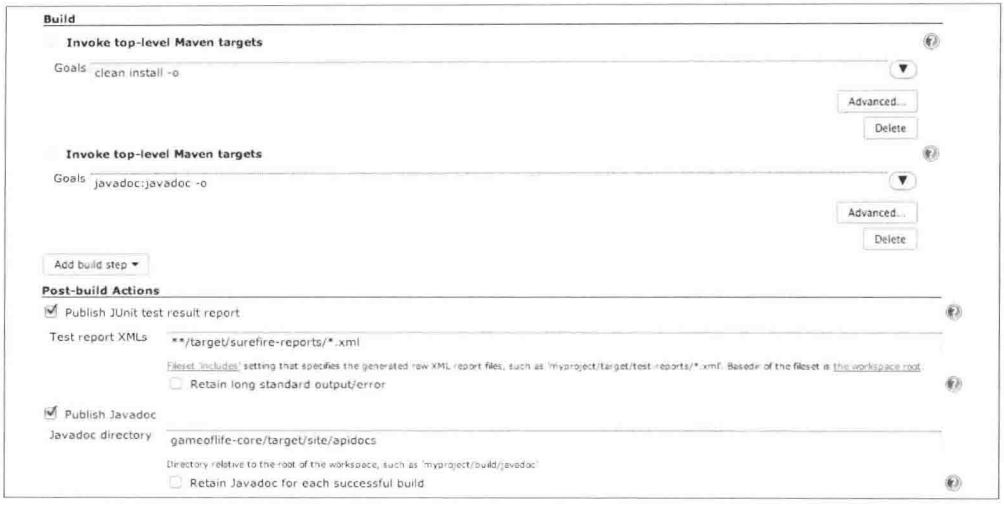


图 2-25　增加新的构建步骤和报告来生成Javadoc

现在在“Post-build Actions”部分勾选“Publish Javadoc”复选框。这个项目是一个多模块项目，因此每个模块都生成单独的子目录（核心、服务、网络，等等）。对于这个示例，我们对显示核心模块的文档感兴趣。在 Javadoc 目录字段，输入 **`gameoflife-core/target/site/apidocs`**——这是 Maven 用来放置为核心模块生成的 Javadoc 的地方。第一次，Jenkins 可能会显示错误信息，称该目录不存在。Jenkins 是正确的——直到我们运行 `javadoc:javadoc` 的目标，该目录才存在，但由于我们还没有运行此命令，所以在这一阶段我们可以安全地忽略该信息。

如果你勾选“Retain Javadoc for each successful build”，Jenkins 还将保留以前构建的 Javadoc——不总是有用，但是它有可以派上用场的时候。

现在手工触发一个构建。你可以从构建作业主页（使用 Build Now 链接），或直接从服务器主页来手工触发构建。一旦构建完成，打开构建作业的总结页面。现在，你应该看到在屏幕上突出显示的 Javadoc 链接——此链接将打开 Javadoc 文档的最新版本（见图 2-26）。如果你让 Jenkins 为每一个构建保存 Javadoc，在构建的详细信息页面也能看到这个链接，它指向该特定构建的 Javadoc。

添加代码覆盖率和其他度量

正如我们之前提到的，生成报告是 Jenkins 的强项之一。我们已经看到它显示测试结果并发布 Javadoc 是多么容易，但你也可以使用 Jenkins 的插件发布大量的其他非常有用的报告。

插件是 Jenkins 的另一个卖点——插件可以做任何事情，从集成新的构建工具或者版本控制系统到通知机制和报告。此外，Jenkins 插件很容易安装并且和现有的 Jenkins 架构无缝地集成。

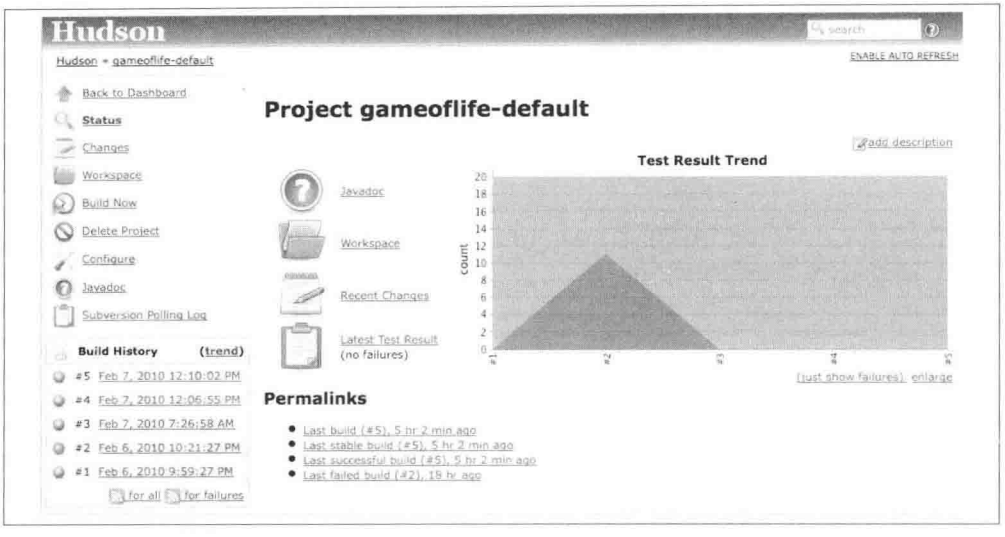


图2-26　Jenkins为构建产物添加Javadoc链接

为了查看这些插件是如何工作的，我们打算使用 Cobertura 插件来集成代码覆盖率度量。代码覆盖率表明在测试过程中有多少应用程序的代码实际被执行——它是一个非常有用的工具，尤其是在用于去发现那些未经你的测试集测试的代码区域的时候。它也可以表明，一个团队应用良好的测试实践方法（如测试驱动开发或行为驱动开发）的程度。

Cobertura（*http://cobertura.sourceforge.net*）是一个支持 Maven 和 Jenkins 开放源代码覆盖率的工具。我们的 Maven 示范项目已经配置记录代码覆盖率，所以我们要做的是安装 Jenkins Cobertura 插件并为 Jenkins 记录和显示生成代码覆盖率度量。

若要安装新的插件，可到 Manage Jenkins 页面，单击 Manage Plugins 条目。这会显示可用的插件以及在你的服务器上已经安装的插件列表（见图 2-27）。如果你的构建服务器没有互联网链接，则你还可以从别的地方下载插件并将其上传到 Jenkins 安装目录（打开 Advanced 选项卡，见图 2-27）或者将插件复制到 *$JENKINS_HOME/plugins* 目录，然后手工安装。

在本例中，我们对 Cobertura 插件感兴趣，所以打开 Available 选项卡并向下滚动，直到你在 Build Reports 部分找到 Cobertura Plugin 条目。勾选该复选框，然后单击屏幕底部的安装按钮。

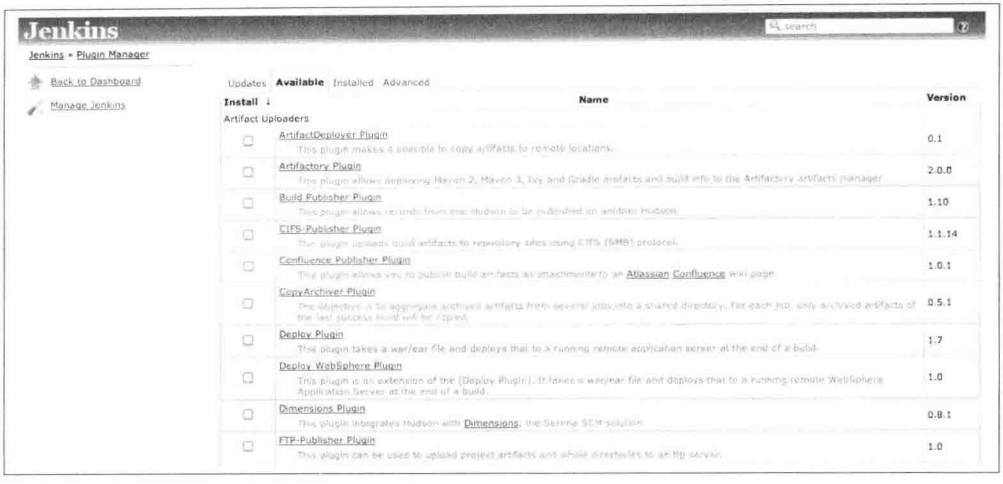


图2-27　Jenkins有大量可用的插件

这会为你下载并安装插件。一旦安装完成，你将需要重启 Jenkins 实例来查看自己的工作成果。当你重启 Jenkins 之后，回到 Manage Plugins 屏幕并且单击 Installed 选项卡——现在，在此页面上已安装插件的列表里应该有 Cobertura Plugin 条目。

一旦你确定插件已成功安装，则去 *game-of-life-default* 构建作业的配置页。

若要在我们的项目中设置代码覆盖率度量，需要做两件事。首先，我们需要以 Jenkins 可以使用的 XML 格式生成 Cobertura 覆盖率数据；然后，我们需要配置 Jenkins 显示覆盖率报告。

我们的 Game of Life 项目已经配置了生成 XML 代码覆盖率报告。所有你需要做的是运行 `mvn cobertura:cobertura` 以生成 XML 格式的报告。Cobertura 也能生成 HTML 格式的报告，但在我们的例子中让 Jenkins 负责处理报告，这样能节省构建的时间。为简单起见，我们将 `cobertura:cobertura` 目标添加到第二个构建步骤里（见图 2-28）。你还可以只为代码覆盖率度量添加一个新的构建步骤。在实际项目中，代码质量度量通常放在一个不同的构建作业中，低于默认的构建运行的频率。

接下来，我们需要让 Jenkins 跟踪代码覆盖率度量。向下滚动到“Post-build Actions”部分。在此可看到 Publish Cobertura Coverage Report 复选框。当你安装新的插件时，Jenkins 经常会添加像这样的 UI 元素。当你勾选此复选框时，Jenkins 将显示早前安装的 Cobertura 插件的配置选项（见图 2-29）。

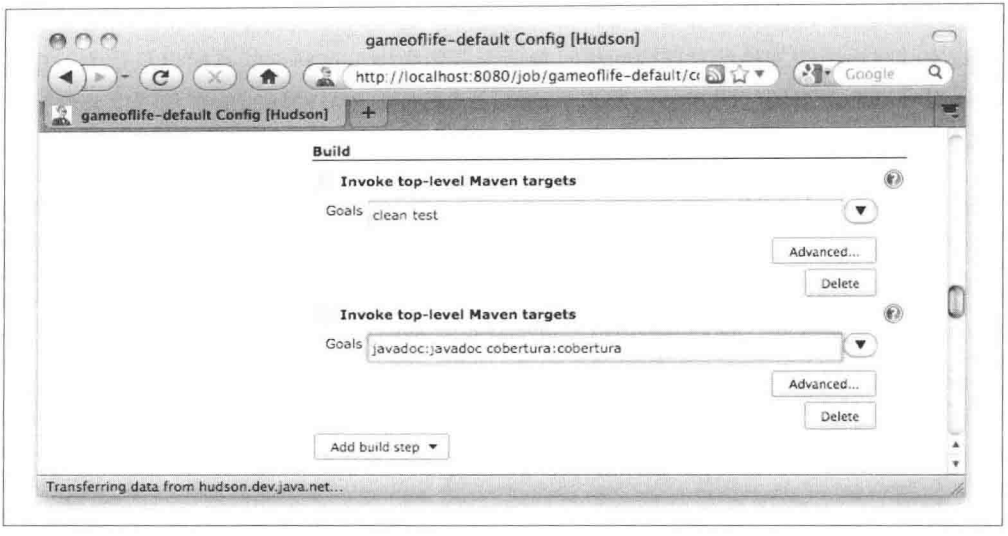


图2-28　将另一个Maven目标添加到生成测试覆盖率度量

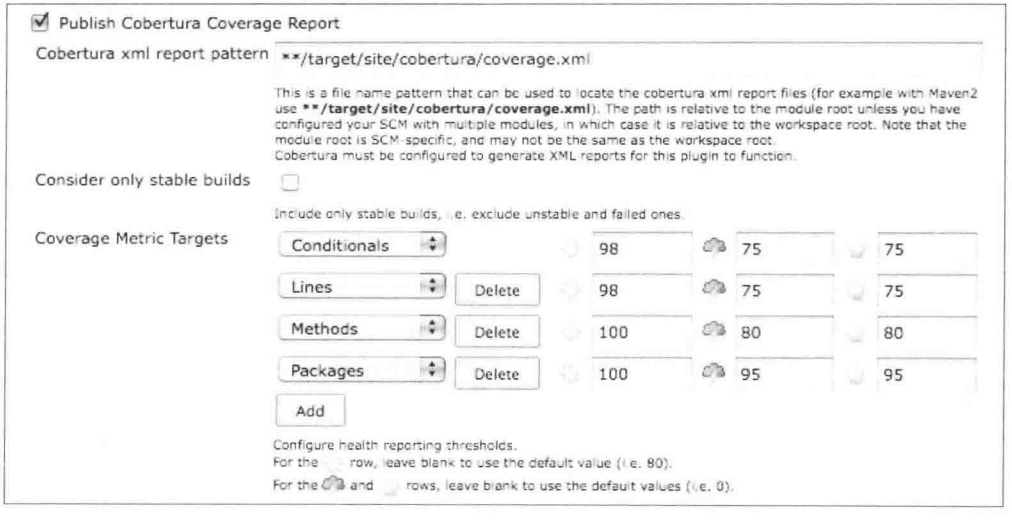


图2-29　在Jenkins中配置测试覆盖率度量

像其他 Jenkins 支持的代码质量相关的插件一样，Cobertura 插件允许你微调 Jenkins 显示报告数据和如何解释数据。在 Coverage Metrics Targets 部分，可以定义你认可的代码覆盖率的最低可接受水平。在图 2-29 中，我们让 Jenkins 列出低于 50% 测试覆盖率的构建，认为这些是不稳定的（以黄色的球表示）并且相应地通知团队。

微调经常在我们的实际构建中派上用场。例如，你可能想在发布版本强制实施一个特定

的代码覆盖率，以确保发布版本的高代码覆盖率。另一种对于遗留项目有帮助的策略就是随着时间逐步增加最低可接受的代码覆盖率范围级别。这样你能够避免只是为了提高代码覆盖率而对遗留代码增加大量的单元测试，但是你要鼓励所有的新代码以及修复的问题都得到很好的测试。

现在手工触发一个构建。你第一次运行配置了 Cobertura 报告的构建作业，你会在构建主页看到覆盖率统计信息，并提供了 Coverage Report 链接可以看到更多的细节（见图 2-30）。Cobertura 报告显示了我们运行的构建不同类型的代码覆盖率。因为我们仅运行一次测试覆盖率度量，覆盖率会以红色和绿色的柱形图显示。

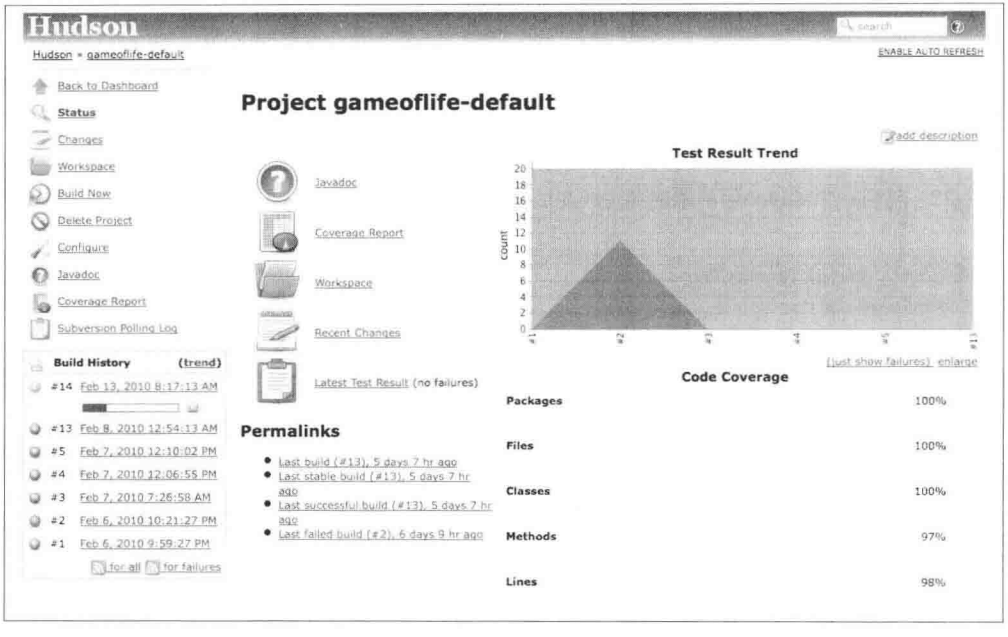
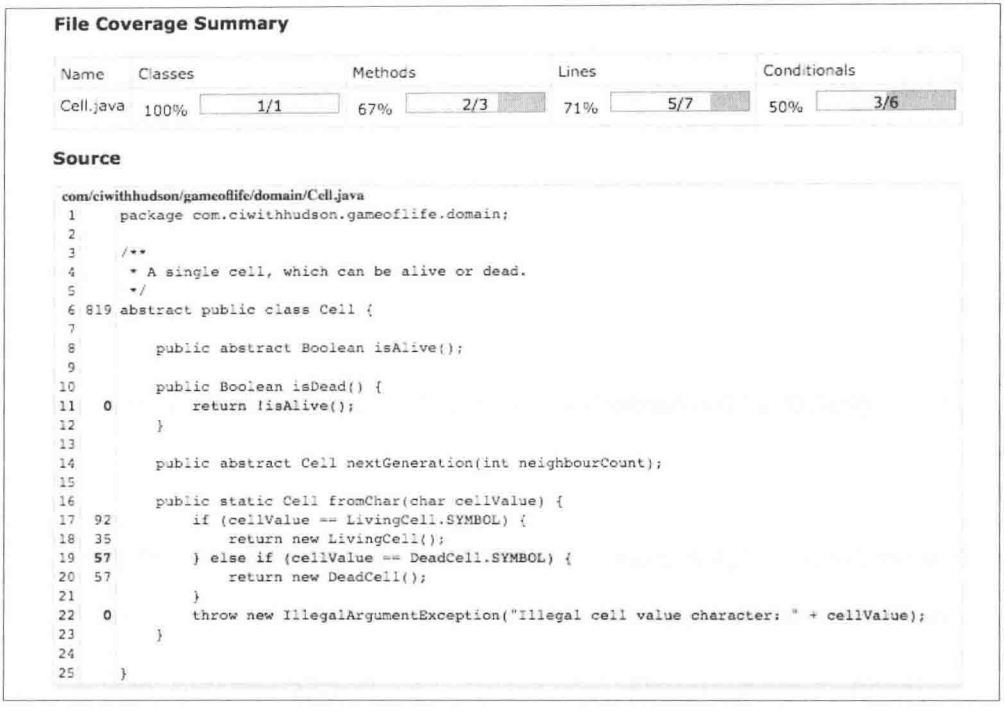


图2-30　Jenkins在构建主页显示代码覆盖率度量

如果你单击 Coverage Report 图标，则会看到应用程序每个包里面的代码覆盖率，甚至可以查看每个类的代码覆盖率（见图 2-31）。当你查看这一级别时，Jenkins 显示这个类的总体覆盖率统计信息，并且会把已经执行的行标注为绿色，没有执行的行标注为红色。

随着时间的推移，这个报告的作用日益明显。Jenkins 不仅报告最新构建的度量数据，而且会随着时间的推移跟踪度量的变化，这样你就能够看到整个项目生命周期它们是如何演变的。



File Coverage Summary

Name	Classes	Methods	Lines	Conditionals
Cell.java	100% 1/1	67% 2/3	71% 5/7	50% 3/6

Source

com/ciwithhudson/gameoflife/domain/Cell.java

```
 1      package com.ciwithhudson.gameoflife.domain;
 2
 3      /**
 4       * A single cell, which can be alive or dead.
 5       */
 6  819 abstract public class Cell {
 7
 8          public abstract Boolean isAlive();
 9
10          public Boolean isDead() {
11    0         return !isAlive();
12          }
13
14          public abstract Cell nextGeneration(int neighbourCount);
15
16          public static Cell fromChar(char cellValue) {
17    92        if (cellValue == LivingCell.SYMBOL) {
18    35            return new LivingCell();
19    57        } else if (cellValue == DeadCell.SYMBOL) {
20    57            return new DeadCell();
21              }
22    0         throw new IllegalArgumentException("Illegal cell value character: " + cellValue);
23          }
24
25      }
```

图2-31　Jenkins能够显示包以及类的代码覆盖率度量

例如，如果你深入查看覆盖率报告，就会注意到此代码的某些部分没有测试过（例如图2-31的`Cell.java`类）。

代码覆盖率度量对于隔离尚未经过测试的代码非常有用，比如可以为在初始开发的过程中某些并没有得到正确测试的案例添加额外的测试。Jenkins代码覆盖率图也是一个很好的方式：你能随着项目的增长跟踪你的代码覆盖率度量。事实上，当你添加新的测试时，就会发现Jenkins将随着时间的推移显示代码覆盖率图，而不只是最新的结果（见图2-32）。

请注意，在这里我们的目标不是为了提高代码覆盖率而提高代码覆盖率——我们增加一些额外的测试来验证那些没有被测试过的代码，这样代码覆盖率就提高了。这里还有微妙但重要的区别——代码覆盖率，与任何其他的度量一样，在很大程度上是一个达到目的的手段（高的代码质量和低廉的维护成本），而不是目的本身。

尽管如此，这样的度量也可以帮助你深入了解项目的健康状态，并且Jenkins会用一种非常容易的方式来展现它们。

这仅仅是为Jenkins编写的代码质量度量插件之一，其实还有很多插件（在本书写作之际，有50多个插件）。我们会在第9章再介绍一些插件。

40

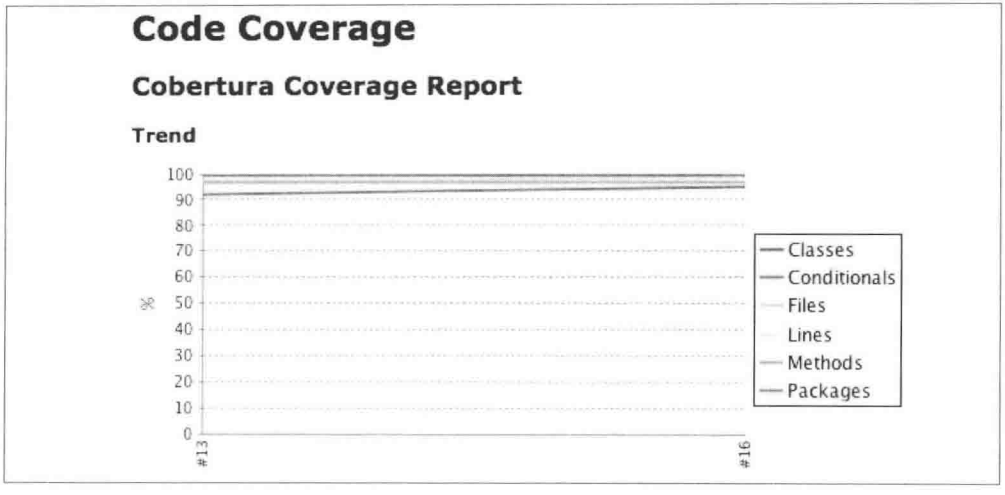


图2-32 Jenkins显示的随时间推移的代码覆盖率图

小结

在本章中，我们已经讲解了开始构建Jenkins的必要知识。现在你应该能够设置新的构建作业了，并且可以设置JUnit测试结果报告以及Javadoc了。你已经看到了如何添加报告插件和监视代码覆盖率。干得好！但Jenkins还有更多内容需要了解——在下面的各章中，我们将看到Jenkins在许多其他领域如何帮你提高构建自动化进程。

第 3 章 41

安装Jenkins

简介

你可能注意到的第一件关于 Jenkins 的事情是它的安装有多么简单。确实，在不到 5 分钟的时间内你就可以安装和运行一个 Jenkins 服务器。然而，通常在现实世界里，事情并不总是那么简单，当你安装 Jenkins 服务器用于生产时，有一些细节需要考虑。在本章中，我们将一起看看如何安装 Jenkins 到本地服务器或者完全成熟的构建服务器上。我们也会探讨一旦 Jenkins 运行起来应该怎样照料这个 Jenkins 安装，以及怎样执行基本的维护任务（如备份和升级）。

下载和安装 Jenkins

Jenkins 易于安装，且在任何地方都可以运行。你可以把它当作一个独立的应用程序进行运行，或者部署在传统的 Java 应用服务器（如 Tomcat 和 JBoss）上。支持单机部署的选项使得它易于在本地机器上安装和试验，你可以在大约几分钟内就启动运行一个基础的安装。

因为 Jenkins 是一个 Java 应用程序，所以需要在机器上运行最新的 Java 版本。更准确地说，你至少需要安装 Java 5。事实上，在构建服务器上，你几乎肯定会需要 Java 开发工具包（JDK）5.0 的所有特性或其他更好的特性去执行自己的构建。如果你不能确定，则可以在自己的机器上通过执行 `java -version` 命令检查 Java 版本：

```
$ java -version
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b04-248-10M3025)
Java HotSpot(TM) 64-Bit Server VM (build 14.3-b01-101, mixed mode)
```

42 Jenkins是以分布式的形式打包Java Web应用程序的（一个WAR文件）。你可以从Jenkins官网（*http://jenkins-ci.org*，见图3-1）或本书的网站下载最新版本。Jenkins是一个动态的项目，新版本会定期发布。

对于Windows用户，有一个带图形界面的Windows安装包用于Jenkins。安装程序被打包成一个ZIP文件的形式，其中包含一个Jenkins的MSI包。假如你的机器还没有安装.NET库的话，还有一个可以帮助你安装.NET库的*setup.exe*文件。在大多数情况下，你所需要做的就是解压ZIP文件并运行里面的*jenkins-x.x.msi*（见图3-2）。MSI安装程序捆绑了JRE，所以不需要单独的Java安装。

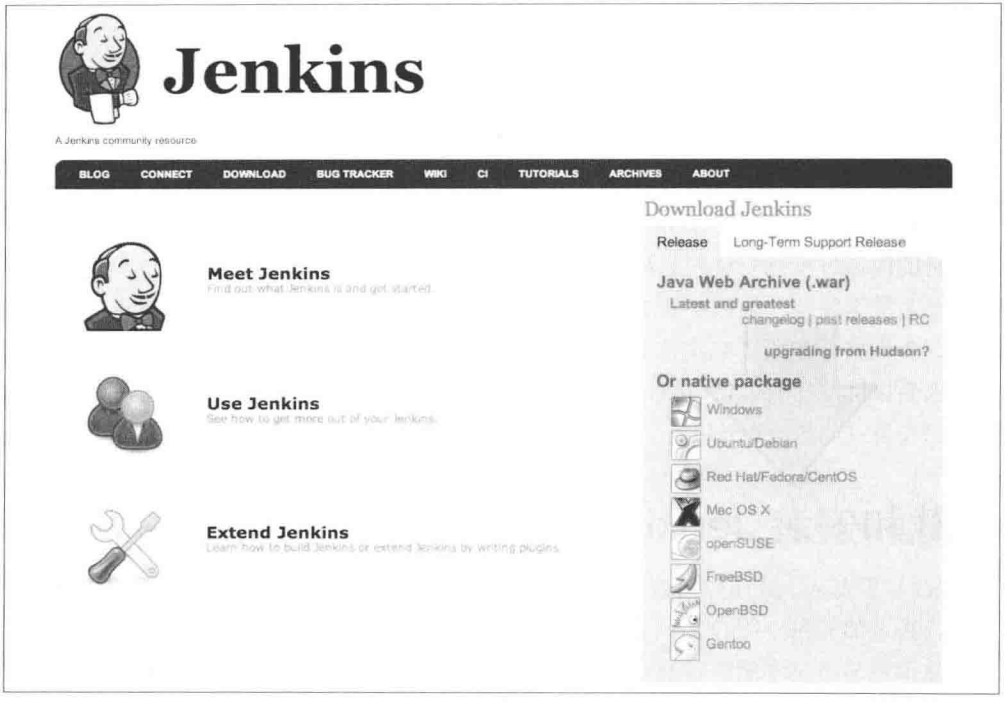


图3-1 你可以从Jenkins官网上下载Jenkins二进制文件

一旦你运行了安装程序，Jenkins会自动运行在8080端口上（见图3-3）。安装程序将为你创建一个新的Jenkins服务，你可以像其他Windows服务一样启动和停止它。

也有优秀的本地包提供给Mac OS X和其他大多数主要的Linux分支，包括Ubuntu、Red Hat（包括CentOS和Fedora）及Open Solaris。下面将讨论如何在Ubuntu和Red Hat上安装Jenkins。

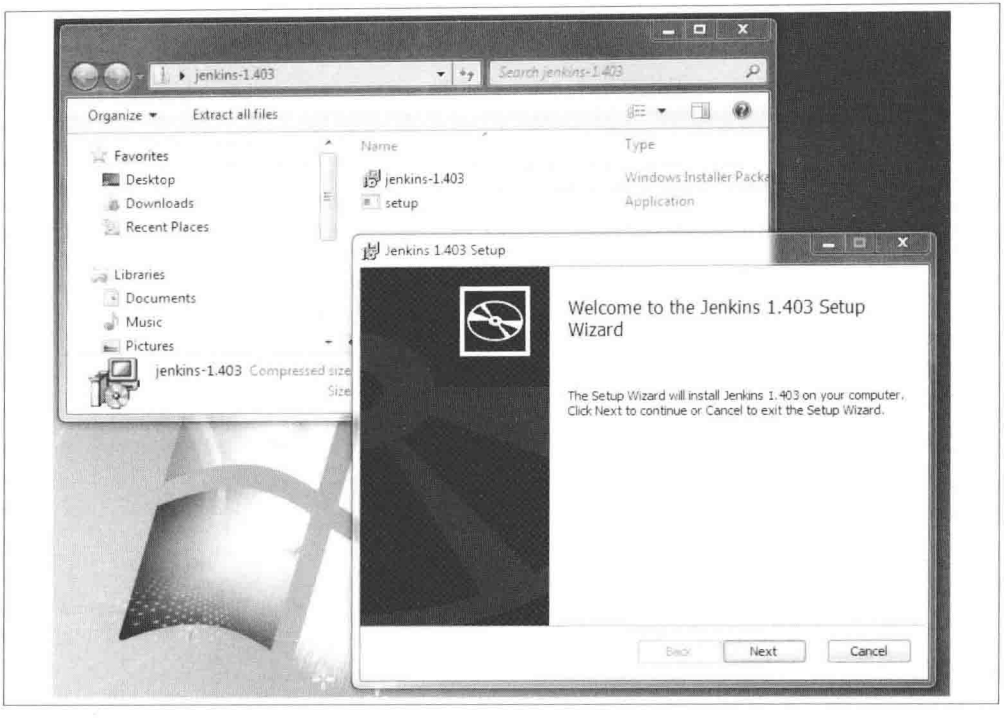


图 3-2　在Windows中的Jenkins安装向导

假如你没有使用本地安装包来安装 Jenkins 的话，则可以简单地从 Jenkins 官网上下载最新的二进制发行版。下载了最新的 Jenkins 发行版后，把它放在你构建服务器上合适的目录下。在 Windows 环境下，你可能把它放在名为 *C:\Tools\Jenkins*（一个好的做法是不把 Jenkins 放在一个包含空格的路径中，如 *C:\Program Files*，因为一些情况下这样做会为 Jenkins 带来问题）的目录下。在 Linux 或 UNIX 上，它可能会在 */usr/local/jenkins*、*/opt/jenkins* 或者一些其他目录下（依赖于你的习惯或者系统管理员当时的想法）。

在我们继续下去之前，先启动 Jenkins 来看一看。如果你在前面的章节没有试一试的话，现在是时候上手练习一下了。在包含文件的目录下打开一个控制台并且运行以下命令：

```
java -jar jenkins.war
[Winstone 2008/07/01 20:54:53] - Beginning extraction from war file
...
INFO: Took 35 ms to load
...
[Winstone 2008/07/01 20:55:08] - HTTP Listener started: port=8080
[Winstone 2008/07/01 20:55:08] - Winstone Servlet Engine v0.9.10 running:
    controlPort=disabled
[Winstone 2008/07/01 20:55:08] - AJP13 Listener started: port=8009
```

Jenkins 现在应该已经运行在 8080 端口上了，打开你的浏览器，在 *http://localhost:8080* 看一下（见图 3-3）。

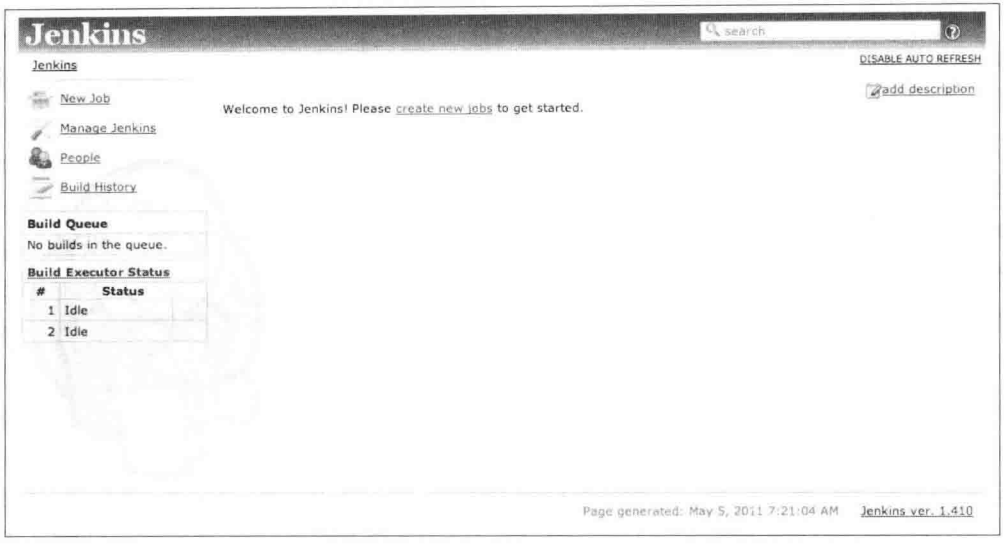


图 3-3　Jenkins启动界面

为 Jenkins 准备好构建服务器

在本地部署机器上安装 Jenkins 是一件简单的事情，但怎样在一个合适的构建服务器上安装 Jenkins 就需要你多花些心思了。

在你开始安装之前，首先需要的是一个构建服务器。为了工作正常，Jenkins 需要足够的处理器能力和内存资源。Jenkins 本身是一个相对适度的 Java Web 应用程序。然而，在大多数的配置里，至少会有一些构建将主要运行在构建服务器上。构建往往是一些内存和处理器密集型的操作，Jenkins 可以配置为并行地运行几个构建。依赖于你管理的构建作业的数量，Jenkins 也需要拥有自己的内存用于自身使用。所需要的内存数量很大程度上取决于你构建的性质，但现在内存已经很便宜了（至少在非宿主环境下），最好不要吝啬。

一个构建服务器也需要 CPU 马力。一般来说，每个并行的构建都需要一个处理器，尽管在实践中，你可以利用 I/O 延迟做得更好。尽可能多地贡献你的构建服务器给运行持续构建的作业，这会让你获取最佳利益。特别地，你需要避免使用内存或 CPU 密集型应用，如测试服务器、使用率高的企业级应用，以及企业级数据库（如 Oracle、企业级邮件服务器等）。

在现今的许多公司中，一个实用的选择是使用一个虚拟机。使用这种方式，可以为你的初始化安装选择自己认为合适的内存大小和处理器数量，并且可以在之后需要的时候很容易地添加更多的内存和处理器。另外，假如使用虚拟机的话，需要确保有足够的内存来支持你期望运行并发构建的最大数量。持续集成服务器的内存使用就是最佳描述——Jenkins 会根据构建作业的需求创建额外的 JVM，这些都需要内存。

另一个有用的方法是设置多个构建服务器。Jenkins 能非常简便地在其他机器上设置“slaves”，并将它们用于运行额外的构建工作。这些“slaves”会保持不活跃的状态，直到有一个新的构建工作请求——然后 Jenkins 把构建作业分派到 slave 节点上并报告结果。这是应付突然飙升的构建活动（比如说在主要产品发布之前）的一个非常好的方式。如果某些重量级的构建有破坏主要的构建服务器的倾向时，它也是一个非常有用的策略——就把它们放在它们自己专用的构建代理上。我们将在本书的后面看看怎么在细节上做到这些。

如果你是在 Linux 或 UNIX 构建服务器上安装的 Jenkins，一个好的做法是为 Jenkins 创建一个特殊的用户（和用户组）。这使得它在监控 Jenkins 构建所使用的系统资源及在现实条件下排查问题构建方面变得更简单。下面要讨论的本地二进制安装包就是做这件事的。如果你没做过上述两件事，则可以通过下面展示的命令行创建一个专用的 Jenkins 账户：

```
$ sudo groupadd build
$ sudo useradd --create-home --shell /bin/bash --groups build jenkins
```

具体细节可能取决于你的环境。比如，你可能更喜欢图形化的管理控制台而不是命令行；或者在一个基于 Debian 的 Linux 服务器（比如 Ubuntu）上，你可以使用更友好的 `adduser` 和 `addgroup` 命令。

在大多数环境下，你需要正确地为用户配置 Java。例如，你可以通过把 `JAVA_HOME` 和 `PATH` 变量定义在 *.bashrc* 文件中来做这件事，如下：

```
export JAVA_HOME=/usr/local/java/jdk1.6.0
export PATH=$JAVA_HOME/bin:$PATH
```

现在你就可以在一个独立的环境中使用这个用户运行 Jenkins 了。

Jenkins 主目录

46

在我们安装 Jenkins 之前，你需要知道关于 Jenkins 怎样存储它的数据的一些事。事实上，不管你把 Jenkins 的 WAR 文件存放在哪儿，Jenkins 都把其所有重要数据存放在一个专用的、隔离的，称为 Jenkins 主目录的目录下。在这里，Jenkins 存储关于构建服务器的

配置信息、构建作业、构建产物和其他有用的信息，当然也包括你安装了的任何插件。Jenkins 主目录格式是跨版本向后兼容的，所以你可以自由地更新或者重新安装 Jenkins 而不影响 Jenkins 主目录。

不用说，这个目录将会耗费大量的磁盘空间。

默认情况下，Jenkins 主目录被称为 *.jenkins*，会被放置在你的当前用户主目录下。例如，假如你在 Windows 7 下运行了一个机器，且用户名是“john”，你会发现 Jenkins 主目录在 *C:\Users\john\.jenkins* 下；在 Windows XP 系统下，主目录是 *C:\Documents and Settings\John\.jenkins*；在 Linux 机器上，主目录大多数可能是在 */home/john/.jenkins* 下；等等。

你可以通过定义 `JENKINS_HOME` 环境变量来强行使 Jenkins 使用一个不同的目录作为其主目录。要做到这件事，你可能需要遵从本地目录设置习惯来修改构建服务器上的配置，或者要让你的系统管理员觉得这是可接受的。比如，假如你的 Jenkins WAR 文件安装在 */usr/local/jenkins* 下，且 Jenkins 主目录需要放在 */data/jenkins* 目录下，你可能需要根据下面几行示例写一个启动脚本：

```
export JENKINS_BASE=/usr/local/jenkins
export JENKINS_HOME=/var/jenkins-data
java -jar ${JENKINS_BASE}/jenkins.war
```

假如你把 Jenkins 运行在一个 Java EE 容器里，比如 Tomcat 或者 JBoss，你可以配置 webapp 来显示它自己的环境变量。例如，假如你在使用 Tomcat，则可以在 *$CATALINA_BASE/conf/localhost* 目录下创建一个名为 *jenkins.xml* 的文件：

```
<Context docBase="../jenkins.war">
  <Environment name="JENKINS_HOME" type="java.lang.String"
               value="/data/jenkins" override="true"/>
</Context>
```

以前，Jenkins 被称为 Hudson。Jenkins 对以前的 Hudson 安装保留兼容性，从 Hudson 升级到 Jenkins 只需要简单地把旧的 *hudson.war* 文件替换成 *jenkins.war*。Jenkins 将在以下地方寻找它的主目录（按优先顺序）：

1．一个被称为 JENKINS_HOME 的 JNDI 环境项。

2．一个被称为 HUDSON_HOME 的 JNDI 环境项。

3．一个被称为 JENKINS_HOME 的系统属性。

4．一个被称为 HUDSON_HOME 的系统属性。

5．一个被称为 JENKINS_HOME 的环境变量。

6．一个被称为 HUDSON_HOME 的环境变量。

7．在用户主目录下的 *.hudson* 目录（如果它已经存在）。

8．在用户主目录下的 *.jenkins* 目录。

在 Debian 或者 Ubuntu 系统上安装 Jenkins

假如你想在 Debian 或者 Ubuntu 上安装 Jenkins，为这些平台安装本地的二进制包是一件非常容易的事。这确实非常容易，尽管由于高频率的更新会导致这些二进制包不在标准的仓库中提供。首先，你需要向系统中添加 key，如下：

```
$ wget -q -O - http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key \
  | sudo apt-key add -
$ sudo echo "deb http://pkg.jenkins-ci.org/debian binary/" > \
  /etc/apt/sources.list.d/jenkins.list
```

然后，更新 Debian 的包仓库：

```
$ sudo aptitude update
```

这些做完以后，你就可以使用 *aptitude* 安装工具安装 Jenkins 了：

```
$ sudo aptitude install -y jenkins
```

这种方式是把 Jenkins 当作一个服务来安装，在 */etc/init.d/jenkins* 配置了正确的启动脚本，同时还有一个相应的系统用户名为“jenkins”。如果你还没在服务器上安装 Java，则需要安装 Java 的 OpenJDK。默认情况下，你会在 */usr/share/jenkins* 目录下找到 Jenkins 的 WAR 文件，Jenkins 的主目录为 */var/lib/jenkins*。

安装程序应该已经启动了 Jenkins。一般来说，要启动 Jenkins，只需简单地调用这个脚本：

```
$ sudo /etc/init.d/jenkins start
```

Jenkins 现在应该已经运行在默认端口 8080 上（*http://localhost:8080/*）。你可以停止 Jenkins：

```
$ sudo /etc/inid.d/jenkins stop
```

Jenkins 会把日志文件写到 */var/log/jenkins/jenkins.log*。你也可以在 */etc/default/jenkins* 中调整配置参数。如果你想修改 Java 的启动参数（JAVA_ARGS）的时候，这非常有用。

你还可以使用这个文件来配置自己想要传给 Jenkins 的参数，例如 HTTP 端口或者 Web 应用程序上下文（参见本章的“以单机应用形式运行 Jenkins”一节）。

48 在 Red Hat、Fedora 或者 CentOS 上安装 Jenkins

也有本地二进制安装包给 Red Hat、Fedora 和 CentOS 使用。首先，你需要设置仓库如下：

```
$ sudo wget -O /etc/yum.repos.d/jenkins.repo \
http://jenkins-ci.org/redhat/jenkins.repo
$ sudo rpm --import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key
```

如果是一个全新的安装，则可能需要安装 JDK：

```
$ sudo yum install java-1.6.0-openjdk
```

下一步，你就可以像下面这样安装包了：

```
$ sudo yum install jenkins
```

这样会把最新的 Jenkins 安装到 */usr/lib/jenkins* 目录下，默认的 Jenkins 主目录是 */var/lib/jenkins*。

现在，你可以启动 Jenkins 使用 `service` 命令了：

```
$ sudo service jenkins start
```

Jenkins 现在应该已经运行在默认的 8080 端口上了（*http://localhost:8080/*）。

Jenkins 的配置参数放在 */etc/sysconfig/jenkins* 文件中。然而，在写配置项的时候会比 Ubuntu 提供的包有更多的限制：例如，你可以使用 JENKINS_PORT 参数来定义 HTTP 端口，但指定一个应用程序上下文的话，就需要手动修改启动脚本。这里列出了主要的配置选项。

JENKINS_JAVA_CMD

> 你想要用来运行 Jenkins 的 Java 版本。

JENKINS_JAVA_OPTIONS

> 要传给 Java 的命令行选项，比如内存选项。

JENKINS_PORT

> Jenkins 运行所绑定的端口。

在 SUSE 或者 OpenSUSE 上安装 Jenkins

二进制安装包也提供给 SUSE 和 OpenSUSE，所以在这些平台上的安装过程也很简单。首先，你需要添加 Jenkins 仓库到 SUSE 仓库列表里：

```
$ sudo zypper addrepo http://pkg.jenkins-ci.org/opensuse/ jenkins
```

最后，你只需使用 zypper 命令安装 Jenkins：

```
$ sudo zypper install jenkins
```

正如你从控制台输出中了解到的一样，这将安装最新版本的 Jenkins 以及 Sun 的 JDK（如果后者还没有安装的话）。OpenSUSE 的安装通常会带有 Java 的 OpenJDK 版本，但是 Jenkins 更喜欢 Sun 的版本。当下载了 Sun JDK 后，它会提示在你可以继续安装之前需要先验证 Sun Java 的许可证明。

这个安装过程也会创建一个 *jenkins* 用户并且把 Jenkins 当作服务来安装，所以不管什么时候启动机器，Jenkins 都会自动启动。要想手动启动 Jenkins 的话，你可以调用 Jenkins 在 */etc/init.d* 目录下的启动脚本：

```
$ sudo /etc/init.d/jenkins jenkins start
```

Jenkins 现在应该已经运行在默认的端口 8080 上了（*http://localhost:8080/*）。

配置选项和 Red Hat 上的安装基本类似（参见本章的“在 Red Hat、Fedora 或者 CentOS 上安装 Jenkins”一节）。你可以在 */etc/sysconfig/jenkins* 文件中定义有限数量的配置变量，但是对于任何更高级的配置选项，你需要修改在 */etc/init.d/jenkins* 中的启动脚本。

zypper 工具也可以使更新 Jenkins 实例变得容易：

```
$ sudo zypper update jenkins
```

这条命令会从 Jenkins 官网上下载和安装 Jenkins 的最新版本。

以单机应用形式运行 Jenkins

你可以使用两种方式运行 Jenkins 服务器：要么作为一个独立的应用程序，要么部署为一个标准的 Web 应用，运行在 Java Servlet 容器里或像 Tomcat、JBoss 或者 GlassFish 等应用程序服务器。这两种方法有它们各自的优缺点，所以我们都会探讨。

Jenkins 打包成一个 WAR 文件，所以你可以直接使用内嵌的 Servlet 容器运行它。Jenkins 使用一个轻量的 Winstone Servlet 引擎来让你能够立即运行服务器，而不需要自己去配置一个 Web 服务器。这或许是最简单的启动方式，能让你在几分钟之内就启动和运行

Jenkins。如果你部署 Jenkins 到一个传统的应用程序服务器的话，这也是一个非常灵活的选项，它提供一些额外的不可用的功能。特别地，假如你正在运行 Jenkins 作为一个独立的服务器，你可以在它运行的时候安装插件和进行升级，然后直接从管理员屏幕重启 Jenkins 就可以了。

使用内嵌的 Servlet 容器运行 Jenkins，只需在命令行下输入以下内容即可：

```
C:\Program Files\Jenkins>
        java -jar jenkins.war
[Winstone 2011/07/01 20:54:53] - Beginning extraction from war file
[Winstone 2011/07/01 20:55:07] - No webapp classes folder found - C:\Users\john\
    .jenkins\war\WEB-INF\classes
50 jenkins home directory: C:\Users\john\.jenkins
...
INFO: Took 35 ms to load
...
[Winstone 2011/07/01 20:55:08] - HTTP Listener started: port=8080
[Winstone 2011/07/01 20:55:08] - Winstone Servlet Engine v0.9.10 running:
    controlPort=disabled
[Winstone 2011/07/01 20:55:08] - AJP13 Listener started: port=8009
```

在 Linux 环境下，这个过程是类似的。注意我们是如何从之前创建的“jenkins”用户账户启动这个 Jenkins 服务器的：

```
john@lambton:~$ sudo su - jenkins
jenkins@lambton:~$ java -jar /usr/local/jeknins/jenkins.war
[Winstone 2011/07/16 02:11:24] - Beginning extraction from war file
[Winstone 2011/07/16 02:11:27] - No webapp classes folder found - /home/jenkins/
    .jenkins/war/WEB-INF/classes
jenkins home directory: /home/jenkins/.jenkins
...
[Winstone 2011/07/16 02:11:31] - HTTP Listener started: port=8080
[Winstone 2011/07/16 02:11:31] - AJP13 Listener started: port=8009
[Winstone 2011/07/16 02:11:31] - Winstone Servlet Engine v0.9.10 running:
    controlPort=disabled
```

这将会在控制台窗口中启动一个内嵌的Servlet引擎。Jenkins Web应用程序现在就可用了，而且服务绑定在 8080 端口上。当你使用内嵌服务器运行 Jenkins 的时候，是没有 Web 应用程序上下文的，所以你可以直接使用服务器 URL（例如，*http://localhost:8080*）来访问 Jenkins。

要停止 Jenkins 的话，只需按下 Ctrl-C 组合键。

默认地，Jenkins 会运行在 8080 端口上。如果这样不适合你的环境，则可以手动指定端口，使用 `--httpPort` 选项：

```
$ java -jar jenkins.war --httpPort=8081
```

在实际架构中，Jenkins 可能不是唯一运行在你的构建服务器上的 Web 应用程序。依赖于你的服务器的能力，Jenkins 可以与其他的 Web 应用程序共存，例如 Maven 仓库管理器。如果你的 Jenkins 运行在另一个应用服务器上，例如 Tomcat、Jetty 或者 GlassFish，你还需要覆盖 ajp13 端口，使用 `--ajp13Port` 选项：

```
$ java -jar jenkins.war --httpPort=8081 --ajp13Port=8010
```

一些其他有用的选项如下：

`--prefix`

这个选项可以帮助你为 Jenkins 服务器定义一个上下文路径。默认情况下 Jenkins 运行在 8080 端口并且没有上下文路径（*http://localhost:8080*）。不管怎样，假如你使用了这个选项，则可以强制 Jenkins 去使用任何适合你的上下文路径，例如：

```
$ java -jar jenkins.war --prefix=jenkins
```

在这种情况下，可以在 *http://localhost:8080/hudson* 访问 Jenkins。

当想要集成一个独立的 Jenkins 实例到 Apache 的时候，这个选项经常会被使用。

51

`--daemon`

如果你在一台 UNIX 机器上运行 Jenkins，则可以使用这个选项以后台任务的形式启动 Jenkins，作为一个 UNIX 守护进程来运行。

`--logfile`

默认情况下，Jenkins 会把日志文件写到当前目录下。然而，在一个服务器上，你经常需要把日志文件写到一个预先确定的目录中。你可以使用这个选项将消息重定向到其他文件：

```
$ java -jar jenkins.war --logfile=/var/log/jenkins.log
```

使用 Ctrl-C 组合键来停止运行 Jenkins 有点太野蛮，当然——在实践中，你将会写一个脚本去自动启动和停止服务器。

如果你使用了内嵌的 Winstone 应用服务器运行 Jenkins，也可以优雅地直接调用 Winstone 服务器来重启或者关闭 Jenkins。为了做到这一点，你需要在启动 Jenkins 的时候指定 `controlPort` 选项，如下：

```
$ java -jar jenkins.war --controlPort=8001
```

在 UNIX 环境下，一个稍完整的示例可能看起来会像这样：

```
$ nohup java -jar jenkins.war --controlPort=8001 > /var/log/jenkins.log 2>&1 &
```

这里的关键就是 `controlPort`。这个选项给了你通过 Winstone 工具直接停止或者重启 Jenkins 的手段。唯一的问题就是你需要一个匹配的 Winstone JAR 文件版本。幸运的是，你的 Jenkins 安装包捆绑了一个，所以你不需要再去找。

要重启服务器，你可以运行一下命令：

```
$ java -cp $JENKINS_HOME/war/winstone.jar winstone.tools.WinstoneControl reload: \
  --host=localhost --port=8001
```

要完全关闭它，你可以使用如下命令：

```
$ java -cp $JENKINS_HOME/war/winstone.jar winstone.tools.WinstoneControl shutdown \
  --host=localhost --port=8001
```

另一种完全关闭 Jenkins 的方法是调用特殊的“/exit”URL，如下：

```
$ wget http://localhost:8080/exit
```

在一个真正的服务器上，你通常会需要设置安全配置，使得只有系统管理员可以访问这个 URL。在这种情况下，你将需要提供一个用户名和密码：

```
$ wget --user=admin --password=secret http://localhost:8080/exit
```

52 注意，你实际上可以从不同的服务器上做这些，而不仅仅是本地服务器：

```
$ wget --user=admin --password=secret http://buildserver.acme.com:8080/exit
```

注意，虽然这两种方法都可以相对干净地关闭 Jenkins（例如，好过直接杀死进程），但它们会中断任何正在进行中的构建。所以推荐的操作是通过使用 Manage Jenkins 界面上的 Prepare for Shutdown 按钮来进行干净的关闭操作（参见第 4 章的“配置面板——Manage Jenkins 界面”一节）。

作为一个独立的应用程序运行 Jenkins 可能并不符合每个人的口味。对于一个生产环境的服务器，你可能希望利用一个有更复杂的监控和管理功能的全面的 Java 应用程序服务器，比如 JBoss、GlassFish 或者 WebSphere 应用程序服务器。系统管理员可能会在使用相对不知名的 Winstone 服务器的时候很谨慎，或者简单地更喜欢把 Jenkins 融入一个已知的 Java Web 应用程序开发模式中。如果是这样的话，你可能更喜欢，或者说有义务，去把 Jenkins 部署为一个标准的 Java Web 应用程序。我们会在下面探讨这个选项。

基于 Apache 服务器运行 Jenkins

假如你正在一个 UNIX 环境中运行 Jenkins，你可能想要把它隐藏在 Apache HTTP 服务器之后，以便更好地协调服务器 URL 和简化维护及访问。通过这种方式，用户可以使用一个像 *http://myserver.myorg.com/jenkins* 这样的 URL 而不是 *http://myserver.myorg.com:8081* 去访问 Jenkins 服务器。

可以做到这一点的一种方式就是使用 Apache 的 `mod_proxy` 和 `mod_proxy_ajp` 模块。这些模块可以让你使用 AJP13（Apache JServer Protocol 版本 1.3）在 Apache 服务器上实现代理。通过使用这个模块，Apache 会在你的 Apache 服务器（运行在 8080 端口）上将请求转换为一个特定的 URL 模式，直接发送到运行在一个不同的端口上的 Jenkins 服务器。所以当一个用户打开一个 URL（比如 *http://www.myorg.com/jenkins*）时，Apache 会透明地把流量转发到你运行在 *http://buildserver.myorg.com:8081/jenkins* 的 Jenkins 服务器上。从技术上讲，这就是所谓的“反向代理”，客户端并不知道服务器端正在做的任何代理，或者代理服务器放在了哪里。所以你可以放心地把 Jenkins 服务器隐藏在防火墙之后，同时通过一个面向公众的 URL 来继续提供更广泛的对 Jenkins 实例的访问。

这个模块的具体配置将依赖于你的 Apache 版本细节和安装细节，但是一个可能的途径在这里展示一下。

首先，如果作为一个独立的应用程序运行 Jenkins 的话，确保你使用了 `--prefix` 选项来启动 Jenkins。你选择的前缀必须和你想要在面向公众的 URL 中使用的后缀相匹配。所以如果你想要通过 URL *http://myserver.myorg.com/jenkins* 去访问 Jenkins 的话，你需要提供 `jenkins` 作为前缀：

```
$ java -jar jenkins.war --httpPort=8081 --ajp13Port=8010 --prefix=jenkins
```

如果你在一个应用程序服务器（比如 Tomcat）上运行 Jenkins，则它已经运行在一个特定的 Web 上下文（默认是 `/jenkins`）。 53

接下来，确保 `mod_proxy` 和 `mod_proxy_ajp` 模块是活动状态的。在你的 *http.conf* 文件中（通常在 */etc/httpf/conf* 目录下），你需要有如下一行：

```
LoadModule proxy_module modules/mod_proxy.so
```

代理实际上被配置在 *proxy_ajp.conf* 文件里（通常在 */etc/httpd/conf.d* 目录下）。注意代理的路径名（在例子中是 `/jenkins`）必须和 Jenkins 正在使用的前缀或者 Web 上下文相匹配。这里也给出了一个配置文件的例子：

```
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
ProxyPass /jenkins http://localhost:8081/jenkins
```

```
ProxyPassReverse /jenkins http://localhost:8081/jenkins
ProxyRequests Off
```

一旦这些配置完成，你就只需重启 Apache 服务器：

```
$ sudo /etc/init.d/httpd restart
Stopping httpd:                                            [ OK ]
Starting httpd:                                            [ OK ]
```

现在你应该可以通过使用 URL *http://myserver.myorg.com/jenkins* 来访问 Jenkins 服务器了。

在应用服务器上运行 Jenkins

由于 Jenkins 是作为一个普通的 WAR 文件被分发的，所以在任何标准的 Java 应用程序服务器，如 Tomcat、Jetty 或者 GlassFish 上，部署它会很容易。在应用程序服务器上运行 Jenkins 可以说是设置和维护都会更复杂。你也会丢失一定管理方面的优秀特性，比如直接从 Jenkins 内升级或者重启服务器的能力。另一方面，相对于 Winstone 服务器来说，你的系统管理员可能更熟悉于维护一个运行在 Tomcat 或者 GlassFish 上的应用程序。

让我们来看一下怎样典型地部署 Jenkins 到 Tomcat 服务器上。最简单的方法无疑是简单地解压 Tomcat 的二进制发行版到你的磁盘（倘若还尚未安装），然后把 *jenkins.war* 文件复制到 Tomcat *webapps* 目录下。你可以从 Tomcat 官网（*http://tomcat.apache.org*）上下载二进制安装文件。

你可以通过运行在 Tomcat bin 目录下的 *startup.bat* 或者 *startup.sh* 脚本来启动 Tomcat。当你启动 Tomcat 后，Jenkins 就可以访问了。你需要注意的是，在这种情况下，Jenkins 将执行它自己的 Web 应用上下文（通常是“jenkins”），所以你需要在自己使用的 URL 里包含这个去访问你的 Jenkins 服务器（例如 *http://localhost:8080/jenkins*）。

54 然而，这种方式并不一定是最灵活和最健壮的选项。假如你的构建服务器是 Windows，则你可能需要安装 Tomcat 为一种 Windows 服务，以便你能确保不管服务器什么时候重启，它都能自动启动。类似地，如果你在 UNIX 环境下安装 Tomcat，它应该被配置为一个服务。

内存方面的考虑

持续集成服务器会使用很多内存。这是它本身就有的特性——构建会消耗内存，多个构建并发运行也会消耗更多的内存。所以在不管你想要同时运行多少构建的情况下，你均应该确保构建服务器有足够的 RAM 来应对。

Jenkins 本身自然也需要 RAM 来运行，但是如果你需要支持大量的构建过程，则仅仅分配给 Jenkins 大量内存是不够的。事实上，每次开始一个构建的时候，Jenkins 都会

跨越一个新的 Java 进程，所以在一个大型的构建当中，是构建过程需要内存，而不是 Jenkins。

你可以为 Jenkins 构建作业定义特定构建内存选项——我们会在本书的后面部分看到怎样做这些。然而，假如你有很多构建需要维护，你可能需要定义 JAVA_OPTS、MAVEN_OPTS 和 ANT_OPTS 这些构建中默认使用的环境变量的值。JAVA_OPTS 选项将应用到主要的 Jenkins 进程中，而其他两个选项将在 Jenkins 分别为 Maven 和 Ant 构建作业创建新的 JVM 过程时使用。

这里有一个怎样在 UNIX 机器的 *.profile* 文件中配置这些变量的例子：

```
export JAVA_OPTS=-Djava.awt.headless=true -Xmx512m -DJENKINS_HOME=/data/jenkins
export MAVEN_OPTS="-Xmx512m -XX:MaxPermSize=256m"
export ANT_OPTS="-Xmx512m -XX:MaxPermSize=256m"
```

安装 Jenkins 为一个 Windows 服务

假如你在一个 Windows 环境下运行了一个用作生产环境的 Jenkins 安装，把它运行为一个 Windows 服务是至关重要的。在这种方式下，不管服务器什么时候重启，Jenkins 都能自动启动，并且可以使用标准的 Windows 管理工具来进行管理。

在应用程序服务器如 Tomcat 上运行 Jenkins 的好处之一就是，它一般能很容易地去配置这些服务器运行为一个 Windows 服务。它也很容易把 Jenkins 安装为一个服务，而不需要安装 Tomcat。

Jenkins 设计了一个非常方便的特性，可以使得把 Jenkins 安装成一个 Windows 服务变得简单。目前还没有一个图形化的安装程序能够帮你做这件事，但是你可以使用另一个好东西——一个基于 Web 的图形化安装程序。

首先，你需要在目标机器上启动 Jenkins 服务器。最简单的方法就是使用 Java Web Start 来运行 Jenkins（见图 3-4）。或者，你可以通过下载 Jenkins 并且从命令行运行它，就像我们之前讨论的那样：

55

```
C:\jenkins> java -jar jenkins.war
```

当 Jenkins 默认的 8080 端口已经被另外一个应用程序占用的时候，第二个选项就显得有用了。你使用了哪个端口实际上并不重要——你可以以后再更改它。

只要你成功运行了 Jenkins，就可以连接到服务器，然后进入 Jenkins 的管理界面。在这里你会发现有一个按钮是 Install as Windows Service。这将在服务器上创建一个 Jenkins 服务，它会自动且有条理地开始和停止 Jenkins（见图 3-5）。

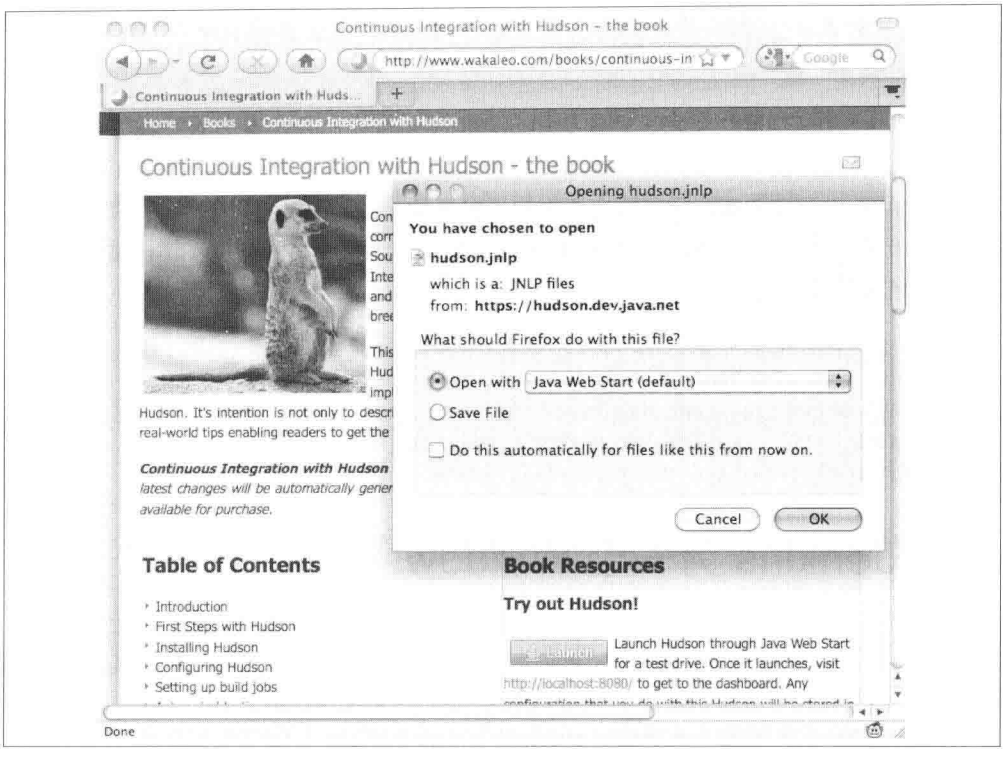


图 3-4　使用Java Web Start启动Jenkins

56



图 3-5　以Windows服务的方式安装Jenkins

Jenkins 会提示你指定一个安装目录。这个就是 Jenkins 的主目录（JENKINS_HOME）。默认的值是 JENKINS_HOME 变量的值：在当前用户主目录下的一个叫作 *.jenkins* 的目录。但通常这对于 Windows 安装不是一个好的选择。当在 Windows XP 上运行 Jenkins 的时候，你需要避免把 Jenkins 主目录安装在靠近 *C:\Documents And Settings* 的地方——不仅是因为它是一个长得出奇的名字，还因为这些空格会对你的 Ant 和 Maven 构建或者其他任何使用基于 classpath 资源的测试造成严重破坏。一个好的做法是使用一个简短并且合乎情理的名字，比如 *C:\Jenkins*。Windows Vista 和 Windows 7 的主目录路径如 *C:\Users\john* 也是可以的。

有时也因为其他一些原因而要求提供一个简短的主目录路径。在很多版本的 Windows 系统（Windows XP、Windows Server 2003 等）下，文件路径长度被限制在 260 个字符左右。假如你把一个嵌套的 Jenkins 工作目录和一个深的 class path 结合起来使用，你会经常遇到溢出情况，这将导致非常模糊的构建错误。为了将 Windows 文件路径限制溢出的风险降到最低，你需要重新定义 JENKINS_HOME 环境变量来指向一个更短的路径，就像我们上面讨论的一样。

这种方法并不总是起作用，比如在 Windows Vista 或者 Windows 7 上。一个可供选择的策略是使用 *jenkins.exe* 程序，这个程序在 Web Start 安装过程中就应该已经安装在你指定的目录里了。以管理员身份打开命令行提示符（右击，选择“以管理员身份运行”）并以 install 选项运行 *jenkins.exe* 可执行程序：

```
C:\Jenkins> jenkins.exe install
```

这个基本的安装在一个简单的上下文环境中能正常工作，但是你要经常调整你的服务。例如，默认情况下，Jenkins 服务会运行在本地系统账户下。但是，假如你在使用 Maven，Jenkins 就需要一个 *.m2* 的目录以及主目录下需要有一个 *settings.xml* 的文件。类似地，假如你在使用 Groovy，你可能需要一个 *.groovy/lib* 目录，等等。为了允许这样做，以及使得测试 Jenkins 安装变得更容易，需要确保你是在一个真实的用户账户以及正确的开发环境设置下运行这个服务的（见图 3-6）。另外，以系统用户的身份运行这个应用程序，但使用 Jenkins 上的系统信息（System Information）页面去检查 *${user.dir}* 目录，并且把所有必须要放置的文件放置在用户主目录下。

57

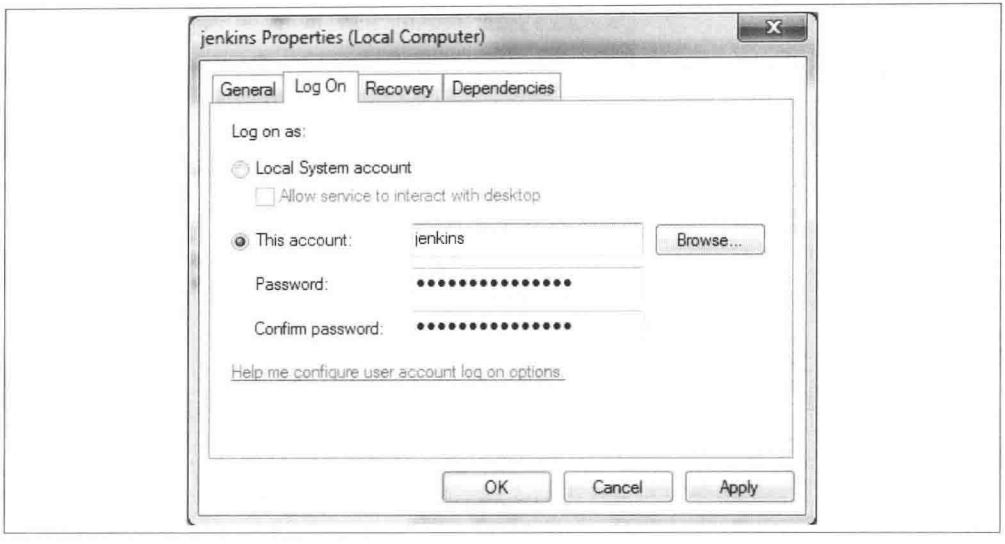


图 3-6　配置Jenkins Windows服务

你可以在一个叫作*jenkins.xml*的文件中配置 Jenkins 服务的细节，它和你的*jenkins.war*文件在同一个目录中。在这里，你可以配置（或重新配置）端口、JVM 选项和 Jenkins 的工作目录。在下面的例子中，我们稍微多给了 Jenkins 一点内存，并且把它运行在 8081 端口上：

```
<service>
  <id>jenkins</id>
  <name>Jenkins</name>
  <description>This service runs the Jenkins continuous integration system
    </description>
  <env name="JENKINS_HOME" value="D:\jenkins" />
  <executable>java</executable>
  <arguments>-Xrs -Xmx512m
  -Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecycle
  -jar "%BASE%\jenkins.war" --httpPort=8081 --ajp13Port=8010</arguments>
</service>
```

58 最后，如果你需要卸载 Jenkins 服务，以下两种方法可以任选其一。最简单的做法是运行 Jenkins `uninstall` 可执行选项：

```
C:\jenkins> jenkins.exe uninstall
```

另一个选项是使用 Windows 服务工具 `sc`：

```
C:> sc delete jenkins
```

Jenkins 主目录里有什么

Jenkins 主目录下包含了所有 Jenkins 服务器配置的细节，以及你在 Manage Jenkins 界面做的配置细节。这些配置细节存储为一组 XML 文件的形式。比如，很多核心的配置是存储在 *config.xml* 文件里的。其他指定工具的配置存储在其他适当命名的 XML 文件里：例如，Maven 安装的细节存储在名为 *hudson.tasks.Maven.xml* 的文件里。你很少需要手动去修改这些文件，不过偶尔也会派上用场。

Jenkins 主目录还包含了许多子目录（见图 3-7）。并不是所有的文件和目录在全新安装之后就会有，因为有一些是当 Jenkins 需要的时候才会创建它们。如果你查看一个已有的 Jenkins 安装，就会发现一些额外的与 Jenkins 配置和插件相关的 XML 文件。

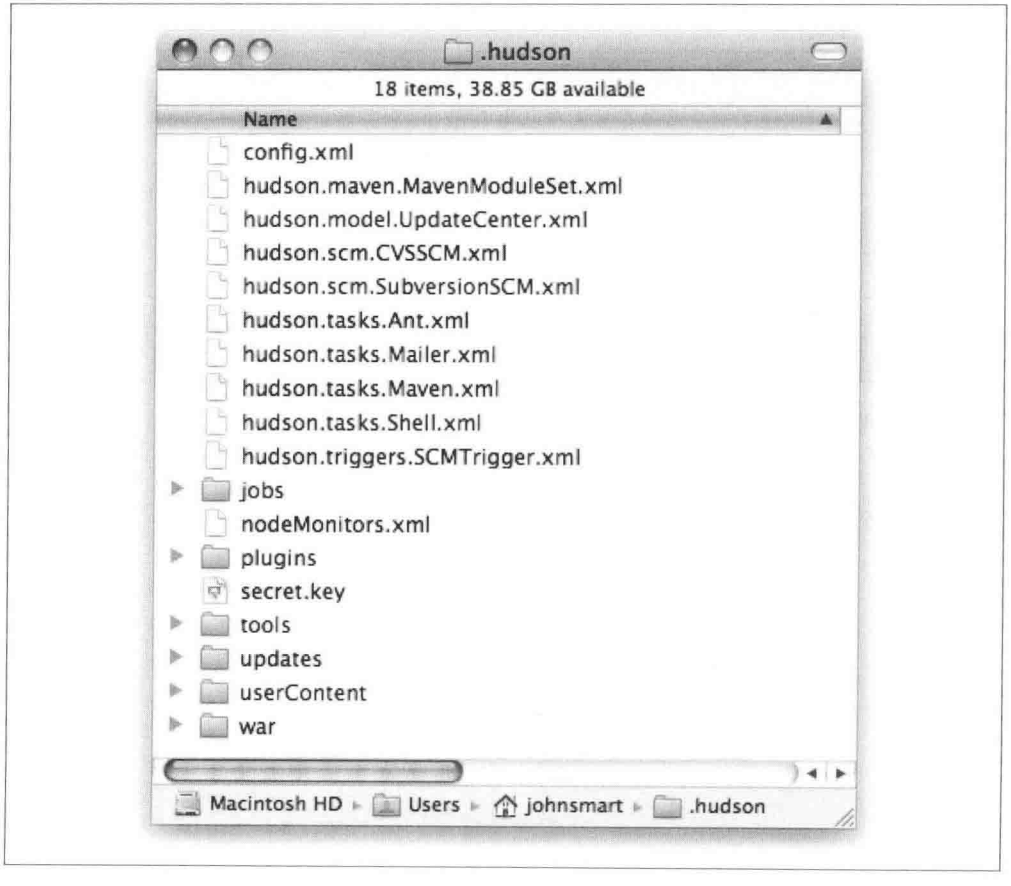


图 3-7　Jenkins主目录

这些主要的目录在表 3-1 中有更详细的描述。

表 3-1　Jenkins主目录结构

目录	描述
.jenkins	默认的 Jenkins 主目录（在旧的安装中可能是 *.hudson*）
fingerprints	这个目录被 Jenkins 用来跟踪人工操作的痕迹。我们会在本书的后面看到如何对其进行跟踪
jobs	这个目录包含 Jenkins 管理的构建作业的配置细节，以及这些构建所输出的产物以及数据。我们会在下面详细地看一下这个目录
plugin	这个目录包含所有你已经安装了的插件。插件可以使你通过添加额外的功能来扩展 Jenkins。需要注意的是，除了 Jenkins 的一些核心插件（subversion、cvs、ssh-slaves、maven 和 scid-ad）外，插件不存储为可执行的 `jenkins` 文件，也不存储在扩展的 Web 应用程序目录。这意味着你可以更新你的 Jenkins 而不需要重新安装插件
updates	这是 Jenkins 使用的一个内部目录，用来存放可用的插件更新
userContent	你可以使用这个目录存放你自己为 Jenkins 服务器定制化的一些内容。你可以在 *http://myserver/hudson/userContent*（如果你正在一个应用服务器上运行 Jenkins）或 *http://myserver/userContent*（如果运行在一个单机模式）里访问在这个目录里的文件
users	如果你使用的是 Jenkins 的本地用户数据库，用户账户信息会被存放在这个目录下
war	这个目录包含了扩展的 Web 应用程序。当你以一个单机应用程序的形式运行 Jenkins 时，它会把 Web 应用程序解压到这个目录

59

作业目录是整个 Jenkins 目录结构里至关重要的一部分，值得更多的关注。你可以在图 3-8 中看到一个真实的 Jenkins 作业目录的例子。

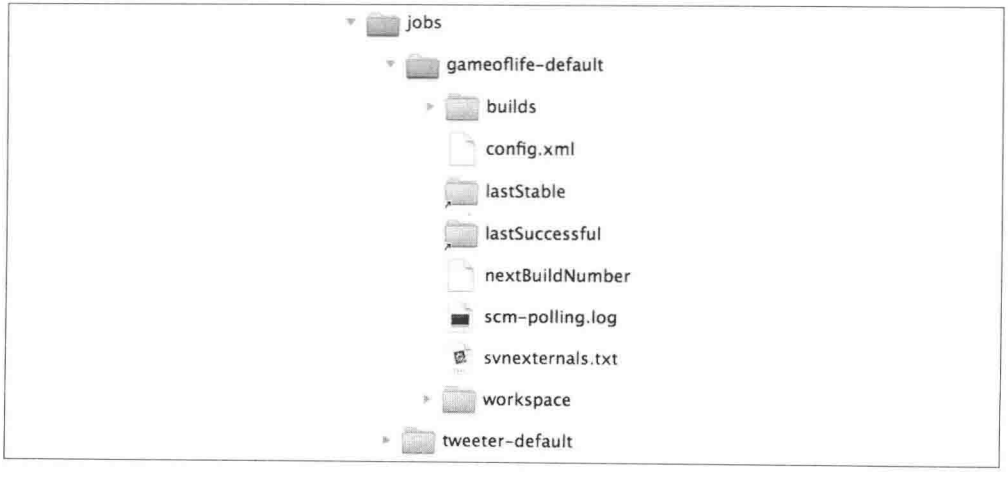


图 3-8　Jenkins作业目录

这个目录包含 Jenkins 实例管理的每一个 Jenkins 构建作业的子目录。每个工作目录包含两个子目录：*builds* 和 *workspace* 以及一些其他的文件。特别是，它包含构建作业的 *config.xml* 文件，正如你所期望的那样，这个文件包含了这个构建作业的所有配置细节。还有一些其他 Jenkins 内部使用的文件，你通常不需要动它们，比如 *nextBuildNumber* 文件（包含你在这个构建中想要分配给下一个构建的数量），以及大多数最近成功的构建和最后一个稳定构建的符号链接。一个成功的构建指没有任何的编译错误。一个稳定的构建就是一个成功的构建（不论你为这个构建配置了怎样的质量标准，比如单元测试、代码覆盖率等）。

builds 和 *workspace* 目录都是非常重要的。*workspace* 目录是 Jenkins 对你的项目进行构建的地方：它包含 Jenkins 检验过的源代码，以及构建本身生成的所有文件。这个工作空间会被每一个连续构建所重复使用——每一个项目只能有一个 *workspace*，并且它需要的磁盘空间往往是相对稳定的。

builds 目录包含为这个作业所执行的构建的历史。你很少需要直接干预这些目录，但是知道它包含了什么对你来说是有用的。你可以在图 3-9 中看到一个真实的构建目录的例子，图里执行了三个构建。Jenkins 存储每一个在目录里执行的构建的历史和人工操作，并打上时间戳（图 3-9 中的“2010-03-12_20-42-05”等）。它还包含指向构建历史目录的带有实际构建号的符号连接。

每个构建目录都包含一些信息，比如构建产物日志文件、这个构建所使用的 Subversion 修订号（假如你在使用 Subversion）、可以触发这个构建的一些变动及任何你让 Jenkins 跟踪的数据或度量。例如，假如你的构建作业跟踪单元测试结果或测试覆盖率度量，每一个构建的这些数据都会存放在这个目录下。这个构建目录页包含你存放的任何构建产物——二进制构建产物，当然也包括一些其他的生成文件，比如 javadoc 或者代码覆盖率度量。某些构建作业的类型，比如 Jenkins Maven 构建作业，默认情况下也会归档为二进制版本。

builds 目录的大小会随着时间自然而然地增大，因为会有构建历史的堆积。你可能想要在设计构建服务器目录结构的时候把这个考虑进去，特别是你的构建服务器运行在一个有多个磁盘分区的 UNIX 风格环境下的时候。这些数据大部分都是文本文件或者 XML 形式的文件，这种形式不需要为每个构建消耗大量额外的空间。然而，假如你的构建获取了一些构建版本时，比如 JAR 或者 WAR 文件，它们也会存放在这里。这些构建产物的大小应该被考虑到你的空间磁盘需求当中。稍后我们会看一下在磁盘空间是个问题的情况下，怎样限制一个特殊构建作业的构建数量的存储。限制 Jenkins 存放的构建作业的数量通常需要在磁盘空间和能够保持有用的构建统计之间做出权衡，因为 Jenkins 强大的报告功能也依赖于这个构建历史。

61

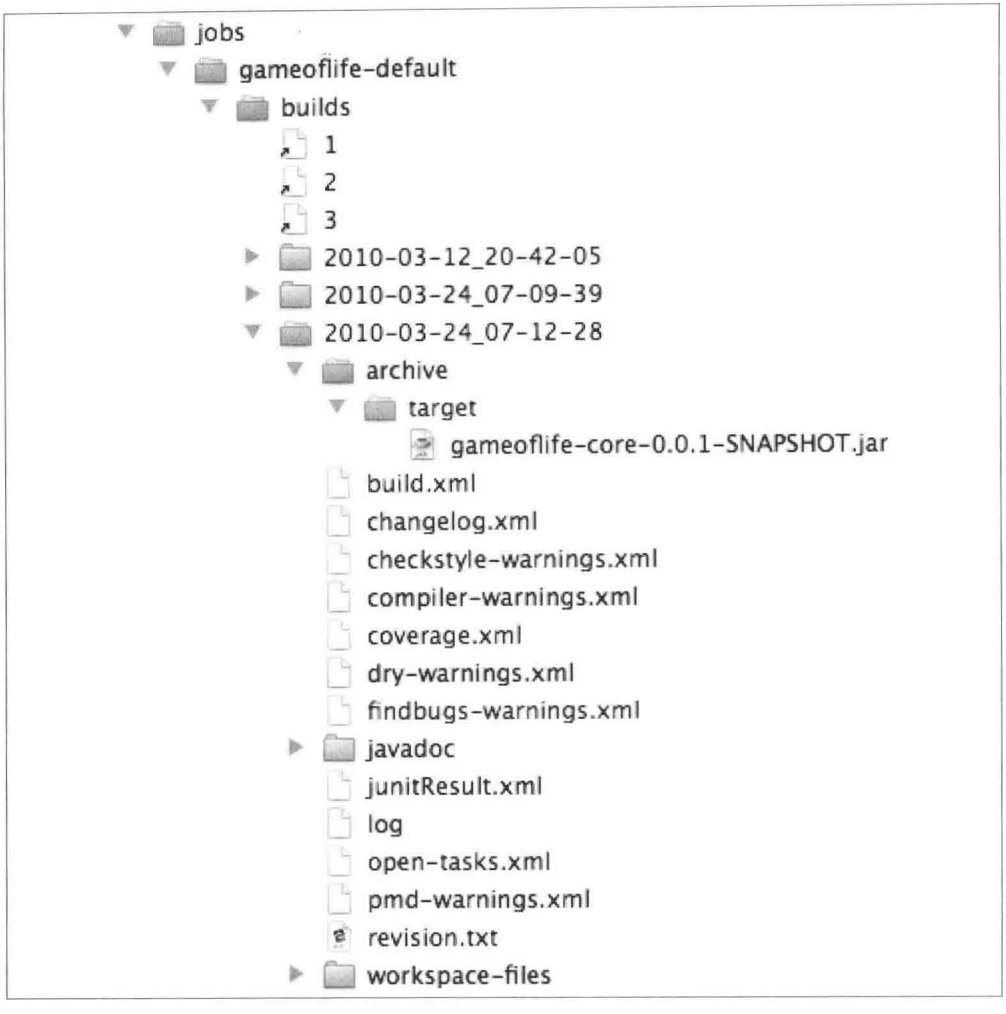


图 3-9　构建目录

Jenkins 广泛地使用这个目录下的文件来显示构建历史和度量数据，所以你应当特别注意在你不确定自己在做什么的情况下，不要删除任何构建历史目录。

备份 Jenkins 数据

确保你的 Jenkins 数据能定期备份是非常重要的。特别是 Jenkins 主目录数据需要备份，这里有你服务器的配置细节，也有你的构建版本和构建历史。这个目录应该经常自动备份。Jenkins 本身相对并不那么重要，因为它能够很容易地重装而不影响你的构建环境。

升级 Jenkins 安装

Jenkins 升级是非常容易的——你只需要简单替换自己本地的 *jenkins.war* 文件然后重启 Jenkins。当然，你应该确保当你重启服务器的时候没有构建正在运行。由于你的构建环境配置细节、插件和构建历史存放在 Jenkins 的主目录下，所以升级 Jenkins 将不会影响你的安装。你可以经常检查自己当前正在运行的 Jenkins 版本（可以通过查看每个界面右下角的版本号来检查）。

假如你使用 Linux 软件包之一安装了 Jenkins，就可以使用和服务器上其他系统软件包一样的过程去升级 Jenkins。

如果你作为一个独立实例在运行 Jenkins，则也可以直接从 Web 接口升级 Jenkins（在 Manage Jenkins 部分）。假如有一个更新的版本可用，Jenkins 会提示并让你选择是手动下载它还是自动更新（见图 3-10）。



图 3-10　从Web接口升级Jenkins

只要 Jenkins 完成升级文件的下载，你就可以在没有作业运行的时候让它重启。这可能是最简便的升级 Jenkins 的方式，尽管这并不是在所有环境里都有用。特别地，你需要作为一个独立应用程序运行 Jenkins，并且运行 Jenkins 的用户需要有读写 *jenkins.war* 文件的权限。

假如你在一个应用程序服务器上运行 Jenkins，比如 Tomcat 或者 JBoss，在升级 Jenkins 实例的时候你可能需要多做一些整理。比如 Tomcat，会把编译了的 JSP 页面放在

CATALINA_BASE/work 目录下。当你升级 Jenkins 版本的时候，这些文件需要被删除，防止任何过期的页面被引用的可能。

所有你安装了的插件将不会被 Jenkins 升级所影响。另外，插件也是可以升级的，不过是相对独立于 Jenkins 主执行程序的。你可以在 Jenkins Web 应用程序中使用 Jenkins 插件管理器（Plugin Manager）直接对插件进行升级。我们会在本书的后面进一步详细讨论插件。

小结

在本章中，我们看到了怎样在不同的环境下安装和运行 Jenkins，以及学习了一些关于如何维护运行之后的 Jenkins 安装的基本技巧。Jenkins 非常易于安装（不管是作为一个独立的应用程序还是作为一个部署在已有的应用程序服务器上的 WAR 文件）。在选择一个构建服务器作为 Jenkins 宿主机的时候，你主要需要考虑的是 CPU、内存和磁盘空间。

配置Jenkins服务器

简介

在你能开始创建 Jenkins 构建作业之前，需要做一点配置，确保 Jenkins 服务器在你的工作环境中能顺利工作。Jenkins 具有高可配置性，尽管大多数选项都提供了适当的默认值，也可以在系统路径和环境变量中找到正确的构建工具，但通常情况下一个好的做法就是，你要确切地知道你的构建服务器正在做什么事。

Jenkins 整体来说是非常容易配置的。管理界面很直观，且有详细的在线帮助文档（每个输入框旁边的蓝色问号图标），在本章中，我们将从细节上看看如何搭建配置基础服务器，包括对于不同 Java 版本，Ant、Maven 等构建工具，以及 CVS、Subversion 等 SCM 工具的 Jenkins 配置。我们也会进一步在书中关注更高级的服务器配置，如其他版本控制系统、通知工具等。

配置面板——Manage Jenkins 界面

在 Jenkins 中，你可以用可视化的 Manage Jenkins 界面管理所有的系统配置（参考图 4-1）。你也可以在 Jenkins 任何界面的搜索框输入“manage”直接到达界面。这个界面会因你安装的插件而发生变化，所以当你看到的比书中展示的多时，不要惊讶。

你可以在这个界面进行 Jenkins 服务器的各方面配置。页面上的每个链接可以导航到 Jenkins 服务器不同部分的配置界面，在这里讨论一些比较有意思的选项。

配置系统

你可以在这里管理在构建中使用的工具路径，例如 JDK，不同版本的 Ant 和 Maven，

以及安全选项、邮件服务器和其他系统配置。你安装的众多 Jenkins 插件都将需要在这里进行配置——当你安装插件时，Jenkins 会动态地添加相关字段。

66



图4-1　在Manage Jenkins界面进行安装配置

从硬盘加载配置

就像在之前章节看到的那样，Jenkins 用 XML 文件存储所有它自己的以及构建作业的配置细节，并放在 Jenkins 的主目录（参见第 3 章的“Jenkins 主目录”一节），同样在这个目录下存储了所有的构建历史数据。如果你将构建作业从一个 Jenkins 实例迁移到另外一个 Jenkins 实例，或者是归档旧的构建作业，则需要添加或者删除对应的构建作业目录到 Jenkins 的 *builds* 目录。你不需要停掉 Jenkins 服务器——可以使用“从硬件加载配置”（Reload Configuration from Disk）选项直接加载 Jenkins 系统和构建作业配置。如果有很多的构建历史数据，则这个处理会有一点慢，因为 Jenkins 不止加载构建配置，还会加载所有的构建历史数据。

管理插件

Jenkins 最好的特性是其扩展性架构。它拥有一个巨大的第三方开源插件生态系统，使你能够为构建服务器添加额外的功能，从支持不同的 SCM 工具，如 Git、

Mercurial 或者 ClearCase，到代码质量和代码覆盖率度量报告。我们将在本书中看到很多受欢迎且好用的插件。插件可以通过 Manage Plugins（管理插件）界面进行安装、升级和卸载。注意卸载插件时需要谨慎，因为它可能会影响 Jenkins 实例的稳定性，我们将在第 13 章的“构建迁移”一节看到更多细节。

系统信息

这个界面显示所有当前 Java 系统属性和系统环境变量列表。在这里你可以查看 Jenkins 正在使用的 Java 版本，其下都有哪些用户，等等。你也可以查看 Jenkins 当前正在使用的正确环境变量设置。它主要用于故障排除，以便你能确保自己的服务器使用的系统变量及变量值和你预期的一样。

系统日志

通过系统日志界面可以很方便地实时查看 Jenkins 日志文件。此外，其主要用途是为了故障跟踪。

你能订阅 RSS 多种级别的日志信息。比如：作为 Jenkins 管理员，你可能想订阅所有的错误和警告日志信息。

负荷统计

Jenkins 依据并发构建作业的数量和构建队列的长度（它会给出你的构建作业需要等待多久才可以执行）跟踪你的服务器繁忙情况。这些统计可以为你提供关于是否需要增加容量和扩展基础设施构建节点的依据。

脚本控制台

这个界面可以让你在服务器上执行 Groovy 脚本，它对于高级故障跟踪是非常有用的：高级故障跟踪取决于强大的 Jenkins 内部架构背景知识。它主要还是对插件开发者是非常有用处的。

管理节点

Jenkins 也可以很好地处理并行和分布式构建。在这个界面，你想要多少构建就可以配置多少。如果你使用分布式构建，并且搭建了分布式节点，则 Jenkins 支持节点并行执行。Jenkins 可以在一个位于另外机器的构建节点上执行它的构建作业。我们将在第 11 章讨论如何进行分布式构建配置。

关机准备

如果你需要关停 Jenkins，而当前 Jenkins 服务器正在运行，并且一个构建作业正

在执行，则最好不要直接关停。若想彻底关停 Jenkins，你可以使用 Prepare for Shutdown（关机准备）链接来阻止任何新的构建请求。最终当所有的构建作业完成时，你将能彻底地关停 Jenkins。

我们将在书中的后面章节看到这些功能的具体细节。接下来，我们将关注于如何配置最重要的 Jenkins 系统参数。

配置系统环境

最重要的 Manage Jenkins 界面是 Configure System（配置系统）界面（见图 4-2）。在这里，你能设置 Jenkins 做日常工作的大部分必要工具。这个默认界面包含了多个部分，每部分涉及不同的配置区域或扩展工具。此外，当你安装插件时，大部分插件的系统级配置也同样在这个界面进行配置。

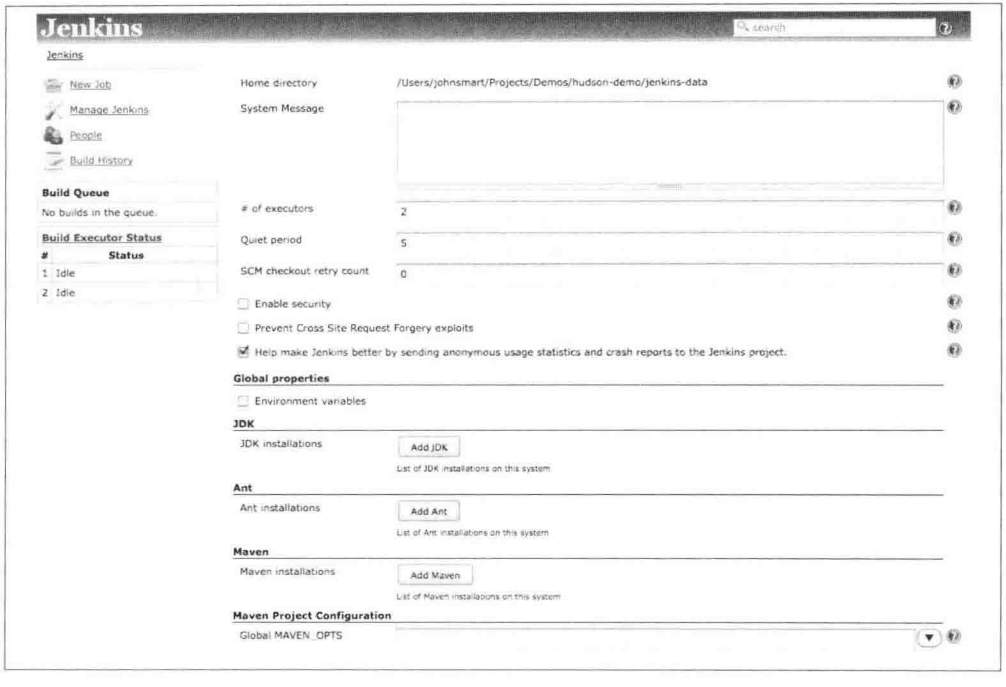


图4-2　Jenkins系统配置

这个配置系统界面可为你提供 Jenkins 安装的全局参数的定义，以及构建流程所需的扩展工具。界面的第一部分允许你定义一些通用的系统级参数。

Jenkins 主目录显示在界面上，以供参考。这样，你就能检查自己正在使用的主目录是否

正确。提醒一下，你可以在自己的环境里通过设置 JENKINS_HOME 环境变量来改变这个路径（参见第 3 章的“Jenkins 主目录”一节）。

System Message（系统信息）字段有几个有用的用途。这个文本内容会显示在 Jenkins 主页的头部。你能使用 HTML 标签，所以通过使用服务器名称和一段简短的用途描述来定制化你的构建服务器就会是一种非常简单的方式。你也能用它来为所有用户显示通知，比如声明系统停机维护等。

静默期（Quiet Period）对于类似于 CVS 将文件逐个地进行提交而不是将文件组合在一起作为一个单一的原子事务进行提交的 SCM 工具是有用处的。通常，Jenkins 会在检测到源码库的一个变更时立刻会触发一个构建。然而，这不适用于所有情况。如果你正在使用类似于 CVS 的 SCM 工具，你不会希望 Jenkins 在有变更提交后就立刻开始构建，因为源码库需要等所有变更全部提交后才能保持一致状态。你可以使用静默期来避免这类问题。如果你在此设置了值，Jenkins 将在触发构建前等待设定的时长来确保变更都已提交。这有效确保在开始构建前所有的变更都提交了，并且源码库处于稳定的状态。

大多数现代版本控制系统，如 Subversion、Git 或 Mercurial 的提交方式都是原子性的。也就是说多个文件中的变更被作为一个单元提交到源码库中，从而使源码库一直保持稳定状态。然而，一些团队仍使用一种单一逻辑变更集分多次提交的操作方式。在这种情况下，你可以使用静默期来确保每次构建总是使用稳定的源码版本。

这里指定的静默期值实际上是默认作用于系统范围的，如果有需要，你可以分别为每个项目重新定义不同的值。

在此你还可以管理用户账号和用户权限，默认情况下，Jenkins 允许用户做任何事情。如果你想要更多限制性的方式，则需要在这里通过 Enable Security（开启安全）选项来激活 Jenkins 安全设置。有很多方法可以做到这一点，我们将在之后看到 Jenkins 的这方面内容（请看第 7 章）。

配置全局属性

Global properties（全局属性，见图 4-3）部分可以让你集中管理用于所有构建作业中的定义变量。你能在这里添加很多你想要的属性，而且可以在你的构建作业中使用它们。Jenkins 将使它们在你的构建作业环境中生效，所以你可以任意地在自己的 Ant 和 Maven 构建脚本中使用它们。注意属性名字中不能包含句点（“.”），因为这样不能被正确处理，所以 `ldapserver` 或 `ldap_server` 都可以，但是 `ldap.server` 不行。

你通常有两种使用这些属性的方式。首先，你能在自己的构建脚本中直接使用它们，写

法为 ${key} 或 $key（上面的例子可写成 ${ldapserver} 和 $ldapserver）。这是最简单的使用方法，但是意味着在你的构建作业配置和构建脚本间有着深度的耦合。

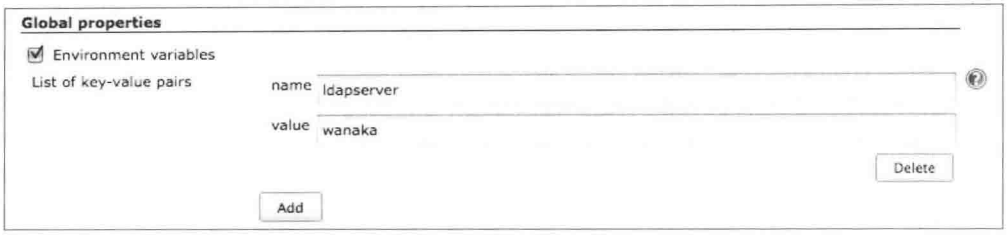


图4-3　在Jenkins中的环境变量配置

如果你的脚本使用不同的属性名字（例如，包含点号），你也可以将值放入构建作业配置里的脚本中。在图 4-4 中，我们把定义在图 4-3 中的 ldapserver 属性值放入 Maven 构建作业中。使用 -D 选项意味着在构建脚本里可以访问这个值。这是一种灵活的方式，因为我们可以在 Jenkins 中为脚本定义全局属性，然后在构建脚本中使用那些变量。例如，在图 4-4 中，ldapserver 属性可以在 Maven 构建中通过内部的 ${ldap.server} 属性进行访问。

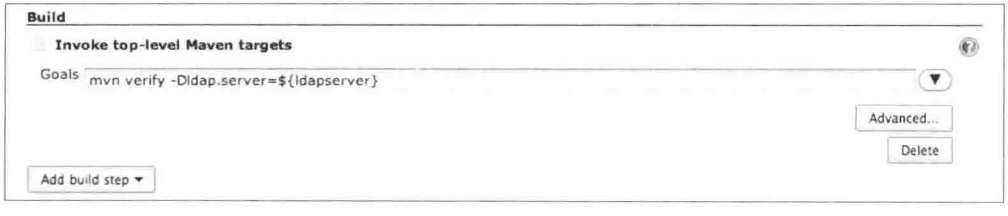


图4-4　环境变量的使用

配置 JDK

总的来说，Jenkins 的最重要用途之一是用于构建 Java 应用。所以，Jenkins 自然为 Java 提供了优秀的内置支持。

默认情况下，Jenkins 使用在系统路径上查找到的任意 Java 版本去构建 Java 应用，通常是 Jenkins 自己正在使用的版本。然而，对于一个生产构建服务器，你可能想要更多的控制权。例如，由于性能的原因，可能你的 Jenkins 服务器运行在 Java 6 上，但是你的生产服务器却可能正运行在 Java 5 或者 Java 1.4 上。大型机构对于其生产环境 Java 版本的升级往往是谨慎的，并且市场上一些相对重量级的应用服务器对最新版的 JDK 验证是出了名的慢。

在任何情况下，使用一个接近生产服务器上运行的 Java 版本往往是明智的做法。尽管一个应用程序使用 Java 1.4 编译通常可以稳定地在 Java 6 上运行，但反之则不行。或者你可能有不同的应用需要使用不同的 Java 版本进行构建。

Jenkins 提供了与多个 JVM 协作的良好支持。事实上，Jenkins 让配置和使用众多 Java 版本变得很容易。像大多数系统级的配置一样，我们在配置系统界面（Configure System，见图 4-2）进行配置。在这里，你将找到一段关于 JDK 的配置，在此你可以管理需要与 Jenkins 一起协作的 JDK。

最简单的声明一个 JDK 的方式是提供一个恰当的名字（在之后配置你的构建时，用于指定这个 Java），以及 Java 安装的路径（和 JAVA_HOME 变量一样的路径），如图 4-5 所示。尽管你需要手动输入路径，但 Jenkins 将实时地检查这个路径是否存在并且检查其是否是个有效的 JDK 目录。



图4-5　在Jenkins中的JDK配置

你也可以让 Jenkins 为你安装 Java。在这种情况下，Jenkins 将下载 JDK 安装包并且在你的机器上安装一份拷贝（见图 4-6）。第一次构建时需要使用 JDK，Jenkins 将下载指定 JDK 版本并将其安装到 Jenkins 主目录下的 *tools* 目录中。如果构建正运行在一个没有安装 JDK 的新构建代理上，它还会下载 JDK 并安装到代理机器上。

这也是一个配置代理服务器非常好的方式。之后我们会在本书中看到，Jenkins 可以将构建工作委托给其他机器，或者创建代理。构建代理（又名“从节点”）是指 Jenkins 可以用来运行一些构建作业的其他机器。如果你使用 Jenkins 的自动安装选项，则无须手动地在构建代理机器上安装你需要的 JDK 版本，Jenkins 会在第一次用到时自动完成安装。

默认情况下，Jenkins 建议从 Oracle 网站下载 JDK。如果你的 Jenkins 服务器由代理服务器代理，那么你可能需要进行代理配置，以确保 Jenkins 可以访问外部下载网站（参见本章的“配置代理”一节）。另一种方法是提供一个 URL 指向自己内部的 JDK 副本（一个 ZIP 或 Gzip 压缩的 TAR 文件），其存储在内部的一个本地服务器上。在这里还可以提供在本地服务器上的标准安装文件，以使其更加快速地自动安装。当你使用这个配置项时，Jenkins 还会让你指明一个标签，将用于限制只有带有这个标签的构建节点可以使用。当你需要在一些特定构建机器上安装一个工具的特定版本时，这是一个有用的方法。同样地，这也可以应用于其他构建工具（比如 Maven 和 Ant）。



图4-6　自动安装JDK

自动安装程序不能在所有工作环境（例如，如果在操作系统上无法找到或识别到所需要满足的条件，则可能安装会失败）下工作，但它依然是一个用来搭建新的构建服务器或者分布式构建代理服务器有用且方便的方式。

配置构建工具

构建工具是任何构建服务器的基础，Jenkins 也不例外。Jenkins 支持三个主流构建工具：Ant、Maven 和基本的 Shell 脚本（或 Windows 批处理脚本）。通过使用 Jenkins 插件，你还能为其他的构建工具和其他语言添加支持，如 Gant、Grails、MSBuild 等。

Maven

Maven 是一个高级 Java 构建脚本框架，它使用标准目录结构和标准的生命周期，协定统一配置，并通过声明依赖管理来简化很多在 Ant 中存在的典型低级脚本。在 Maven 中，你的项目使用标准、优化定义的生命周期——编译、测试、打包、部署等。每一个生命周期阶段都一一对应着 Maven 插件。多种多样的 Maven 插件都使用标准目录结构，在最小化干预下执行任务。你也可以通过配置其他插件覆盖默认插件来扩展 Maven。

Jenkins 为 Maven 提供了很好的支持，并精通 Maven 项目结构和依赖关系。你可以自动安装指定的 Maven 版本（就像我们 Maven 3 的例子），或是提供一个本地 Maven 安装路径（见图 4-7）。你可以为构建项目配置你需要的多个 Maven 版本，并为不同的项目使用不同的 Maven 版本。

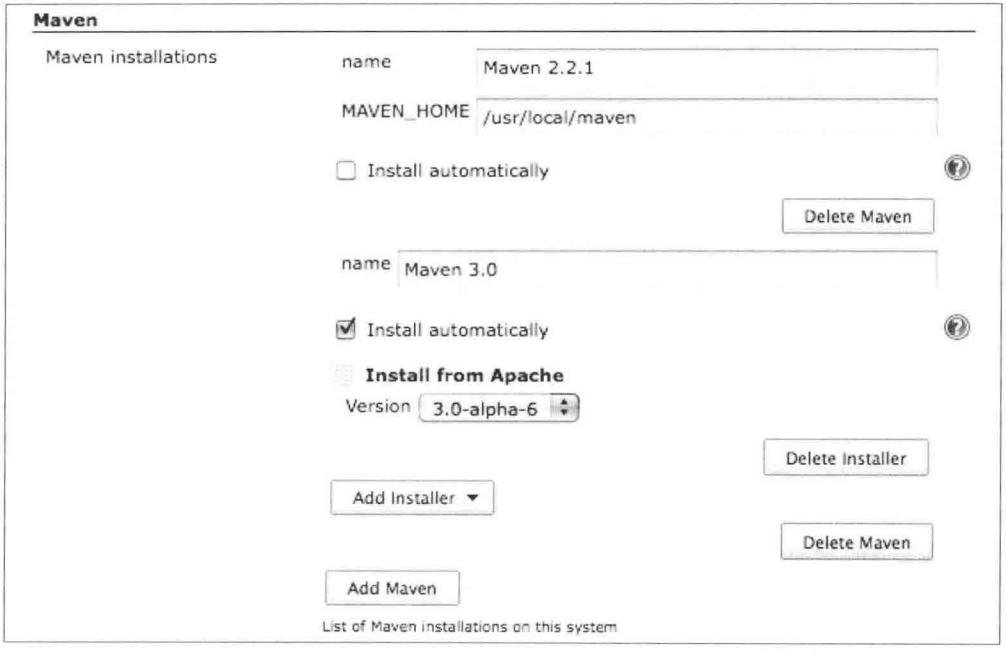


图4-7　在Jenkins中的Maven配置

如果你勾选 Install automatically 复选框，Jenkins 将下载并安装对应的 Maven 版本。你可以让 Jenkins 从 Apache 网站直接下载 Maven，或从本地选择路径。当你使用分布式构建时，这是一个很好的选择，因为 Maven 是交叉平台，它可以工作在任何一台机器上。你无须在首次使用 Maven 时在每台机器上进行安装，Jenkins 会自动下载并将 Maven 安装到其主目录下的 *tools* 目录中。

有时你需要将 Java 系统参数传给 Maven 构建进程，比如经常用于 Maven 中类似代码覆盖或生成站点等一些比较耗内存的作业时。你可以通过设置 MAVEN_OPTS 属性进行配置。在 Jenkins 中，你可以设置系统级范围的默认值，并可在所有项目中调用（见图 4-8）。如果你想在所有项目中使用标准内存选项（例如），则无须手动在每个项目中做设置。

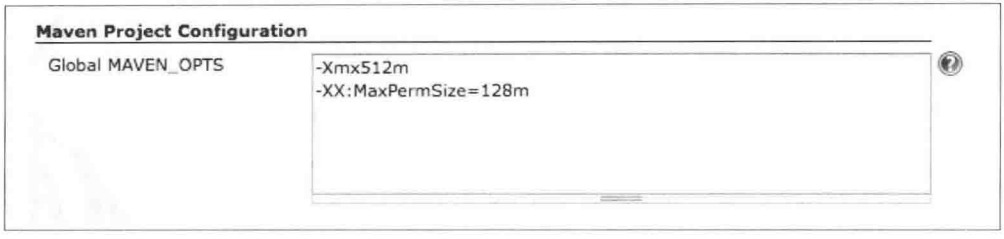


图4-8　配置系统范围参数MAVEN_OPTS

Ant

Ant 是一个被广泛应用并且非常知名的构建 Java 的脚本语言。它是一种应用于大量开源项目中灵活、可扩展、相对简单的脚本语言。一个 Ant 构建脚本（通常是 *build.xml*）由多个 *target* 组成。每个 target 在构建过程中负责执行特定的工作，比如编译你的代码或运行你的单元测试。它是通过执行任务来做到这一点的，这些任务会执行构建作业的指定部分，比如调用 *javac* 编译代码，或创建新目录。target 之间也有依赖（dependency），来指明构建作业的执行顺序。例如，编译代码需要在执行单元测试之前进行。

Jenkins 为 Ant 提供了优秀的内置支持，可以从构建作业中直接调用 Ant 的 target，提供属性来定制流程。我们将在本书的后面章节中详细解释具体操作。

75 如果 Ant 存在于系统中，则 Jenkins 可以查找到它，然后，如果你想知道自己正在使用的是哪个 Ant 版本，或你需要分别为不同的构建作业使用不同的 Ant 版本，则可以配置安装所有你需要的 Ant 版本（见图 4-9）。只需在配置系统界面中的 Ant 配置处提供一个名字和安装目录即可。之后，你将可以为自己的每个项目指定需要的 Ant 版本。

如果你勾选 Install automatically 复选框，Jenkins 将下载 Ant 并安装到 Jenkins 主目录下的 *tools* 目录中，这与 Maven 一样。它将在构建作业第一次使用 Ant 时，从 Apache 网站或是本地路径下载一个 Ant 的安装文件。同样，这是一个很好的可以让构建服务器达到标准化的方法，并能使在已有架构上添加新的分布式构建服务器变得更容易。



图4-9　在Jenkins中的Ant配置

Shell 脚本

如果你的构建服务器是运行在 UNIX 或 Linux 上的，则 Jenkins 支持在构建作业中添加 Shell 脚本。这可以方便地解决那些你不想在 Ant 或 Maven 中执行底层、系统相关的作业。在 Shell 配置部分，执行 Shell 脚本时将使用你定义的默认 Shell。默认情况下，就是 */bin/sh*，但是有时你可能想要修改成 *bash* 或是 *Perl* 等其他的解释器。

在 Windows 中，Shell 配置部分不适用——你需使用 Windows 批处理脚本代替它。所以在 Windows 构建服务器上，你应该让这个配置项保留空白。

配置版本控制工具

Jenkins 预装了 CVS 和 Subversion 的插件。其他版本控制系统插件可以通过 Manage Plugins 界面进行下载。

配置 Subversion

76

Subversion 无须特殊的配置，因为 Jenkins 使用本地 Java 库与 Subversion 仓库进行交互。如果连接到仓库需要验证，则 Jenkins 会在你进入 Subversion 的 URL 时，在构建作业配置处进行提示。

配置 CVS

CVS 只需很少的配置或根本无须配置。默认情况下，Jenkins 将在系统路径上查找像 CVS 这样的工具；如果它不在系统路径上，你也可以提供明确的路径。CVS 将登录账

号和密码等细节信息保留在一个叫 *.cvspass* 文件中，通常这个文件在你的 home 目录中。如果并非如此，则你可以提供一个让 Jenkins 能够找到这个文件的路径。

配置邮件服务器

最后一个你需要在基本配置选项界面做设置的是邮件服务器。email 是 Jenkins 最基本的通知技术——在一个构建失败后，它将给提交代码变更的开发人员及可选的其他团队成员发邮件。所以，Jenkins 需要知道你的邮件服务器（见图 4-10）。



图4-10　在Jenkins中的邮件服务器配置

系统管理员的 email 地址是 Jenkins 发送通知的邮件发送方。你还可以单击 Test configuration（测试配置）按钮来检查它，Jenkins 会发一个测试邮件到这个地址。

在许多机构中，你可以获得用户登录名加其机构域名的 email 地址。例如，在 ACME 中，
77 用户 John Smith 有一个登录名“jsmith”和一个 email 地址“jsmith@acme.com”。如果延伸到版本控制系统，Jenkins 可以节省大量的配置工作。在前面的示例中，你可以简单地指定 acme.com 作为默认用户 email 的后缀，然后 Jenkins 会补全剩余部分。

你还需要给 Jenkins 服务器（不适用 localhost）提供一个 URL，Jenkins 将这个 URL 放入通知邮件中，以便用户可以直接从 email 中链接到 Jenkins 的构建失败界面。

Jenkins 还提供了更复杂的邮件配置，使用如 SMTP 认证和 SSL 等高级特性。可以通过单击 Advanced（高级）按钮来配置这些选项。

例如，很多机构使用 Google Apps 的邮件服务，你可以配置 Jenkins 与 Gmail 服务协作，如图 4-11 所示。你需要做的就是使用 Gmail 的 SMTP 服务器并在 SMTP 认证中提供你的 Gmail 用户名和密码（你还需要使用 SSL 和非标准端口 465）。



图4-11　在Jenkins中使用Google Apps域配置邮件服务器

配置代理

在大多数企业环境中，Jenkins服务器是在防火墙之后的，也不会直接访问互联网。Jenkins需要上网从外部站点下载插件和更新，或是安装诸如JDK、Ant和Maven等工具。如果你需要通过一个HTTP代理服务器上网，则可以在Configure System界面（见图4-12）的高级（Advanced）中配置链接细节（服务器和端口，以及可能需要的用户名和密码）。

78

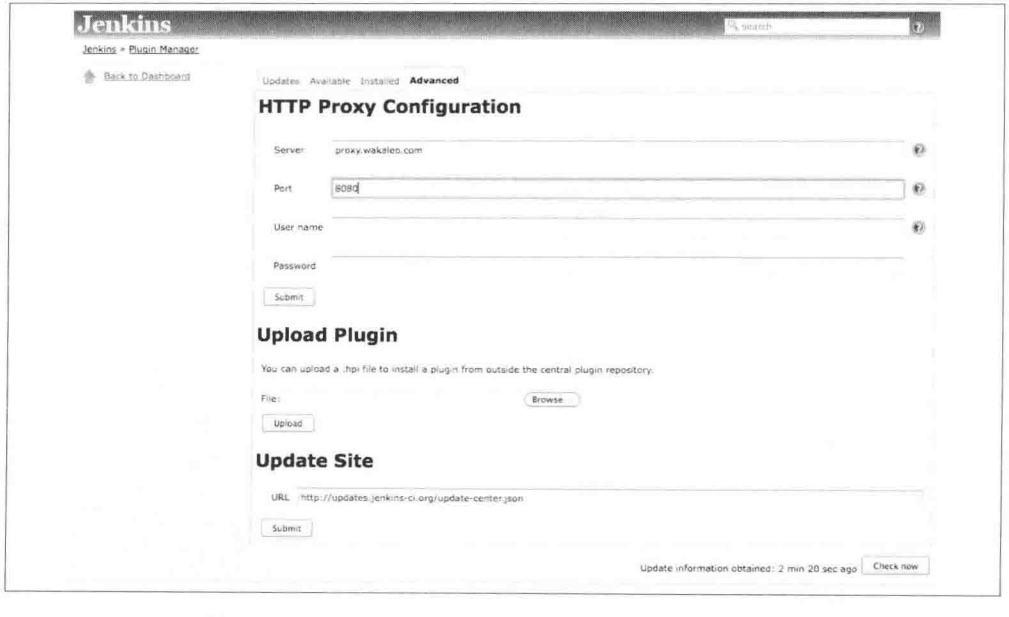


图4-12　为Jenkins配置代理

如果你的代理使用微软 NTLM 身份认证方案，那么你需要提供域名和用户名。你可以在用户名输入框中加入这两项：输入域名，其后紧跟一个反斜杠（\），再跟用户名，就像“MyDomain\Joe Bloggs”这样。

最后，如果你是在 Jenkins 构建服务器上设置代理的话，则请注意，所有服务器上运行的工具最好也都配置此代理。特别是像 Subversion（如果你访问外部的仓库）和 Maven（如果你没用企业级仓库管理器）这类的工具。

小结

你在开始用 Jenkins 时不需要大量的配置工作。所需要的配置是相当简单的，并主要集中在配置系统（Configure System）界面。一旦做完以上配置，就可以准备去创建第一个 Jenkins 构建作业了！

第 5 章 设置构建作业

简介

构建作业是一个持续集成服务器的基本职能。

构建作业是一种用来进行编译、测试、打包、部署或者其他一些项目相关操作的特殊方式。构建作业的形式多种多样：你可能要编译和单元测试应用程序，报告源代码相关的代码质量度量，生成文档，生成程序组建包，部署到生产环境，运行自动化冒烟测试，或者其他类似的作业。

软件项目通常会包含几个相关的构建作业。例如，你可能会选择启动专门的构建作业用来运行所有的单元测试。如果这些测试通过，则你可能进行一个执行更长时间的集成测试，运行代码质量度量，或生成技术文档，最后捆绑你的 Web 应用程序并将其部署到测试服务器。

在 Jenkins 中，构建作业很容易建立。在本章中，我们将看看主要类型的构建作业，以及如何配置它们。在后面的各章中，我们将进一步深入学习，看一下如何组织多个构建作业，如何建立作业管道，以及如何把部署过程自动化。但现在，让我们从如何在 Jenkins 中设置基本构建作业开始吧。

Jenkins 构建作业

在 Jenkins 中创建新的作业相当简单：只需单击 Jenkins 仪表盘上的“New Job”菜单项。Jenkins 支持几种不同类型的构建作业，当你选择构建一个新的作业时会看到（见图 5-1）。

自由式软件项目

自由式构建作业是通用的构建作业，这提供了最大限度的灵活性。

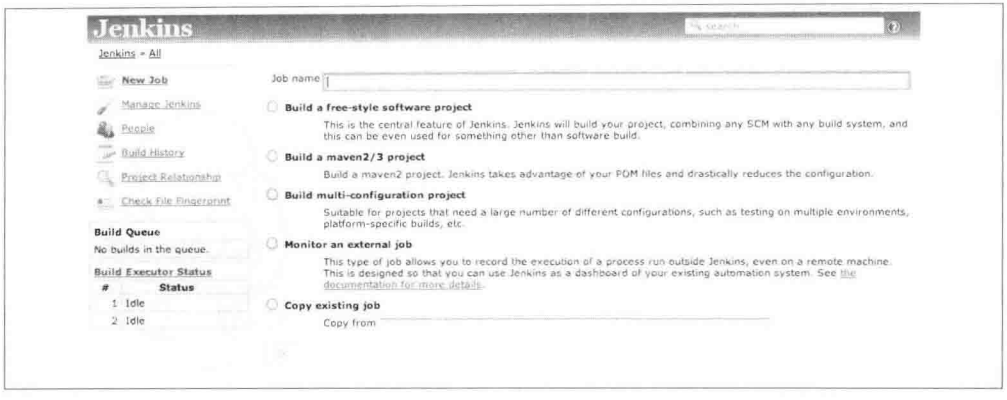


图5-1　Jenkins支持四种类型的构建作业

Maven 项目

“maven2/3 project”是专门适用于 Maven 项目的构建作业。Jenkins 了解 Maven *pom* 文件和项目结构，并可以使用 *pom* 文件中收集的信息减少你需要设置项目执行的工作。

监视外部作业

“监视外部作业”构建作业让你留意非交互的过程，例如 cron 作业。

多组态作业

“多组态项目”（也被称为“矩阵项目”）可以允许你在许多不同的配置下运行相同的构建作业。这种强大的功能可用于在许多不同的环境中，不同的数据库，甚至在不同的构建机器上测试应用程序。在本书后面我们将看到如何配置多组态构建作业。

你还可以复制现有的作业，这会创建和现有的构建作业非常相似的新的作业（只要修改配置细节就可以了）。

在本章中，我们将专注于构建作业最常用的前两种类型。我们会在接下来的章节探讨其他的构建作业类型。让我们从最灵活的开始：自由式构建作业。

创建自由式构建作业

自由式构建作业是最灵活和可配置的选项，并且可用于任何类型的项目。它的配置相对简单，我们在此处配置的许多选项也会出现在其他构建作业中。

通用选项

当你创建一个新的自由式作业时，在第一部分你看到它包括一些通用的信息，比如唯一的名字、描述，以及其他信息，如构建作业如何以及在何处执行（见图 5-2）。

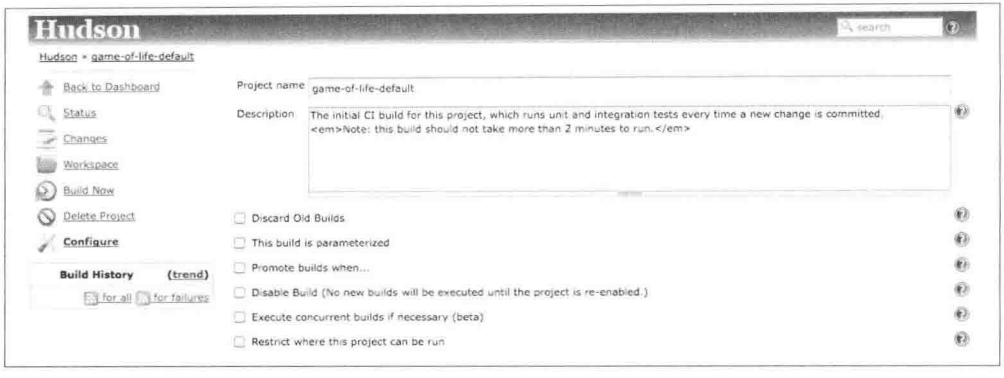


图5-2　创建新的构建作业

项目名称可以是任何你喜欢的名字，但是值得注意的是它将用于项目目录和构建作业 URL，所以我通常避免使用带有空格的名称。项目描述会在项目主页上——使用它可以提供关于构建作业的目的和上下文的视图。在这个字段也支持 HTML 标签。

其他的选项更加技术性，我们在本书的后面再详细探讨。

你应该提前考虑的一个重要方面是你希望如何处理构建历史。构建作业会消耗大量的磁盘空间，尤其是你存储构建产物（由你的构建作业生成的二进制文件，比如 JAR、WAR、TAR 等）。即使没有构建产物，记录每一个构建作业也会消耗额外的磁盘空间和内存（根据你的构建作业性质，可能合理，也可能不合理）。例如，生成关于随着时间的推移静态分析和代码覆盖率度量的报告的代码质量度量作业，可能要为整个项目生成记录。然而，对于自动部署应用程序到测试服务器的构建作业，保留构建历史和构建产物可能不太重要。

Discard Old Builds 选项允许你限制在构建历史记录的作业数。你可以告诉 Jenkins 只保留最近的构建（Jenkins 会在一定的天数后删除构建）或者保留不超过指定数量的构建。如果某个构建有特殊敏感价值，则在构建细节页面单击 Keep this build forever 按钮，你可以告诉 Jenkins 永远保留这个构建（见图 5-3）。请注意只有你在 Jenkins 设定丢弃旧的构建时，该按钮才会显示。

82

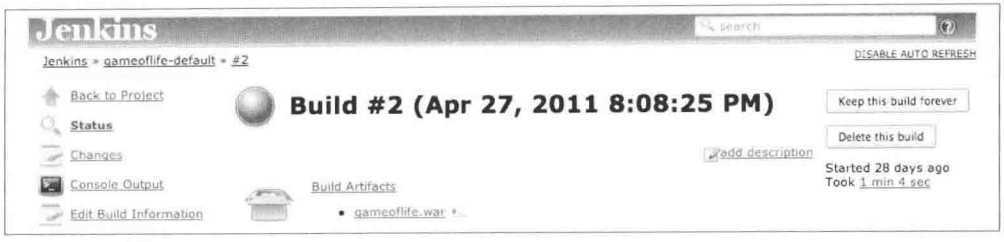


图5-3　永久保留构建作业

此外，Jenkins 永远不会删除最后一个稳定和成功的构建（不管过了多久）。例如，如果你限制 Jenkins 只保留最后 20 个构建，且上一个成功的构建是 30 个之前，则 Jenkins 会继续保留成功的构建作业以及最后 20 个失败的构建。

你还可以选择禁用该构建。一个禁用的构建直到你重新启用它才会被执行。当你创建一个新的构建作业后使用此选项是相当少见的。另一方面，在维护或重大重构时临时暂停一个构建，此选项就派上用场了，这时通知大家构建失败并不是很有用。

高级项目选项

顾名思义，Advanced Project（高级项目）选项包含较少需要的配置选项。你要单击 Advanced 按钮让它们显示出来（见图 5-4）。

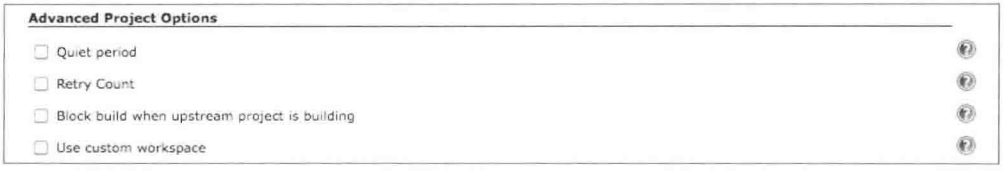


图5-4　单击Advanced按钮，显示高级选项

在构建作业配置的 Quiet period 选项可以覆盖在 Jenkins 系统配置中定义的系统级别（参见第 4 章的“配置系统环境”一节）的安静期。这个选项主要用来不支持原子提交的版本控制系统，比如 CVS，但是有时候也用在开发人员有多次小的提交代码习惯的团队。

当几个相关的项目受到单个提交影响，并且必须以特定的顺序构建时会用到“Block build when upstream project is building”选项。如果你激活此选项，Jenkins 会在开始此构建之前一直等到任何上游的构建作业（参见本章的“Build Triggers”一节）完成。

83 例如，当你发布一个新版本的多模块的 Maven 项目时，项目模块的版本号会经常更新。例如，假定我们在第 2 章 Game of Life 添加了一个 Web 应用程序，设置它为一个单独的 Maven 项目。当我们发布这个项目的一个新版本时，核心和 Web 应用程序的版本号码都

会被更新（见图 5-5）。在我们构建 Web 应用之前，需要构建原始 Game of Life 核心模块的新版本。然而如果每个模块你都有独立自由式构建，那么核心和 Web 应用程序的构建作业会同时开始。如果核心构建作业还没有生成新版本的核心模块，Web 应用程序的构建作业就会失败。

要避免此问题，你可以设置等到核心构建成功之后再开始 Web 应用程序的构建作业。但是这意味着如果核心模块没有修改，仅仅是 Web 应用程序做了修改，则它永远都不能构建。一个更好的方法是使用“Block build when upstream project is building”选项。在这种情况下，当版本控制系统的版本号更新时，Jenkins 会编排执行两个构建。然而它会等待，直到核心完成后，才开始 Web 应用程序的构建。

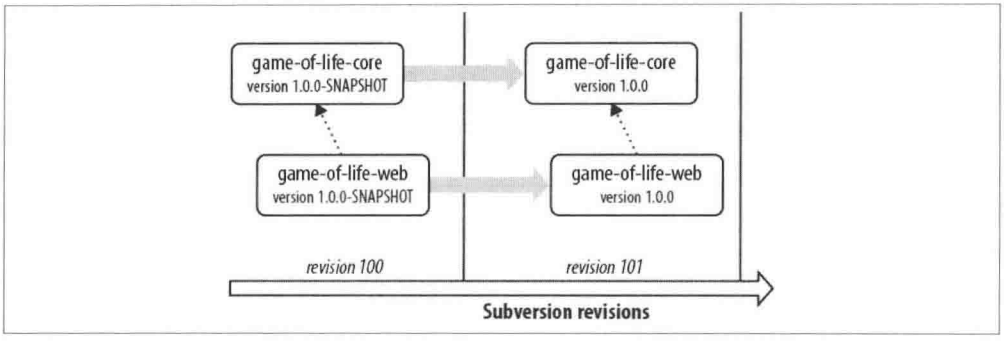


图5-5 当一个提交会影响几个相关的项目时会用到“Block build when upstream project is building”选项

你还可以重写 Jenkins 用以检出源代码默认的工作区间和构建你的项目。通常情况下，Jenkins 为你的项目创建特定的工作空间目录，该目录可在项目的构建作业目录（参见第 3 章的“Jenkins 主目录里有什么”一节）中找到。这在几乎所有情况下正常工作。但是，有时候你需要重写此选项，并迫使 Jenkins 使用一个专门的目录。一个常见的例子是你想让几个构建项目先后在同一个目录中工作。通过勾选“Use custom workspace”选项，并提供自己的路径，你可以覆盖默认的目录。路径可以是绝对的，也可以是 Jenkins 主目录的相对路径。

在本书后面的章节我们会看到一些其他更高级的选项。

84

配置源代码管理

持续集成服务器最基本的作用是监视你的版本控制系统，当有新的更改发生时检出代码。然后服务器编译、测试最新版本的代码。或者，它可能只是定期检出并构建你的源代码的最新版本。在任一情况下，与你的版本控制系统紧密集成是必不可少的。

由于其基础性的作用，在 Jenkins 里面，SCM 配置选项在所有的构建作业中都是完全一样的。Jenkins 开箱即用式支持 CVS 和 Subversion，内置支持 Git，还通过插件与大量的其他版本控制系统集成。在本书写作之际，SCM 对插件的支持包括 Accurev、Bazaar、BitKeeper、ClearCase、CMVC、Dimensions、Git、CA Harvest、Mercurial、Perforce、PVCS、StarTeam、CM/Synergy、Microsoft Team Foundation Server 甚至 Visual SourceSafe。下面我们将看看如何配置一些更常见的 SCM 工具。

用 Subversion 工作

Subversion 是最广泛使用的版本控制系统之一，Jenkins 捆绑完全支持 Subversion（见图 5-6）。为了从 Subversion 仓库使用源代码，你需要提供相应的 Subversion 的 URL——它对三种 Subversion 协议（http、svn 或者 file）都支持得很好。你输入后 Jenkins 会检查 URL 的有效性。如果仓库要求身份认证，Jenkins 会自动提示你输入相应的密码，并将它们存储起来给访问此仓库的其他构建作业使用。

85

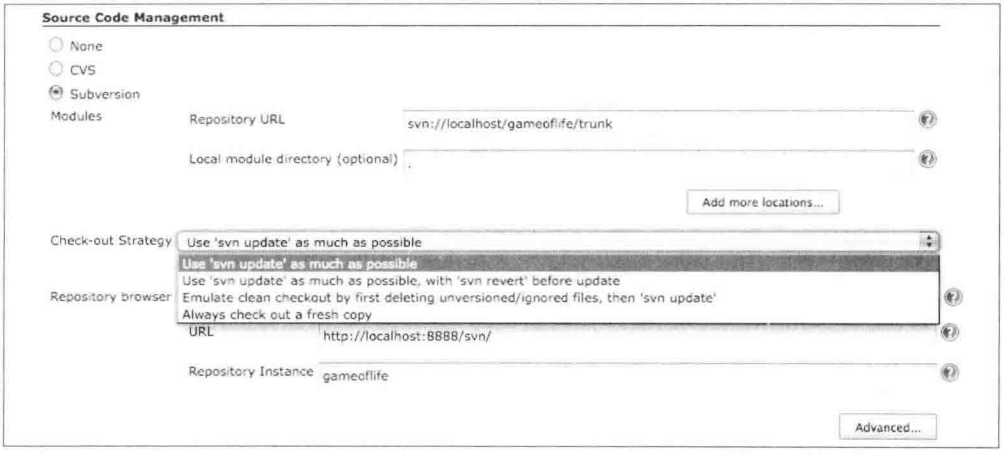


图5-6　Jenkins提供Subversion的内建支持

默认情况下，Jenkins 把代码库的内容检出到你的工作空间的子目录，它的名字将匹配 Subversion URL 的最后一个元素。所以如果你的 Subversion URL 是 svn://localhost/gameoflife/trunk，Jenkins 会检出代码库的内容到名为 *trunk* 的构建作业空间。如果你更喜欢其他的目录名称，则在 *Local module directory* 字段输入你想要的目录名。如果想要 Jenkins 把源代码直接检出到工作空间，则在这里写上“.”。

有时候你可能需要从多个 Subversion URL 获取源代码。在这种情况下，使用“Add more locations”按钮，添加你需要的源代码仓库。

一个精心设计的构建过程不应该修改源代码，或者留下任何可能混淆你的版本控制系统或者构建过程的额外文件。生成的构建产物和临时文件（例如日志文件、报告、测试数据或者基于文件的数据库）应该在一个预留的特定目录中（如 Maven 构建的 *target* 目录），和（或）将其配置为被你的版本控制仓库忽略。一旦构建完成，它们也应该作为构建过程的一部分被删除。这也是确保一个干净和可重复的构建过程的重要组成部分——对于一个给定版本的源代码，你的构建应该表现为完全相同的方式，不论它何时何地运行。本地更改的源代码文件及临时文件的存在都可能有潜在的危害。

通过在 Check-out Strategy 下拉列表中选择合适的值，你可以微调 Jenkins 从 Subversion 仓库获得最新的源代码的方式。如果你的项目一切正常，则可以通过选择“Use ‘svn update’ as much as possible”显著地加快速度。这是最快的选项，但是在你的工作空间可能会留下早期的构建产物和文件。为了安全起见，你可能想使用第二个选项（“Use ‘svn update’ as much as possible, with ‘svn revert’ before update”），其中在系统地运行 `svn update` 之前运行 `svn revert`。这将确保没有本地文件已被修改（虽然它不会删除任何已在构建过程中创建的新文件）。或者，你应该让 Jenkins 在执行 `svn update` 之前删除没有版本的或被忽略的文件，或者为每个构建检出完全干净的副本。

另一个非常有用的功能是 Jenkins 和源代码浏览器集成。好的源代码浏览器是持续集成配置的重要组成部分。它可以让你一眼就能看出什么变化触发给定的构建，这对于分析中断的构建的原因非常有用（见图 5-7）。Jenkins 和绝大多数源代码浏览器集成，包括开源的工具如 WebSVN 和 Sventon，以及像 Atlassian's FishEye 这样的商业软件。

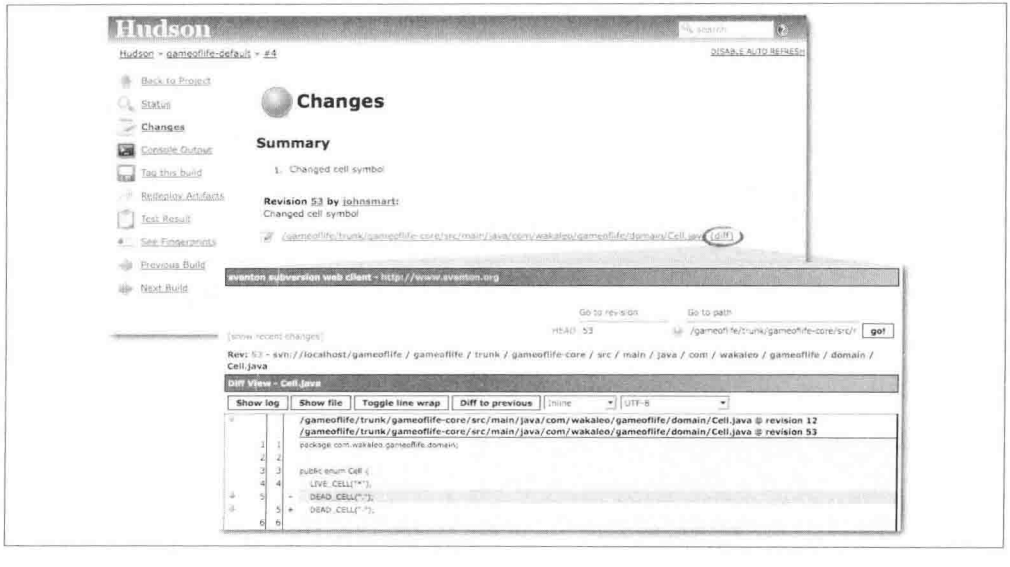


图5-7　源代码浏览器显示哪些代码更改触发了构建

Jenkins 还允许你调整触发构建的变化。在 Advanced 部分，你可以使用 Excluded Regions 域字段告诉 Jenkins 如果只更改了某些文件并不会触发构建。此字段通过正则表达式的列表，识别那些不应该触发构建的文件。例如，如果只是图像被更改，则你不想让 Jenkins 开始一个新的构建。要做到这一点，你可以使用类似于以下内容的一组正则表达式：

```
/trunk/gameoflife/gameoflife-web/src/main/webapp/.*\.jpg
/trunk/gameoflife/gameoflife-web/src/main/webapp/.*\.gif
/trunk/gameoflife/gameoflife-web/src/main/webapp/.*\.png
```

或者，如果你只对部分源代码目录结构变化感兴趣的话，则可以指定 Included Regions。你甚至可以合并 Excluded Regions 和 Included Regions 字段——这种情况下，如果修改的文件在 Included Regions 但是不在 Excluded Regions，这会触发一个构建。

你也可以忽略来自某些用户（Excluded Users），或者某些提交信息（Excluded Commit Messages）的更改。例如，如果你的项目使用 Maven，你可能想用 Maven Release 插件来促进自己的应用程序从快照版本到正式发布。这个插件会自动增加你的应用程序的版本号，从开发期间使用的快照版本（如 1.0.1-SNAPSHOT）到发布版本（1.0.1），然后用这个版本打包和部署你的应用程序，之后在正在进行的开发过程中移到下一个快照编号（例如 1.0.2-SNAPSHOT）。在此过程中，Maven 负责许多 SCM 记录的任务，比如提
87 交源代码的发布版本号，为你的应用程序发布版本创建标签，然后用新的快照版本号码提交源代码。

现在假设你有一种使用这个过程生成一个新版本的特殊的构建作业。在 Jenkins 中，Maven Release 插件生成的许多提交通常会触发构建作业。然而，因为发布构建作业已经编译、测试此版本的应用程序，所以你不需要 Jenkins 在一个独立的构建作业里面再做一次。为了确保 Jenkins 在这种情况下不会触发构建，你可以在 Excluded Commit Messages 字段使用下面的值：

```
[maven-release-plugin] prepare release.*
```

这将确保 Jenkins 会忽略和新的发布版本相关的变化，而不是下一个快照版本相应的变化。

使用 Git

（由 Matthew McCullough 撰写）

Git（*http://git-scm.com/*）是流行的分布式版本控制系统，它是 Subversion（*http://subversion.tigris.org/*）的逻辑继承者，也是 Mercurial（*http://mercurial.selenic.com/*）的竞争者。在 Jenkins 中对于 Git 的支持相当成熟并且十分全面。在 Jenkins 中有许多插件

提供了对 Git 的支持。我们首先从 Git 插件开始，在 Jenkins 中，它提供了对 Git 的核心支持。我们稍后会讨论补充插件。

安装插件

在 Jenkins 插件管理器（*Jenkins Plugin Manager*）中可以获得 Git 插件，并记录在其自己的 wiki（*http://wiki.hudson-ci.org/display/HUDSON/Git+Plugin*）页面。假设插件 Git（1.3.3 或更高的版本）已经安装在你的构建服务器上，所以你需要确保自己确实已经在构建服务器上安装了 Git。你可以在构建服务器上运行以下的命令：

```
$ git --version
git version 1.7.1
```

下一步，回到 Jenkins，在 Jenkins 插件管理器页面中勾选对应的复选框并且单击 Install 按钮。

插件的系统配置。安装 Git 插件后，在 Manage Jenkins → Configure System 页面（见图 5-8）有一小组新的配置选项。尤其是，你需要提供 Git 可执行文件的路径。如果 Git 已经安装在系统路径上，则只要写上“git”就可以了。

SSH 密钥配置。如果你使用 SSH 非密码认证的方式访问 Git 仓库——例如，访问地址类似于 `git@github.com:matthewmccullough/some-repo.git`——你需要提供私钥文件 `~/.ssh/id_rsa` 对应的公钥，`~` 是 Jenkins 运行的用户账户的主目录。

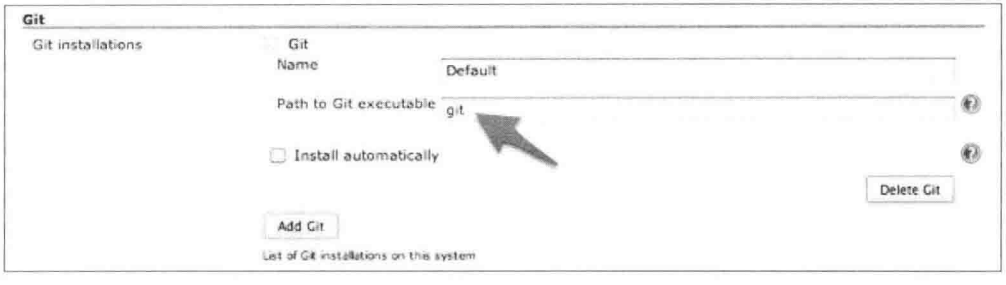


图5-8　Git插件的系统配置

远程服务器的指纹还需要放在 `~/.ssh/known_hosts` 里面，以防止第一次访问 Git 服务器时，Jenkins 无形的提示访问授权。

或者，如果启动 `jenkins` 用户可以登录，则以 `jenkins` 用户 SSH 到 Jenkins 机器上，手工尝试 Git 克隆远程仓库。这会检测你的私钥的设置，在 `~/.ssh` 目录创建 `known_hosts` 文件。这可能是对于不熟悉复杂 SSH 配置的用户，最简单的选择。

使用插件

在现有的或者新的 Jenkins 项目中，会显示 Git 源代码管理（Source Code Management）选项。在这里，你可以配置一个或者多个仓库地址（见图 5-9）。对大多数项目而言一个仓库通常就够了：在更加复杂的情况下，添加第二个仓库也是有用的，并且允许你为 `pull` 和 `push` 指定不同的命名位置。

高级的每个项目源代码管理配置。在大多数情况下，你在使用的 Git 仓库的 URL 应该足够了。然而，如果你需要更多的选项，则可单击 Advanced 按钮（见图 5-10）。这为 `pull` 行为提供了更加精确的控制。

仓库的名称是一个用于速记给定仓库的标题（`remote` 是远程 Git 仓库的一种表示方法），你可以在后面的合并操作配置中引用这个名称。

Refspec 是用于精确控制从远程服务器检索的内容和存储在本地命名空间的特定 Git 语言（*http://progit.org/book/ch9-5.html*）。

构建的分支。branch specifier（见图 5-11）是 Jenkins 要构建的通配符模式或者特定分支名字。如果为空，则所有的分支都会被构建。在本书写作之际，在第一次用一个空白的分支做构建设置时，在保存作业之后，它就填充了 **，这意味着“构建所有的分支”。

89

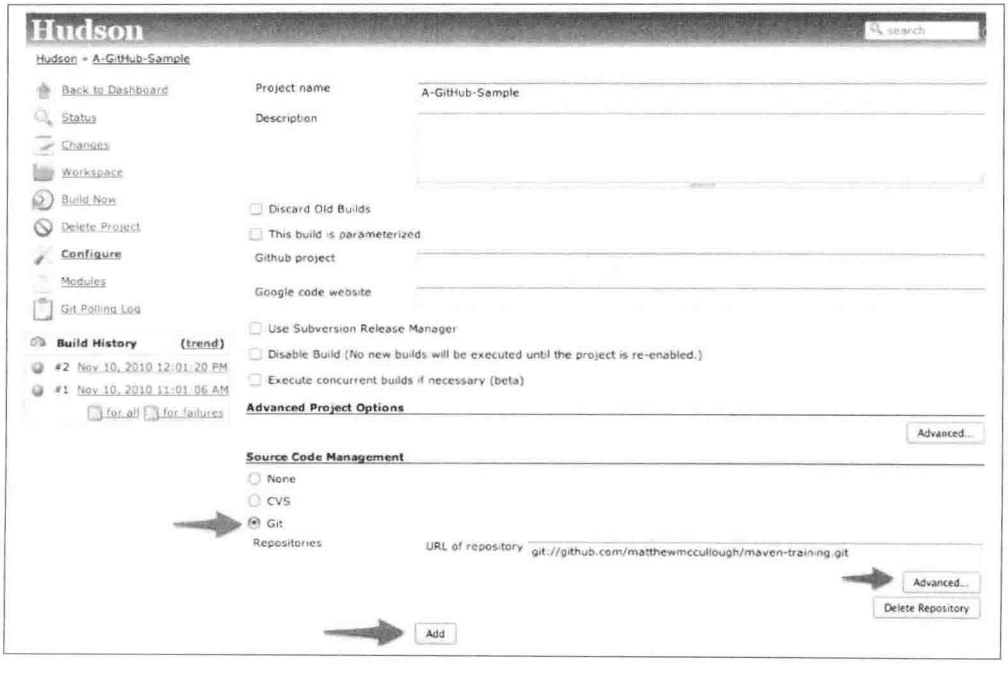


图5-9　输入Git仓库URL

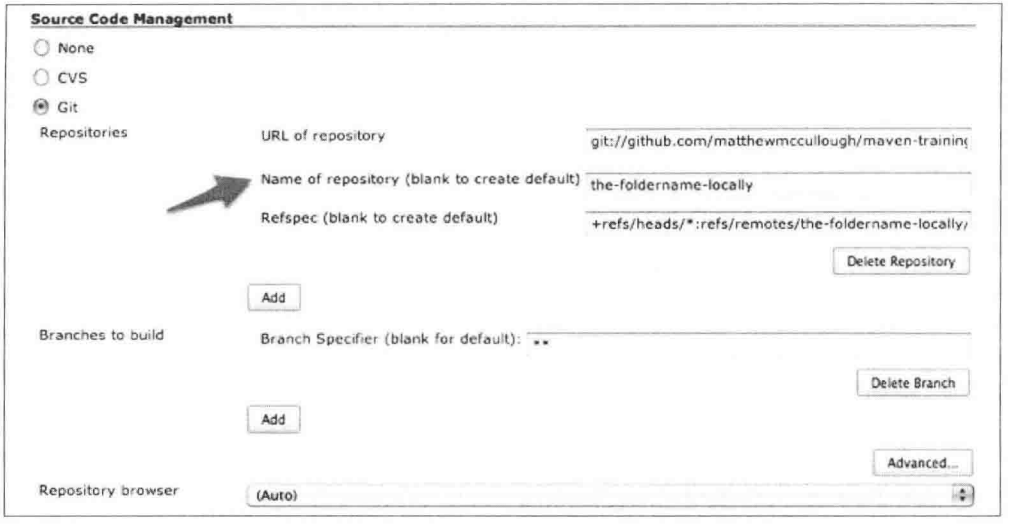


图5-10　Git仓库URL的高级配置

Excluded Regions。区域（见图 5-12）在代码库中是一种特殊命名或通配符路径，即使发生更改时也不应该触发构建。通常这些都是没有被编译的文件（比如本地化包或者图像），这是可以理解的，可能不会对单元或集成测试产生影响。

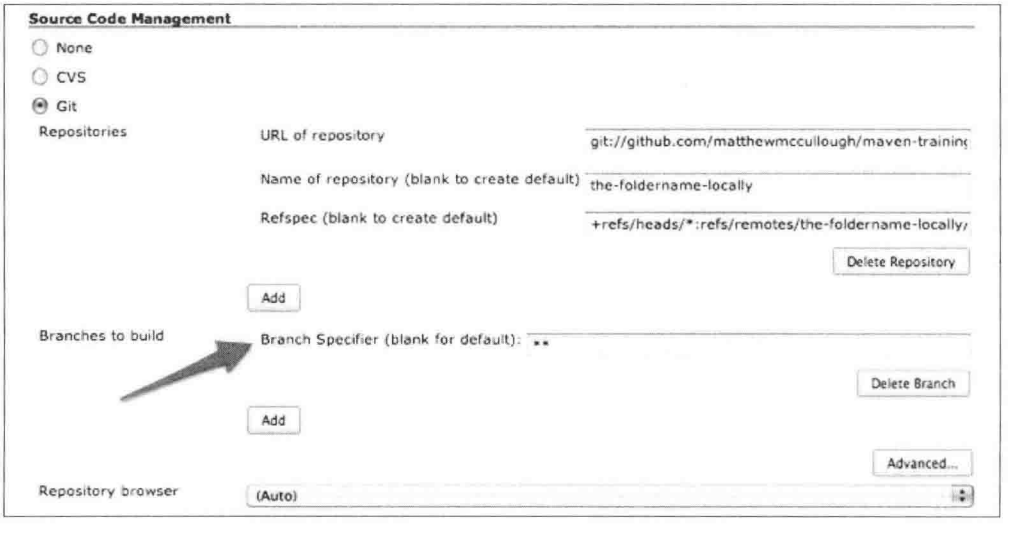


图 5-11　构建Git分支的高级配置

Excluded Users。Git 插件还允许你忽略某些用户(即使他们改变的代码通常会触发构建)。这不像听起来那么恶意：被排除的用户通常是自动化用户，不是开发人员，碰巧在源控

制系统具有提交权限的不同账号。这些自动化的用户经常执行小的变化（比如在 *pom.xml* 文件增加版本号），而不是实际的逻辑改变。如果你想排除几个用户，则把它们放在单独的行。

Checkout/merge to local branch。有时候，你可能想要从你指定的树创建一个本地分支，而不是只使用提交的哈希直接分离出的 HEAD 检出。在这种情况下，在“Checkout/merge to a local branch”字段指定你的本地分支。

用一个例子来解释更加容易一些。如果不指定一个本地分支，插件会做这样的事情：

```
git checkout 73434e4a0af0f51c242f5ae8efc51a88383afc8a
```

另一方面，如果你使用名为 `mylocalbranch` 的本地分支，Jenkins 会做下面的事情：

```
git branch -D mylocalbranch
git checkout -b mylocalbranch 73434e4a0af0f51c242f5ae8efc51a88383afc8a
```

91

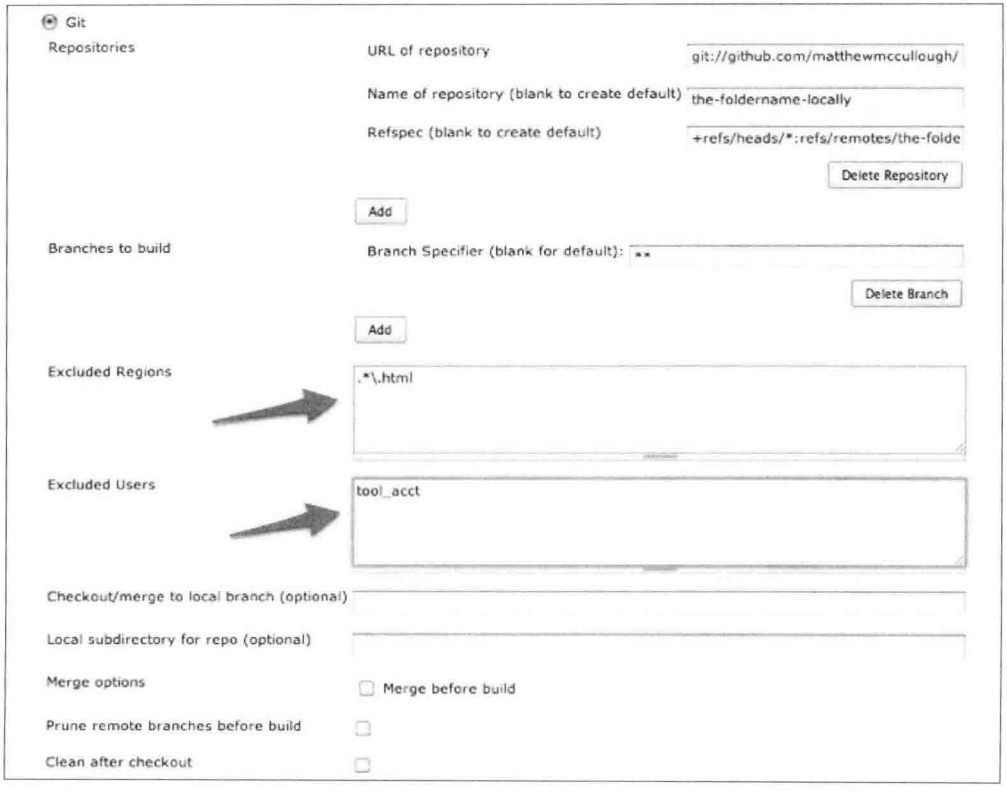


图5-12　分支和域

Local subdirectory for repo。默认情况下，Jenkins 直接把 Git 仓库克隆到构建作业工作空间。如果你想使用一个不同的目录，则可以在这里指定它。请注意，你所指定的目录是对构建作业工作空间的相对路径。

Merge before build。使用此选项的典型情况是将一个集成分支合并到类似于主分支（`master`）上。请记住，只有没有冲突的情况下才会自动合并。需要人工干预的更复杂合并将导致构建失败。

除非启用了构建后push行为，否则由此产生的合并分支不会自动被push到另一个仓库中。

Prune remote branches before build。精简操作会删除那些以前克隆到本地但却在远程服务器上已经被删除了的远程分支的本地副本。总之，这是清理本地克隆并且与远程保持完美的同步。

Clean after checkout。激活 Git 设施来清除任何不用的文件或文件夹，让你的工作副本回到原始状态。

Recursively update submodules。如果你在项目中使用 Git 子模块的功能，则这个选项让你确保每个子模块用显式调用保持更新（即使子模块嵌套在其他子模块）。

92

Use commit author in changelog。Jenkins 在总结视图里面跟踪并显示更改代码的作者。Git 分别跟踪提交者和代码的作者，这个选项允许你在修改日志中切换显示哪个用户名。

Wipe out workspace。通常 Jenkins 会重用工作空间，仅在必要的时候刷新检出。如果你开启了“Clean after checkout”选项，就会清理未跟踪的文件。然而，如果你希望拥有一个完全的干净的工作空间，则可以使用“Wipe out workspace”选项删除并且从头开始构建。铭记，这可能会显著延长初始化和构建项目所需的时间。

Choosing strategy。Jenkins 基于策略（见图 5-13）来决定构建哪些分支。用户可以影响此分支搜索过程。默认的选择是搜索所有分支头。如果安装了 Gerrit 插件，则可以看到所有 Gerrit-notified 提交构建的附加选项。



图5-13　选择策略

Git executable。在 Jenkins 全局选项（见图 5-14），可以为每个构建设置不同的 Git 可执行路径。这并不常用，只有当克隆或者其他 Git 操作对于某个特定版本的 Git 高度敏感

时才会用得到。Git 对版本的支持非常灵活，稍旧的仓库可以用较新版本的 Git 克隆，反之亦然。

93

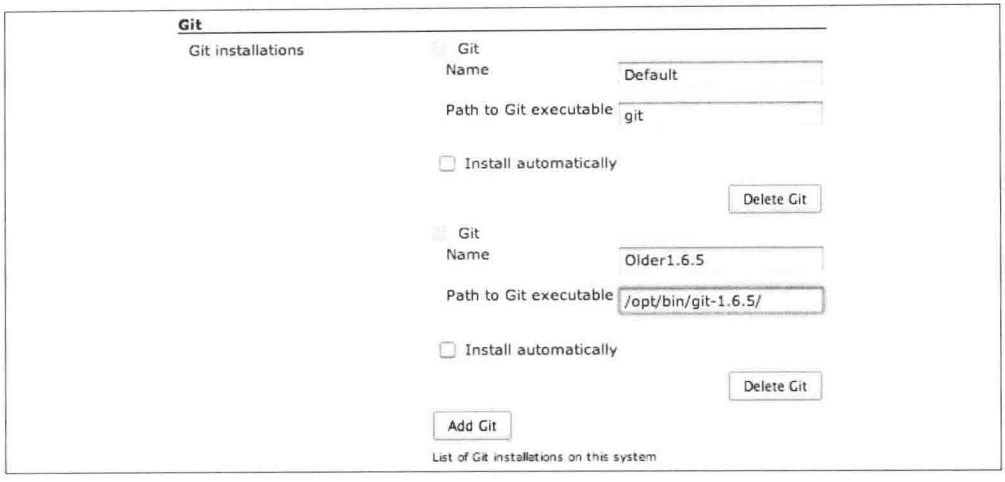


图5-14　Git可执行路径全局设置

Repository browser。像 Subversion 一样，Git 有几个你能够使用的源代码浏览器。最常见的是 Gitorious、Git Web 或者 GitHub。如果你提供了相应的仓库浏览器的 URL，Jenkins 会显示能够触发构建的源代码更改链接（见图 5-15）。

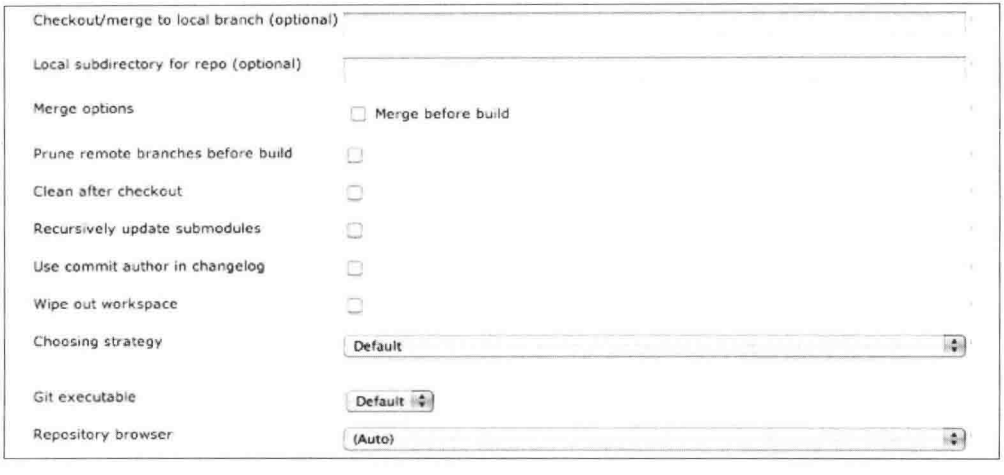


图 5-15　仓库浏览器

Build triggers

基本的 Git 插件提供了定时的基础上 *Poll SCM* 的能力，搜索自上次查询后的变化。如果

发现有变化，会开始一个构建。当查看某一特定的作业时，Polling Log（如图 5-16 所示）可以通过页面左侧导航栏的链接访问。它提供了关于仓库最后一次被轮询的时间的信息，还可能提供一个记录了更改的列表（见图 5-17）。

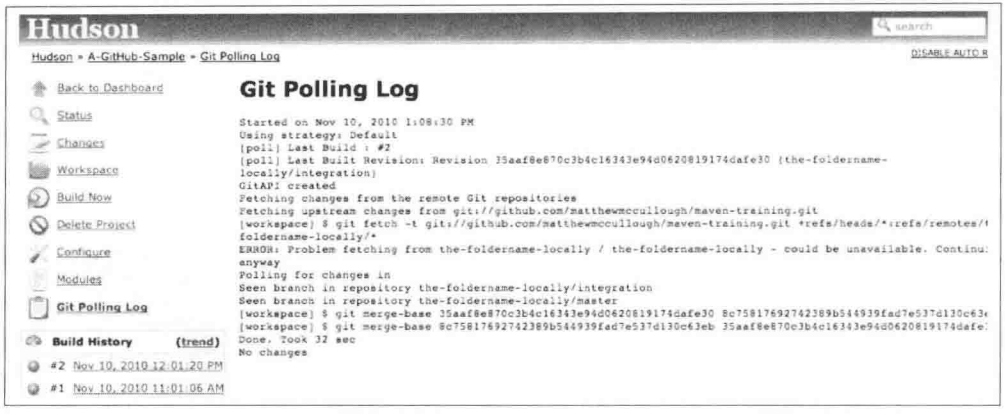


图 5-16　轮询日志

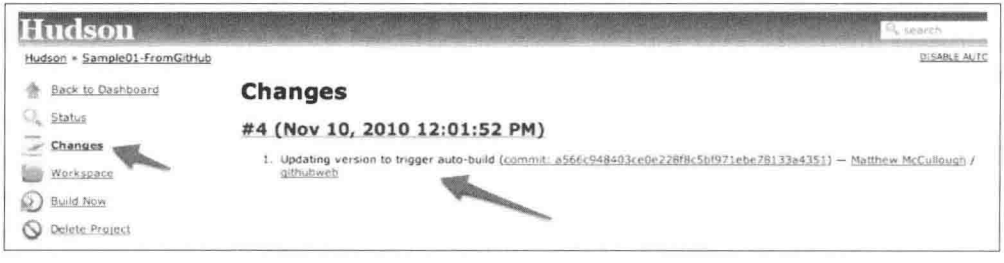


图 5-17　Git轮询的结果

Git 轮询可以提炼出更多对开发人员有用的格式，显示提交的评论、超链接用户名和更改的文件，以及更加详细的视图。

在安装 *Gerrit Build Trigger* 时添加 *Gerrit event* 选项比简单轮询仓库更加有效和精确。

Gerrit Trigger。Gerrit(*http://code.google.com/p/gerrit/*)是一个开源代码的 Web 应用程序，有利于对托管在 Git 版本控制系统的项目源代码进行代码审查（*https://review.source.android.com/#q,status:open,n,z*）。它读取传统的 Git 仓库，提供并排比较。在代码被审查之后，Gerrit 提供了发表评论的位置，可以把补丁设为*打开*、*合并*或*废弃*状态。

Gerrit Trigger（*http://wiki.hudson-ci.org/display/HUDSON/Gerrit+Trigger*）是一个 Jenkins 插件，当在 Git 仓库（见图 5-18）用户指定的项目发生任何特定的用户活动时，能触发 Jenkins 代码构建。它可用于替代更加常用的 *Build periodically* 或 *Poll SCM*。

这个插件的配置很简单，且专注于项目类型与模式和分支类型与模式。在每一个对中，类型可以是 *Plain*、*Path* 或者 *RegExp*——想要观察的类型模式——然后是想要评估的类型模式的值，正如向导中所使用的类型。

95

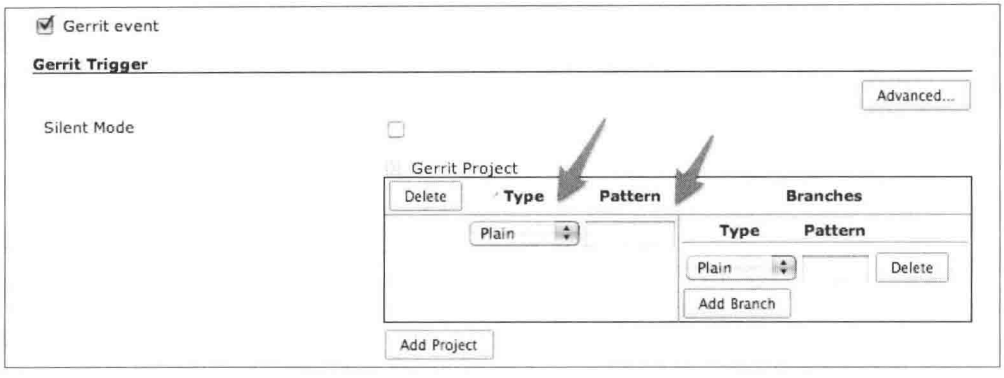


图 5-18　Gerrit Trigger

构建后的操作

在 Jenkins 中的 Git 插件将 Git 特定的功能添加到构建产物的后处理行为。具体而言，Git Publisher（见图 5-19）提供了合并和推动的功能。勾选 Git Publisher 复选框来显示四个选项。

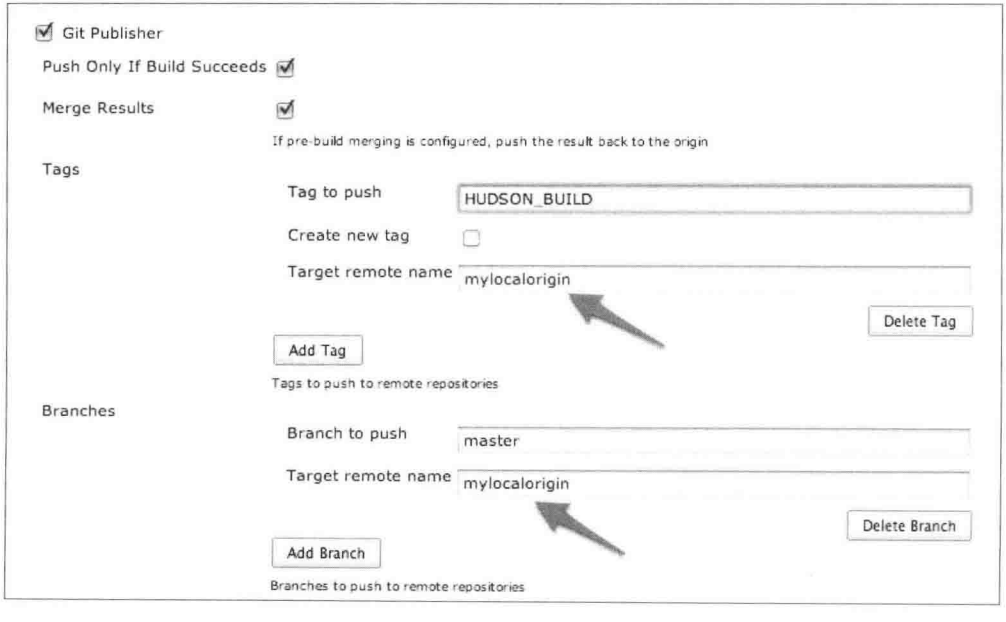


图 5-19　Git发布服务器

Push Only If Build Succeeds。如果在 Jenkins 构建过程中，合并或者其他产生提交的行为已经发生了，就可以推到远程。

Merge Results。如果配置了预构建的合并，则将合并结果的分支推送到它的起源（见图 5-20）。 96

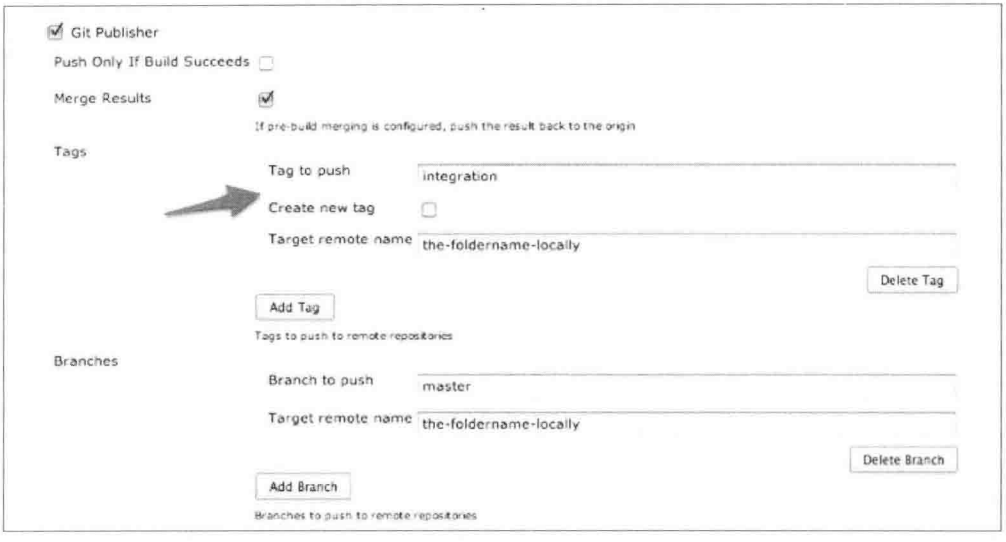


图 5-20　合并结果

Tags(标签)。在推送标签时，每个标签可以被命名，如果不存在的话则可以选择被创建（如果已经存在就会失败）。在标签名字中可以嵌套环境变量。实例包括进程 ID，例如 `HUDSON_BUILD_$PPID` 甚至是由 Jenkins 插件如 `$HUDSON_AUTOTAG_$BUILDNUM` 提供的构建版本号。标签可以针对特定的远程分支比如 `origin` 或 `intergrationrepo`。

Branches(分支)。在 Jenkins 应用程序的构建中使用的当前 HEAD 可以作为构建之后的步骤推送到其他远程分支。你只需要提供目标分支名称和远程名称。

远程分支的名字根据插件以前的配置进行验证。如果远程分支不存在，则会显示警告。

GitHub 插件

GitHub 插件提供了两个集成点。首先，它提供了到项目 GitHub 主页的可选链接。只要输入项目的 URL（不需要树 / 主干或者树 / 分支）就可以了。例如，`http://github.com/matthewmccullough/git-workshop`。

其次，GitHub 插件提供了每个文件的更改链接，可以在一个作业的 *Source Code Management* 配置里的 *Repository browser* 部分找到（见图 5-21）。

97



图5-21　GitHub仓库浏览器

选择 `githubweb` 仓库浏览器，所有更改 - 检测到的文件将链接到 GitHub 相应的源代码页面（图 5-22）。

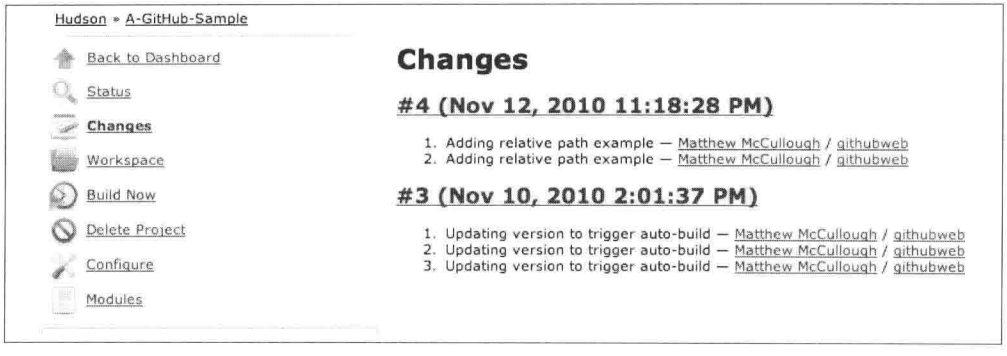


图5-22　GitHub仓库浏览器

Build Triggers

一旦配置了版本控制系统，你需要告诉 Jenkins 什么时候触发构建。可在 Build Triggers 部分进行设置。

在一个自由式的构建中，有三种基本方式可以触发构建作业（见图 5-23）：

- 一旦另一个构建作业完成之后，触发一个构建作业
- 定期开始构建
- SCM 轮询更改

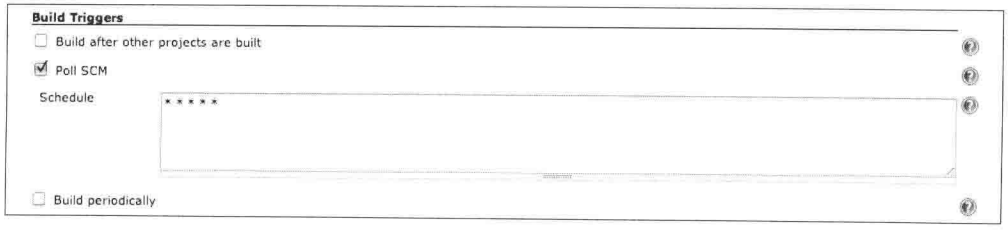


图5-23　有多种方式配置Jenkins来启动构建作业

构建作业完成后触发下一个

第一个选项允许你设置只有另一个构建完毕之后，才会开始一个构建。这是设置一个构建管道的简单方法。例如，你可以设置初始的构建作业来运行单元和集成测试，紧接着另一个单独的构建作业来运行更多 CPU 密集型代码质量度量。你只需要在此字段中输入前面构建作业的名称。如果该构建作业能够被其他的构建作业触发，则只要以逗号分隔，在这里列出其名字就可以了。在这种情况下，一旦列表中的构建作业完成就能触发构建作业。

在前一个构建作业的 Post-build 行为部分有个对应的字段，叫作“Build other projects”。当你修改“Build after other projects are built”字段时，该字段会在相应的构建作业中被自动更新。然而，不像“Build after other projects are built”字段，即使构建不稳定，这个字段也会让你选择触发构建(见图 5-24)。这是非常有用的，例如，在默认的构建作业中，即使有单元测试失败，你也想要运行代码质量度量构建作业。

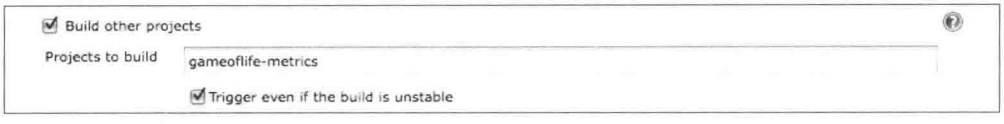


图5-24　即使目前的构建不稳定也会触发其他的构建作业

计划构建作业

另一种策略是在固定的时间间隔触发构建作业。值得注意的是，这不是真正的持续集成——它只是计划的构建，例如 UNIX cron 作业。在自动化构建的早期，甚至现今的很多情形下，构建并不是响应提交到版本控制的更改，而仅仅是每晚运行。然而，要更加有效，持续集成服务器应该比一天一次更快地提供反馈。

然而，有些情况下，计划的构建确实有意义。这包括构建作业运行的时间非常长，快速反馈不是那么重要。例如，强度综合负载和性能测试可能持续几个小时，或是 Sonar 构建作业。Sonar 是跨项目和随着时间的推移，密切关注代码质量度量的极好方法，但是 Sonar 服务器每天只存储一组数据集，所以更加频繁地运行 Sonar 构建没什么意义。

对于所有的调度任务，Jenkins 使用 cron 风格的语法，包括五个由空格分隔的字段：

MINUTE HOUR DOM MONTH DOW

每个字段用下面的值：

MINUTE

小时内的分钟数（0~59）。

HOUR

一天的小时数（0~23）。

DOM

本月的天数（1~31）。

MONTH

月份（1~12）。

DOW

本周的一天（0~7），其中 0 和 7 是星期日。

还有一些快捷键：

- “*”代表一个字段所有可能的值。例如，“* * * * *”表示“每分钟一次”。
- 你可以使用“M-N”符号定义范围。例如，DOW 中的“1-5”表示“星期一到星期五”。
- 你可以使用 / 定义范围间隔时间。例如，分钟字段“*/5”表示“每 5 分钟”。
- 逗号分隔的列表表示有效值的列表。例如，分钟字段的“15，45”表示“在过去的每个小时的第 15 和 45 分钟运行”。
- 你也可以使用数值的快捷表示，比如“@yearly”、“@annually”、“@monthly”、“@weekly”、“@daily”、“@midnight”及“@hourly”。

通常，你只需在这个字段中写一行；但对于更复杂的调度配置，你可能需要多行。

轮询 SCM

正如我们所见，计划的构建作业通常并不是大多数 CI 构建作业的最佳策略。任何反馈的价值与你收到的反馈速度成正比，持续集成也不例外。这就是为什么轮询 SCM 通常是更好的选择。

轮询涉及定期询问版本控制服务器是否提交了任何改变。如果项目中的源代码进行了任何更改，Jenkins 会触发构建。轮询操作通常相对便宜，所以你可以频繁地轮询以确保提交更改后构建快速发生。轮询越频繁，启动的构建越快，得到的关于哪些变化导致了构建中断的反馈也会越准确。

在 Jenkins 中，SCM 轮询很容易配置，并使用我们前面讨论过的 cron 相同的语法。

100 SCM 轮询的诱惑是尽可能经常轮询一次（例如，使用“* * * * *”，或者每分钟一次）。

因为 Jenkins 只是查询版本控制系统，只有在源代码修改的时候才触发构建，这种方法对于小型项目往往是适用的。如果有大量的构建作业就有其局限性了，因为这会导致 SCM 服务器配置和网络查询饱和（即使许多是不必要的）。在这种情况下，最好是设定更加精确的方法，仅当 SCM 收到变化时才会触发 Jenkins 构建作业。我们在下面的“远程触发构建”一节讨论此选项。

在很多项目中，如果更新频繁地提交到版本控制系统，这可能会导致许多构建作业进行排队，反过来会导致反馈时间延长。通过降低轮询的频率，你可以在一定程度上减少构建队列，但代价是较不精确的反馈。

如果你正在使用 CVS，轮询并不是一个好的选择。当 CVS 检查项目中的更改时，它会一个接一个地检查文件，这是一个缓慢和烦琐的过程。最好的解决方案是迁移到现代的版本控制系统如 Git 或者 Subversion。第二个最佳的解决方案是在非常稀疏的间隔（比如每隔 30 分钟）使用轮询。

远程触发构建

轮询对于较小的项目是一种有效的策略，但是并不特别适合规模化——对于大量的构建作业，它浪费了网络资源，并且在所提交代码更改和构建作业开始之间总有小的延迟。一个更加精确的策略是任何时候提交更改时，让 SCM 系统触发 Jenkins 构建。

从远程启动 Jenkins 构建作业是相对容易的。你只需要调用以下形式的 URL：

http://SERVER/jenkins/job/PROJECTNAME/build

例如，如果我的 Jenkins 服务器运行在 *http://myserver:8080/jenkins*，我可以通过调用下面的 URL 使用像 `wget` 或者 `curl` 这样的工具：

```
$ wget http://myserver:8080/jenkins/job/gameoflife/build
```

技巧就是每当更改被提交时，让你的版本控制服务器做这个。如何做到这一点的细节对于每个版本控制系统是不同的。例如，在 Subversion，你需要写一个提交后的钩子脚本，来触发构建。你也可以写一个 Subversion 钩子脚本用来解析仓库 URL 以提取项目名字，并在相应的构建作业的 URL 完成 `wget` 操作：

```
JENKINS_SERVER=http://myserver:8080/jenkins
REPOS="$1"
PROJECT=<Regular Expression Processing Goes Here>❶
/usr/bin/wget $JENKINS_SERVER/job/${PROJECT}/build
```

❶ 这里使用正则表达式从 Subversion 仓库中获取你的项目名称。

然而，这种方法只能触发一个特定的构建，并且依赖于默认构建作业基于 Subversion 仓库名称的规则。Subversion 更加灵活的方法是直接使用 Jenkins Subversion API，如下所示：

```
JENKINS_SERVER=http://myserver:8080/jenkins
REPOS="$1"
REV="$2"
UUID='svnlook uuid $REPOS'
/usr/bin/wget \
  --header "Content-Type:text/plain;charset=UTF-8" \
  --post-data "'svnlook changed --revision $REV $REPOS'" \
  --output-document "-" \
  --timeout=2 \
  $JENKINS_SERVER/subversion/${UUID}/notifyCommit?rev=$REV
```

这会自动启动监视 Subversion 仓库的任意 Jenkins 构建作业。

如果你已激活了 Jenkins 安全规则，事情就变得有点复杂了。最简单的情况（任何用户可以做任何事情）下，你需要激活“Trigger builds remotely”选项（见图 5-25），并且提供一个特殊的字符串，可以在 URL 中使用：

http://SERVER/jenkins/job/PROJECTNAME/build?token=DOIT

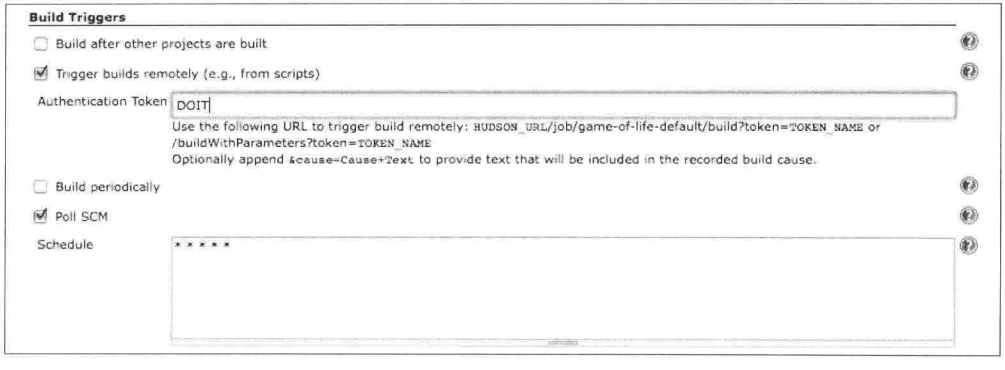


图5-25　使用URL的token触发构建

如果用户需要登录才能触发构建（例如，如果你正在用矩阵或基于项目的安全），这就不会起作用。在这种情况下，你需要提供用户名和密码，如下面的示例所示：

102

```
$ wget http://scott:tiger@myserver:8080/jenkins/job/gameoflife/build
```

或

```
$ curl -u scott:tiger http://scott:tiger@myserver:8080/jenkins/job/gameoflife/build
```

手工构建作业

构建并没有被自动触发。一些构建作业应该只能由人工干预开始。例如，你可能想设置自动化部署到用户验收测试环境中，这只能由 QA 人员请求触发。这种情况下，你可以简单地把 Build Triggers 部分留下空值。

构建步骤

现在 Jenkins 应该知道在何处以及多久获得项目源代码。你需要向 Jenkins 解释的下一件事是用源代码做什么。在自由式的构建中，你可以通过定义构建步骤来做这个。构建步骤是 Jenkins 自由构建过程的基本构建块。它们能让你准确地告知 Jenkins，你到底想要如何构建项目。

构建作业可能包括一个或者更多的步骤。它甚至偶尔什么步骤都没有。在自由式构建中，你可以在项目配置（见图 5-26）的 Build 部分添加许多你需要的步骤。在基本的 Jenkins 安装中，你将能够添加步骤来调用 Maven 和 Ant，运行特定操作系统的 Shell 或者 Windows 批处理命令。通过安装额外的插件，你还可以集成其他的构建工具，比如 Groovy、Gradle、Grails、Jython、MSBuild、Phing、Python、Rake 以及 Ruby，这里只提出了一些比较知名的工具。

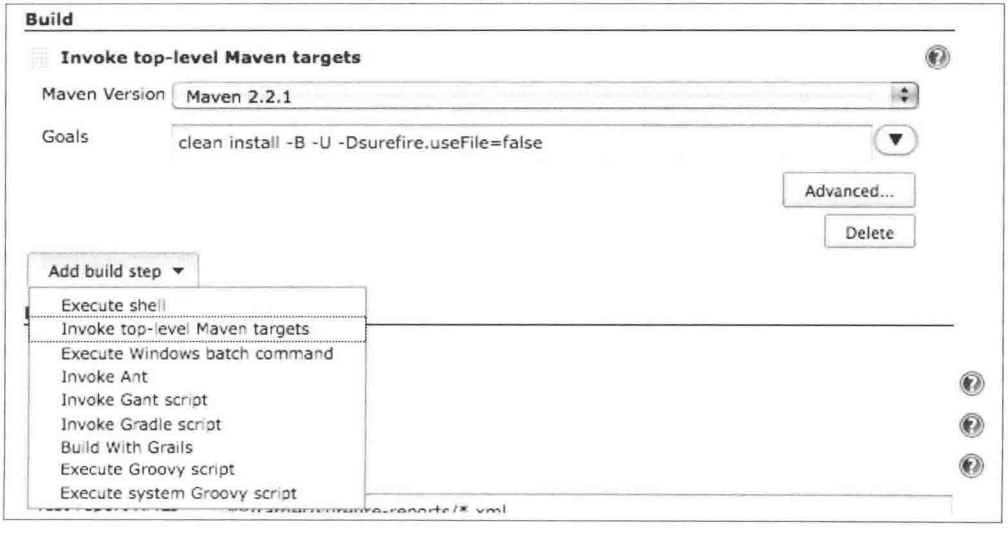


图5-26　在自由式构建的作业中添加构建步骤

在本节的其余部分，我们将深入探讨一些更常见类型的构建步骤。

Maven 构建步骤

Jenkins 对 Maven 支持得很好，并且 Maven 构建步骤很容易配置并且非常灵活。只要从构建步骤列表中选择“Invoke top-level Maven targets”，选择一个要运行的 Maven 版本（如果你已经安装了多个版本），然后输入你想运行的 Maven 目标即可。Jenkins 自由构建作业对于 Maven 2 和 Maven 3 都支持得很好。

就像在命令行一样，你可以设定尽可能多的目标。你还可以提供命令行选项。在 CI 上下文中有许多有用的 Maven 选项：

103

`-B, --batch-mode`

该选项告诉 Maven 如果需要任何输入，则不提示用户任何输入，只使用默认值。如果在 Jenkins 构建过程中，Maven 提示任何输入，那么构建将陷入无限期。

`-U, --update-snapshots`

强迫 Maven 检查依赖于远程仓库更新版本和快照。这将确保你在使用最新和最优的快照依赖关系进行构建，而不是使用可能没有与源代码的最新版本同步的旧的本地副本。

`-Dsurefire.useFile=false`

该选项强制 Maven 把 JUnit 输出到控制台，而不是到目标目录的文件文件。以这种方式，任何测试失败的详细信息都直接显示在构建作业控制台输出。Jenkins 测试报告需要的 XML 文件仍会生成。

高级选项，也值得调查（单击 Advanced 按钮）。

可选的 *POM* 字段允许你覆盖 Maven *pom.xml* 文件的默认值。这相当于从命令行使用 `-f` 或 `--file` 选项运行 Maven。对于一些多模块的 Maven 项目这是有用的，这些项目聚合的 *pom.xml* 文件（包括 `<modules>` 部分）位于子目录而不是顶层目录。

104

属性字段允许你设置被传递到 Maven 构建过程中的属性值，在这里使用标准属性文件格式说明：

```
# Selenium test configuration
selenium.host=testserver.acme.com
selenium.port=8080
selenium.broswer=firefox
```

这些属性作为传递给 Maven 的命令行选项，如下所示：

```
$ mvn verify -Dselenium.host=testserver.acme.com ...
```

JVM 选项（Options）字段允许你为构建作业设置任何标准的 Java Virtual Machine 选项。所以如果你的构建过程是内存密集型的，则可以通过 `-Xmx` 选项添加一些额外的堆空间（例如，`-Xmx512m` 会设定最大的堆的大小是 512MB）。

最后一个选项允许你为此构建作业配置专用的 Maven 仓库。通常情况下，Maven 使用默认的 Maven 仓库（通常位于用户主目录的 *.m2/repository* 文件夹）。偶尔地，这可能导致构建作业彼此互相干扰，或者从一个构建到另一个构建使用不一致的快照版本。为确保你的构建在干净的实验室条件下运行，你可以激活此选项。你的构建作业将得到自己的专用仓库，保留为自己独占使用。不利的方面：构建作业首次运行，可能需要一些时间来下载所有 Maven 的构建产物，私有仓库会占用大量的空间。然而，它是保证你的构建在真正的隔离环境中运行的最佳方式。

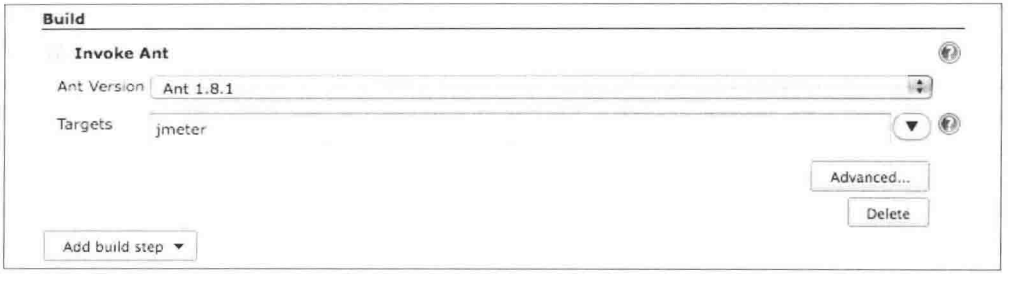
Ant 构建步骤

自由式构建作业同样支持 Ant。Apache Ant（*http://ant.apache.org/*）是一个广泛使用的和非常知名的 Java 构建脚本工具。事实上，大量的 Java 项目依靠 Ant 构建脚本。

Ant 不仅是主要的构建脚本工具——即使你的项目使用 Maven，你仍可能调用 Ant 脚本来做更具体的任务。Ant 库支持许多开发工具和低级的任务，比如使用 SSH 或者与专有的应用程序服务器一起工作。

就其最基本的形式而言，配置 Ant 构建步骤确实相当简单——你需要提供想要使用的 Ant 版本以及你想要调用的目标的名称。例如，在图 5-27 中，我们调用一个 Ant 脚本来运行 JMeter 测试脚本。

正像 Maven 构建步骤那样，Advanced 按钮给你提供了更多细节的选择，例如指定一个不同的构建脚本，或者在不同目录的构建脚本（默认是根目录的 *build.xml*）。你也可以像 Maven 那样指定 JVM 选项的属性。



105

图5-27　配置Ant构建步骤

执行 Shell 或 Windows 批处理命令

有时你可能需要直接在操作系统级别执行命令。一些旧式的构建过程依赖于特定的操作系统的脚本。在其他情况下，你可能需要执行一个低级别的操作，这是操作系统级别的命令最容易完成的。

你可以在 Jenkins `Execute Shell`（UNIX）或者 `Execute Windows Batch`（Windows）来执行系统级别的命令。作为一个例子，在图 5-28 中，我们已经添加了一个步骤来执行 UNIX 的 `ls` 命令。

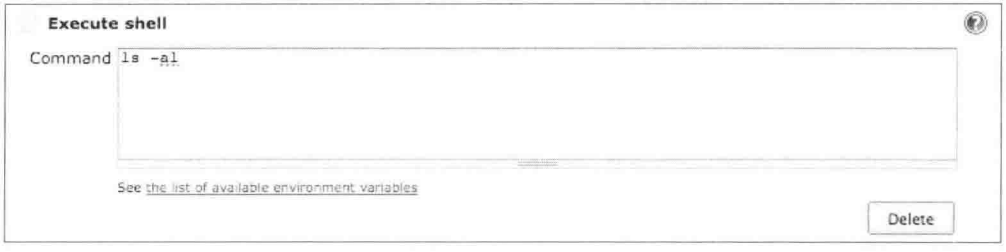


图5-28　配置可执行的Shell步骤

此构建步骤的输出如下所示：

```
[workspace] $ /bin/sh -xe /var/folders/.../jenkins2542160238803334344.s
+ ls -al
total 64
drwxr-xr-x   14 johnsmart staff   476 30 Oct 15:21 .
drwxr-xr-x    9 johnsmart staff   306 30 Oct 15:21 ..
-rw-r--r-- @  1 johnsmart staff   294 22 Sep 01:40 .checkstyle
-rw-r--r-- @  1 johnsmart staff   651 22 Sep 01:40 .classpath
-rw-r--r-- @  1 johnsmart staff   947 22 Sep 01:40 .project
drwxr-xr-x    5 johnsmart staff   170 22 Sep 01:40 .settings
-rw-r--r-- @  1 johnsmart staff   437 22 Sep 01:40 .springBeans
drwxr-xr-x    9 johnsmart staff   306 30 Oct 15:21 .svn
-rw-r--r-- @  1 johnsmart staff  1228 22 Sep 01:40 build.xml
-rw-r--r-- @  1 johnsmart staff   l50 22 Sep 01:40 infinitest.filters
-rw-r--r--    1 johnsmart staff  6112 30 Oct 15:21 pom.xml
106 drwxr-xr-x    5 johnsmart staff   170 22 Sep 01:40 src
drwxr-xr-x    3 johnsmart staff   102 22 Sep 01:40 target
drwxr-xr-x    5 johnsmart staff   170 22 Sep 01:40 tools
```

你可以执行特定的操作系统的命令（如 `ls`），或者在将一个更加复杂的脚本作为文件存储在你的版本控制系统，并执行该脚本。如果你正在执行脚本，则只需要填写相对于工作目录的脚本名称。

Shell 脚本使用 -ex 选项执行——命令会被打印到控制台，正如结果的输出那样。如果任何执行的命令返回非零值，构建将会失败。

当 Jenkins 执行一个脚本时，它会设置脚本里面用到的环境变量。在下一节中我们会更加详细地讨论这些变量。

事实上，有很多好的理由可以解释为什么你应该在你的构建作业中尽量避免使用系统级别的脚本，如果你能尽量避免的话。它使你的构建作业在特定的操作系统上能处于最好的情况下，并且最大程度减少了对精确的机器配置的依赖。一个更加方便的替代方式是用更加便携的脚本语言例如 Groovy 或者 Gant 执行操作系统脚本。

在构建中使用 Jenkins 环境变量

可以用在几乎任何构建步骤的一个有用的技巧，是从 Jenkins 中可以获得目前构建作业信息。事实上，当 Jenkins 开始构建步骤时，构建脚本可以使用下面的环境变量。

`BUILD_NUMBER`

当前的构建号，比如“153”。

`BUILD_ID`

以当前时间戳生成格式 YYYY-MM-DD_hh-mm-ss 的构建。

`JOB_NAME`

作业的名字，比如 *game-of-life*。

`BUILD_TAG`

方便地确定当前构建作业，为 `jenkins-${JOB_NAME}-${BUILD_NUMBER}` 形式（例如，`jenkins-game-of-life-2010-10-30_23-59-59`）。

`EXECUTOR_NUMBER`

标识在同一个机器上不同执行者中运行该构建的执行者。这是你在“build executor status”看到的数字，但该数字从 0 而不是从 1 开始。

`NODE_NAME`

如果构建运行在 slave 上，该节点的名字；如果构建运行在 master 上则是空值。

NODE_LABELS

运行该构建的节点标签列表。

107 JAVA_HOME

如果你的构建配置使用特定的 JDK，这个变量设置为指定 JDK 的 JAVA_HOME 。当设置此变量时，PATH 会被更新成包含 *$JAVA_HOME/bin*。

WORKSPACE

工作空间的绝对路径。

HUDSON_URL

Jenkins 服务器完整的 URL，例如 *http://ci.acme.com:8080/jenkins/*。

JOB_URL

此构建作业完整的 URL，例如 *http://ci.acme.com:8080/jenkins/game- of-life*。

BUILD_URL

此构建完整的 URL，例如 *http://ci.acme.com:8080/jenkins/game-of- life/20*。

SVN_REVISION

对于基于 Subversion 的项目，该变量包括当前的版本号码。

CVS_BRANCH

对于基于 CVS 的项目，此变量包含该模块的分支。如果配置 CVS 为检出主干，则将不会设置此环境变量。

很容易访问这些变量。在 Ant 脚本中，你可以使用 <property> 标签来访问它们，如下所示：

```
<target name="printinfo">
    <property environment="env" />
    <echo message="${env.BUILD_TAG}"/>
</target>
```

在 Maven 中，你可以以同样的方式（使用“env.”前缀）访问变量或者直接使用 Jenkins 环境变量。例如，在下面的 *pom.xml* 文件中，项目 URL 会指向运行 mvn site 的 Jenkins 构建作业：

```
<project...>
   ...
  <groupId>com.wakaleo.gameoflife</groupId>
  <artifactId>gameoflife-core</artifactId>
  <version>0.0.55-SNAPSHOT</version>
  <name>gameoflife-core</name>
  <url>${JOB_URL}</url>
```

或者，如果你在构建一个 Web 应用程序，也可以使用 *maven-war-plugin* 把构建作业编号插入 Web 应用程序清单，例如：

```
<project>
  ...
  <build>
    ...
    <plugins>
       <plugin>
         <artifactId>maven-war-plugin</artifactId>
         <configuration>
           <manifest>
            <addDefaultImplementationEntries>true</addDefaultImplementationEntries>
           </manifest>
           <archive>
            <manifestEntries>
              <Specification-Title>${project.name}</Specification-Title>
              <Specification-Version>${project.version}</Specification-Version>
              <Implementation-Version>${BUILD_TAG}</Implementation-Version>
            </manifestEntries>
          </archive>
        </configuration>
       </plugin>
      ...
    </plugins>
  </build>
  ...
</project>
```

108

这会产生如下内容的 *MANIFEST.MF* 文件：

```
Manifest-Version: 1.0
Archiver-Version: Plexus Archiver
Created-By: Apache Maven
Built-By: johnsmart
Build-Jdk: 1.6.0_22
Jenkins-Build-Number: 63
Jenkins-Project: game-of-life
```

```
Jenkins-Version: 1.382
Implementation-Version: jenkins-game-of-life-63
Specification-Title: gameoflife-web
Specification-Version: 0.0.55-SNAPSHOT
```

在 Groovy 脚本中，可以通过 `System.getenv()` 方法获得：

```
def env = System.getenv()
env.each {
   println it
}
```

或

```
def env = System.getenv()
println env['BUILD_NUMBER']
```

运行 Groovy 脚本

Groovy 不仅仅是一种流行的 JVM 动态语言，它也是一个方便的低级脚本语言。Jenkins Groovy 插件（*http://wiki.jenkins-ci.org//display/HUDSON/Groovy+Plugin*）允许你运行任意的 Groovy 的命令，或者作为构建过程的一部分调用 Groovy 脚本。

109 一旦你以通常的方式安装了 Groovy 插件，就需要在系统配置页面添加一个 Groovy 安装的引用（见图 5-29）。

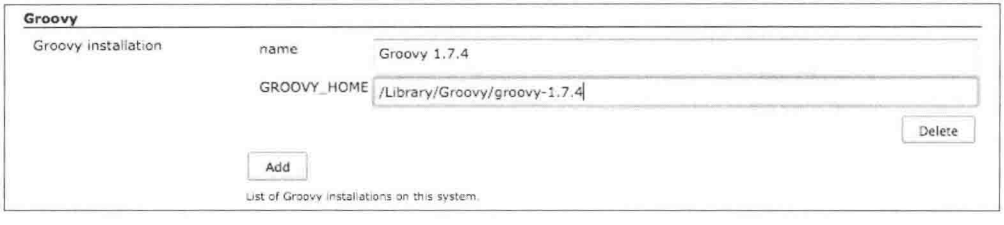


图5-29　把Groovy安装添加到Jenkins中

现在你可以为构建作业添加 Groovy 脚本了。当你单击“Add build step”时，在下拉菜单中可以看到两个新的条目：“Execute Groovy script”与“Execute system Groovy script”。通常你想用第一个选项——这将在一个单独的 JVM 中执行 Groovy 脚本，就好像从命令行里面调用 Groovy 一样。第二个选项会在 Jenkin 自己的 JVM 运行 Groovy 命令，并且能够完全访问 Jenkins 的内部，其主要用来操纵 Jenkins 构建作业或者构建过程本身。这是一个更高级的主题，我们会在本书后面讨论。

一个 Groovy 构建步骤可以采取两种形式之一。对于简单的情况，你可以添加一小段

Groovy，如图 5-30 所示。对于更复杂的情况，你可能会写 Groovy 脚本，并将其放在版本控制之下。一旦你的脚本安全地在 SCM 中，则可以通过选择“Groovy script file”选项并提供到你脚本的路径（相对于你的构建作业工作空间）运行它。

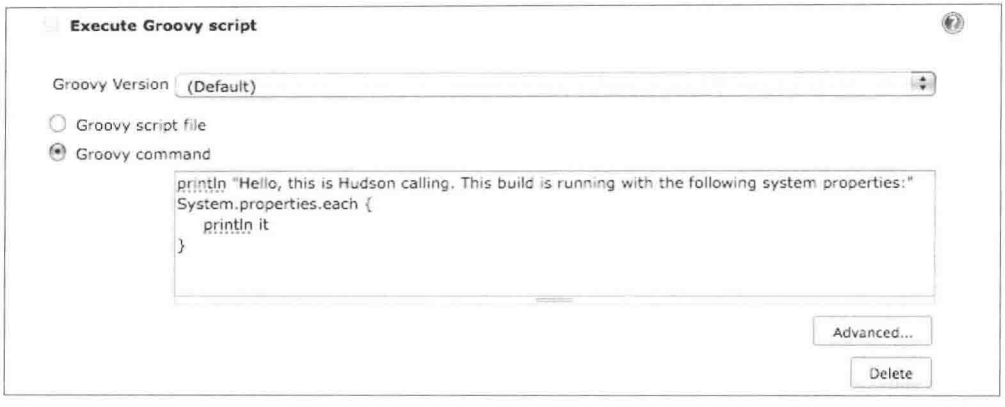


图 5-30　作为构建作业的一部分运行Groovy命令

在图 5-31 中，你可以看到一个稍微复杂的例子。在这里，我们正在运行 *run-fitness-tests.groovy* 脚本，在脚本目录中可以找到这个脚本。该脚本把要执行的测试套件作为参数——我们在 Script parameters 字段提供这些值。如果我们想为 Groovy 本身提供任何选项，则可以把这些值放在 Groovy parameters 字段。另外，我们可以在 Properties 字段提供命令行属性——这为通过使用 -D 命令行选项把属性值传递到 Groovy 脚本提供了更方便的方式。

110

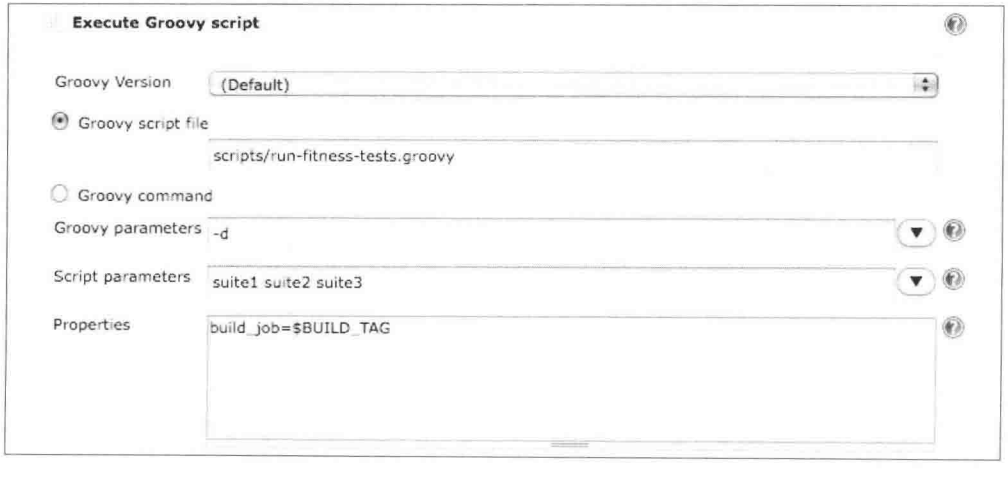


图 5-31　作为构建作业的一部分运行Groovy脚本

用其他语言构建项目

Jenkins 是一个灵活的工具，它不仅仅可以用于 Java 和 Groovy。例如，Jenkins 也适用于 Grails、.Net、Ruby、Python 和 PHP, 这里只列举了几例。使用其他语言时，你通常需要安装插件来支持你喜爱的语言，为这种语言增加新的构建步骤类型。我们会在本章的“用其他语言使用 Jenkins”一节看一些例子。

构建后的操作

一旦构建完成，你仍然需要做几件事情。你可能需要归档某些生成的构建产物，报告测试结果，并把结果通知给相关人员。在本节中，我们来看看一些构建完成后需要配置的较常见的作业。

报告测试结果

构建作业最明显的需求之一是报告测试结果。不仅需要报告是否有测试失败，也包括执行了多少测试、它们执行了多长时间，等等。在 Java 世界里，JUnit 是最广泛使用的测试库，JUnit XML 格式的测试结果被广泛地使用并且得到其他工具的很好支持。

111

Jenkins 为测试报告提供了很好的支持。在自由式构建作业中，你需要勾选“Publish JUnit test result report”选项，并且提供 JUnit 报告文件的路径（见图 5-32）。你可以使用通配符表达式（例如在 Maven 项目中使用 `**/target/surefire-reports/*.xml`）来包含来自不同目录的 JUnit 报告——Jenkins 将汇总结果到一份报告。

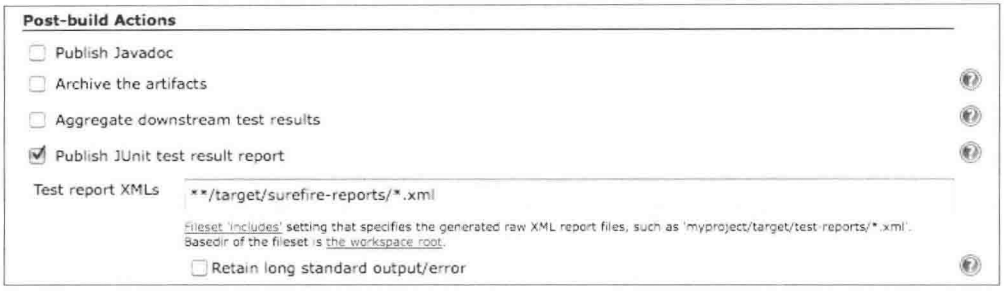


图 5-32　报告测试结果

在第 6 章我们会详细地探讨自动化测试。

归档构建产物

除了少数例外，构建作业的主要目标通常是进行构建并生成一些东西。在 Jenkins 中，

我们把生成的这些东西称为构建产物。构建产物可能是二进制可执行文件（例如 Java 项目的 JAR 或者 WAR 文件），或是其他一些相关可交付结果，例如文件或源代码。构建作业能够保存一个或者多个不同的构建产物；只保留最新的副本或者每次构建的构建产物。

配置 Jenkins 来存储你的构建产物很容易——只需在 Post-build Actions 勾选“Archive the artifacts”复选框，并且指定你想要保存的构建产物（见图 5-33）。

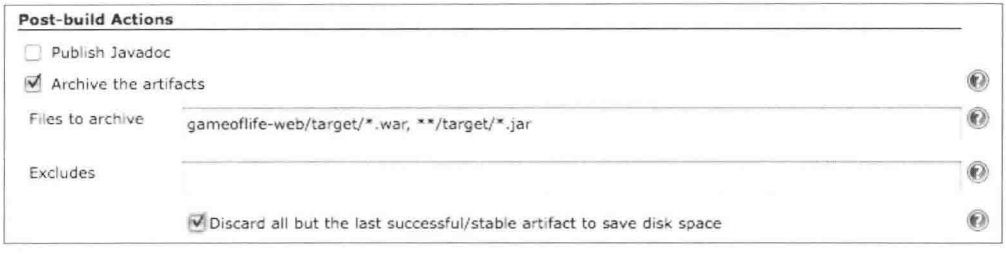


图5-33　配置构建产物

在“Files to archive”字段，可以提供要归档（相对于作业工作空间）文件的完整路径，或者使用类似 Ant 的通配符（如 `**/*.jar`，在工作空间任何地方的所有 JAR 文件）。使用通配符的优点之一是它使你的构建较少依赖于版本控制设置。例如，你在使用 Subversion（参见本章的“配置源代码管理”一节）时，Jenkins 把项目直接检出到工作空间或者到子目录中（取决于你如何设置）。如果你使用像 `**/target/*.war` 这样的通配符，无论项目位于任何目录，Jenkins 都会发现该目录下的文件。

112

像往常一样，Advanced（高级）按钮提供了一些额外的选项。如果你使用通配符来查找构建产物，则可能需要从搜索中排除某些目录。你可以通过填写 Excludes 字段做到这一点。可输入一个模式来匹配你不想要的归档文件，即使它们通常会包括在“Files to archive”字段。

归档的构建产物可能占用大量的磁盘空间（尤其是如果构建频繁的话）。为此，你可能希望只保留最后成功的一个版本。要做到这一点，只需勾选“Discard all but the last successful/stable artifact”选项。Jenkins 将保留最后一个稳定版本的构建产物（如果有的话）。它也会保留跟在稳定的构建产物之后最新的不稳定的构建产物（如果有的话）。

已经归档的构建产物出现在构建产物页面（见图 5-34）。最新的构建产物也会显示在构建作业主页。

你也可以使用永久的 URL 来访问最新的构建产物。这提供了一个很好的方式来从你的构建中重用最新的构建产物，不管是在其他 Jenkins 构建作业还是外部脚本中。有三个 URL 可用：最后的稳定构建、最后成功的构建和最后完成的构建。

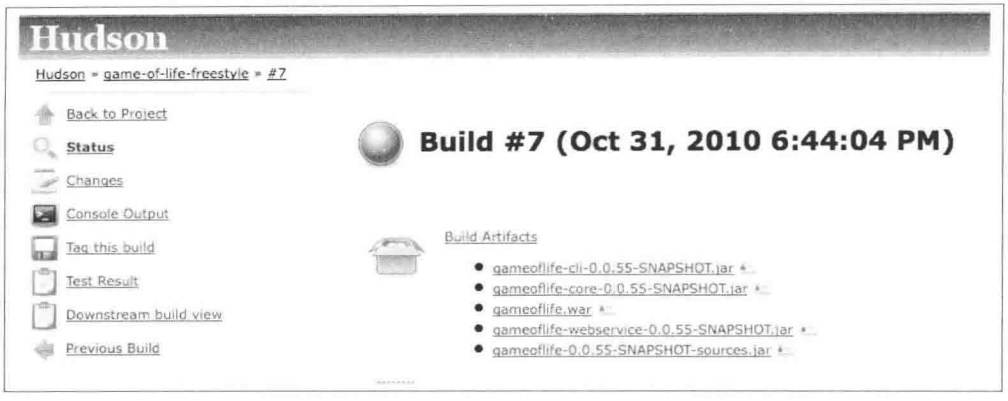


图 5-34　构建产物显示在构建产物页面和构建作业主页

在我们看 URL 之前，讨论*稳定*和*成功*的构建的概念。

当编译没有报告任何错误时构建就是*成功*的。

113 如果它构建成功并且没有发布者报告它是不稳定的，该构建就被认为是*稳定*的。例如，取决于你项目的配置、单元测试失败、不充分的代码覆盖率或其他代码质量度量问题，可能导致将构建标记为不稳定。所以一个稳定构建总是成功的，但是反过来不一定成立——构建可以成功但是不稳定。

已完成的构建只是构建完成而无论结果如何。请注意的是不论构建产物是什么，归档的步骤一定发生。

构建产物 URL 的格式非常直观，并且采用以下格式。

最新的稳定构建

<server-url>/job/<build-job>/lastStableBuild/artifact/<path-to-artifact>

最新的成功构建

<server-url>/job/<build-job>/lastSuccessfulBuild/artifact/<path-to-artifact>

最新完成的构建

<server-url>/job/<build-job>/lastCompletedBuild/artifact/<path-to-artifact>

通过一些实例最能说明问题。假定你的 Jenkins 服务器运行在 *http://myserver:8080*，你的构建作业被称为 *game-of-life*，你在存储名为 *gameoflife.war* 的文件，该文件位于你的工作空间的目标目录。这个构建产物的 URL 如下所示。

最新的稳定构建

http://myserver:8080/job/gameoflife/lastStableBuild/artifact/target/gameoflife.war

最新的成功构建

http://myserver:8080/job/gameoflife/lastSuccessfulBuild/artifact/target/gameoflife.war

最新完成的构建

http://myserver:8080/job/gameoflife/lastCompletedBuild/artifact/target/gameoflife.war

构建产物不一定是可执行的二进制文件。想象一下，假定你的构建过程包括将每个构建自动部署到测试服务器。为方便起见，你要保存与每个部署的 WAR 文件相关联的确切源代码的副本。一种方法是生成源代码相关的构建，并把该文件和 WAR 文件归档。我们可以生成包括应用程序源代码的 JAR 包文件（例如，在 Maven 项目中使用 Maven Source Plugin），然后把这个加入要存储的构建产物列表中（见图 5-35）。

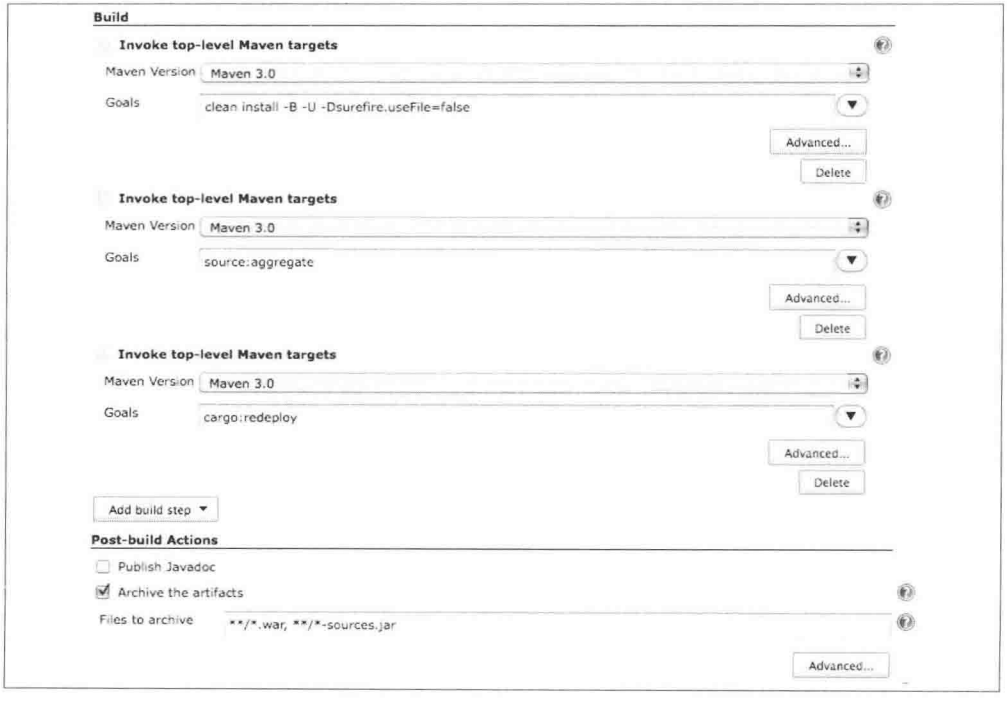


图5-35　把源代码和二进制包归档

当然，这个例子有点学术：只使用此构建的修订号（显示在构建产物页）来从你的版本控制系统检索源代码可能会更简单。但是你要学会这种思想。

请注意，如果你在使用企业仓库管理器（比如 Nexus 或者 Artifactory）来存储二进制构建产物时，你可能不需要在 Jenkins 服务器上保留它们。你可能希望只是作为构建的一部分，把你的构建产物自动化部署到企业仓库管理器并在需要的时候从这里检索它们。

通知

CI 服务器的意义是让人们知道什么时候构建中断。在 Jenkins 中，这会出现在通知标题的下面。

Jenkins 对电子邮件通知提供了开箱即用的支持。你可以在 Post-build Actions（见图 5-36）勾选“E-mail Notification”复选框。然后输入需要知道构建中断的团队成员的电子邮件地址。当构建中断时，Jenkins 将发送一封友好的电子邮件以及指向中断了的构建的链接给列表中的用户。

115

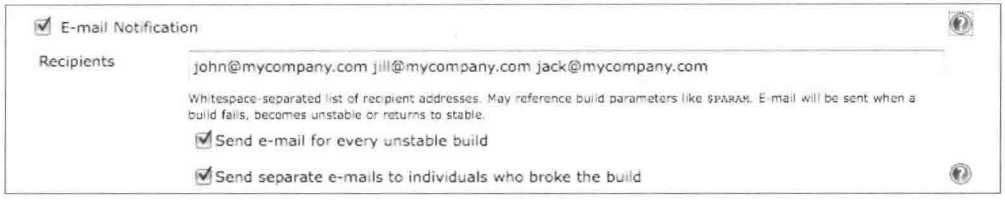


图5-36　电子邮件通知

你还可以选择发送单独的电子邮件给破坏构建的用户。要使这个生效，你需要在 Jenkins 服务器上激活了安全（见第 7 章）。

通常情况下，只要构建失败（例如因为编译错误），Jenkins 就会发送电子邮件通知。当构建第一次变得不稳定（例如，单元测试失败）时，它也会发出通知。除非你将其配置为这样做，否则 Jenkins 不会为每个不稳定的构建发送电子邮件，而仅会处理第一个不稳定的构建。

最后，以前失败或者不稳定构建一旦成功，Jenkins 会发送消息，让每个人都知道问题已得到解决。

构建其他的项目

你也可以使用“Build other projects”选项，在 Post-build Actions 中启动其他的构建作业。如果你想以几个更小的步骤而不是一个很长的构建作业组织你的构建过程，这非常有用。只需在上个作业之后列出你想开始的项目即可。通常情况下，这些项目只在构建稳定的情况下才会被触发，但你可以选择即使在目前的构建不稳定的情况下也触发另一个构建

作业。这可能是有用的，比如即使主要构建有一些测试失败，你仍想在项目主要构建作业之后运行代码质量度量报告构建作业。

运行新的构建作业

现在所有你要做的就是保存新的构建作业。然后你能手动触发第一个构建，或者等待它自己触发。一旦构建完成，你可以单击构建号查看自己的工作成果。

使用 Maven 构建作业

在本节中，我们将看看其他最常用的构建作业：Maven 2/3 构建作业。

Maven 构建作业专门适用于 Maven 2 和 Maven 3 构建。创建一个 Maven 构建作业比配置等效的自由构建作业需要少得多的工作。Maven 构建作业支持高级的 Maven 相关的功能，例如在多模块项目基础上的增量构建，从快照依赖关系的改变触发构建，以及更加容易地进行配置和生成报告。 116

然而，这里有一个问题：Maven 2/3 构建作业不如自由式构建作业灵活，并且不支持在同一个构建作业中的多个构建步骤。有些用户还报告说，配置为 Maven 而不是自由式构建作业时，大型的 Maven 项目倾向于运行速度更慢和使用更多的内存。

在本节中，我们将调查如何配置 Maven 2/3 构建，在什么情况下你可以使用它们，以及它们的优点和局限性。

要创建一个新的 Maven 构建作业，只要在新的作业页面（见图 5-37）选择“Build a maven2/3 project”。

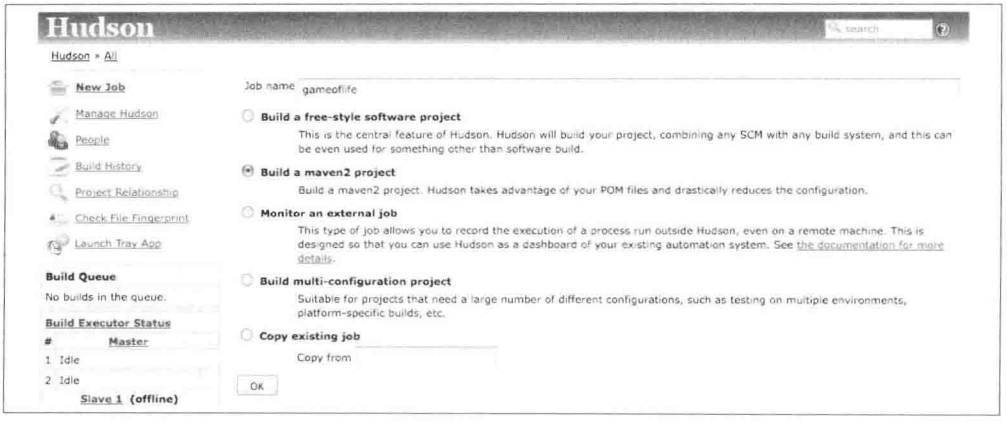


图 5-37　创建一个新的Maven构建作业

每当 SNAPSHOT（快照）依赖被建立时都要进行构建

乍一看，Maven 2/3 构建作业配置屏幕和我们在上一节看到的自由式构建非常相似。你可能会注意到的第一个区别是 Build Triggers 部分。在本节中，一个额外的选项是“Build whenever a SNAPSHOT dependency is built”。如果你选择此选项，Jenkins 会检查你的 *pom.xml* 文件以查看是否有任何快照依赖正在被其他的构建作业构建。如果有任何其他构建作业更新你的项目使用的快照依赖，Jenkins 也会构建你的项目。

通常在 Maven 中，快照依赖用于与同一团队中的其他项目共享最新版本的库。由于它们
117 被定义为不稳定，所以不建议依赖来自其他团队或者外部资源的快照做法。

例如，假定你正在开发一个新的 *game-of-life* Web 应用程序。该项目正在使用 Maven，所以你可以在 Jenkins 中使用 Maven 构建作业。你的团队也在做一个被称为可重用的 *cooltools* 的库。由于这两个项目都是由同一个团队开发的，你在 *game-of-life* Web 应用程序中使用一些最新的 *cooltools* 功能；所以你在 *game-of-life* *pom.xml* 文件 <dependencies> 部分有快照依赖：

```
<dependencies>
    <dependency>
        <groupId>com.acme.common</groupId>
        <artifactId>cooltools</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <scope>test</scope>
    </dependency>
    ...
</dependencies>
```

在 Jenkins 服务器上，你已经为 *cooltools* 和 *game-of-life* 应用程序设置了 Maven 构建作业。由于你的 *game-of-life* 项目需要最新的 *cooltools* 快照版本，勾选“Build whenever a SNAPSHOT dependency is built”选项。这样，只要重建 *cooltools* 项目，*game-of-life* 项目就将被自动重建。

配置 Maven 构建

你会发现变化的下一个区域在 Build 部分。在 Maven 构建作业中，构建部分完全用于运行单个 Maven 目标（见图 5-38）。在本节中，你指定要执行的 Maven 版本（记住，在这里仅仅 Maven 能够工作）、*pom.xml* 文件的位置、要调用 Maven 的目标。你还可以添加自己所需要的任何命令行选项。

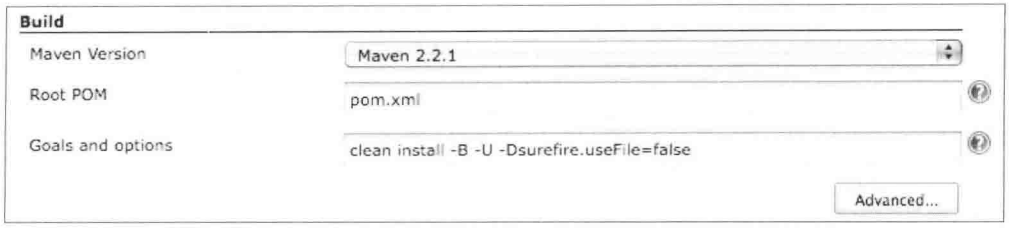


图5-38　指定Maven目标

在许多情况下，这是所有你需要配置的 Maven 构建作业。然而，如果你单击 Advanced 按钮，则可以挑选一些更高级的功能（见图 5-39）。

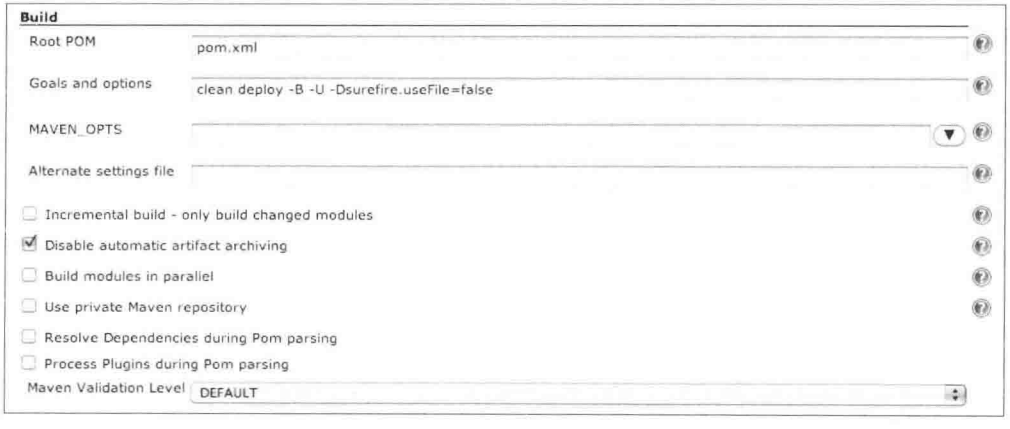


图5-39　Maven构建作业——高级选项

“Incremental build-only build changed modules”选项对于大型、多模块的 Maven 构建非常适用。如果你勾选此选项，则当对其中一个项目模块更改时，Jenkins 只重构那个模块以及使用更改的模块。它通过使用 Maven 2.1 中引入的一些新的 Maven 功能（如果你使用 Maven 2.0.x，则不被支持）执行该功能。Jenkins 检测哪些模块有了变化，然后使用 `-pl(--project-list)` 选项构建只更新的模块，以及 `-amd(--also-make-dependents)` 选项来构建使用更新模块的模块。如果在源代码中改变了什么，所有的模块都被构建。

默认情况下，Jenkins 把所有 Maven 构建作业产生的构建产物归档。有时候这可以派上用场，但是磁盘存储是非常昂贵的。如果想关闭此选项，则只需勾选“Disable automatic artifact archiving”选项。或者，你可以通过使用配置页面顶部的“Discard Old Builds”选项限制存储的构建产物。

“Build modules in parallel”选项告诉 Jenkins 把每个独立的模块作为单独的构建并行运行。

理论上，这可能会加速你的构建。在实践中，如果你的模块是完全独立的（也就是，你不使用聚合）才会起作用，但是这种情况是很少的。如果你认为并行构建模块确实能够真正加快你的多模块项目，则你可能想尝试使用 Maven 3 的自由构建及其新的并行构建功能。

另一个有用的选项是“Use [a] private Maven repository”。通常情况下，当 Jenkins 运行 Maven 时，它与命令行中运行 Maven 完全相同：它会保存构建产物并从本地 Maven 仓库检索构建产物（如果你没有在 *settings.xml* 文件中重新配置，则会在 *~/.m2/repository* 中找到它）。在磁盘空间方面这是有效的，但并不是理想的 CI 构建。事实上，如果几个构建作业正在使用同一个快照构建产物，这些构建最终可能互相干扰。

119 选中此选项后，Jenkins 将告诉 Maven 使用 *$WORKSPACE/.repository* 作为本地 Maven 仓库。这意味着每个作业都有自己独立的 Maven 仓库。它通过消耗额外的磁盘空间修复了上述问题。

选择此选项，Maven 为此构建作业使用位于 *$WORKSPACE/.repository* 目录专用的 Maven 仓库。这需要更多的磁盘空间，但是保证了构建作业之间更好的隔离。

解决这个问题的另一个方法是通过使用 `maven.repo.local` 属性来重写默认仓库位置，如下所示：

```
$ mvn install -Dmaven.repo.local=~/.m2/staging-repository
```

这种方法的优点是能够跨几个构建作业共享仓库（如果你需要做一系列相关的构建，这是有用的）。它还支持自由式作业。

构建后期操作

Maven 构建作业构建后期（post-build）的操作比自由式作业配置相对简单。这只是因为：由于这是一个 Maven 构建，Jenkins 知道到哪里去找到构建输出。构建产物、测试报告、Javadoc 等都产生在标准目录中，这意味着你不必告知 Jenkins 在哪里可以找到构建输出。因此 Jenkins 会自动发现并报告 JUnit 测试结果。在本书后面，我们将看到 Maven 项目如何简化许多代码质量度量工具和报告的配置。

大多数生成的构建后期操作和我们在自由构建作业中看到的那些类似。

部署到企业仓库管理器

在 Maven 构建作业中，一个额外的选项是能够将你的构建产物部署到 Maven 仓库（见图 5-40）。企业仓库管理器是一个作为公用的 Maven 构建产物 proxy/cache 的服务器，

也是你自己内部构建产物的中央存储服务器。开源的企业仓库管理器像 Nexus（来自 Sonatype）和 Artifactory（来自 JFrog）提供了强大的维护和管理功能，使配置和维护你的 Maven 仓库简单得多。这两种产品都有商业版本，附带针对更加复杂或高端构建基础架构的功能。

让 Jenkins 来部署你的构建产物（而不是简单地运行 `mvn deploy`）的优点是，如果你有多模块 Maven 构建，一旦整个构建已经成功完成，则构建产物只部署一次。例如，假定你有一个五个模块的多模块 Maven 项目。如果你运行 `mvn deploy`，三个模块之后构建失败，前两个模块已经部署到你的仓库，但不是最后三个，这让你的仓库处于不稳定状态。让 Jenkins 做部署能够确保构建成功完成之后构建产物才会部署。



120

图5-40　部署构建产物到Maven仓库

要做到这一点，只需勾选在“Post-build Actions”的“Deploy artifacts to Maven repository”选项。你需要指定自己想要部署的仓库的URL。这需要仓库的完整URL(例如：*http://nexus.acme.com/nexus/ content/repositories/snapshots*，而不是 *http://nexus.acme.com/nexus*)。

大部分仓库让你在部署构建产物前需要进行身份认证。

标准的 Maven 方式是在本地 *settings.xml* 文件设定 `<server>` 条目：

```
<settings...>
  <servers>
      <server>
        <id>nexus-snapshots</id>
         <username>scott</username>
         <password>tiger</password>
      </server>
      <server>
         <id>nexus-releases</id>
         <username>scott</username>
         <password>tiger</password>
      </server>
  </servers>
</settings>
```

出于更加安全的考虑，若是需要，你还可以加密这些密码。

然后，在 Jenkins 的 Repository ID 字段输入相应的 ID 值。Jenkins 之后能查找正确的用户名和密码，并部署你的构建产物。一旦构建完成，该构建产物在你的 Maven 企业仓库（见图 5-41）可用。

使用此选项，你不必总是立即部署——你可以回来继续部署上一次构建完之后的构建。只要单击左边的“Redeploy Artifacts”菜单，并指定你要部署的构建产物仓库的 URL（见图 5-42）即可。在前面的示例中，Advanced 按钮让你为本地 *settings.xml* 文件提供 <server> 项的 ID。正如本书后面所述，你可以使用此部署作为构建进阶过程中的一部分，当某些质量度量均能满足时，配置自动部署到不同的仓库中。

121

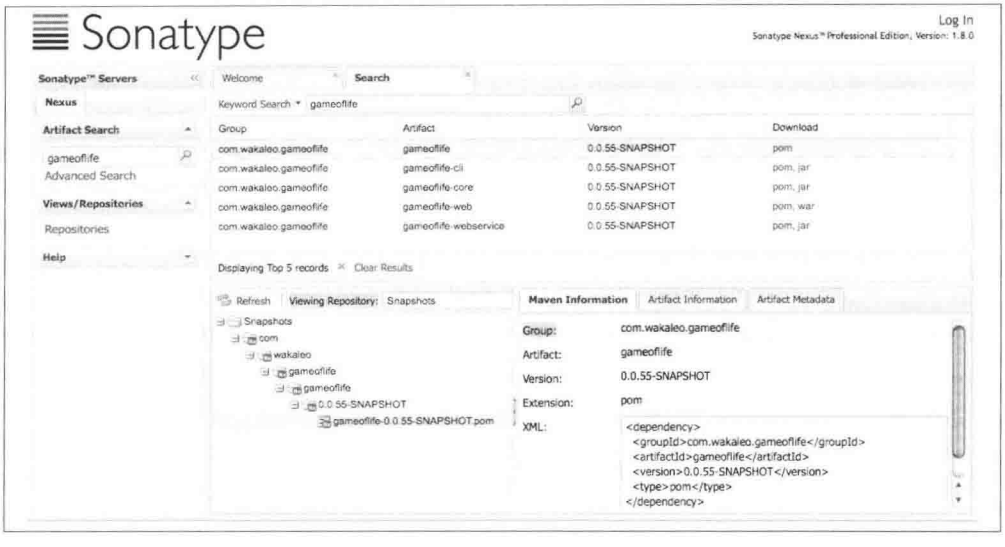


图5-41　部署完成后的构建产物在你的企业仓库管理器可用

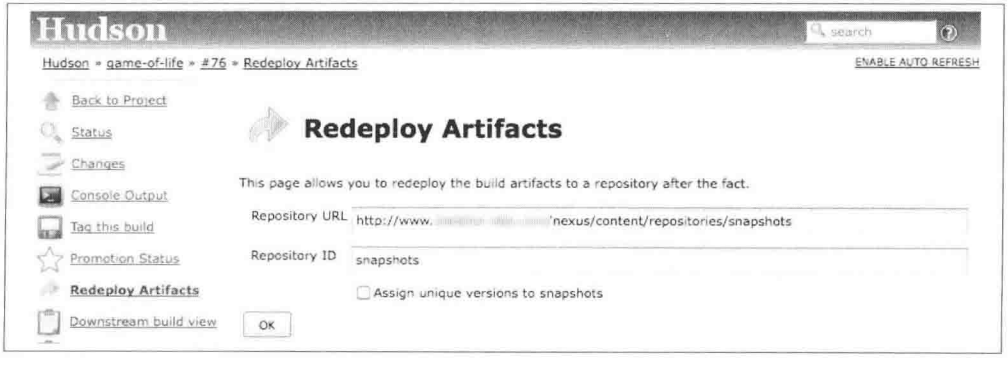


图5-42　重新部署构建产物

这种方法对于任何企业仓库管理器都是适用的。然而，如果你在使用 Artifactory，则可能更愿意安装 Jenkins Artifactory 插件（*http://wiki.jenkins-ci.org/display/JENKINS/Artifactory+Plugin*），它提供了与 Artifactory 企业仓库管理器的紧密双向集成。它的工作原理是在部署期间，发送附加的消息给 Artifactory 服务器，允许服务器回指生成给定构建产物的精确构建。一旦你安装了这个插件，就可以在你的 Maven 构建作业中 Post-build Actions 下勾选“Deploy artifacts to Artifactory”选项启用它。然后从服务器仓库列表中选择你的项目要部署到的仓库，以及进行部署所需的用户名和密码（见图 5-43）。

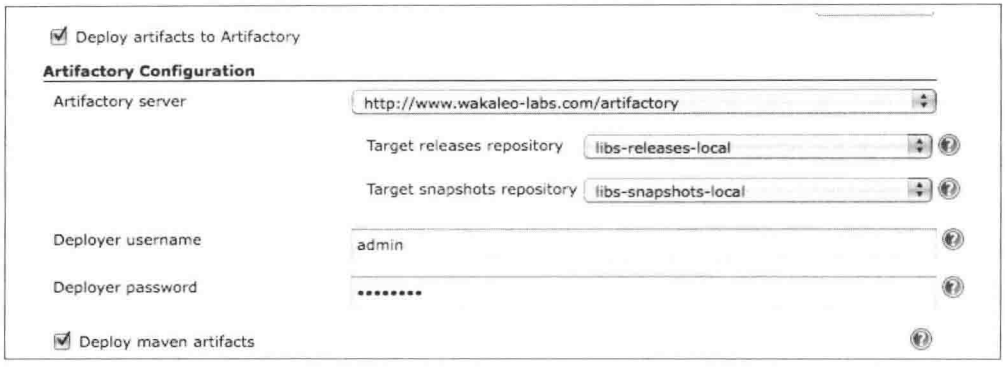


图5-43　从Jenkins部署到Artifactory

现在，你的构建作业将自动部署到 Artifactory。此外，服务器上构建产物的链接现在显示在构建作业主页和构建结果页面（见图 5-44）。



图5-44　Jenkins显示到相应的Artifactory仓库的链接。

此链接将你带到包含部署构建产物（见图 5-45）的 Artifactory 服务器页面。在此页面上，也有返回构建产物页面的链接。

123



图5-45　在Artifactory中查看已部署的构建产物

部署到商业企业仓库管理器

企业仓库管理器是任何基于 Maven 的软件开发基础架构的重要组成部分。对于依赖标准 Maven 仓库，使用像 Ivy 和 Gradle 工具的非 Maven 项目，它们也发挥了关键作用。

两个主要的企业仓库管理器，Nexus 和 Artifactory，提供的专业版具有与 Jenkins 集成的额外功能。在本书后面，我们将讨论如何使用 Nexus Pro 分期和发布管理的高级功能来实现复杂构建进阶策略。在部署方面，商业版的 Artifactory（Artifactory Pro Power Pack）延伸了我们前面看到的双向集成。当你在仓库浏览器查看构建产物时，“Builds”选项卡显示了 Jenkins 构建创建构建产物的细节，以及到 Jenkins 构建页面的链接（见图 5-46）。Artifactory 还跟踪 Jenkins 构建中使用的依赖；如果你尝试从仓库删除它们，则会发出警告。

管理模块

使用 Maven 时，把一个项目分成几个模块是很常见的。Maven 构建作业能够理解多模块项目，并增加了一个模块菜单项，可以让你一目了然地显示项目的结构（见图 5-47）。

单击任何一个模块都会把你带到模块的构建页面。在这里，你可以查看每个模块详细的构建结果；触发隔离模块的构建；如果必要的话，调整单个模块的配置，重写整个项目的配置。

Maven 构建作业中额外的构建步骤

默认情况下，Maven 构建作业只允许单一的 Maven 目标。有些时候，当你想在主构建之前或者之后添加一些额外的步骤时有一点局限性。你可以用 Jenkins M2 Extra Steps 插件。此插件让你在主要的 Maven 目标之前或者之后添加正常的构建步骤，给予你自由式构建的灵活性，同时仍然拥有方便的 Maven 构建作业配置。

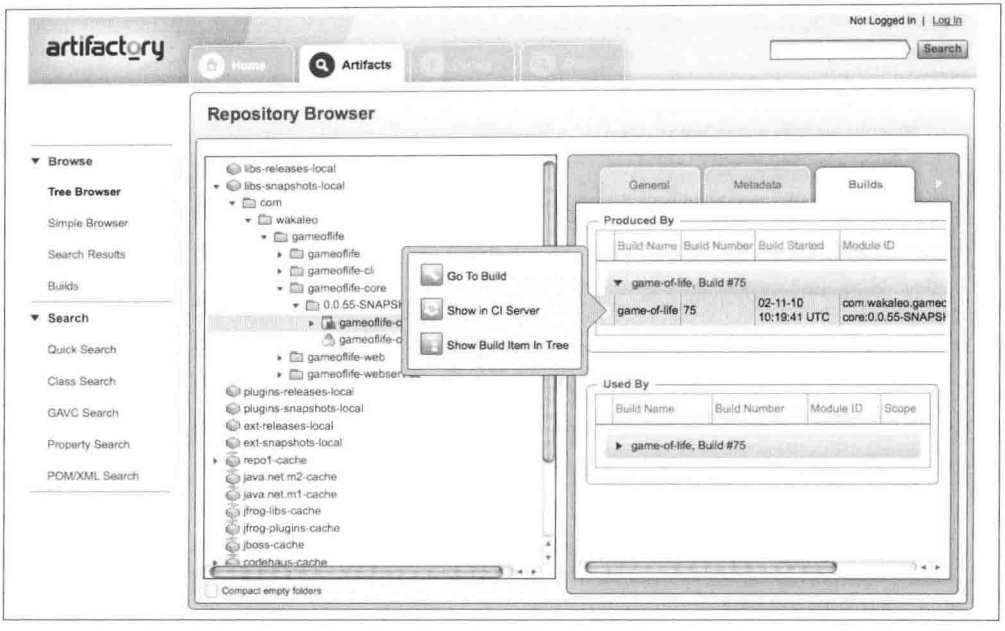


图5-46　在Artifactory查看部署的构建产物和相应的Jenkins构建

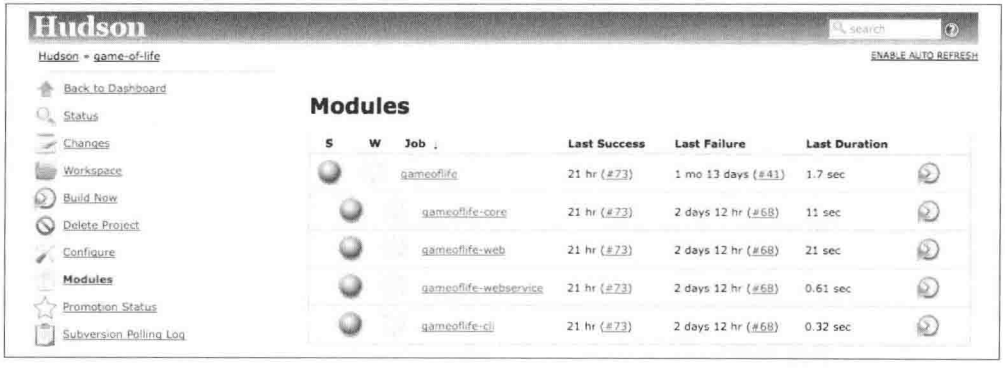


图5-47　在Maven构建作业中管理模块

安装这个插件，然后进入构建作业的 Build Environment 部分。勾选“Configure Extra M2 Build Steps”选项。你现在应该能够在主要 Maven 目标执行之前或者之后添加要执行的构建步骤了（见图 5-48）。

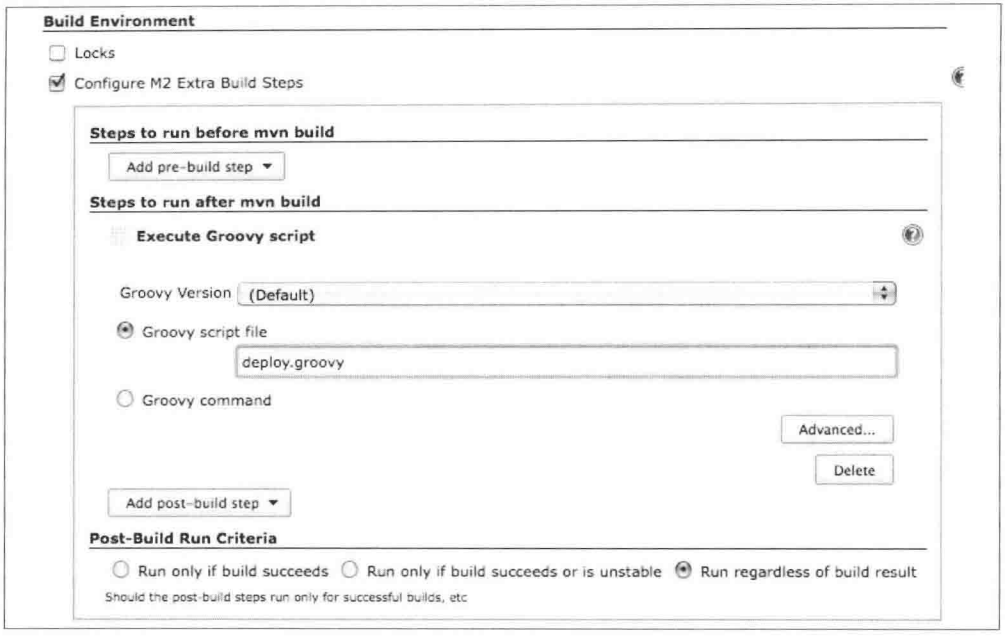


图5-48　配置额外的Maven构建步骤

用其他语言使用 Jenkins

正如我们前面提到的，Jenkins 为其他语言提供了出色的支持。在本节中，我们将看看如果用比较常见的其他语言使用 Jenkins。

用 Grails 构建项目

Grails 是建立在 Groovy 和许多知名的 Java 开源框架如 Spring 和 Hibernate 基础上的开源动态 Web 应用程序框架。

Jenkins 为 Grails 构建提供了出色的支持。首先，你需要安装 Jenkins Grails 插件（*http://wiki.jenkins-ci.org/display/HUDSON/Grails+Plugin*）。一旦你已经完成安装并且重启 Jenkins，就需要在配置系统屏幕（见图 5-49）的 Grails Builder 部分提供至少一个版本的 Grails 让 Jenkins 使用。

现在，你可以配置自由式构建作业来构建 Grails 项目。Grails 插件添加了“Build with Grails”构建步骤，你可以用来构建 Grails 应用程序（见图 5-50）。在这里，你要提供 Grails 目标或者你想要执行的目标。与命令行不同，你可以在同一个命令里执行几个目标。然而，如果你需要传递任何参数到一个特定的目标，则应该把目标及其参数用

双引号括起来。在图 5-50 中，我们运行 `grails clean`，其次运行 `grails test-app -unit -non-interactive`。为了能正确运行，我们用引号括起第二个命令的选项，即 `grails clean "test-app -unit -non-interactive"`。

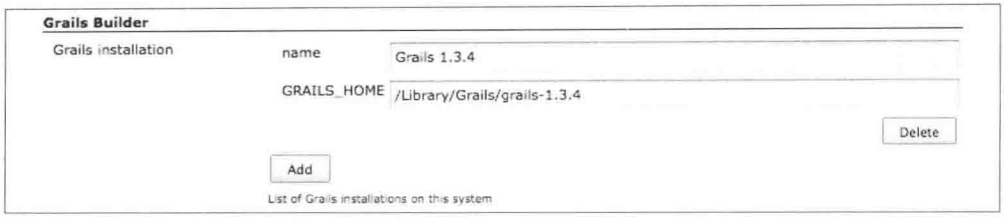


图5-49　在Jenkins中添加Grails安装

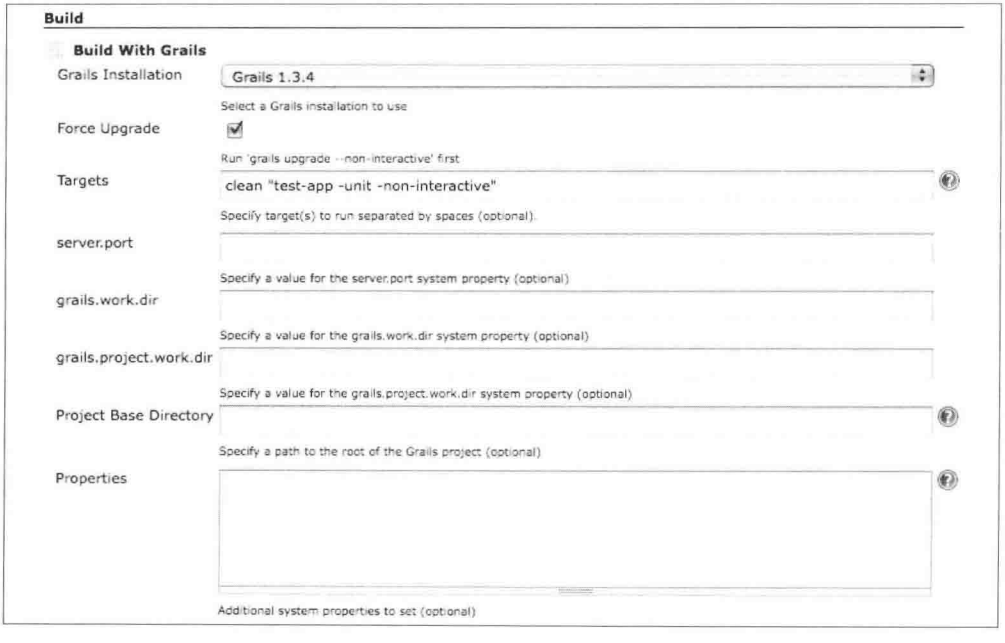


图5-50　配置Grails构建步骤

Grails 构建步骤需要多个可选参数。例如，Grails 对于版本比较苛刻——如果你的项目是由一个旧版本创建的，则 Grails 要求你升级。为了安全起见，你可能需要勾选 Force Upgrade 复选框，这可以确保在运行主要目标之前运行 `grails upgrade --non-interactive`。

你也可以指定服务器端口（如果正在执行 Web 测试，则用得到），以及任何你想要传递到构建的其他属性。

127 用 Gradle 构建项目

由 Rene Groeschke 撰写

相对于构建工具 Ant 和 Maven，Gradle（*http://gradle.org*）对于 Java Virtual Machine 而言是相对较新的开源构建工具。Gradle 构建脚本是基于 Groovy 用 Domain Specific Language（DSL）语言编写的。Gradle 实现约定优于配置，允许直接访问 Ant 任务，并使用类似 Maven 的声明依赖管理。Groovy 脚本自然简洁，允许你用很少的代码写出非常具有表现力的构建脚本（尽管不能像既定的工具 Ant 和 Maven 那样支持 IDE）。

Jenkins 运行 Gradle 构建有两种方式。你可以使用 Jenkins Gradle 插件或者 Gradle Wrapper 功能。

Jenkins Gradle 插件

你可以像通常的方式那样安装 Gradle 插件——到 Manage Plugins（管理插件）屏幕，选择 Jenkins Gradle 插件，单击安装并重新启动 Jenkins 实例。

Jenkins 重新启动后，你需要配置新的 Gradle 插件。现在你应该在配置系统屏幕找到新的 Gradle 部分。在这里，你需要添加想要使用的 Gradle 安装。该过程类似于其他工具的安装。首先，单击添加 Add Gradle 按钮来添加一个新的 Gradle 安装，然后输入适当的名称（见图 5-51）。如果在构建服务器上已经安装了 Gradle，则你可以指向本地 Gradle 主目录。或者，你可以使用“Install automatically”功能直接从 URL 下载 ZIP 或者 GZipped TAR 安装文件。你可以使用公共的 URL（*http://gradle.org/downloads.html*），或更喜欢使用本地服务器可用的安装包。

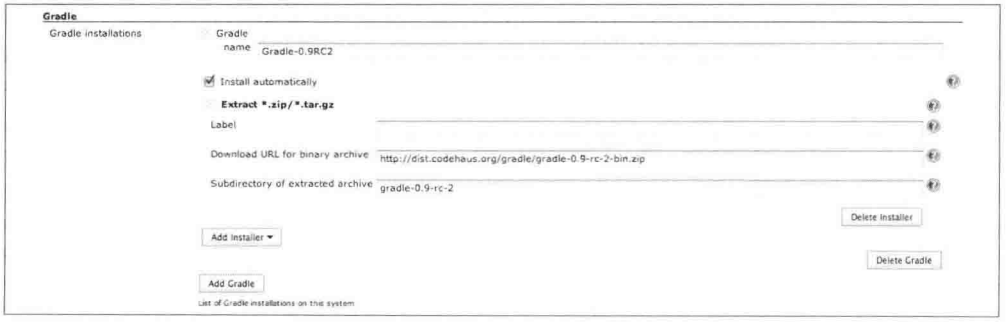


图5-51 配置Gradle插件

你通常使用自由构建作业来配置你的 Gradle 构建。当你将构建步骤添加到自由式构建时，你现在有个称为“Invoke Gradle script”的选项，它可以让你把 Gradle 具体设置添加到自己的构建作业中。

作为一个例子，这里有个非常简单的Gradle构建脚本。它是一个使用Maven目录结构和Maven仓库管理器的简单Java项目。有一个叫作*uploadArchives*的可定制的任务，可将生成的归档部署到本地企业仓库管理器：


```
apply plugin:'java'
apply plugin:'maven'

version='1.0-SNAPSHOT'
group = "org.acme"

repositories{
  mavenCentral()
     mavenRepo urls:'http://build.server/nexus/content/repositories/public'
}
dependencies{
   testCompile "junit:junit:4.8.2"
}
uploadArchives {
  repositories.mavenDeployer {
    configuration = configurations.archives
      repository(url: "http://build.server/nexus/content/repositories/snapshots")
{
        authentication(userName:"admin", password: "password")
    }
  }
 }
```

在图5-52中，我们使用刚刚配置的“Gradle-0.9RC2”实例来运行此Gradle构建。在这种情况下，我们希望运行JUnit测试并将生成的构建产物上传到我们的本地Maven仓库。此外，我们配置作业从***/build/test-results*收集测试结果，该目录是Gradle存储测试结果的默认目录。

增量构建

当用不变的源运行Gralde构建作业时，Gradle运行增量构建。如果Gradle任务的输出仍然可用，并且自从上次构建完后源码没有变化，则Gradle能够跳过任务的执行并且把任务标记为最新。增量构建的特性可以大大地减少正在运行构建作业的持续时间。

如果当Gradle评估认为测试任务已经是最新的，而你的单元测试执行却是被跳过了，这将会导致在Jenkins运行你的Gradle构建时出现问题。在我们前面的构建作业示例中，我们配置了一个构建后动作来发布我们构建的JUnit报告。如果测试任务被Gradle跳过，Jenkins作业将被标记为失败并出现以下消息：

129 Test reports were found but none of them are new. Did tests run?

你可以通过将下面的代码段添加到 Gradle 文件，使输出无效并且强制重新执行你的测试解决此问题：

```
test {
   outputs.upToDateWhen { false }
}
```



图5-52 设置Gradle构建作业

把上面的代码段添加到你的构建文件后，作业控制台输出应该看起来像图 5-53 这样。



图5-53 增量Gradle作业

正如你看到的那样，除了 *test* 和 *uploadArchives* 外的所有任务都被标识为最新的并且没有被执行。

用 Visual Studio MSBuild 构建项目

Jenkins 是一个 Java 应用程序，但是它也为 .NET 项目提供了出色的支持。

要在 Jenkins 中构建 .NET 项目，你需要安装 MSBuild 插件（*http://wiki.jenkins-ci.org/display/HUDSON/MSBuild+Plugin*）。

你也可能想要安装 MSTest 插件（*http://wiki.jenkins-ci.org//display/HUDSON/MSTest+Plugin*）和 NUnit 插件（*http://wiki.jenkins-ci.org//display/HUDSON/NUnit+Plugin*），以显示你的测试结果。

一旦你已经安装了 .NET 插件并重启 Jenkins，则需要配置 .NET 构建工具。转到配置系统页面并且指定 MSBuild 可执行文件的路径（见图 5-54）。

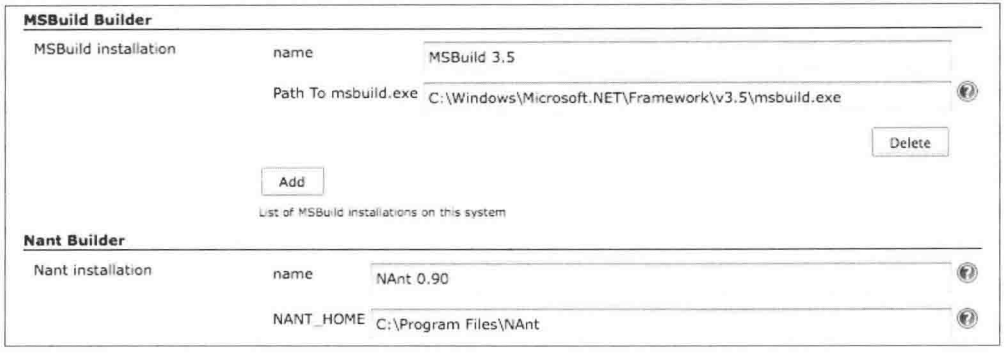


图5-54　在Jenkins中配置.NET构建工具

一旦有此设置，你就可以返回自由式项目并且添加 .NET 构建步骤的配置。

转到 Build 部分，在 Add Build Step 菜单中选择“Build a Visual project or solution using MSBuild”选项。然后输入 MSBuild 构建脚本的路径（*.proj* 或 *.sln* 文件），以及你的构建需要的任何命令行选项（见图 5-55）。

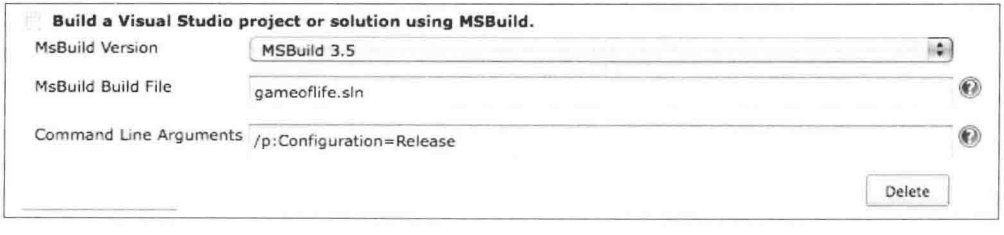


图5-55　使用MSBuild构建步骤

131 用 NAnt 构建项目

另一种构建你的 .NET 项目的方式是使用 NAnt。NAnt 是广泛应用于 Java 世界 Ant 构建脚本工具的 .NET 版本。NAnt 构建脚本是（通常以 *.build* 为扩展名）XML 文件，与 Ant 构建脚本使用的格式非常类似。

为了在 Jenkins 中用 NAnt 构建，你需要安装 Jenkins NAnt 插件（*http://wiki.jenkins-ci.org/display/HUDSON/NAnt+Plugin*）。一旦你已经安装了插件并重启了 Jenkins 服务，则转到配置系统页面并且在 Nant Builders 部分指定 NAnt 安装目录（见图 5-54）。

现在跳转到自由项目的 Build 部分并选择“Execute NAnt build”（见图 5-56）。你在此处指定构建脚本和你想要调用的目标。如果单击“Advanced”选项，你还可以设定传递给 NAnt 脚本的属性值。

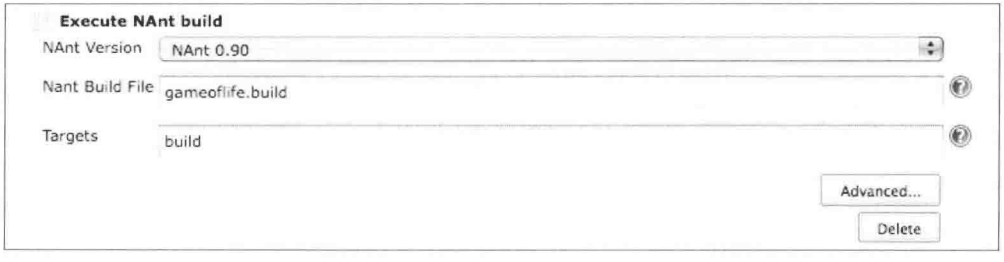


图5-56 使用NAnt的构建步骤

使用 Ruby 和 Ruby on Rails 构建项目

当谈到集成 CI 到 Ruby 以及 Ruby on Rails 项目时，Jenkins 可作为优秀的选择。Rake 插件可以让你向构建作业中添加 Rake 构建步骤。你还可以使用 Ruby 插件来在构建作业中直接运行 Ruby 脚本。最后，Ruby Metrics 插件提供了诸如 RCov、Rails stats 以及 Flog Ruby 代码质量度量工具。

在这个领域的另一个宝贵工具是 `CI:Reporter`。该库是 `Test::Unit`、`RSpec` 和 `Cucumber` 的附加组件，为你的测试生成 JUnit 兼容的 XML 报告。正如我们所看到的那样，Jenkins 能直接使用 JUnit-compatible 的测试结果来汇报你的测试结果。你需要使用 Gem 安装 CI:Reporter，正如此处所示：

```
$ sudo gem install ci_reporter
Successfully installed ci_reporter-1.6.4
1 gem installed
```

接下来，你需要在 Rakefile 中加入以下内容：

```
require 'rubygems'
 gem 'ci_reporter'
require 'ci/reporter/rake/test_unit' # use this if you're using Test::Unit
```

在第 9 章中，我们讨论整合代码质量度量到你的 Jenkins 构建中。Jenkins 还为 Ruby 代码覆盖率度量提供支持。Ruby Metrics 插件支持使用 *rcov* 诊断的代码覆盖率度量以及使用 *Rails stats* 的普通代码统计度量。要安装 *rcov* 插件，你应该首先需要运行下面的内容：

132

```
$ ./script/plugin install http://svn.codahale.com/rails_rcov
```

一旦配置完成，你可以在 Jenkins 显示测试结果和测试结果趋势。

最后，你可以使用 Rake 构建步骤配置 Rake 构建，如图 5-57 所示。

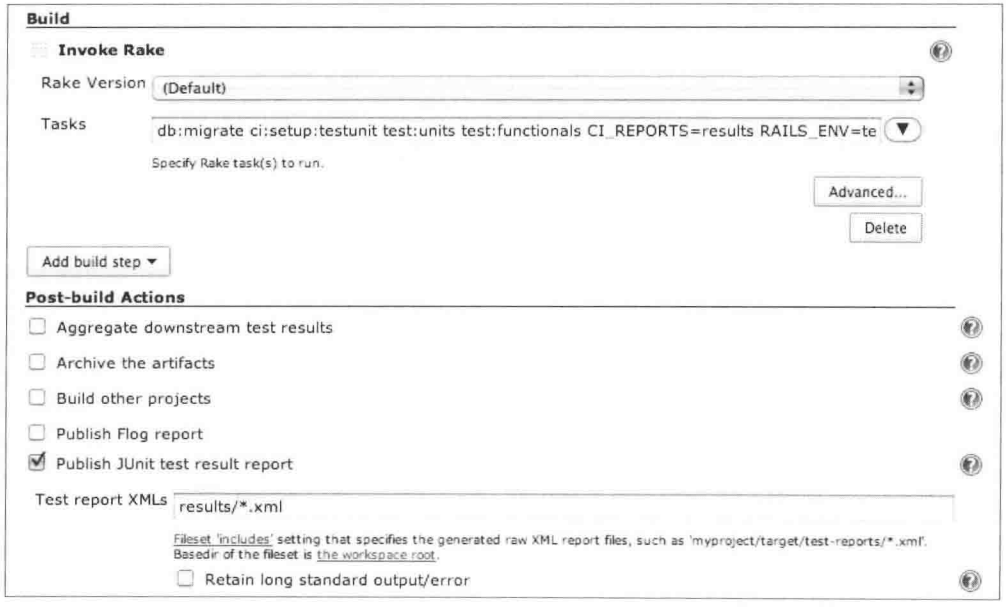


图5-57　使用Rake的构建步骤

你还需要配置 Jenkins 来报告测试和质量度量结果。这可以通过激活“Publish JUnit test result report”、“Publish Rails stats report”和“Public Rcov report”选项来进行（见图 5-58）。JUnit XML 报告可以在 *results* 目录中找到（在“Test report XMLs”字段输入 **`results/*.xml`**），以及在 *coverage/units* 目录找到 Rcov 日期。

133

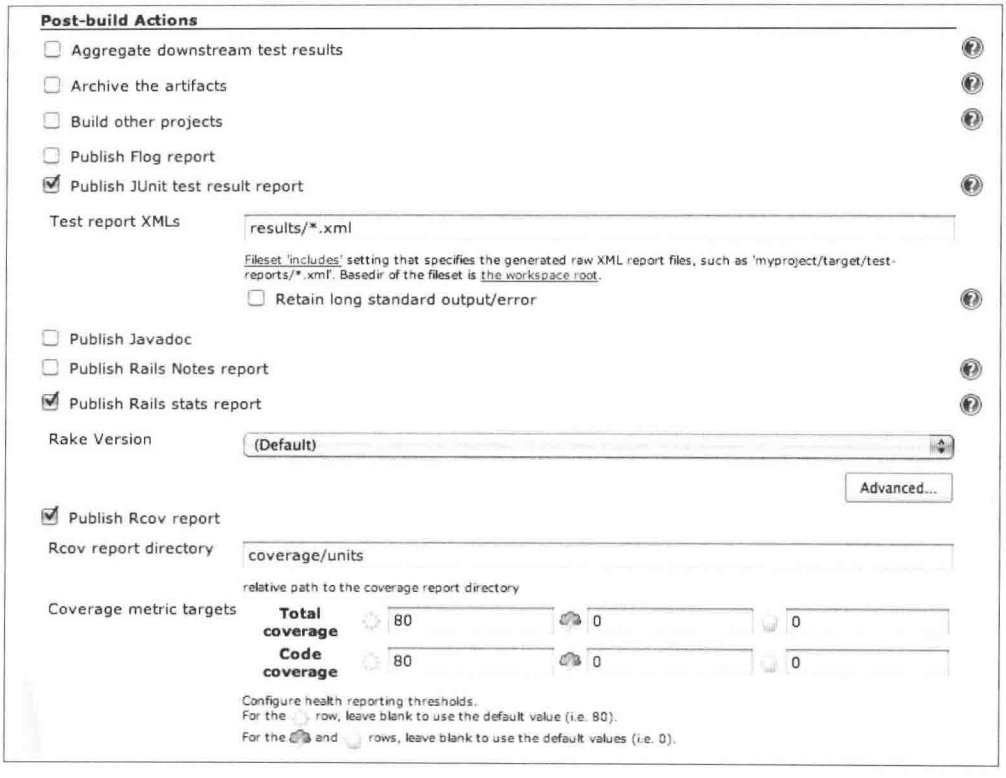


图5-58　为Ruby和Rails发布代码质量度量

小结

在本章中我们已经涉及了你很可能会遇到的最常见的创建新构建作业的基础知识。在本书后面，我们将在这些基础上讨论更多高级的选项，比如参数化构建、矩阵构建以及构建进阶策略。

第 6 章

自动化测试

简介

假如你没有在持续集成设置中使用自动化测试，那你是真的错失了很大的一块东西。相信我——没有自动化测试的持续集成真的只是自动化计划构建的小改进。现在别误会我的意思，假如你从一无所有起步，这就已经是向前的一大步了——但是你可以做得更好。简而言之，如果你使用的 Jenkins 没有任何自动化测试，你就没有获取到你本应从持续集成基础架构中得到的价值。

持续集成的基本原则之一就是构建应该能被验证。你必须能够客观地确定一个特定的构建是否已经准备好被推送到构建过程的下一个阶段。要做到这一点，最简便的方法就是使用自动化测试。如果不使用自动化测试的话，你会发现自己必须保留很多构建版本并且手动测试它们，这是违背持续集成的理念的。

把自动化测试集成到你的应用程序中有很多方法。要写出高质量的测试，最有效的方法之一就是先写测试，使用如测试驱动开发（TDD）或者行为驱动开发（BDD）的技术。这种方法常用在许多敏捷开发的项目中，单元测试的目的就是既能阐明你对代码行为的理解，又能写一个自动化测试去测试代码是不是确实完成了这个行为。专注于测试预期的行为，而不是实现，代码也有助于更全面和更准确的测试，从而帮助 Jenkins 提供更多相关的反馈。

当然，更典型的是只要代码完成了就做单元测试，这也是另一种常用的方法，总比根本没有测试好得多。

不过，Jenkins 并不局限于单元测试。有很多其他类型的自动化测试（包括集成测试、

Web 测试、功能测试、性能测试、负载测试等）是你需要考虑的，这取决于你的应用程序的性质。所有这些都在一个自动化构建设置中有其一席之地。

136 Jenkins 也可以和行为驱动开发或验收测试驱动开发这些技术结合使用，作为一个开发人员和其他项目相关人员的交流工具。BDD 框架如 easyb、fitnesse、jbehave、rspec、Cucumber 以及很多其他的东西，都试图能明确地呈现验收性测试以使得测试人员、产品所有者以及最终用户都能够明白。使用这些工具，Jenkins 能够使用业务术语来报告项目进展，以便于一个团队中开发人员和非开发人员的沟通。

对于那些已有的或者遗留下来的且配置了很少或者没有自动化测试的应用程序，为代码添加全面的单元测试是一件非常耗时且困难的事情。此外，测试可能不是很有效，因为它们往往是去验证已有的实现，而不是验证预期的业务行为。在这种情况下一种有用的方式是，编写一个自动化功能测试（“回归”）来测试一个模拟用户操作应用程序的常见方式。例如，自动化测试工具如 Selenium 和 WebDriver 可以有效地用于高水平地测试 Web 应用程序。虽然这种方式并不像集合了高质量的单元测试、集成测试以及验收测试那样全面，但它仍然是将自动化回归测试集成进一个已有应用程序中的一个非常有效且低成本的方式。

在本章中，我们会看看 Jenkins 是如何帮助你跟踪自动化测试结果以及你如何使用这些信息去监控和分析你的构建过程的。

自动化单元和集成测试

首先让我们来看看如何把你的单元测试集成到 Jenkins 中去。不管你是正在实践测试驱动开发模式还是在使用一个更传统的方法写单元测试，这些都可能是你想要集成到 Jenkins 里的第一个测试。

Jenkins 在报告你的测试结果方面做得非常出色。然而，这取决于你写了合适的测试用例并且配置了你的构建脚本去自动化地运行它们。幸运的是，集成单元测试到你的自动化构建中一般来说是一件相对容易的事情。

还有很多单元测试工具，它们都属于 xUnit 家族的一员。在 Java 世界中，JUnit 是事实上的标准，尽管 TestNG 是另一个流行且有大量创新功能的 Java 单元测试框架。对于 C# 应用程序，NUnit 测试框架能做类似于 JUnit 提供的一些功能，`Test::Unit` 对于 Ruby 也一样。对于 C/C++，有一个 CppUnit；PHP 开发人员可以使用 PHPUnit。而这不是一个完整的清单！

这些工具也可以为集成测试、功能测试和 Web 测试等服务。许多 Web 测试工具比如 Selenium、WebDriver 和 Watir，生成与 xUnit 兼容的报告。行为驱动开发模式和自动化

验收测试工具如 easyb、Fitnesse、Concordion 也和 xUnit 友好兼容。在下面的几节中我 137
们不区分这些不同类型的测试，正如从一个配置的角度来看，Jenkins 会以完全相同的方式来对待它们。可是，你几乎肯定需要在自己的构建作业中做出一些区分。为了得到最快的反馈循环，你的测试应该分组到一个定义好的类别中，在最后运行较慢的功能和 Web 测试之前，要先开始运行比较快的单元测试，然后进行集成测试。

关于更详细地讨论如何去自动化你的测试已经超出了本书的范畴，但是我们在附录 A 中仍涵盖了一些对 Maven 和 Ant 有用的技术。

在 Jenkins 中配置测试报告

一旦构建生成了测试结果，你就需要配置 Jenkins 构建作业去显示它们。就像上面提到的，Jenkins 适用于任何与 xUnit 兼容的测试报告，不管它们是使用何种编程语言编写的。

对于 Maven 构建作业，不需要特殊的配置——只需要确保你要调用的目标程序会运行你的测试，比如 `mvn test`（单元测试）或 `mvn verify`（单元测试和集成测试）。Maven 构建作业配置的一个例子如图 6-1 所示。

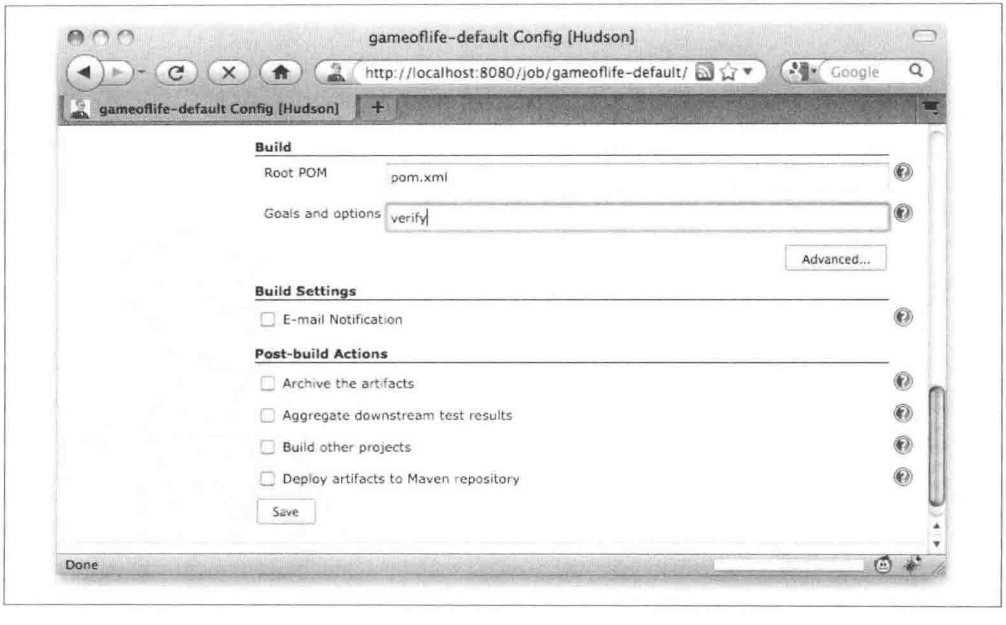


图 6-1　在Manage Jenkins界面配置Jenkins安装

对于自由风格的构建作业，你需要做更多的配置工作。除了需要确保你的构建确实运 138
行了测试用例，你还需要告诉 Jenkins 去发布 JUnit 的测试报告。你可以在“Post-build

Actions”部分配置这些（见图 6-2）。在这里，你需要为 JUnit 或者 TestNG XML 报告提供一个路径。确切的位置依赖于这个项目——对于 Maven 项目来说，大多数项目可在类似于 ***/target/surefire-reports/*.xml* 的路径找到它们。对于一个基于 Ant 的项目来说，它取决于你怎样配置 Ant JUnit 任务，就像我们上面讨论过的那样。

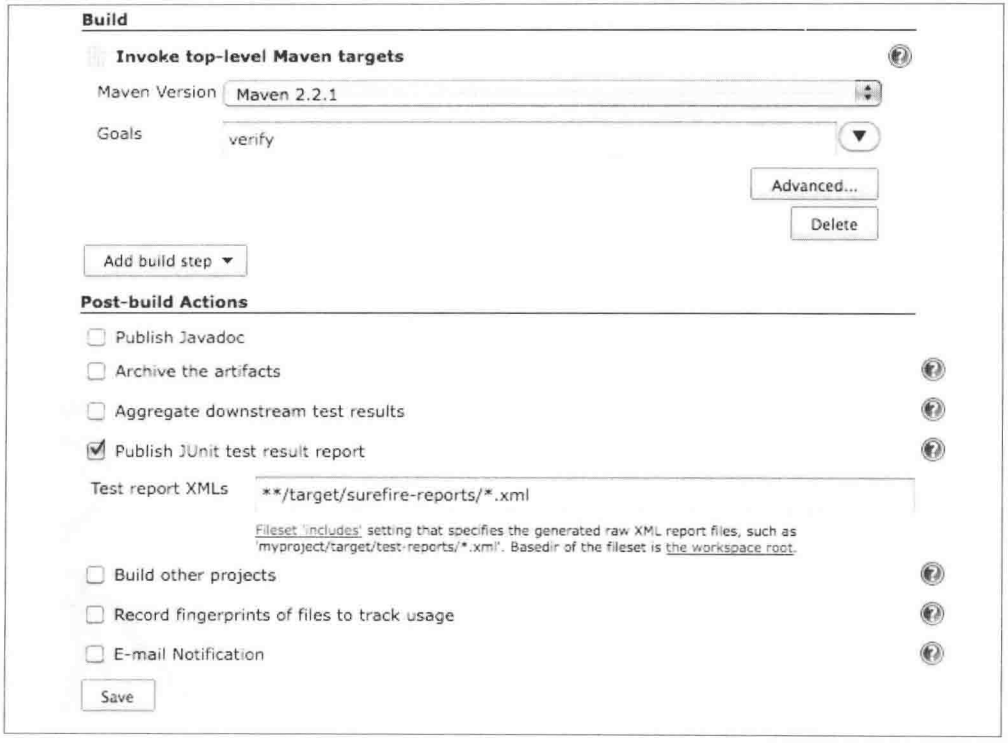


图 6-2　在一个自由风格的项目中配置Maven测试报告

对于 Java 项目来说，不管它们使用的是 JUnit 还是 TestNG，Jenkins 都能做出优异的工作。假如你在为非 Java 项目使用 Jenkins，你可能需要 xUnit 插件。这个插件可以让 Jenkins 以一种非常简便的方式处理来自非 Java 工具的测试报告。它为 MSUnit 和 NUnit（对于 C# 和其他 .NET 语言）、UnitTest++ 和 Boost Test（对于 C++）、PHPUnit（对于 PHP）及包括一些其他通过额外插件获得的 xUnit 库提供支持（见图 6-3）。

139

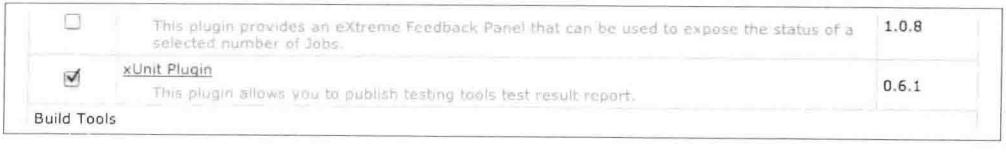


图 6-3　安装xUnit插件

一旦你安装了 xUnit 插件，就需要在“Post-build Actions”部分为你特定的 xUnit 报告做配置。检查“Publish testing tools result report”复选框，并输入测试库生成的 XML 报告的路径（见图 6-4）。当构建作业运行的时候，Jenkins 会把这些报告转化成 JUnit 报告，以便它们可以显示在 Jenkins 中。

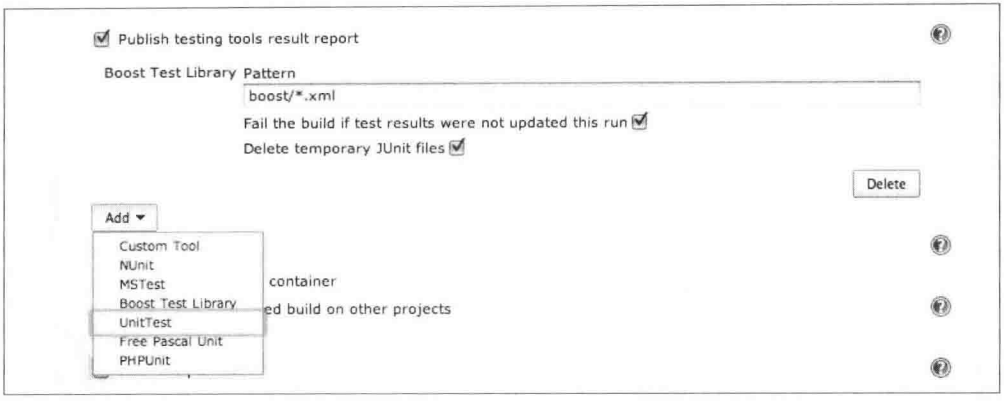


图6-4　发布xUnit测试结果

显示测试结果

只要 Jenkins 知晓了在哪儿能找到这些测试报告，它就能在报告这些结果上做得非常出色。事实上，Jenkins 的主要工作之一就是检测和报告构建中的失败。一个失败的单元测试就是众多明显的症状之一。

如前所述，Jenkins 区分失败的构建和不稳定的构建。一个失败的构建（红球表示）表明测试失败，或者说一个构建作业被一些行为残酷地破坏了，比如一个编译错误。一个不稳定的构建，从另一方面来讲，是一个质量不够好的构建。这样的说法有点含糊：如何定义“质量”，从某种意义上来讲这取决于你自己，但是通常它和代码质量度量比如代码覆盖率或者编码标准相关，稍后我们会在本书中讨论。现阶段，我们主要关注那些失败了的构建。

在图 6-5 中我们能看到 Jenkins 如何显示一个包含测试错误的 Maven 构建作业。这是构建作业的主页，是你构建失败后第一个要去的地方。当一个构建里有测试错误的时候，Latest Test Result 链接处会显示这个构建作业当前的测试错误号（在说明中有“5 个错误”），也包括从上一个构建起的测试错误数量的变化（在说明中的“+5”——5 个新的测试错误）。你也能够看到这些测试随着时间的推移是怎么变化的——之前的构建中的测试错误也会在测试结果趋势图中显示为红色。

140

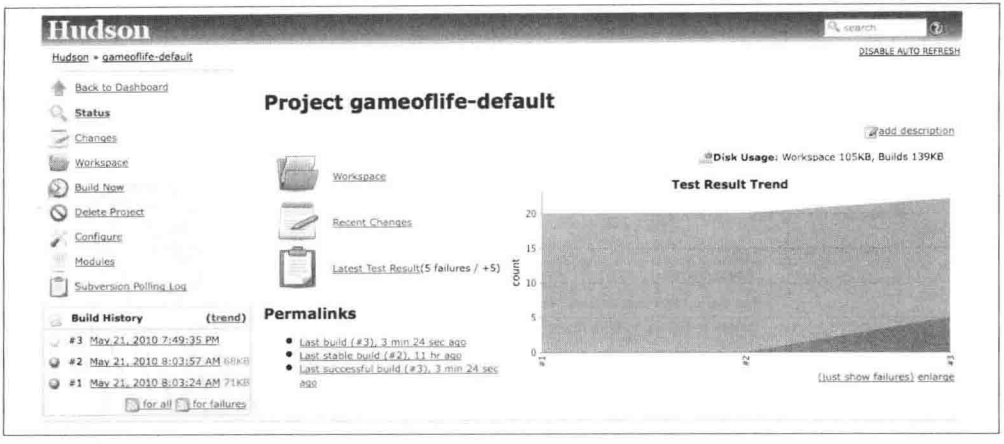


图 6-5　Jenkins在项目主页显示测试结果趋势

如果你单击了 Latest Test Result 链接的话，Jenkins 会向你提供当前测试结果的纲要（见图 6-6）。Jenkins 能够理解 Maven 的多模块项目结构，并且对于一个 Maven 构建作业，Jenkins 首先显示每个模块的测试结果的汇总视图。欲了解在特定模块中关于失败测试的更多细节，只需在你感兴趣的模块上单击即可。

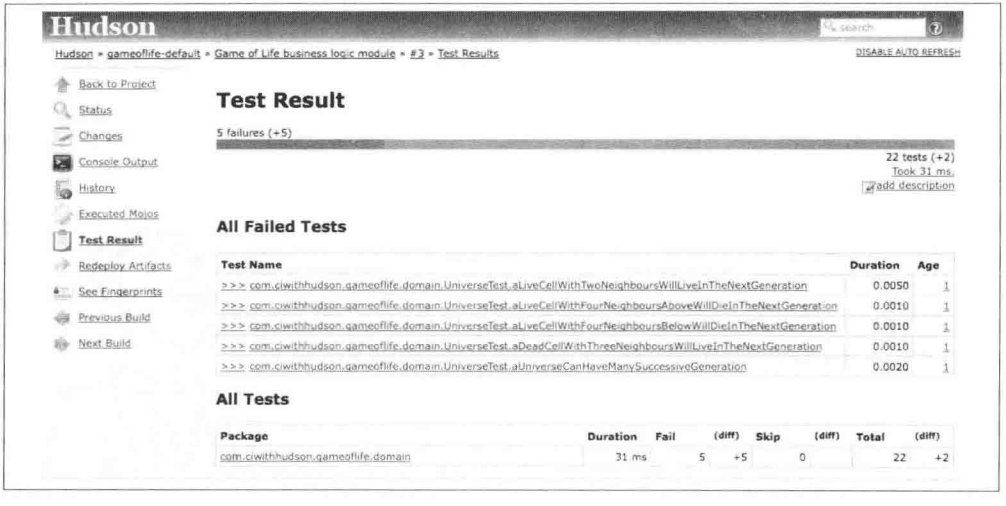


图 6-6　Jenkins显示测试结果的汇总

对于那些自由风格的构建作业，Jenkins 会直接给出测试结果的汇总，但是是通过高级的软件包而不是模块来组织的。

141　在两种情况下，Jenkins 通过为每一个包呈现测试结果的汇总开始。从这里，你可以继续

深入，看到每个测试类的测试结果，最后是测试类本身内部的测试。如果有任何失败了的测试，这些会突出地显示在页面的顶部。

这个全视图能够向你提供一个关于测试的当前状态的很好概述及其历史的显示。年龄（Age）列可以告诉你一个测试已经被破坏多长时间了，还有一个超链接可以带你到这个测试失败了的第一个构建。

你还可以使用屏幕右上角的 Edit Description 链接为测试结果添加描述。这是个为失败构建注释一些额外细节的好办法，以便添加一些关于这些失败测试的起源的额外信息或者一些关于怎么去修复它们的笔记。

当一个测试失败的时候，你一般会想知道为什么失败。想要看一个特定的测试错误的细节的话，只需在屏幕上单击相应的链接即可。这将会显示所有令人厌恶的细节，包括错误消息和堆栈跟踪，也包括测试失败了多久的一个提醒（见图 6-7）。你应该小心那些已经导致不止一两个构建失败的测试——这表明了一个可能需要进行调查的棘手的技术问题，或者一个对于失败构建的自满的态度（开发人员可能正在忽视这些构建错误），这是更严重的并且绝对应该进行调查。



图 6-7　一个测试错误的细节

确保你也对测试运行了多长时间保持关注，而不仅仅是关注它们是通过了还是失败了。单元测试应该被设计为运行快速，过于长时间运行的测试可以作为性能问题的一个标志。慢的单元测试也会拖延反馈的速度，在持续集成里，快速反馈才是最重要的。例如，能在五分钟内运行 1 000 个单元测试是非常好的——但花费掉 1 小时时间就不太好了。所以，最好是定期检查你的单元测试运行花费时间；如果有必要的话；要调查一下为什么它们会花费那么长的时间。

幸运的是，Jenkins 可以很容易地告诉你测试已经运行超时了多长时间。在构建作业的主页面上，单击屏幕左边的 Build History 框里的“trend”链接。这里会给你展示一张历次运行的线形图表，显示每个构建运行了多长时间，如图 6-8 所示。现在，测试已经不是一个构建作业里唯一发生的事情了，但是如果有很多需要你担心的测试的话，它们可能会花费你大部分时间。所以此图也是一种查看你的测试执行情况如何的好方式。

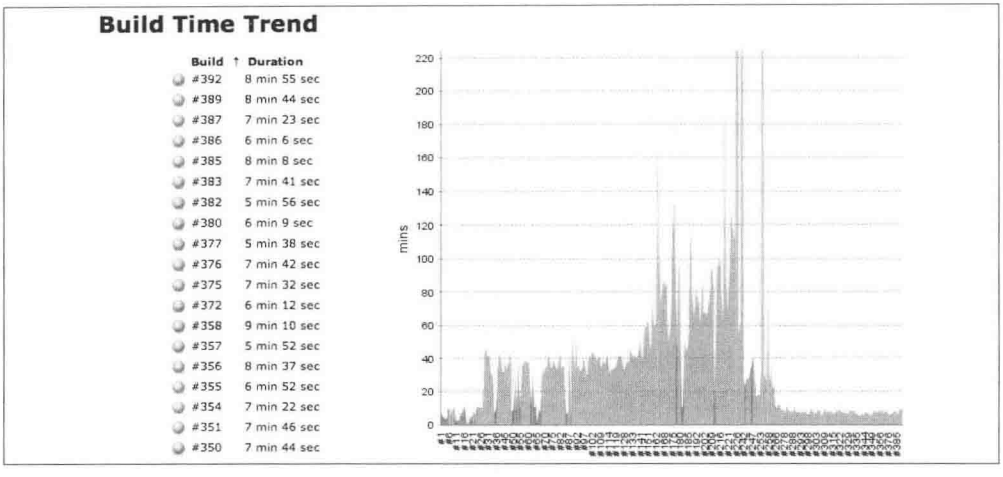


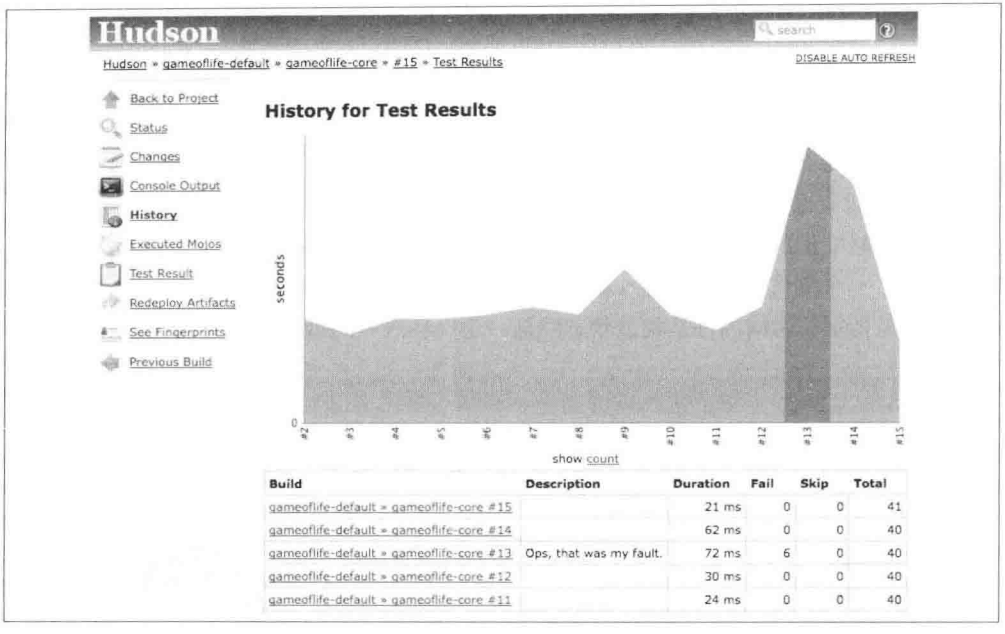
图6-8　构建时间趋势图能够提供关于你的测试运行速度的形象展示

当你正处于 Test Results 页面上时（见图 6-6），你还可以更深入到一个特定模块、包或者类下查看一下这些测试分别运行了多长时间。只需要单击在测试结果页面上的测试持续时间（图 6-6 中“花费 31ms”）去查看一个包、类或者单个测试的测试历史（见图 6-9）。这使得隔离一个在实际中更加耗时的测试，甚至是决定对你的单元测试需要进行常规优化的时机都变得更加容易。

忽略测试

Jenkins 区分失败的和跳过的测试。跳过的测试就是那些已经失效了的测试，比如在 JUnit 4 中使用 `@Ignore` 注释：

```
    @Ignore("Pending more details from the BA")
    @Test
    public void cashWithdrawalShouldDeductSumFromBalance() throws Exception {
    Account account = new Account();
143 >   account.makeDeposit(100);
    account.makeCashWithdraw(60);
    assertThat(account.getBalance(), is(40));
}
```



Build	Description	Duration	Fail	Skip	Total
gameoflife-default » gameoflife-core #15		21 ms	0	0	41
gameoflife-default » gameoflife-core #14		62 ms	0	0	40
gameoflife-default » gameoflife-core #13	Ops, that was my fault.	72 ms	6	0	40
gameoflife-default » gameoflife-core #12		30 ms	0	0	40
gameoflife-default » gameoflife-core #11		24 ms	0	0	40

图6-9　Jenkins也能让你知道自己的测试运行花费了多长时间

在某些情况下，跳过一些测试是完全合法的，比如放置一个自动化验收测试，或者在你实现了低水平的技术测试之后，设置一些更高级的技术测试。在这种情况下，你不想被失败的验收测试打扰，但又不想忘记测试的实际存在。使用诸如`@Ignore`注释这样的技术要比简单地在测试之外做注释或者重命名（在JUnit 3）它要更好，因为它允许Jenkins为你监控忽略了的测试。

在TestNG里，你可以使用`enabled`属性跳过测试：

```
@Test(enabled=false)
public void cashWithdrawalShouldDeductSumFromBalance() throws Exception {
    Account account = new Account();
    account.makeDeposit(100);
    account.makeCashWithdraw(60);
    assertThat(account.getBalance(), is(40));
}
```

在TestNG里，你也可以在测试之间定义依赖，以便某些测试只会在另一个测试或者另一组测试运行之后才会开始运行，正如这里所示：

144

```
@Test
public void serverStartedOk() {...}
```

```
@Test(dependsOnMethods = { "serverStartedOk" })
public void whenAUserLogsOnWithACorrectUsernameAndPasswordTheHomePageIsDisplayed(){..}
```

在这里，如果第一个测试（serverStartedOk()）失败，接下来的测试将会被跳过。

在所有这些情况下，Jenkins 会把那些没运行的测试标记为黄色，包括在整个测试结果趋势图和测试细节里（见图 6-10）。跳过测试并不是和测试失败了一样糟糕，但重要的是不要养成忽略它们的习惯。跳过测试就像在一个版本控制系统里的分支一样：由于一个特殊原因，一个测试应该被跳过；还要清楚什么时候它们能被重新激活。一个被跳过的测试如果保持很长时间内都是跳过状态的话，是存在潜在问题的。

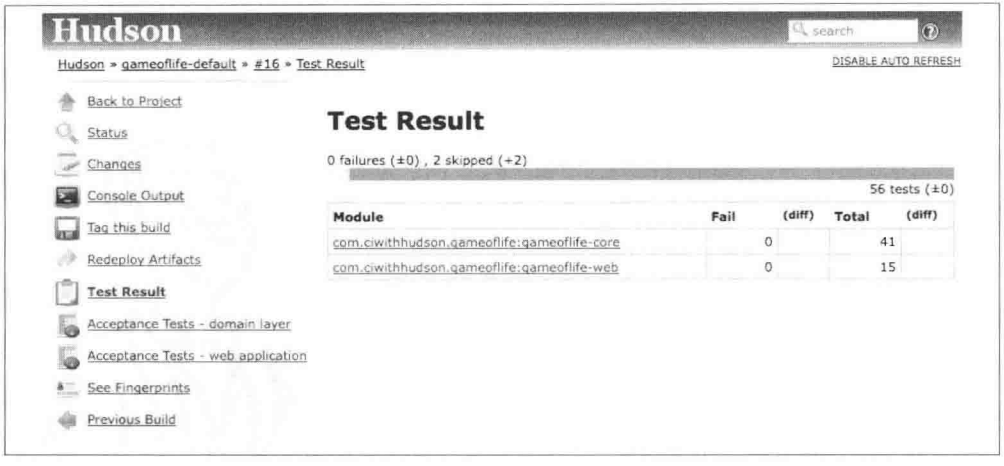


图 6-10　Jenkins显示跳过的测试为黄色

代码覆盖率

另一个非常有用的跟测试相关的度量就是代码覆盖率。代码覆盖率为你显示了在这个测试过程中应用程序的哪些部分正在被执行。虽然这本身不是质量测试的一个充分的展示（执行一整个实际上没有测试任何东西的应用程序是非常简单的，并且代码覆盖率度量并不提供测试质量或者精确度的展示），但是它很好地展示了没有被测试到的那部分代码。而且，如果你的团队正在引入严格的测试实践（比如测试驱动开发模式），代码覆盖率可以作为一个测量实际开发实现情况的合适指标。

145 代码覆盖率分析是一个 CPU 和内存密集型的过程，会显著地降低你构建作业的速度。由于这个原因，你通常需要在一个单独的 Jenkins 构建作业中运行代码覆盖率度量，并在你的单元测试和集成测试成功地完成之后运行。

有很多可用的代码覆盖率工具，其中有些在 Jenkins 里是被支持的，但是它们都要通过

专门的插件来支持。Java 开发人员可以在 Cobertura 和 Emma 里选择一个，这是两款非常流行的开源代码覆盖率工具；或者也可以使用 Clover，它是来自 Atlassian 的一个非常强大的商业代码覆盖率工具。对于 .NET 项目，你可以使用 NCover。

这些工具的行为和配置都是类似的。在本节，我们会着眼于 Cobertura。

使用 Cobertura 测量代码覆盖率

Cobertura（*http://cobertura.sourceforge.net*）是一个针对 Java 和 Groovy 的开源代码覆盖率工具，它易于使用且能和 Maven 及 Jenkins 很好地集成。

就像几乎所有 Jenkins 上的代码质量度量插件一样，Jenkins 上的 Cobertura[注1] 插件也不会为你运行任何测试覆盖率度量。它需要你去生成原始的代码覆盖率数据作为自动化构建过程中的一部分。另一方面，Jenkins 在代码覆盖率度量的报告上也做得非常出色，包括随时跟踪代码覆盖率和提供跨多个应用模块的总覆盖率。

代码覆盖率是一个复杂的业务，它能帮助了解 Cobertura 遵循的基本过程（特别是当你需要把它设置在像 Ant 这样更低级别的构建脚本工具上时）。代码覆盖率分析的工作分三个步骤。首先，它修改（或改装）你的应用程序中的类，从而记录每一行代码执行的次数。[注2] 它们把所有这些数据都存放在一个特殊的数据文件当中（Cobertura 使用一个名为 *cobertura.ser* 的文件）。

当应用程序代码新增加了后，你再针对这些新加的代码运行测试程序。在测试的最后，Cobertura 会生成一个包含了在这个测试过程中每行代码执行次数的数据文件。

一旦生成了这个数据文件，Cobertura 就可以使用这个数据以一种更有用的格式（比如 XML 或者 HTML），去生成一个报告。

在 Maven 中集成 Cobertura

146

在 Maven 中使用 Cobertura 生产代码覆盖率度量是相对简单的。如果你对生产代码覆盖率数据感兴趣，则只需要把 *cobertura-maven-plugin* 添加到 *pom.xml* 文件中的构建部分即可：

```
<project>
  ...
  <build>
    <plugins>
```

注 1：Cobertura：我们会在本书的后面进行讨论。

注 2：这实际上是个轻微的简化描述；事实上，Cobertura 也存储其他数据，比如每一个可能的布尔测试的结果。然而这并不改变一般的方法。

```
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>cobertura-maven-plugin</artifactId>
        <version>2.5.1</version>
        <configuration>
          <formats>
            <format>html</format>
            <format>xml</format>
          </formats>
        </configuration>
      </plugin>
      ...
    </plugins>
  <build>
  ...
</project>
```

这将在你直接调用 Cobertura 插件的时候生成代码覆盖率度量：

```
$ mvn cobertura:cobertura
```

代码覆盖率数据会生成在 *target/site/cobertura* 目录下一个名为 *coverage.xml* 的文件里。

然而，这种方法会测试你的类并且为每一个构建生产代码覆盖率数据，这样是非常低效的。一个更好的办法就是把这些配置文件放在一个特殊的配置文件里，如下所示：

```
<project>
  ...
  <profiles>
    <profile>
      <id>metrics</id>
      <build>
        <plugins>
          <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>cobertura-maven-plugin</artifactId>
            <version>2.5.1</version>
            <configuration>
              <formats>
                <format>html</format>
                <format>xml</format>
              </formats>
            </configuration>
          </plugin>
        </plugins>
      </build>
```

```
    </profile>
    ...
  </profiles>
</project>
```

在本例中，你需要使用度量配置文件调用 Cobertura 插件去生成代码覆盖率数据：

```
$ mvn cobertura:cobertura -Pmetrics
```

另外一种方法就是在你的 Maven 报告中包含代码覆盖率报告。这种方法与仅仅生成覆盖率数据相比较的话，相对较慢，且会耗费更多的内存资源，但假如你同时也想生成其他代码质量度量和报告时就很有意义了。假如你想使用 Maven 2，你还需要在报告部分包含 Maven Cobertura 插件，如下所示：

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>cobertura-maven-plugin</artifactId>
        <version>2.5.1</version>
        <configuration>
          <formats>
            <format>html</format>
            <format>xml</format>
          </formats>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
</project>
```

现在你就可以在为这个项目生成 Maven 网站时生成覆盖率数据了：

```
$ mvn site
```

如果 Maven 项目包含模块（通常在作为较大的 Maven 项目实践时），则你只需要在父 *pom.xml* 文件中设置 Cobertura 相关配置——会为每一个模块生成测试覆盖率度量和报告。假如你在使用 `aggregate` 配置选项，Maven Cobertura 插件也会生成高级别的结合了来自所有模块的覆盖率数据的报告。然而，不管你是不是使用这个选项，Jenkins Cobertura 插件都会从几个文件中拿到覆盖率数据并且把它们结合到一个单独的汇总报告中。

在本书写作之际，Maven Cobertura 插件有个缺陷——代码覆盖率只会为那些在测试生

148 命周期阶段执行的测试进行记录，而不会为集成测试阶段执行的测试进行记录。这会是一个问题，在你使用这个阶段去运行一个要求完全打包和部署了应用的集成或者 Web 测试的时候——在这种情况下，只在集成测试阶段中执行的测试的覆盖率将不会计算在 Cobertura 的代码覆盖率度量中。

Cobertura 与 Ant 结合

集成 Cobertura 到 Ant 构建当中要比集成在 Maven 里更复杂。不过，在覆盖率被测量的时候，它却能让你更好地控制哪些类可被检测。

Cobertura 以捆绑一个 Ant 任务的方式出现，你可以使用这个任务来集成 Cobertura 到 Ant 构建当中。你需要下载最新的 Cobertura 发布版本，并且解压到你硬盘的某个地方。为了让你的构建更便携，也更容易部署到 Jenkins 中，一个好的做法就是把你正在使用的 Cobertura 发布版本放在自己的项目目录内，并把它保存在你的版本控制系统里。这种方式使得不管构建运行在哪里，都能够很容易地确保它会使用 Cobertura 的同一个版本。

假设你已经下载了最新版本的 Cobertura 安装包，并且把它放在你的项目中一个叫作 *tools* 的目录下，你可以这样做：

```
<property name="cobertura.dir" value="${basedir}/tools/cobertura" />❶

<path id="cobertura.classpath">❷
    <fileset dir="${cobertura.dir}">
       <include name="cobertura.jar" />❸
       <include name="lib/**/*.jar" />❹
    </fileset>
</path>

<taskdef classpathref="cobertura.classpath" resource="tasks.properties" />
```

❶ 告诉 Ant 你的 Cobertura 安装文件在哪里。

❷ 我们需要建立一个 Cobertura 可以用来运行的 classpath。

❸ 路径包含 Cobertura 应用程序本身。

❹ 所有的依赖。

接下来，我们需要测量你的应用程序类。你必须注意这些测量类需要放在一个独立的目录里，以防止它们在意外的情况下被打包和部署到生产环境中去。

```
<target name="instrument" depends="init,compile" >❶
  <delete file="cobertura.ser" />❷
```

```
  <delete dir="${instrumented.dir}" />❸
  <cobertura-instrument todir="${instrumented.dir}" >❹
    <fileset dir="${classes.dir}">
      <include name="**/*.class" />
      <exclude name="**/*Test.class" />
    </fileset>
  </cobertura-instrument>
</target>
```


❶ 一旦这些应用程序测试类被编译，我们就只能测量它们一次。

❷ 删除先前的构建生成的所有覆盖率数据。

❸ 删除所有先前测量的类。

❹ 测量应用程序类（但不是测试类），并且把它们放在 *${instrumented.dir}* 目录下。

在这个阶段，*${instrumented.dir}* 目录下包含一个应用程序类的测量版本。现在，我们所需要做的事情就是针对放在目录里的那些类，运行单元测试，最终生成一些有用的代码覆盖率数据。

```
<target name="test-coverage" depends="instrument" >
  <junit fork="yes" dir="${basedir}">❶
    <classpath location="${instrumented.dir}" />
    <classpath location="${classes.dir}" />
    <classpath refid="cobertura.classpath" />❷

    <formatter type="xml" />
    <test name="${testcase}" todir="${reports.xml.dir}" if="testcase" />
    <batchtest todir="${reports.xml.dir}" unless="testcase">
      <fileset dir="${src.dir}">
        <include name="**/*Test.java" />
      </fileset>
    </batchtest>
  </junit>
</target>
```

❶ 针对要被测量的应用程序类，运行 JUnit 测试。

❷ 这些要被测量的类使用了 Cobertura 类，所以这些 Cobertura 库也需要在 classpath 中。

这将会产生原始的测试覆盖率数据，我们需要生成 Jenkins 可以使用的 XML 格式的测试覆盖率报告。事实上要产生这些报告，我们需要调用另外一个任务，如下所示：

```
<target name="coverage-report" depends="test-coverage">
  <cobertura-report srcdir="${src.dir}" destdir="${coverage.xml.dir}"
```

```
                    format="xml" />
</target>
```

最后，在你完成这些之后别忘了进行整理：整理的目标是不仅要删除生成的类，还要删除生成的测量类、Cobertura 覆盖率数据以及 Cobertura 报告：

```
<target name="clean"
        description="Remove all files created by the build/test process.">
    <delete dir="${classes.dir}" />
    <delete dir="${instrumented.dir}" />
    <delete dir="${reports.dir}" />
    <delete file="cobertura.log" />
    <delete file="cobertura.ser" />
</target>
```

150

当这些做完后，你就可以将覆盖率报告集成到 Jenkins 里去了。

安装 Cobertura 代码覆盖率插件

只要代码覆盖率数据作为你的构建过程的一部分被生成，你就可以配置 Jenkins 来对其做报告。这涉及安装 Jenkins Cobertura 插件。我们在第 2 章的“添加代码覆盖率和其他度量”一节讲过相关操作流程，但我们还会再过一遍来帮你记忆。进入 Manage Jenkins 界面，并且单击 Manage Plugins。这样会把你带到一个 Plugin Manager 界面。假如 Cobertura 还没有安装，则你会在 Available 选项卡中的 Build Reports 部分里找到 Cobertura 插件（见图 6-11）。要安装它的话，只需要勾选复选框，然后按下回车键（或向下滚动到屏幕底部，然后单击“Install”按钮）。Jenkins 会为你下载并安装插件。一旦下载完成，你需要重启 Jenkins 服务器。

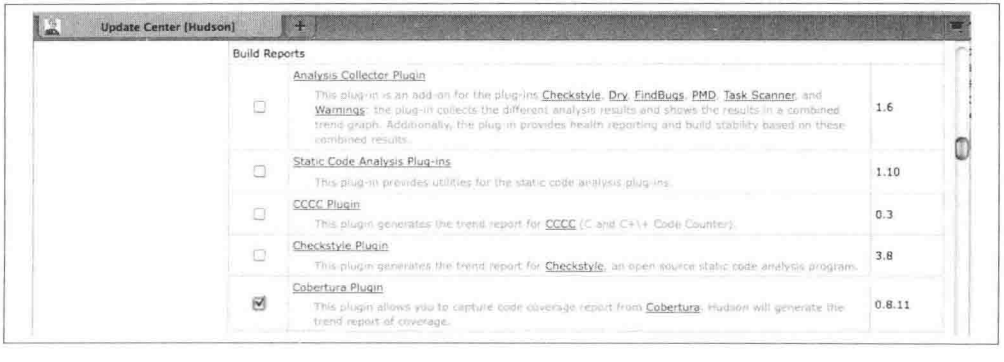


图 6-11　安装Cobertura插件

报告构建的代码覆盖率

只要你已经安装了插件，就可以在构建作业里设置代码覆盖率报告。由于代码覆盖率会非常慢且耗费内存，所以你通常需要为它以及其他代码质量度量创建一个单独的构建作业，在普通的单元和集成测试之后运行。对于那些非常大的项目，你可能也想要把这个设置成一个只在夜间运行的构建。的确，代码覆盖率和其他所谓的度量的反馈通常不像测试结果的反馈那样对时间序列要求严格，这会为受益于快速反馈的构建作业留下空闲的构建执行器。

正如我们先前提到的，Jenkins 本身不做任何代码覆盖率的分析——你需要配置自己的构建去生 Cobertura *coverage.xml* 文件（或多个文件），在你可以生成任何形式的图或者报告之前，通常会使用我们之前讨论的技术之一（见图 6-12）。

151

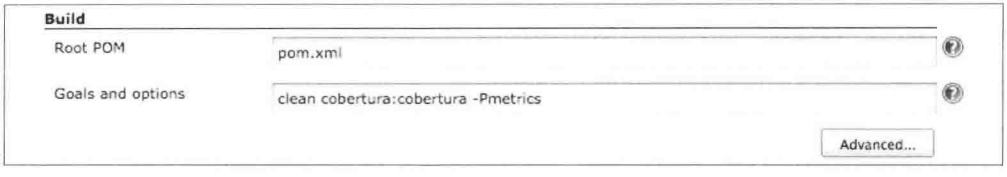


图6-12　代码覆盖率度量构建需要生成覆盖率数据

一旦你已经配置了自己的构建去生成一些代码覆盖率数据，就可以在构建作业的“Post-build Actions”部分配置 Cobertura。当你单击“Publish Cobertura Coverage Report”复选框时，应该能看到像图 6-13 里的一些东西。

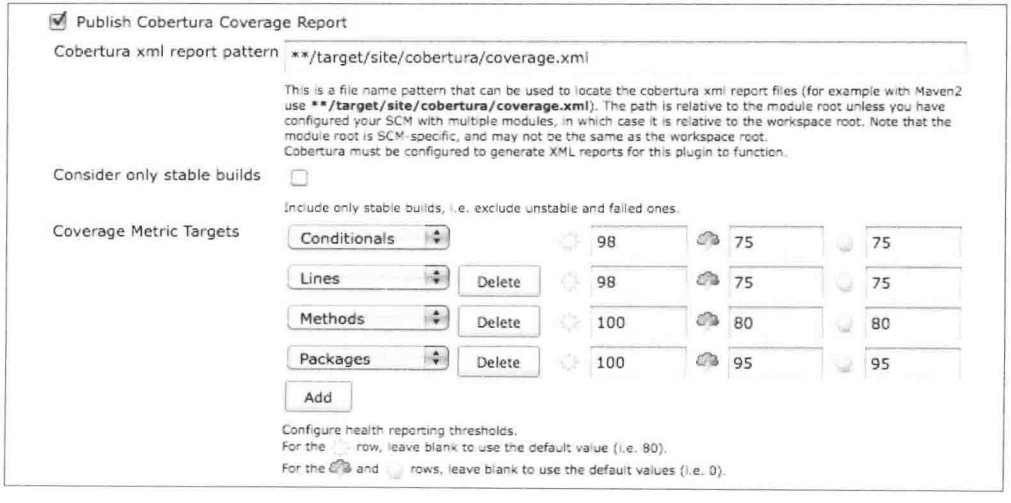


图 6-13　在Jenkins里配置测试覆盖率度量

这里最优先和最重要的地方是我们生成的 Cobertura XML 数据的路径。你的项目可能包含一个或几个单个的 *coverage.xml* 文件。比如，假如你有一个多模块的 Maven 项目，这个 Maven Cobertura 插件就会为每一个模块生成一个分离的 *coverage.xml* 文件。

path 变量接受 Ant 风格的通配符，所以它能很容易地包含来自若干个文件中的代码覆盖率数据。对于任一个 Maven 项目，像 ***/target/site/cobertura/coverage.xml* 这样的路径都会包含这个项目中所有模块的代码覆盖率度量。

事实上有几种类型的代码覆盖率，这在需要区分它们的时候会很有用。最直观的就是行(Line)覆盖率，它计算了在自动化测试期间任何被给出的代码的执行次数。“条件覆盖率”

（也被称为“分支覆盖率”）是考虑在 `if` 或 like 语句中的条件判断中，是否测试了所有的可能情况。比如，考虑下面的代码片段：

```
if (price > 10000) {
  managerApprovalRequired = true;
}
```

要想为这段代码获得完整的条件覆盖，你需要执行两次：一次是使用一个超过 10 000 的值，另一次是使用一个 10 000 或者小于 10 000 的值。

其他基本的代码覆盖率衡量度量包括方法（应用中有多少方法被测试用例测试过）、类以及包。

Jenkins 能够让你定义你想跟踪这些度量中的哪一个。默认情况下，Cobertura 插件会记录条件（Conditional）、行（Line）和方法（Method）覆盖率，这些通常就足够了。然后，假如你认为添加其他的覆盖率度量可能对于你的团队来说是有用的，这个也是很容易的。

Jenkins 代码质量度量并不仅仅是一个被动的报告过程——Jenkins 能让你定义如何让这些度量来影响构建结果。你可以为那些影响构建结果和 Jenkins 仪表盘上的天气报告（见图 6-14）的覆盖率度量定义阈值。

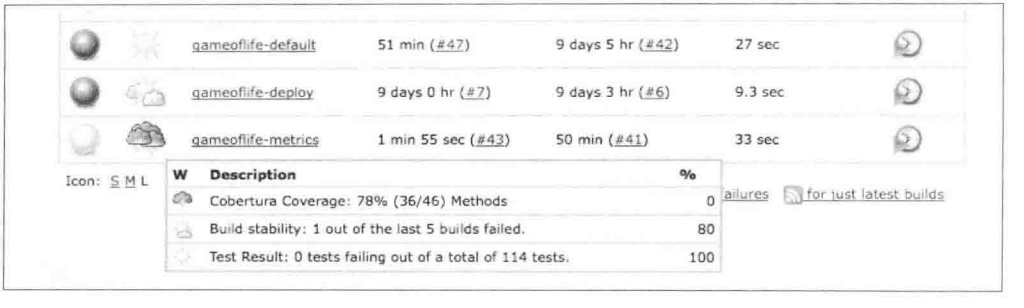


图6-14　在仪表盘上的项目状态中的测试覆盖率结果

第一个就是（就是有阳光图标的那个）构建想要获得一个阳光灿烂的天气图标所必需的最小值。第二个显示了一个低于此值就会把构建归为一个暴风雨天气图标的值。Jenkins会相应地为其他更微妙的天气图标推断出一个介于这些值之间的值。

最后一个阈值若是低于它的话，构建就会被标记为“不稳定的”——黄球。虽然不像红球（表示一个坏了的构建）那么糟糕，但一个黄球也仍然会导致一个消息通知并且在仪表盘上会很难看。

这个功能远非简单的装饰细节——它为你的项目提供了一个有价值的可以客观地设置代码质量目标的方式。尽管它不能被单独解释，但在项目中，代码覆盖率下降一般来说并不是个好的迹象。所以如果你很看重代码覆盖率，当没有达到标准时，使用这些阈值来提供一些难做到的反馈。

解释代码覆盖率

Jenkins在构建作业的主页上显示你的代码覆盖率报告。第一次运行时，它会生成一个简单的条线图（见图2-30）。从第二个构建起，会显示出来一个图，表示你随着时间跟踪着的各种各样类型的覆盖率（见图6-15）。在这两种情况下，这个图也会展示最近一次构建的代码覆盖率度量。

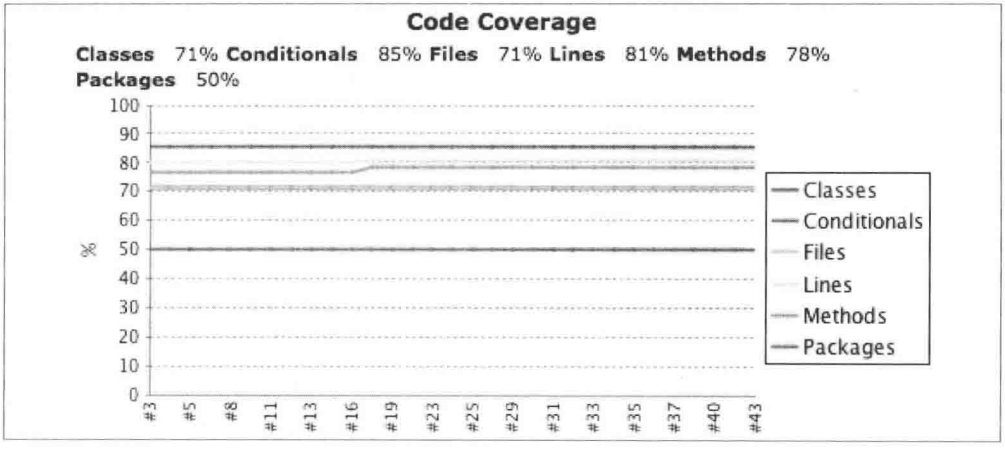


图6-15　在Jenkins中配置测试覆盖率度量

Jenkins也能够很好地让你深入地跟踪到覆盖率度量，显示覆盖率故障的包、包内的类以及类内的代码行（见图6-16）。不管你正在查看什么级别的细节，Jenkins都会在页面的顶端展示一个随着时间变化的代码覆盖率趋势图。再往下看，你会发现包或者类的故障。

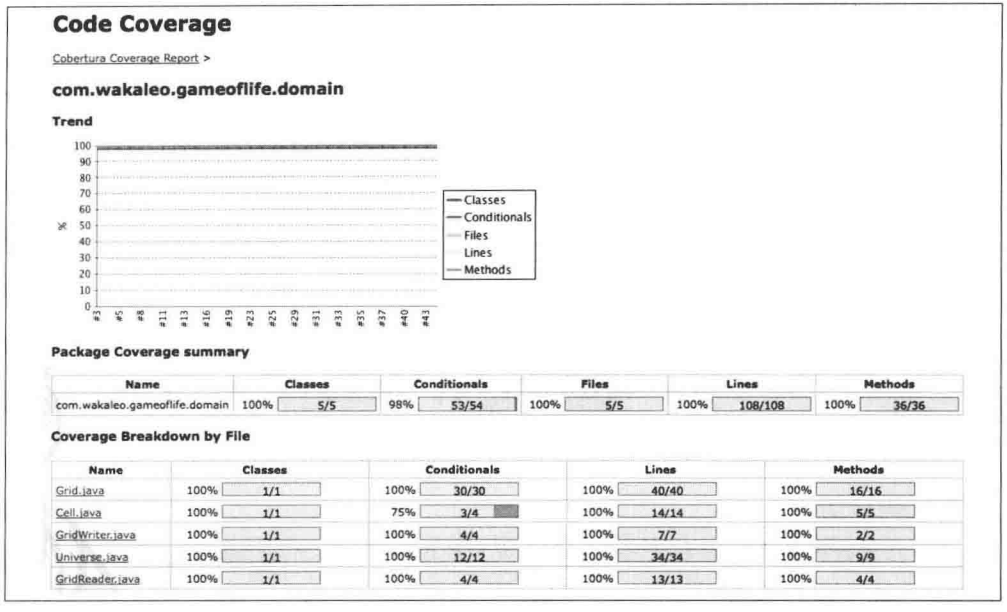


图 6-16　显示代码覆盖率度量

一旦你到达类的细节这个级别，Jenkins 也会显示这个类的源代码，附带了根据它们的覆盖率水平而进行着色的线条。在测试期间，已经被完全执行过的代码行就是绿色的，从来没有被执行的代码行就被标记为红色。竖轴的数字表示对应线所代表的指标被执行的次数。最后，字页边框的黄色阴影用来表示不全面的条件覆盖率（比如，在一个结果中只测试一个 `if` 语句）。

使用 Clover 测量代码覆盖率

Clover 是一款来自 Atlassian（*http://www.atlassian.com/software/clover*）的优秀的商业代码覆盖率工具。Clover 对于使用 Ant、Maven 甚至 Grails 的项目来说工作得非常好。关于 Clover 的配置和使用，在 Atlassian 官方网站上有非常好的文档记录，所以我们将不会再详细描述这些内容。但是，为了给出一些上下文，这里是一个通常情况下在 Jenkins 中使用的 Clover 的 Maven 2 的配置，如下所示：

```
<build>
  ...
  <plugins>
    ...
    <plugin>
      <groupId>com.atlassian.maven.plugins</groupId>
      <artifactId>maven-clover2-plugin</artifactId>
```

```
      <version>3.0.4</version>
      <configuration>
        <includesTestSourceRoots>false</includesTestSourceRoots>
        <generateXml>true</generateXml>
      </configuration>
    </plugin>
  </plugins>
</build>
...
```

这将会生成一个 HTML 和 XML 覆盖率报告，包括不断汇总的数据（假如这个 Maven 项目包含多个模块的话）。

要想集成 Clover 到 Jenkins 里面的话，你需要使用插件管理器界面去安装 Jenkins Clover 插件。一旦你重启了 Jenkins，就可以集成 Clover 代码覆盖率到你的构建里了。

在你的项目中运行 Clover 是一个多步骤的工程：你对自己的应用程序代码进行仪器测量，运行测试，汇总这些测试数据（对于多模块的 Maven 项目来说）并且生成 HTML 和 XML 报告。由于这是个相当缓慢的操作，因此，通常你需要把它当作一个单独的构建作业的一部分来运行，而不是和你那些普通的测试一起。你可以像下面这样做：

155

```
$ clover2:setup test clover2:aggregate clover2:clover
```

接下来，你需要在 Jenkins 里设置 Clover 报告。可勾选 Publish Clover Coverage Report 复选框来设置该项。这些配置和 Cobertura 的配置有些类似——你需要提供能访问到 Clover HTML 报告目录和 XML 报告文件的 path 变量，并且你也可以为充满阳光的和暴风雨的天气，及不稳定的构建来定义阈值（见图 6-17）。

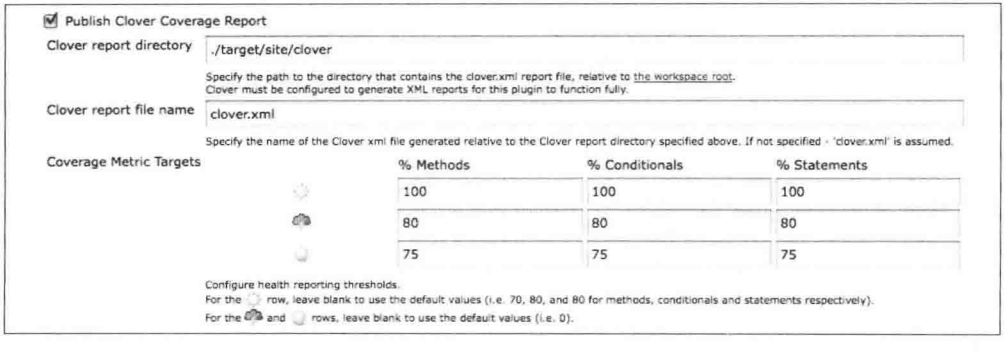


图 6-17　在Jenkins中配置Clover报告

一旦你完成这些设置，Jenkins 就会在你项目构建作业的主页上（见图 6-18）显示当前代码覆盖率的级别，以及一个随时间变化的代码覆盖率的图。

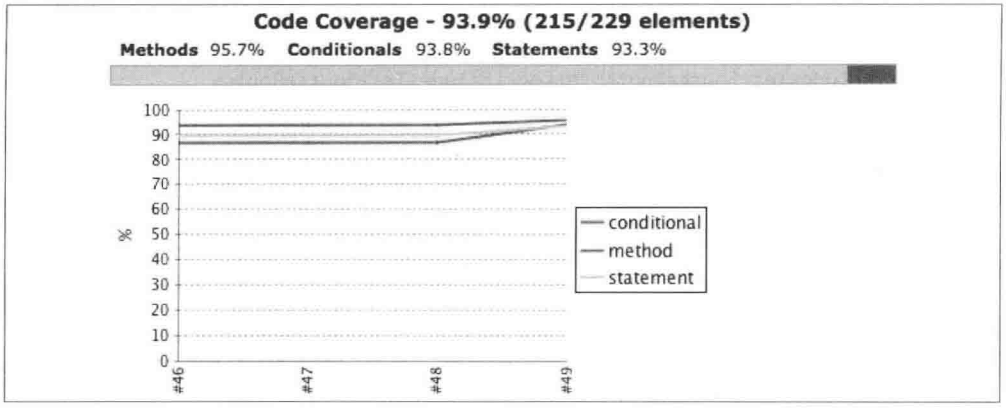


图 6-18 Clover代码覆盖率趋势图

自动化验收测试

自动化验收测试在很多敏捷型的项目中扮演了非常重要的角色，不论是在验证还是在沟通过程中均如此。作为一个验证工具，验收测试扮演了和集成测试相类似的角色，目标是展示应用程序按照预期有效地完成了工作。不过，这几乎是自动化验收测试的次要方面。事实上主要方面聚焦在交流上——精确地向非开发人员展示这个项目目前处于什么阶段（业务所有者、业务分析者、测试人员等）。

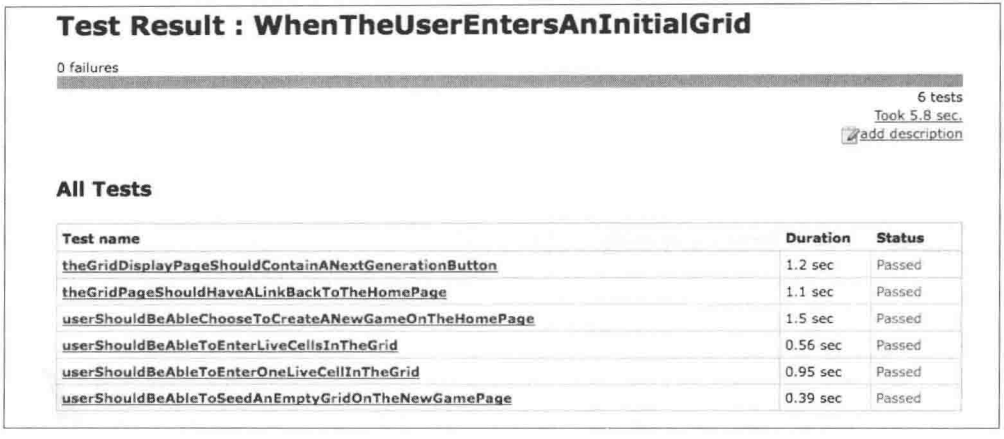
验收测试不应该和开发者关注的测试混为一谈，因为它们的目标和它们的受众都是非常不同的。验收测试应该是这个系统怎么工作的工作实例，强调展示而不是详尽的证明。详尽的测试用例在单元测试级别就应该被完成。

验收测试可以使用一些常见的工具比如 JUnit 来完成自动化，但是为了达到这个目的，还有一个呈增长趋势的就是使用行为驱动开发（BDD）框架，因为它们对于验收测试面向公众化的这个性质来说，倾向于是一个更好的选择。用于自动化验收测试的行为驱动开发工具通常生成带有特定布局的 HTML 报告，这对于非开发人员来说是非常合适的。它们也经常生产兼容 JUnit 的报告，这样的报告能直接被 Jenkins 识别。

行为驱动开发框架也有“等待测试”的概念；也就是说测试用例已经自动化起来，但是开发团队还没有完成。这个区别在和其他相关的非开发人员进行交流的时候扮演着一个非常重要的角色：假如你能在这个过程的早期就把这些测试自动化起来，它们就可以给你一个优秀的指标告诉你哪些功能已经被完成了、哪些已经开始工作以及哪些还没有开始。

一般来说，你的验收测试应该和其他更常见的自动化测试分开来显示。假如它们使用了

和你的标准测试（如 JUnit）一样的测试框架，要确保它们在一个专有的构建作业中执行，以便非开发人员能够查看和专注于业务聚焦的测试，而不需要被低级的或者技术性的问题所打扰。它也可以帮助使业务聚焦和行为命名约定适合于你的测试以及测试类，使它们对于非开发人员来说变得更易于理解（见图 6-19）。在阅读测试报告和理解事实上的业务功能以及被测试的行为时，你命名测试或者测试类的方式会产生巨大的差异。



Test Result : WhenTheUserEntersAnInitialGrid

0 failures

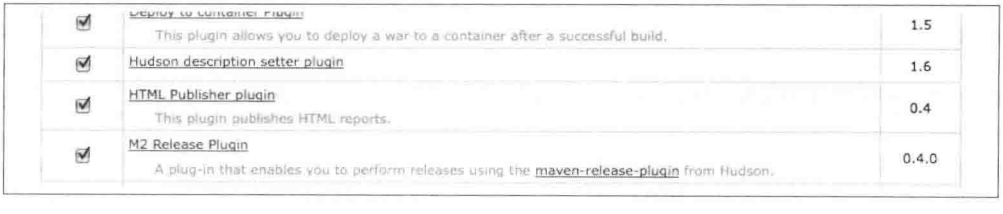
6 tests
Took 5.8 sec.
add description

All Tests

Test name	Duration	Status
theGridDisplayPageShouldContainANextGenerationButton	1.2 sec	Passed
theGridPageShouldHaveALinkBackToTheHomePage	1.1 sec	Passed
userShouldBeAbleChooseToCreateANewGameOnTheHomePage	1.5 sec	Passed
userShouldBeAbleToEnterLiveCellsInTheGrid	0.56 sec	Passed
userShouldBeAbleToEnterOneLiveCellInTheGrid	0.95 sec	Passed
userShouldBeAbleToSeedAnEmptyGridOnTheNewGamePage	0.39 sec	Passed

图 6-19　使用着重业务、行为驱动命名规定的JUnit测试

如果你正在使用一个生成 HTML 报告的工具，则可以在同一个构建里显示它们，就像你那些常见的测试一样（只要它们出现在另一份报告中）。Jenkins 为这种 HTML 报告提供了一个非常方便的插件，称为 HTML Publisher plugin（见图 6-20）。而你仍需确保你的构建生成了正确的报告；不过 Jenkins 可以在你的构建作业主页上显示这些报告，这样使得它们更容易被访问到（对于所有的团队成员来说）。



☑	Deploy to container Plugin This plugin allows you to deploy a war to a container after a successful build.	1.5
☑	Hudson description setter plugin	1.6
☑	HTML Publisher plugin This plugin publishes HTML reports.	0.4
☑	M2 Release Plugin A plug-in that enables you to perform releases using the maven-release-plugin from Hudson.	0.4.0

图 6-20　安装HTML Publisher 插件

这个插件非常易于配置。只需要到“Post-build Actions”部分，然后勾选“Publish HTML reports”复选框（见图 6-21）。接下来，向 Jenkins 提供一个你要生成 HTML 报告到哪里的目录、一个索引页以及你的报告的标题。你也可以要求 Jenkins 为每个构建保存生成的报告，或者只保留最后生成的那个报告。

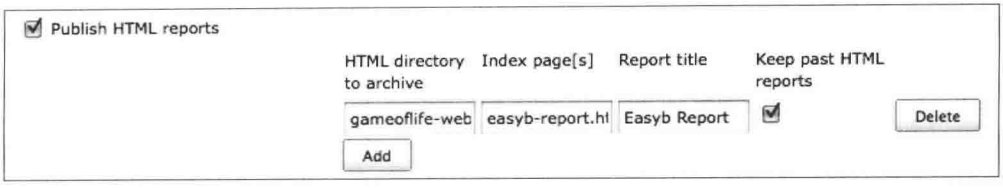


图 6-21　发布HTML报告

一旦完成这些设置，Jenkins 就会在你的构建作业的主页上显示一个特别的图标，以及一个指向 HTML 报告的链接。在图 6-22 中，你可以看到我们先前在活动中配置的 easyb 报告。

158

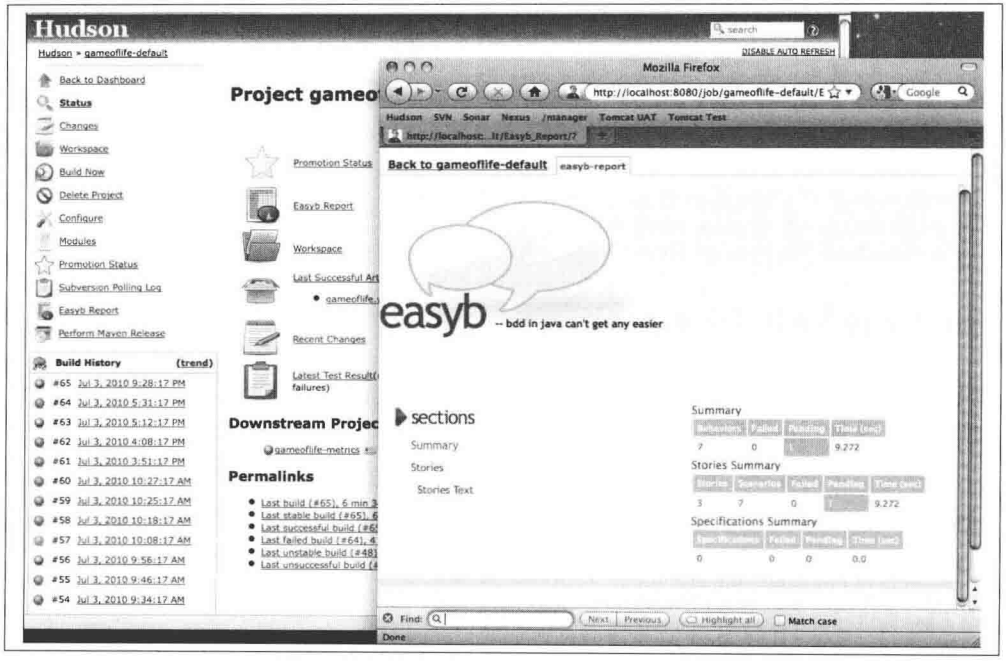


图6-22　Jenkins在构建作业的主页面为你的报告显示出一个特定的链接

HTML Publisher 插件完全适合 HTML 报告。在另一方面，假如你想要（也）发布非 HTML 格式的文档，比如文本文件、PDF 等，这个时候就可以使用 DocLinks 插件。这个插件和HTML Publisher插件类似，但是能够让你把HTML报告和文档归档为其他格式。例如，在图 6-23 中，我们已经配置了一个构建作业去归档为 PDF 文档以及 HTML 报告。现在，这些文档都将会列在构建的主页上。

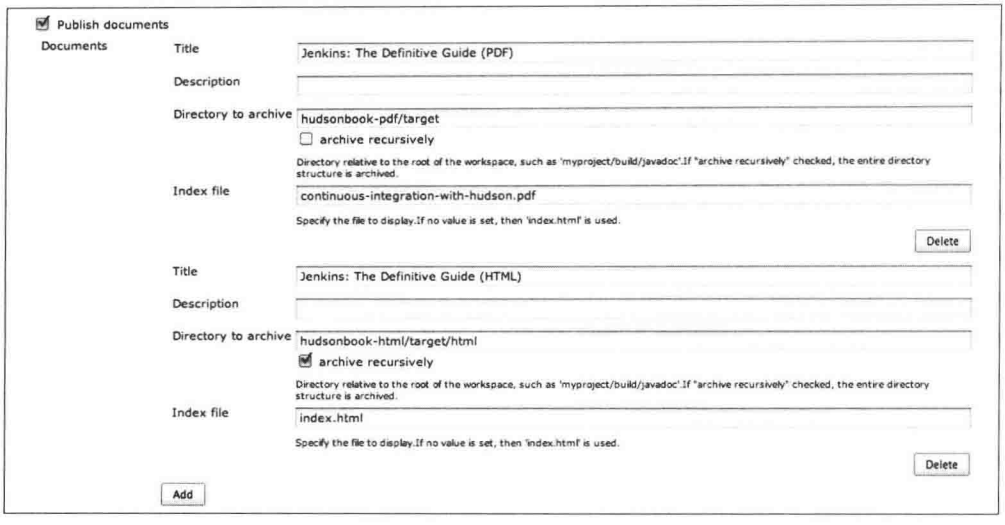


图 6-23　DocLinks插件可以让你归档为HTML或者非HTML格式的文件

JMeter 自动化性能测试

应用程序性能是测试的另一个重要领域。性能测试可以被用来验证很多事情，比如一个应用程序能多快地响应一个给定数目的用户同时发出的请求，或者一个应用程序在应对不断增长的用户数量方面做得能有多好。很多应用程序都有服务水平协议（SLA），它们约定性地定义了这些应用程序应该如何执行。

性能测试通常都是一次性、临时性的活动，只会在一个项目结束的时候或者一些事情开始出错的时候才会进行。不过，性能问题就像其他种类的 bug 一样——在这个过程中它们被检测出来得越晚，在修复它们的时候就会花费越多的时间。因此，自动化这些性能和负载测试是非常有意义的，以便你可以在发布之前精确地找出性能下降的地方。

JMeter（*http://jakarta.apache.org/jmeter/*）是一款非常流行的开源性能和负载测试工具。它的工作原理是，在你的应用程序上模拟负载，并且测量当模拟的用户以及请求的数量增长之后的响应时间。它有效地模拟了一个浏览器或者客户端程序的动作，发送各种类型的请求（HTTP、SOAP、JDBC、JMS 等）到你的服务器。你需要配置一系列的要被发送到应用程序的请求，还要配置随机停顿、条件和循环，以及其他被设计用来更好地模拟真实用户动作的变更。

JMeter 作为一个 Swing 应用程序来运行，你可以在这里配置测试脚本（见图 6-24）。你甚至可以把 JMeter 当作一个代理来运行，然后在一个普通的浏览器里操作应用程序去准备一个测试脚本的最原始版本。

一个关于如何使用 JMeter 的全面指导已经超出了本书的范畴。不过，JMeter 非常易于学习，而且你可以在其官网上找到丰富的关于如何使用它的细节。只需要很少的工作量，你就可以拥有一个非常有质量的测试脚本并且在几小时内就能运行起来。

在这里，我们感兴趣的是把这些性能测试自动化起来的过程。有好几种方法可以把 JMeter 测试集成到你的 Jenkins 构建过程中。尽管在本书写作之际，在 Maven 仓库中还没有一个官方的 JMeter 插件可供使用，不过却有一个 Ant 插件。所以最简单的方法就是写一个 Ant 脚本来运行你的性能测试，然后直接调用这个 Ant 脚本或者（如果你在使用 Maven 项目并且想要通过 Maven 运行 JMeter）使用 Maven Ant 集成去调用来自 Maven 内部的 Ant 脚本。一个简单的运行了一些 JMeter 测试的 Ant 脚本举例如下：

160
```
<project default="jmeter">
  <path id="jmeter.lib.path">
    <pathelement location="${basedir}/tools/jmeter/extras/ant-jmeter-1.0.9.jar"/>
  </path>

  <taskdef name="jmeter"
        classname="org.programmerplanet.ant.taskdefs.jmeter.JMeterTask"
        classpathref="jmeter.lib.path" />


  <target name="jmeter">
    <jmeter jmeterhome="${basedir}/tools/jmeter"
        testplan="${basedir}/src/test/jmeter/gameoflife.jmx"
        resultlog="${basedir}/target/jmeter-results.jtl">
      <jvmarg value="-Xmx512m" />
    </jmeter>
  </target>
</project>
```

这个例子里是假设在你的项目的 *tools* 目录下已经有可用的 JMeter 的安装。把诸如 JMeter 这样的工具放在你的项目结构内是一个好的习惯，因为这使得你的构建脚本可以更便携、更容易地在任何机器上运行，这也正是我们需要 Jenkins 运行它们。

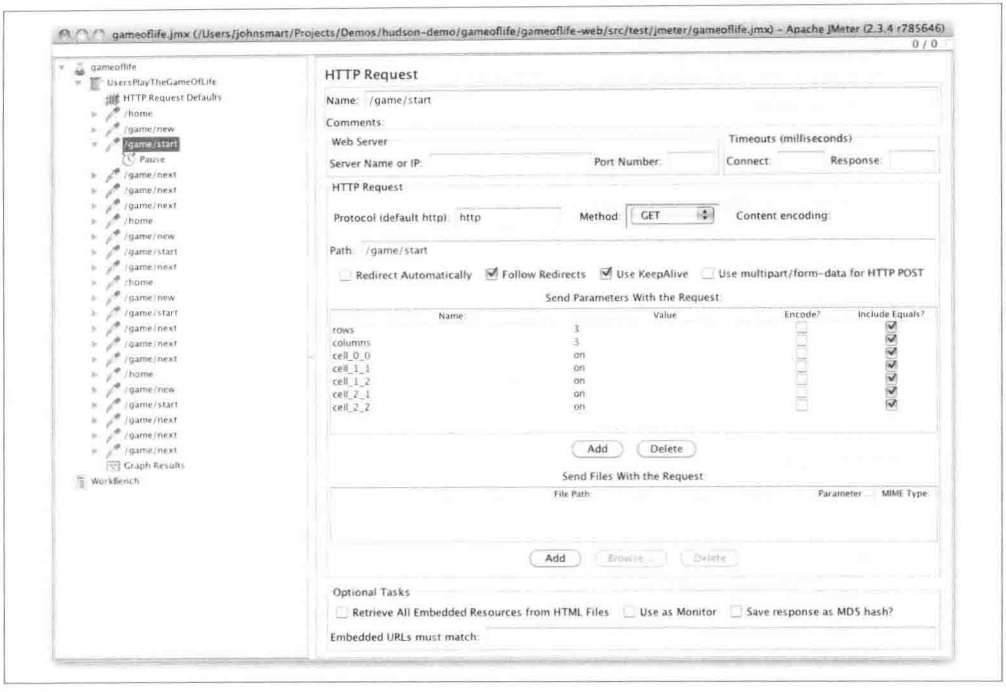


图6-24　在JMeter里准备一个性能测试脚本

需要注意的是，我们也会用一个可选的标签（<jvmarg>）来给 JMeter 提供充足的内存资源——性能测试是一个消耗内存的活动。 <161>

这里展示的脚本会针对一个运行着的应用程序执行 JMeter 性能测试。所以你需要确保自己要测试的应用程序已经在开始测试之前运行起来了。对于那些更重量级的性能测试来说，你通常会想要在开始运行测试之前把应用程序部署到一个测试服务器上。对于大多数应用程序来说这通常不是很困难——比如 Maven Cargo 插件，能让你部署到各种本地或者远程服务器的过程自动化。我们稍后会在本书里看看在 Jenkins 中如何做到这些。

作为另外一种选择，如果你正在为一个 Web 应用程序使用 Maven，则可以使用 Jetty 或者 Cargo 插件来确保应用程序在集成测试开始之前就已经被部署完毕，然后在集成测试阶段从 Maven 内部调用 JMeter Ant 脚本。比如，使用 Jetty 的话，你可以这样做：

```
<project...>
  <build>
    <plugins>
      <plugin>
        <groupId>org.mortbay.jetty</groupId>
        <artifactId>jetty-maven-plugin</artifactId>
```

```
        <version>7.1.0.v20100505</version>
        <configuration>
          <scanIntervalSeconds>10</scanIntervalSeconds>
          <connectors>
            <connector
              implementation="org.eclipse.jetty.server.nio.SelectChannelConnector">
            <port>${jetty.port}</port>
            <maxIdleTime>60000</maxIdleTime>
          </connector>
        </connectors>
        <stopKey>foo</stopKey>
        <stopPort>9999</stopPort>
      </configuration>
      <executions>
        <execution>
          <id>start-jetty</id>
          <phase>pre-integration-test</phase>
          <goals>
            <goal>run</goal>
          </goals>
          <configuration>
            <scanIntervalSeconds>0</scanIntervalSeconds>
            <daemon>true</daemon>
          </configuration>
        </execution>
        <execution>
          <id>stop-jetty</id>
          <phase>post-integration-test</phase>
          <goals>
            <goal>stop</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    ...
  </plugins>
  </build>
</project>
```


这将会先启动一个 Jetty 的实例并把你的 Web 应用程序部署到它上面，然后是集成测试，再之后会停掉这个实例。

最后，你需要在这个阶段运行 JMeter 性能测试。你可以通过使用 *maven-antrun-plugin* 去调用 Ant 脚本来完成这个工作，Ant 脚本是我们先前在集成测试阶段写好了的：

```
<project...>
  ...
  <profiles>
    <profile>
      <id>performance</id>
      <build>
        <plugins>
          <plugin>
            <artifactId>maven-antrun-plugin</artifactId>
            <version>1.4</version>
            <executions>
              <execution>
                <id>run-jmeter</id>
                <phase>integration-test</phase>
                <goals>
                  <goal>run</goal>
                </goals>
                <configuration>
                  <tasks>
                    <ant antfile="build.xml" target="jmeter" >
                  </tasks>
                </configuration>
              </execution>
            </executions>
          </plugin>
        </plugins>
      </build>
    </profile>
  </profiles>
  ...
</project>
```

现在，所有你需要做的就是运行带有性能方面的集成测试来让 Maven 运行 JMeter 测试集。你可以通过调用 *interaction-test* 来做这件事或者验证 Maven 生命周期阶段：

```
$ mvn verify -Pperformance
```

一旦你配置了构建脚本去处理 JMeter，就可以在 Jenkins 里配置起一个性能测试构建。 <163>
对于性能测试构建，我们会使用 Performance Test Jenkins 插件，它可以识别 JMeter 日志并生成很好的统计数据及图标。所以，进入你 Jenkins 服务器上的 Plugin Manager（插件管理器）界面，然后安装这个插件（见图 6-25）。当你完成插件的安装后，需要重启 Jenkins 服务。

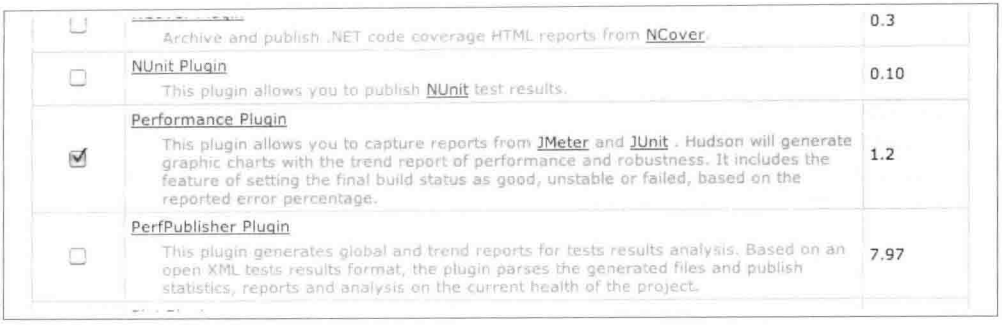


图 6-25　在JMeter中准备一个性能测试脚本

一旦你完成插件的安装，就可以在 Jenkins 中配置一个性能构建作业。这个构建作业通常是跟你的其他构建相分离的。在图 6-26 中，我们已经配置了一个在夜间运行的性能构建，这个构建对于一个长时间运行的负载或者性能测试来说可能已经是足够的了。

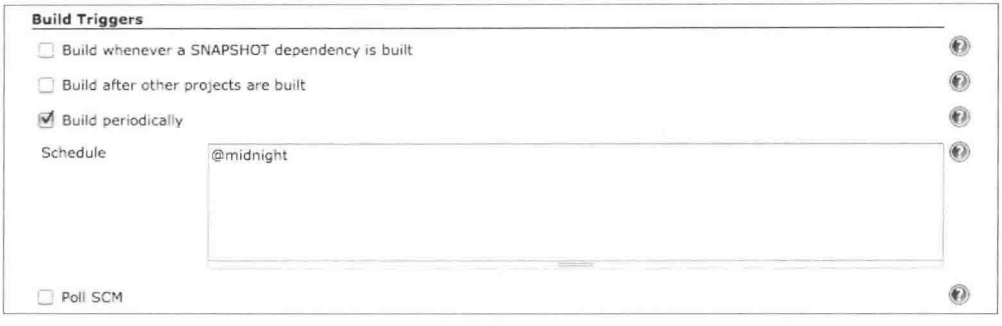


图 6-26　配置一个每天夜里运行的性能构建

剩下的事情就是配置构建作业去运行性能测试。在图 6-27 中，就正在运行我们之前配置了的 Maven 构建。注意我们使用 MAVEN_OPTS 字段来为构建作业提供大量的内存资源。

164

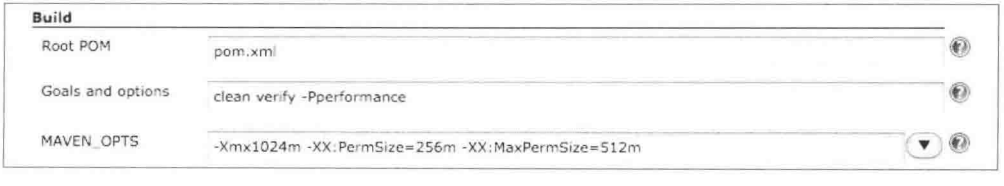


图 6-27　性能测试会需要大量的内存资源

要建立一个性能报告，只需单击 Post-build Actions 部分（见图 6-28）的“Publish Performance test result report”选项。你需要告诉 Jenkins 在哪里能找到 JMeter 测试结果

(是输出文件，而不是测试脚本)。性能插件可以处理多个 JMeter 结果，所以你可以在路径里使用通配符来确保所有 JMeter 报告均能显示出来。

假如你对于性能度量是非常严格的，如果需求的 SLA 不匹配的话，构建就会失败。在一个持续集成环境下，任何种类的度量构建如果不符合最低的质量标准，但却没有显示失败的话，这样的构建往往会被忽略。

你可以配置性能插件来标记一个构建为不稳定或者失败（如果一定百分比的需求最终都出现错误的话）。默认情况下，这些值只出现在真正的应用程序错误（比如 bug）或者服务器崩溃的重大事件中。不管怎样，你真的应该配置 JMeter 测试脚本，为你的请求放置一个最大可接受的响应时间上限。这是特别重要的（如果你的应用程序在这一点上有义务的话）。在 JMeter 中来完成这项工作的一种方法就是通过添加一个 Duration Assertion 元素到你的脚本中。只要有任何请求花费了比实际上固定的执行时间更长的时间的话，就会导致产生错误。



图 6-28　在构建作业中配置性能插件

现在，当构建作业运行的时候，性能插件就会产生一些图来保持跟踪整体响应时间和错误的数量(图 6-29)。你生成的每一个 Jmeter 报告都会有一个独立的图。假如只有一个图，则它会在构建的主页上出现，否则你可以通过 Performance Trend 菜单项在一个专门的页面查看它们。

这个图能够展示最终性能的概述。你通常会使用这个图来确保平均响应时间在你期望的限制之内，并且也能够在平均或者最大响应时间内认出任何异常变化。

假如你需要查出和隔离性能问题，Performance Breakdown 界面就会派上用场了。在 Performance Trend 报告内部，单击屏幕上方的 Last Report 链接。这样就能显示每一个请求的响应时间以及出现的错误（见图 6-30）。你还可以通过单击在构建详情页面的 Performance Report 链接，对之前的构建做同样的操作。

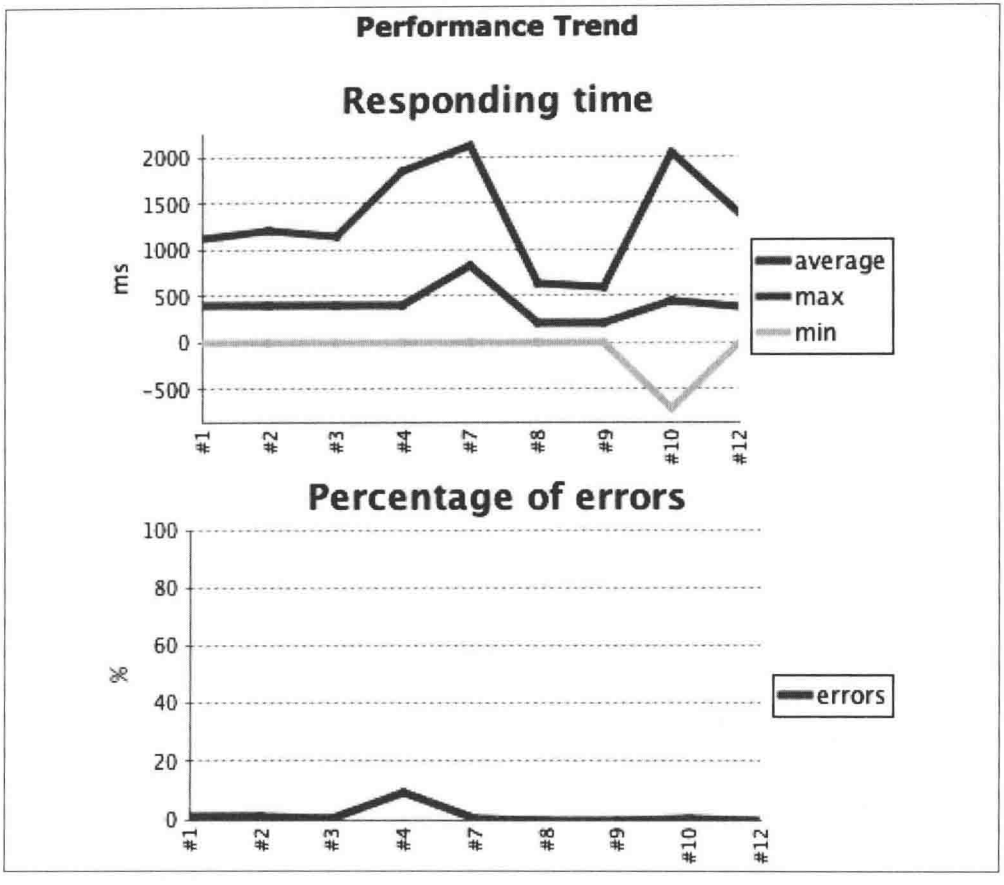
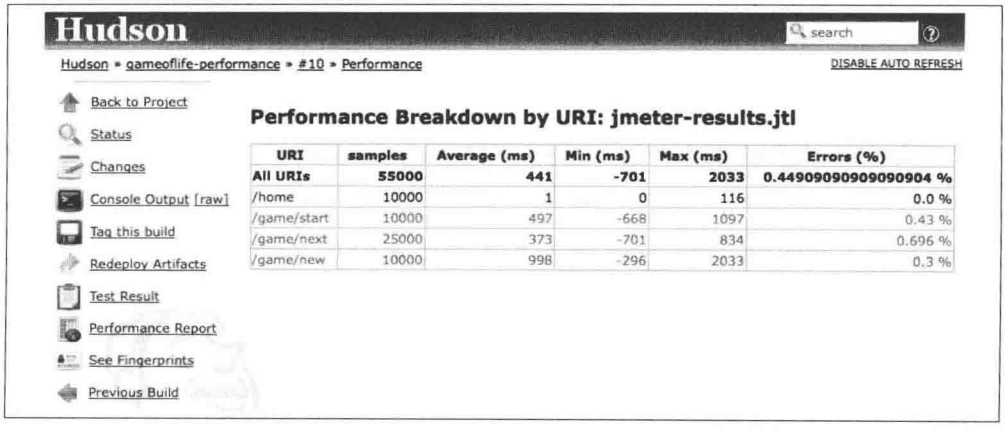


图 6-29　Jenkins Performance插件保持跟踪响应时间和错误



URI	samples	Average (ms)	Min (ms)	Max (ms)	Errors (%)
All URIs	55000	441	-701	2033	0.44909090909090904 %
/home	10000	1	0	116	0.0 %
/game/start	10000	497	-668	1097	0.43 %
/game/next	25000	373	-701	834	0.696 %
/game/new	10000	998	-296	2033	0.3 %

图 6-30　你也可以查看每个请求的性能结果

通过做一些小调整，一个 JMeter 测试脚本就能通过模拟给定的用户量进行工作。通常情况下，无论如何，你都想要看看不同的用户量下你的应用程序表现情况。Jenkins Performance 插件能非常好地处理这个问题，并且可以为多个 JMeter 报告处理图（只需要确保在你告诉 Jenkins 哪里可以找到报告的时候使用通配符表达式）。

当然，能够为每一个运行的测试重用同一个 JMeter 测试脚本也是件非常好的事情。JMeter 支持参数，所以你可以在不同数量的模拟用户的情况下很容易地重用同样的 JMeter 脚本。只需要在你的 JMeter 脚本里使用一个属性表达式，然后在你运行脚本的时候把属性表达式传递给 JMeter。如果你的属性名为 `request.threads`，那么在 JMeter 脚本中的属性表达式就应该是 `${__property(request.threads)}`。然后你可以使用在 `<jmeter>` Ant 任务中的 `<property>` 元素来传属性，当你运行脚本的时候。例如，接下来的 Ant 目标，运行 JMeter 三次，分别针对 200、500 和 1000 个模拟用户：

```
<target name="jmeter">
  <jmeter jmeterhome="${basedir}/tools/jmeter"
        testplan="${basedir}/src/test/jmeter/gameoflife.jmx"
        resultlog="${basedir}/target/jmeter-results-200-users.jtl">
    <jvmarg value="-Xmx512m" />
    <property name="request.threads" value="200"/>
    <property name="request.loop" value="20"/>
  </jmeter>
  <jmeter jmeterhome="${basedir}/tools/jmeter"
        testplan="${basedir}/src/test/jmeter/gameoflife.jmx"
        resultlog="${basedir}/target/jmeter-results-500-users.jtl">
    <jvmarg value="-Xmx512m" />
    <property name="request.threads" value="500"/>
    <property name="request.loop" value="20"/>
  </jmeter>
  <jmeter jmeterhome="${basedir}/tools/jmeter"
        testplan="${basedir}/src/test/jmeter/gameoflife.jmx"
        resultlog="${basedir}/target/jmeter-results-1000-users.jtl">
    <jvmarg value="-Xmx512m" />
    <property name="request.threads" value="1000"/>
    <property name="request.loop" value="20"/>
  </jmeter>
</target>
```

166

请求帮助！我的测试运行得太慢了

设计你的 CI 构建时最根本的原则之一就是关于一个构建的失败次数随着时间快速减少的信息。换句话说，一个构建失败的消息越晚报告给你，它就越没有价值，并且越难以修复。

事实上，假如你的功能或者集成测试的运行花费了几个小时的时间，它们就不会运行每一个更改。它们更可能会被运行为一个夜间构建。这样做的一个问题就是，在 24 小时内什么都有可能发生；并且假如夜间构建失败了，这将很难指出到底应该归咎于在白天提交到版本控制系统里的哪些变更。这是个严重的问题，所以要求你的 CI 服务器能提供最快的反馈才是有帮助的。

当然有一些构建就是挺慢的，这是由于其本来的性质决定的。性能或者负载测试就属于这一类，因为它们是为一个大型项目做一些更重量级的代码质量度量构建。然而，大多数的集成或者功能测试肯定不属于这一类。你应该尽自己的最大努力来让这些测试尽可能快。对于一整个集成 / 功能测试集合来说，在 10 分钟以下可能是可接受的。两个小时就不可接受了。

所以，假如你发现自己需要加快测试，这里有几个策略可能会对你有帮助，其难度相近。

添加更多的硬件

有时候，能够加速构建的最简单方式就是添加更多的硬件设备。这跟你升级构建服务器一样简单。与验证和修复集成相关的 bug 花费的时间和努力相比，购买一个新的构建服务器的花费比较适合。

另一个选项是考虑使用虚拟化或者基于云的方式。在本书的后面，我们将会看到你如何能够使用 VMware 虚拟机或者基于云的基础设施比如 Amazon Web Services（EC2）或者 CloudBees 来提供构建能力，按需分配，而不是投入财力去买一些永久的新机器。

这种方式也可以把你的构建分布到几台服务器上。然而这种方式本身并不会加快测试，假如你的构建服务器处于沉重的需求之下，或者假如构建作业正处在不断排队的情况下，它可以加快反馈速度。

运行更少的集成 / 功能测试

在很多应用程序里，集成测试或者功能测试被默认用作一种标准方法，用于测试系统的几乎所有方面。然而集成或者功能测试却不是检测和验证 bug 的最好方式。由于在一个典型的端到端的测试中大量的组件会被牵扯到，所以很难去知晓到底哪里出了问题。另外，由于有那么多移动的部分，所以就算不是完全不可行的话，想要覆盖一个应用程序所有可能的路径，也是一件极其困难的事情。

正是由于这个原因，因此只要有可能，你就会更喜欢运行快的单元测试，而不是较慢的集成或者功能测试。当你有信心保证每个独立组件都能正常工作时，就可以通过少量端到端的测试用例或者那些在过去发生过问题的测试用例，来为这个系统完成用例的单步

调试。这样会帮助确保哪些组件能正确地组合；也就是，集成测试应做的事情。但是将这些更全面的测试尽可能地转向单元测试。这个策略可能是最可接受的方式来保持快速反馈迭代，不过这确实也要求一定的规则和工作量。

并行运行测试

假如你的功能测试花费了两个小时去运行，那就不太可能要求所有测试都一个接着一个地运行了。也不可能让它们占用你的构建机器上所有可用的 CPU 资源。所以，把你的集成测试分成更小的批次并且并行地运行它们是非常有意义的。

有几种策略可以尝试，你需要付出的工作量可能会取决于你的应用程序的本身情况。比如，其中一种方法就是配置几个构建作业来运行你功能测试里不同的子集，并且是并行运行这些作业的。Jenkins 可以帮助你汇总这些报告。充分利用一个分布式构建架构来进一步加速你的构建是一个很好的方法。这种策略的一个要点就是能够隔离地运行测试子集，这可能会要求一些重构。

从更低的级别上来说，你也可以在构建脚本级别并行地运行测试。就像我们先前看到的一样，TestNG 和最新版本的 JUnit 可以支持并行运行测试用例。不过，你需要确保能够同时运行你的测试用例，这可能会需要做一些重构。例如，在测试用例内的普通文件或者共享实例变量就会引起问题。

一般来说，你需要小心测试之间的交互。假如你的 Web 测试启动了一个内嵌的 Web 服务器（比如 Jetty），你需要确保使用的端口和其他正在运行着的测试所使用的端口不同。

不过，假如你能让你的应用程序实现并行测试，并行地运行你的测试用例是一种加快测试速度更加有效的方式。

小结

自动化测试在任何一个持续集成环境里都是非常关键的一部分，你应该严肃对待。就像 CI 的其他方面一样，快速反馈更是如此，它是最重要的，所以确保你的测试，尤其是集成测试或者功能测试快速运行是非常重要的。

Jenkins安全

简介

Jenkins 支持多种安全模型，可以与多个用户库集成。在小型组织机构中，开发人员都是在一起工作的，你的 Jenkins 机器安全可能不太重要——你可能只是想防止不明身份的用户篡改你的构建作业配置。对于有着众多团队的大型组织机构而言，需要更严格的方法来确保只有团队成员和系统管理员有权去修改其构建作业配置。Jenkins 服务器可能连接到如企业内部网站，甚至互联网类的外部环境，某些构建作业可能对所有用户可见，而另外一些对未授权的用户不可见。在本章中，我们将了解如何在 Jenkins 上为不同环境和情况设置不同的安全配置。

激活 Jenkins 安全

在 Jenkins 中配置安全是非常简单的。在主配置界面中选中 Enable security（启用安全）复选框（见图 7-1）。这里将显示很多选项，本章我们会对细节进行深入了解。第一部分，安全域（Security Realms），判断 Jenkins 从哪里验证用户，包括诸如使用存储在 LDAP 服务器的用户数据选项，或者是使用 UNIX 系统用户账号（当然前提是 Jenkins 运行在 UNIX 机器上）选项以及使用 Jenkins 管理的简单内置用户数据库的选项。

第二部分，授权策略（Authorization），决定用户登录之后可以做什么。策略范围包括“任何用户可以做任何事（Anyone can do anything）”（没有任何限制）或是“登录用户可以做任何事（Logged-in users can do anything）”，以及更复杂的角色和项目矩阵授权策略。

在本章其余部分，我们将学习针对常见场景进行 Jenkins 安全配置。

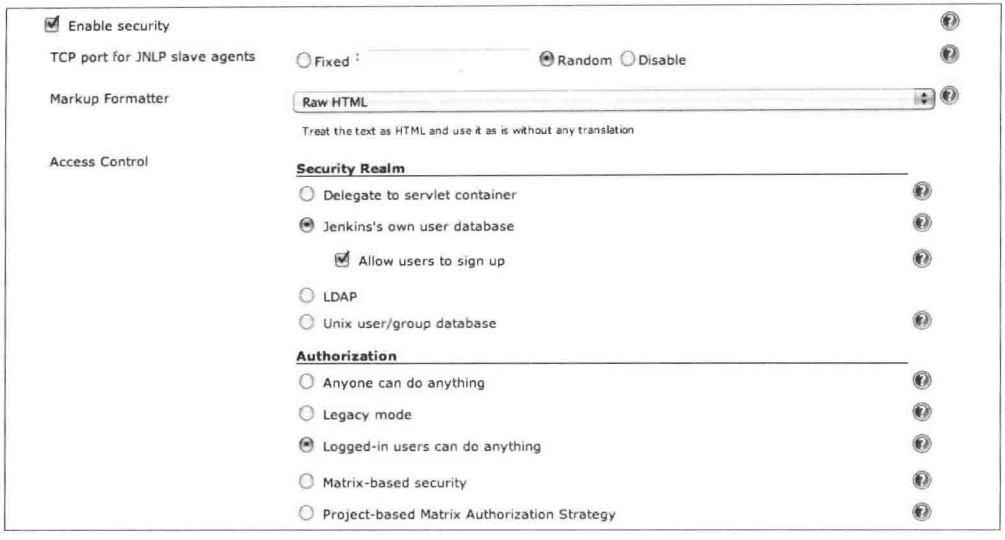


图7-1 启用Jenkins安全

Jenkins 安全实例

在 Jenkins 中最简单可用的安全模型是允许授权用户做任何事情，未授权用户只有对构建作业的只读权限。这对于小型团队很合适，开发人员可以管理构建作业，而其他用户（测试、需求、项目经理等）只能查看相关构建作业状态。事实上，一些构建作业可能正是为这种目的设置的，例如显示自动化验收测试结果或代码质量度量。

你可以在授权部分中选择“Logged-in users can do anything”这种配置。Jenkins 有几种给用户授权的方法（参见本章的“安全域——定义 Jenkins 用户”一节），但是对于这个示例，我们将使用最简单的选项，Jenkins 自带的数据库（参见本章的“使用 Jenkins 内置用户数据库”一节）。配置如图 7-1 所示。

请确保勾选“Allow users to sign up”选项。这个选项将在网页上方处显示一个注册链接来让用户创建他们需要的账号（见图 7-2）。这对于开发人员使用他们的 SCM 用户名是个好的想法：在这种情况下，Jenkins 将能够在用户提交了变更后触发对应的构建作业。

这种方法显然对于很多种情况来说有点太简单了——它对于一起办公的小团队是有用的，它的目标是：知道是谁提交的变更触发了相应的构建，而不是在管理权限方面加以严格的访问限制方式。在接下来的章节，我们将讨论 Jenkins 的安全矩阵：定义你的用户（安全域）和决定允许他们做什么（授权）。



图7-2 Jenkins注册界面

安全域——定义 Jenkins 用户

Jenkins 有多种方式定义和管理用户，从适合小型团队的一个简单、内置的用户数据库到集成企业级目录，以及许多其他的选择。

使用 Jenkins 内置用户数据库

在 Jenkins 里最简单的管理用户账号方法是使用 Jenkins 内置用户数据库。如果你想让事情变得简单，这是一个好的方式，它很少需要安装和配置。People（用户）需要先登录 Jenkins 服务器并给自己创建账号，然后基于安全模型进行选择，用户管理员可以决定允许这些用户做什么。

Jenkins 在监控到一个提交到源码的变更时自动添加所有的 SCM 用户进入这个数据库。这些用户名主要用于记录谁是每个构建作业的负责人。你能通过单击 People（用户）菜单项来查看当前已知用户列表（见图 7-3）。在这里，你可以看到 Jenkins 的用户，同时也能看到最近他们提交过的项目变更。注意这个列表包含 Jenkins 监控到的所有提交过变更的用户——他们可能不是（通常不是）登录 Jenkins 服务器的全部生效用户。



图7-3 Jenkins用户列表

如果你单击列表里的用户，Jenkins 会跳转到显示该用户的详细信息界面，包括用户的全名及其提交过的构建作业（见图 7-4）。从这里，你还可以修改和完善该用户的详细信息，如密码或者电子邮件地址。

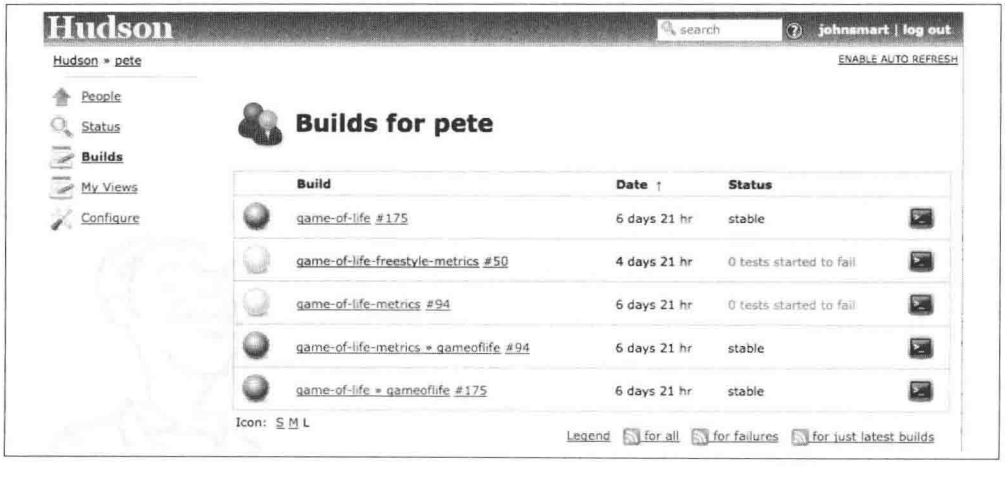


图7-4 用户参与的构建列表

在这个列表中出现的用户不见得就能登录进 Jenkins 系统。要想登录进 Jenkins 系统，用户账户需要设置一个密码。主要有两种方法，如果已经激活了 Allow users to sign up

（允许用户注册）选项，则用户可以方便地用他们的SCM用户名、他们的邮箱地址和密码注册（参见本章的“Jenkins安全实例”一节）。或者，你可以通过用户详情页面的Configure（配置）菜单项，提供一个邮件地址和密码来激活用户（见图7-5）。 173

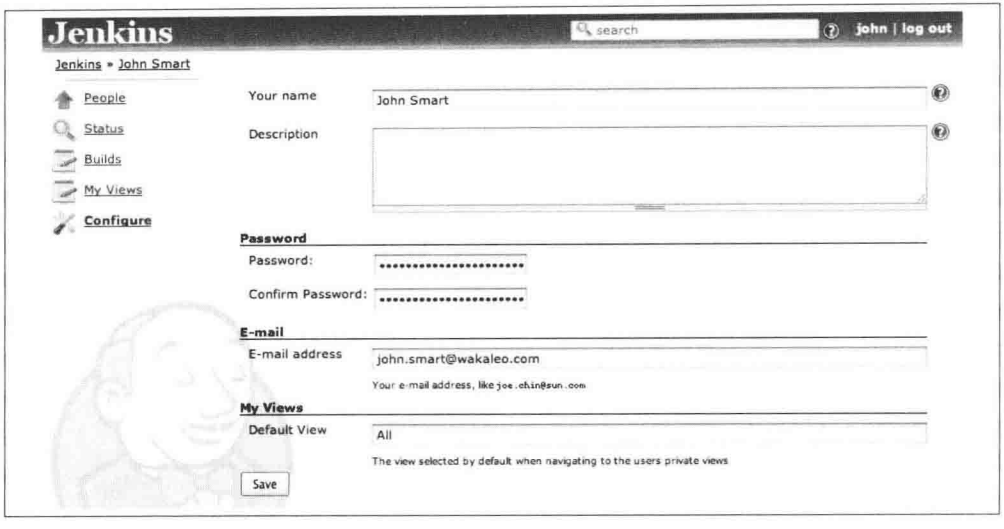


图7-5 通过注册页面创建用户账号

值得注意的是，如果你的电子邮件地址与你的版本控制用户名一样（例如，如果你在acme.com工作，用户“Joe”在版本控制系统有一个电子邮件地址*joe@acme.com*），Jenkins会将配置在电子邮件通知部分（Email Notification，见图7-6）的后缀添加到用户名后成为电子邮件地址。如果你已经做了这类配置，则无须指定新用户的电子邮件地址，除非不符合这个情况。

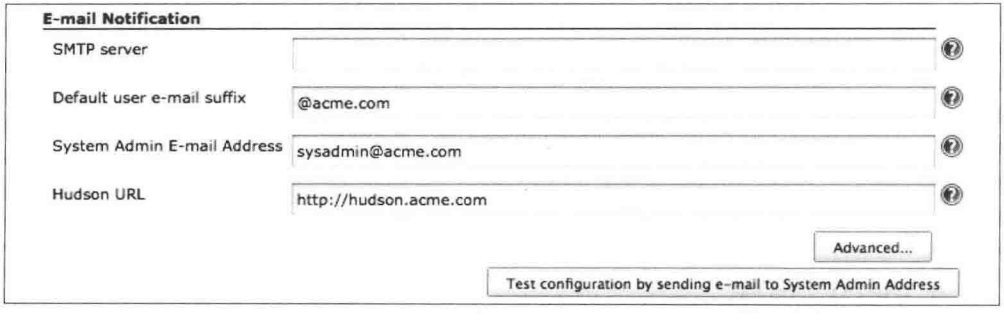


图7-6 同步邮件地址

另一种管理当前激活的用户（那些能真正登录Jenkins的用户）的方式是单击Jenkins配置界面的Manage Users链接（见图7-7）。 174

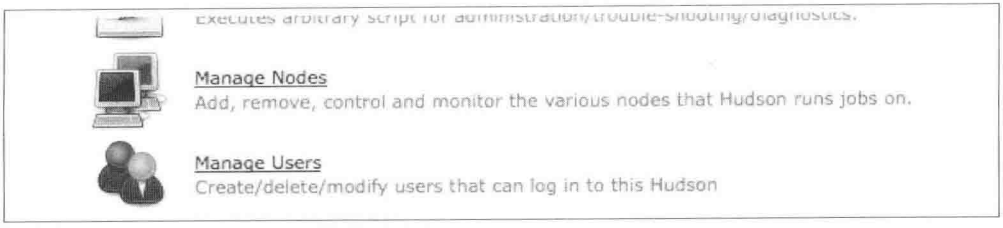


图7-7　你可以在Jenkins配置界面管理Jenkins用户

从这里，你可以查看和编辑能登录进 Jenkins 的用户（见图 7-8）。这包括用户自己手动注册（如果这个选项被激活）和通过设置密码激活的 SCM 用户。你还能编辑用户的详细信息（例如，修改他们的电子邮件地址或是重置密码），甚至是从激活用户列表中删除它们。这样做不会真正把这些账号从用户列表删除（例如这些用户名还会出现在构建历史中），但是他们将不能够登录 Jenkins 服务器。

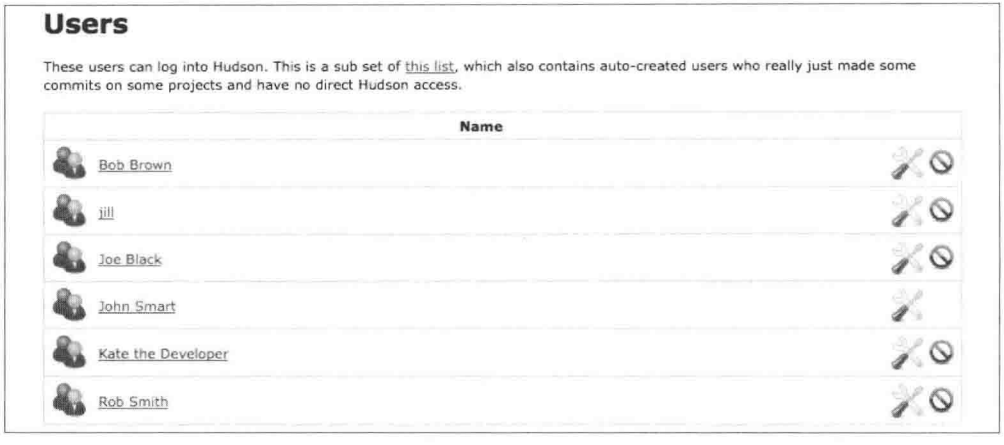


图7-8　Jenkins用户数据库

Jenkins 专有数据库对于许多团队和组织是够用的。然而对于更大型的组织，用它去手动管理大量的用户账号可能变得乏味且重复（特别是当这些信息已经存在时）。下面，我们将看看如何让 Jenkins 连到其他用户管理系统，如 LDAP 仓库及 UNIX 用户和用户组。

175

使用 LDAP 仓库

许多机构使用 LDAP 目录来存储各应用程序用户账户和密码。Jenkins 很好地集成了 LDAP，无须特殊的插件。它可以使用 LDAP 仓库对用户进行身份认证，检查组成员和获取授权用户的电子邮件地址。

使 Jenkins 和你的 LDAP 仓库集成，只需在安全域（Security Realm）部分选择“LDAP”，然后填入相应 LDAP 服务器的详细信息（见图 7-9）。最重要的输入项是仓库服务器。如果你使用一个非标准的端口，则需要提供此端口（例如，*ldap.acme.org:1389*）。或者如果你正在使用 LDAPS，则需要指定它（例如，*ldaps://ldap.acme.org*）。

如果你的服务器支持匿名绑定，以上方法足够你使用。如果不支持匿名绑定，则可以使用高级（Advanced）选项来调整你的配置。

大部分高级选项是可以留白的（除非你有充分的理由去修改它们）。如果你的仓库是非常大的，则你可能希望指定一个基准 DN 值（如：`dc=acme，dc=com`），又或者是一个用户、用户组搜索基准规则（如：`ou=people`）来缩小查询用户的范围。这通常是你发现性能问题时才需要用到的。或者，如果你的服务器不支持匿名绑定，你需要提供一个 DN 管理员和密码，这样 Jenkins 可以连接到服务器执行查询。

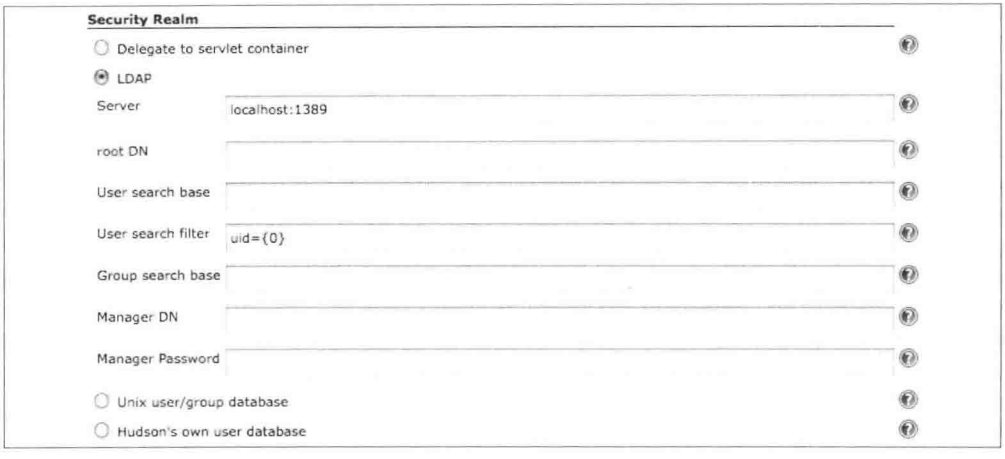


图7-9　Jenkins中的LDAP配置

一旦你设置 LDAP 作为安全域，则可以配置你喜欢的如前所述的安全模型。当用户登录 Jenkins 时，他们将通过 LDAP 仓库进行权限验证。

你还可以使用 LDAP 群组，虽然这个配置项不是显而易见的。假定你在 LDAP 仓库里定义了叫作 JenkinsAdmin 的组，并且 DN 是 `cn=JenkinsAdmin，ou-Groups，dc=acme，dc=com` 的组。使这个组关联到 Jenkins 中，你需要使用大写的公共名称（cn）和带 `ROLE_` 的前缀。所以，`cn=JenkinsAdmin` 变为 `ROLE_JENKINSADMIN`。你能在图 7-10 中看到一个使用这种 LDAP 群组方式的例子。

176

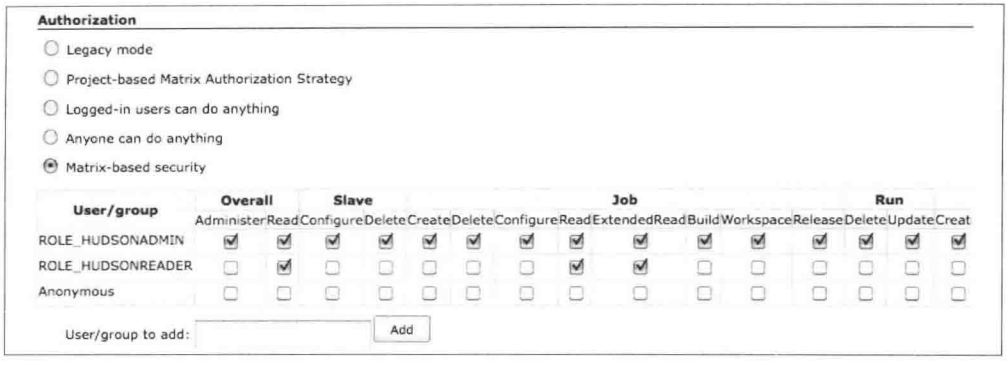


图7-10　在Jenkins中使用LDAP组

使用微软活动目录

微软活动目录是一个广泛用于微软架构的目录服务产品。尽管活动目录（Active Directory）提供LDAP服务，搭建它稍微有一点复杂，但是用Jenkins去对接活动目录服务器会更简单些。幸运的是它有个插件。

Jenkins Active Directory插件能够让你配置Jenkins对微软活动目录进行验证。你可以验证用户身份，还可以从项目矩阵授权策略中检索他们的组。注意，不像传统LDAP集成（参见本章的“使用LDAP仓库”一节），这里不需要带有`ROLE_`的组名前缀——你可以使用活动目录组（如“域管理员”/Domain Admins）目录。

若配置这个插件，你需要提供完整的活动目录服务器域名。如果你有多个域，则可以提供用逗号分隔的列表。如果你提供主域（如acme.com而不是europe.acme.com），则搜索将在全局目录进行。注意，如果你不指定绑定的DN（见下文），则用户必须用类似“europe\joe”或者“joe@europe”来登录。

高级选项允许你指定一个网站名字（通过限制域名控制器来提高Jenkins查询性能）、绑定的DN和密码，当你连接到多个域时，这可以派上用场。你需要提供一个有效的绑定DN和密码，Jenkins可以用它来链接你的服务器，从而建立完整的用户身份认证。这种方式，用户可以简单地输入“jack”或者“jill”，系统会自动拼出*jack@europe.acme.com*或*jack@asia.acme.com*。你需要提供完整的用户名和域名，如*admin@europe.acme.com*；或LDAP形式的专有名称，如`CN=Administrator, OU=europe, DC=acme, DC=com`。

177

这个插件的另一个优点是它对Windows、UNIX环境都兼容。如果Jenkins运行在UNIX服务器上，它仍然可以使用运行在另一台机器上的微软活动目录服务来进行身份认证。

更准确地说，如果 Jenkins 正运行在 Windows 机器上而且你没有指定一个域名，那么负责授权的这台机器必须是域里的一个成员。Jenkins 会通过 ADSI 查出所有的详细信息，所以不需要任何额外的配置。

在一个非 Windows 机器上（或者你指定一个或多个域），你需要告诉 Jenkins 进行身份认证的活动目录域的名称。之后 Jenkins 会使用 DNS SRV 记录和 LDAP 活动目录服务来进行用户认证。

Jenkins 可以判断用户属于哪个活动目录组，所以你可以使用这些作为授权策略的一部分。例如，你可以在安全矩阵中使用这些组，或是允许“域管理员”管理 Jenkins。

使用 UNIX 用户和用户组

如果你的 Jenkins 正运行在 UNIX 机器上，则还可以让 Jenkins 使用机器上定义的用户和用户组。在这种情况下，用户将使用 UNIX 账号和密码登录 Jenkins。这里使用身份认证模块（Pluggable Authentication Modules，PAM），还可以和 NIS 集成工作。

这是最基本的形式，但有些麻烦，因为它需要为每一个 Jenkins 新用户创建并设置新的用户账号。如果这些 UNIX 账号是为其他用途创建设置的，则才是真的有用处。

Servlet 容器授权

另一种识别 Jenkins 用户的方式是让你的 Servlet 容器为你做这些事。当你的 Jenkins 正运行在 Servlet 容器上时，如 Tomcat 或者 GlassFish，并且你已经有确定的方案将 Servlet 容器和你本地企业用户目录进行集成，这种方式是有用处的。例如，Tomcat 允许你使用关系型数据库（使用 JDBC 或数据源）、JNDI、JAAS 或一个 XML 配置文件进行用户身份认证。你还可以使用在 Servlet 容器中的用户目录定义的角色进行安全矩阵的授权策略设置。

在 Jenkins 中，这是容易配置的——只需选择 Security Realm 部分的选项即可（见图 7-11）。当你做完后，Jenkins 会让服务器处理一切。 178

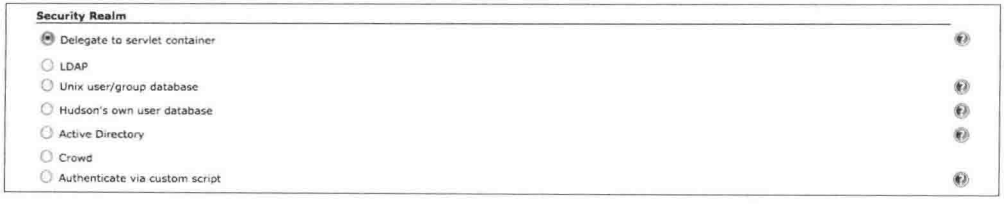


图7-11　选择安全域

使用 Atlassian Crowd

如果你的企业使用 Atlassian 产品，如 JIRA 和 Confluence，则你也可能正在用 Crowd。Crowd 是一个来自 Atlassian 的商用身份管理（Identity Management）和单点登录（Single-Sign On，SSO）的应用，可以让你管理多个系统间公用的账号。你既可以用它管理一个内部的用户、用户组和角色的数据库，也可以管理外部集成的目录，如 LDAP 目录或者用户自定义存储。

使用 Jenkins Crowd 插件，你能使用 Atlassian Crowd 作为 Jenkins 用户和用户组的数据源。在开始之前，你需要在 Crowd 中设置一个新的应用（见图 7-12）。只需新建一个叫“hudson”的（或类似的）普通应用，之后根据选项卡的步骤执行。在 Connection 选项卡中，你需要提供 Jenkins 服务器的 IP 地址，然后将检索 Jenkins 用户账号和组信息映射到 Crowd 目录中，最后，你需要在 Crowd 中设置哪些来自目录的用户能连接到 Jenkins。一个选择是允许授权所有用户，并让 Jenkins 来辨别细节；另一种方式是，你可以列出并设置允许连接到 Jenkins 的那些 Crown 用户组。

179

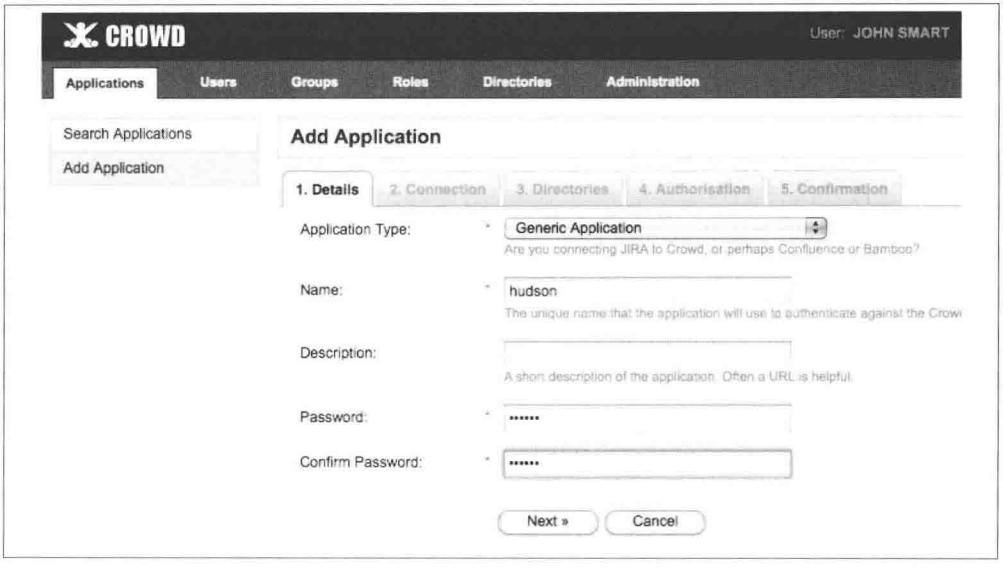


图7-12　使用Atlassian Crowd作为Jenkins安全域（一）

一旦你设置完了，就需要安装 Jenkin Crowd 插件，就像往常一样通过 Jenkins 插件管理器进行安装。一旦你安装好了插件并重启了 Jenkins，则可以在 Jenkins 主配置界面设置 Crowd 作为你的安全域（见图 7-13）。

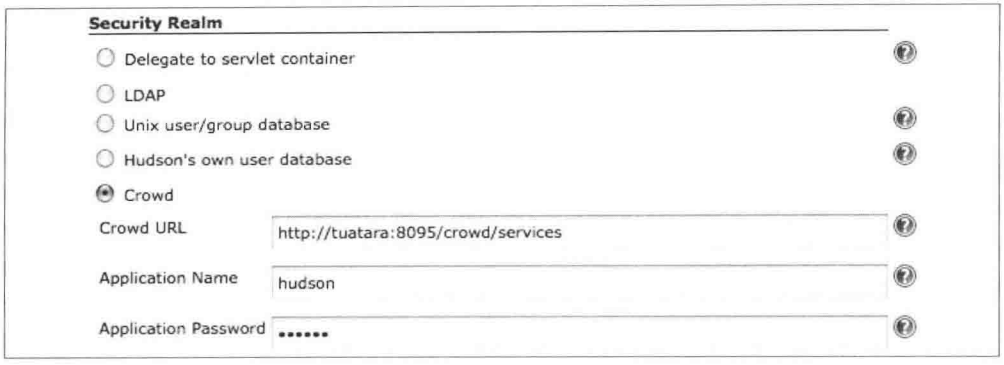


图7-13　使用Atlassian Crowd作为Jenkins安全域（二）

通过对这个插件进行安装和配置，你可以使用来自 Crowd 的用户和用户组配置 Jenkins 授权机制，比如我们在之前章节讨论过的任何授权机制。例如，在图 7-14 中，我们正在主配置界面使用定义在 Crowd 中的用户组来配置安全矩阵。

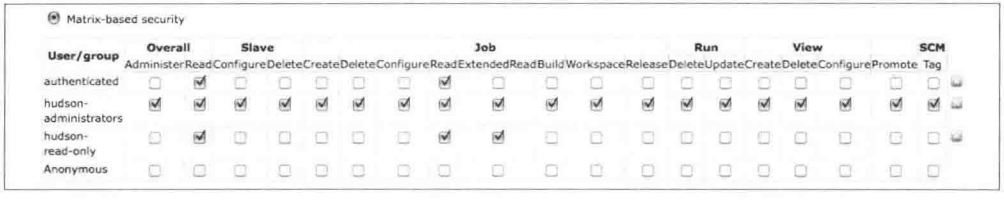


图7-14　在Jenkins中使用Atlassian Crowd组

与其他系统集成

除了这里讨论的身份认证策略外，还有许多其他的插件允许 Jenkins 通过其他系统进行用户身份认证。在本书写作之际，这些包括中央认证服务（CAS）——一个开源的单点登录工具和 Collabnet Source Forge 企业版（SFEE）服务器。

如果没有可用的插件，你可以编写自定义身份认证脚本。为此你需要安装脚本安全域（Script Security Realm）插件。当你安装完并重启 Jenkins 后，可以用你最喜欢的脚本语言写两个脚本。一个脚本验证用户，另一个脚本通过给定的用户判断所属用户组（见图 7-15）。

180

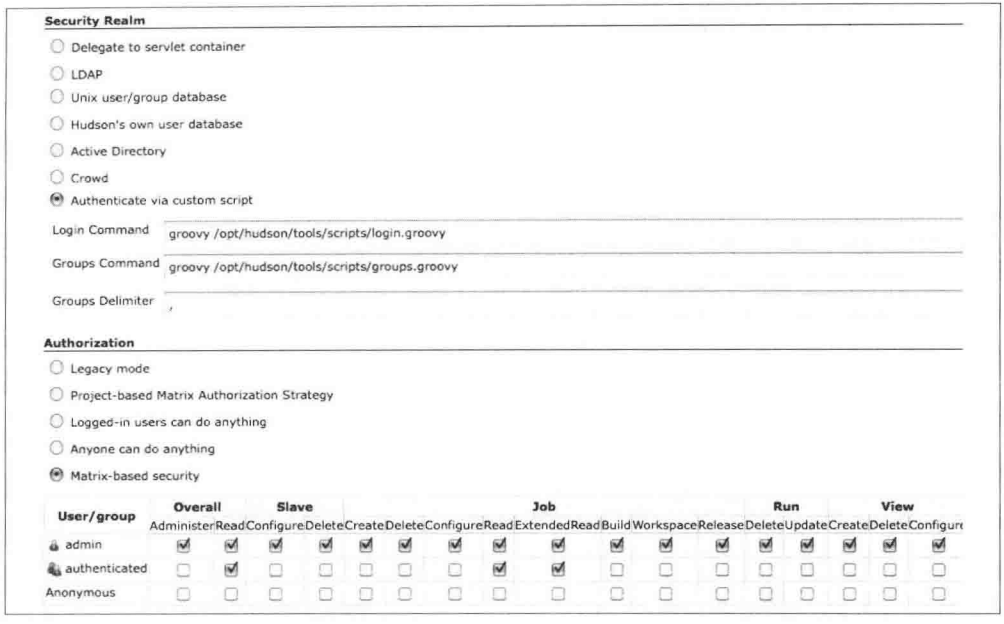


图7-15　使用自定义脚本处理授权

在调用身份认证脚本之前，在Jenkins中设置两个环境变量：U，包含用户名；P，包含密码。这个脚本使用这些环境变量来对指定用户名和密码进行身份认证。如果认证成功则返回0，其他值代表不成功。如果认证失败，处理过程的错误信息报告会报告给用户。这是一个简单的Groovy认证脚本：

```
def env = System.getenv()
def username = env['U']
def password = env['P']

println "Authenticating user $username"

if (authenticate(username, password)) {
181       System.exit 0
} else {
      System.exit 1
}
def authenticate(def username, def password) {
    def userIsAuthenticated = true
    // Authentication logic goes here
    return userIsAuthenticated
}
```

如果你要处理没有分组的基本身份认证，这个脚本就足够了。如果你想在基于矩阵或基于项目的身份认证上使用自定义授权组（参见本章下面的“授权——谁可以做什么”），则可以再写一个脚本来判断一个给定的用户是哪个组的。这个组使用环境变量 U 来判断哪些用户正在登录，并打印一个以逗号分隔的用户组清单。如果你不喜欢逗号，则可以在配置中重写分隔符。一个简单的可以执行这个工作的 Groovy 脚本如下：

```
def env = System.getenv()
def username = env['U']

println findGroupsFor(username)

System.exit 0

def findGroupsFor(def username) {
    return "admin,game-of-life-developer"
}
```

当调用这些脚本进行用户身份认证后，必须返回 0。

授权——谁可以做什么

一旦定义了如何识别用户，你需要决定允许他们做什么。Jenkins 支持多种策略；从简单的登录用户可以做任何事的方式，到更复杂的基于角色、项目的授权策略。

安全矩阵

允许登录的用户做任何事当然是挺灵活的，但可能其仅适用于小团队。对于大型团队和多团队协作的情况，当 Jenkins 在开发环境之外使用时，通常需要更复杂的方法。

安全矩阵是一种更精细的方式，基于角色为不同的用户分派不同的权限。

安全矩阵搭建

182

在 Jenkins 中配置安全矩阵的第一步是创建一个管理员。这是一个重要的步骤，必须在其他步骤之前做完。你可以用一个存在的账户作为系统管理员，或专门为此创建一个账户。如果你想创建一个专门的管理员用户，就像通常情况下那样注册一个用户（见图 7-2）。它没有和 SCM 用户进行关联。

一旦准备好管理员账户，你可以在主配置界面的 Authorization 部分激活“安全矩阵”（Matrix-based security）。Jenkins 会显示一个包括授权用户及你可以分派给这些用户的各种权限复选框的表格（见图 7-16）。

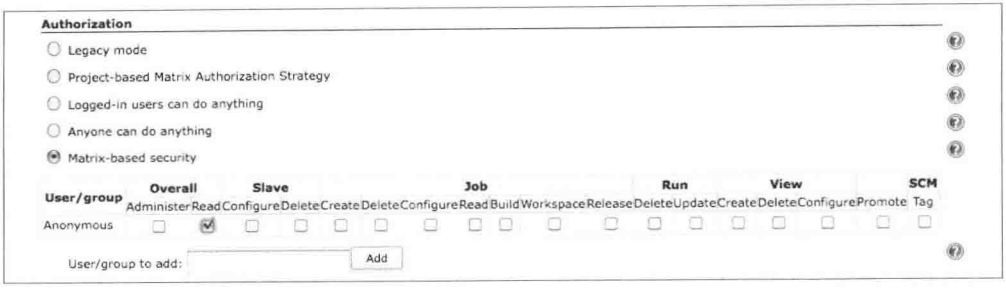


图7-16　安全矩阵配置

这个“anonymous”用户总是存在的。该用户代表未授权用户。通常情况下，你只给未授权用户很有限的权限，如只读权限，或根本没有权限（见图 7-16）。

现在你需要做的第一件事就是给你的管理员分派管理员权限。单击“User/group to add”处的“Add”按钮添加你的管理员用户。你的管理员将立刻出现在权限矩阵中。现在请确保对这个用户分派了每个权限（见图 7-17），然后保存你的配置。你现在可以登录自己的管理员账号（如果还没登录这个账号）并且可以配置其他用户。

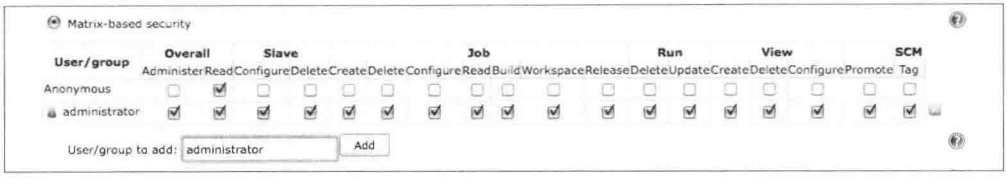


图7-17　设置管理员

183 用户权限调整

当你设置好管理员权限后，可以添加需要访问 Jenkins 实例的其他用户。只需添加用户名和选择需要分派的权限即可（见图 7-18）。如果你正用 LDAP 服务器或者 UNIX 用户及用户组作为身份认证模式（参见本章的“使用 LDAP 仓库”一节），则还可以为用户组配置权限。



图7-18　设置其他用户

你可以授予的权限，由几个组组成：Overall、Slave、Job、Run、View 和 SCM。大多数权限是很直观的，但是有些需要一些解释。各个权限如下所示。

Overall

这个组覆盖系统级基础的权限。

Administer（管理员）

允许用户进行系统级的配置变更操作和其他敏感操作，例如在 Jenkins 配置主界面的操作。这些应该只能让 Jenkins 管理员操作。

Read（只读）

这个权限提供了 Jenkins 所有可视化界面的只读访问。如果你想匿名查看构建作业，而不是去修改或启动它们，可以给匿名用户分派只读权限。如果不是这样的话，则撤回匿名用户的权限。如果你想让所有的用户都能查看构建作业，那么添加一个叫作“authenticated”的用户，并授予其 Overall/Read 权限。

Slave

这个组包括远程构建节点或从节点的权限。

Configure（配置）

创建和配置新节点。

Delete（删除）

删除构建节点。

Job

184

这个组包括作业相关的权限。

Create（创建）

创建新的构建作业。

Delete（删除）

删除一个存在的构建作业。

Configure（配置）

更新一个存在的构建作业配置。

Read（只读）

查看构建作业。

Build（构建）

启动一个构建作业。

Workspace（工作空间）

查看、下载构建作业相关的工作空间内容。工作空间包含源码和构建产品，所以如果你想从普通访问权限上做保护，则需要撤销这个权限。

Release（发布）

启动一个使用 M2Release 插件配置的 Maven 项目的发布。

Run

这个组包括构建历史的相关权限。

Delete（删除）

从一个构建历史中删除一个构建。

Update（更新）

在构建历史中更新构建的描述和其他属性。例如当用户想留下一段关于构建失败原因的笔记，这是很有用处的。

View

这个组包括管理视图。

Create（创建）

创建一个新视图。

Delete（删除）

删除一个存在的视图。

Configure（配置）

配置一个存在的视图。

SCM

关于你的版本控制系统的权限。

Tag

为一个给定的构建，在源码库中创建一个新的标签。

Others（其他）

还可以有其他可用的权限，这主要依赖于安装的插件。

其中一个常用的是：

Promote

如果安装了 Promoted Builds 插件，则这个权限允许用户手动做构建。

求助！我登不进去了

这种情况可能发生在你把自己锁在了 Jenkins 外。例如，你保存了错误的管理员安全矩阵配置。如果发生这样的事情，不要慌张——这里有个简单的解决方法，只需要 Jenkins 的主目录的访问权限。只需打开位于 Jenkins 主目录下的 *config.xml* 文件。里面包含如下的一些内容：

```
<hudson>
    <version>1.391</version>
    <numExecutors>2</numExecutors>
    <mode>NORMAL</mode>
    <useSecurity>true</useSecurity>
    ...
```

找到 `<useSecurity>` 元素，修改值为 false 并进行重启来恢复你对 Jenkins 的访问。你现在又可以访问 Jenkins 了，然后修正你的安全配置。

基于项目的安全

基于项目的安全可构建于我们讨论过的安全矩阵模型之上，并分别应用于各个项目。不止可以给你的用户分派系统级的权限，你还可以单独为项目配置更特殊的权限。

激活项目级别安全，在主配置界面的 Authorization 部分选择“Project-based Matrix Authorization Strategy”（见图 7-19）。在这里，你可以为用户和用户组设置默认权限，就像我们在安全矩阵中看到的那样（参见本章的“安全矩阵”一节）。

所有项目的默认权限无须特别配置。而且，当采用基于项目的安全时，还可以设置特殊的项目特定权限。需在项目配置界面选中“Enable project-based security”（激活基于项目的安全，见图 7-20）。Jenkins 将显示一个项目的特定权限的表格。你可以像在系统级配置页面一样为不同的用户及用户组配置这些权限。这些权限会被添加到系统级权限并产生适合于这些项目的特定项目级权限集合。

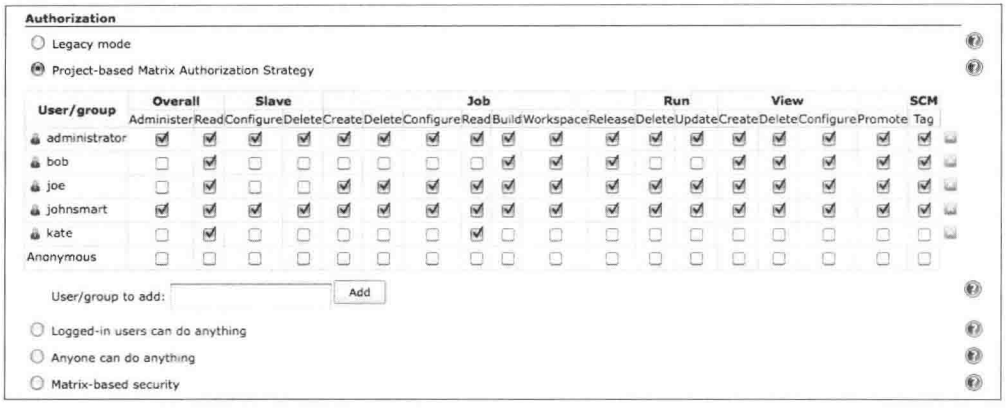


图7-19　基于项目的安全策略

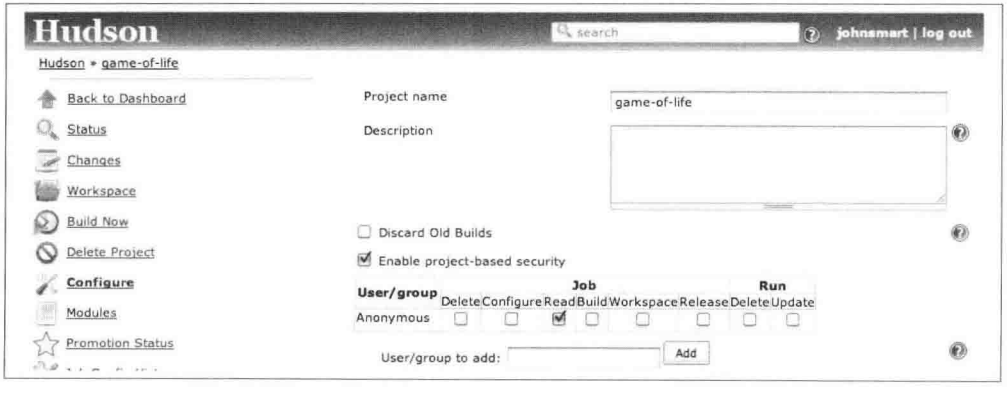


图7-20　配置基于项目的安全策略

用一些实际例子是最容易理解这种工作方法的。例如，在图 7-19 中，没有为匿名用户授予权限，默认情况下所有的构建作业将是只对登录用户可见。然而，我们使用基于项目

的安全，可以逐个地进行覆盖。又比如在图 7-20 中，我们可以为“anonymous”用户设置对 *game-of-life* 项目具有只读权限。

当你保存这个设置时，未授权用户将可以以只读模式来查看 *game-of-life* 项目（见图 7-21），这同样适用于所有的特定项目授权。

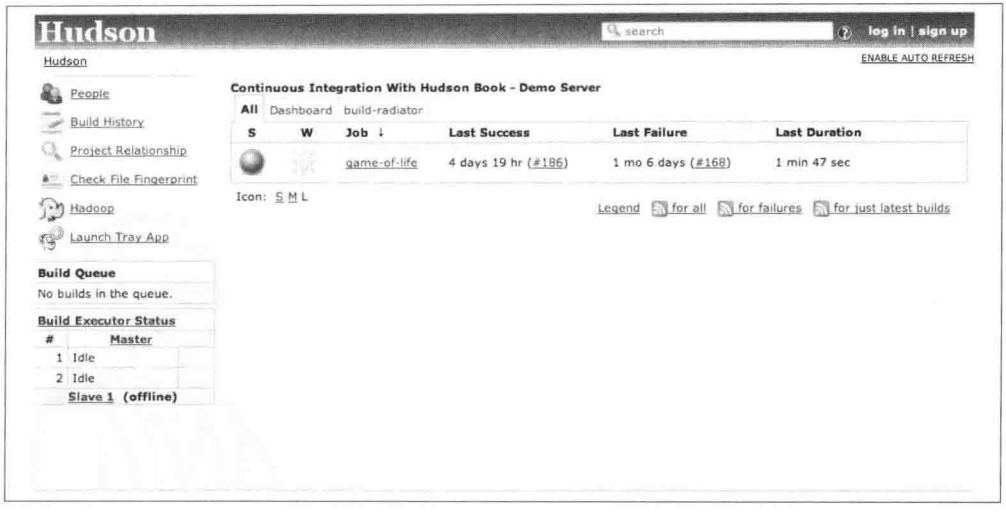


图7-21　查看项目

注意，Jenkins 权限是累积的——在本书写作之际，没有为一个特定项目撤回一个系统级的权限方法。例如，如果是一个匿名用户在系统级别有对构建作业的只读访问权限时，你不能对某个项目撤回这个只读权限。所以当使用基于项目的安全时，使用系统级矩阵来为所有项目定义最小的默认权限，并为项目设置项目级特定授权。

管理项目权限有很多方法，就像基于技术的考虑一样而更多取决于组织文化，一个常见的策略方法是允许团队成员对其拥有的项目有全部的权限，并且对其他项目有只读权限。Extended Read Permission 插件对这个场景很有用处。这个插件允许你让其他团队的用户以只读方式查看你的项目配置，无须修改任何配置（见图 7-22）。这对于与其他团队分享构建配置实践及技巧而又不允许他们篡改你的构建是一个非常棒的方法。

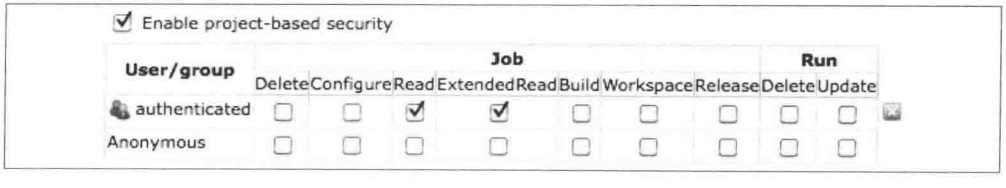


图7-22　设置扩展只读权限

值得注意的是，当涉及大型及多团队协作的情况时，Jenkins内置数据会很快到达它的极限，集成更专业的目录服务是值得考虑的事情，如LDAP服务器、活动目录或是Atlassian Crowd，以及像基于角色安全的更加复杂的权限系统（我们会在下一节讨论）。

188

基于角色的安全

有时单独管理用户权限是低效的，你可能也不想用集成LDAP服务器进行组设置的方法。最接近的替代选择是使用Role Strategy插件，可以允许你定义全局的和项目级的角色，并给用户分派这些角色。

在Plugin Manager中，像之前一样安装这个插件。安装完后，你可以在主配置界面激活授权策略（见图7-23）。



图7-23　设置基于角色的安全

一旦设置好，就可以定义重组相关权限集合的角色。在Manage Jenkins里的Manage Roles界面建立并配置各角色，将这些角色分派给你的用户（见图7-24）。

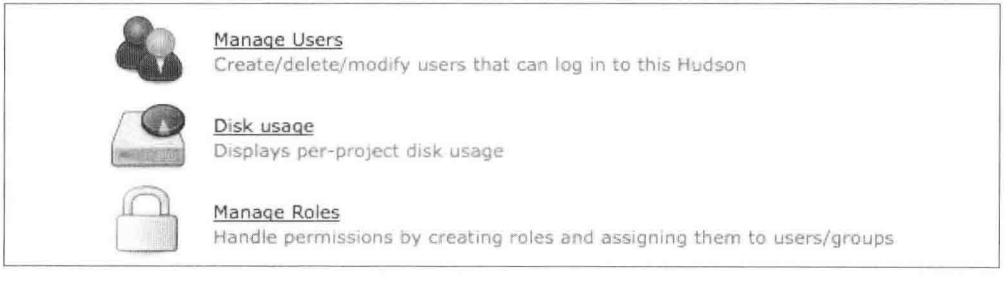


图7-24　Manage Roles配置菜单

在Manage Roles界面，你可以建立全局和项目级的权限。全局权限应用于所有项目，以及典型的系统级管理或通常的访问权限（见图7-25）。角色安全设置与其他我们见过的用户权限设置的安全模型是相似的。

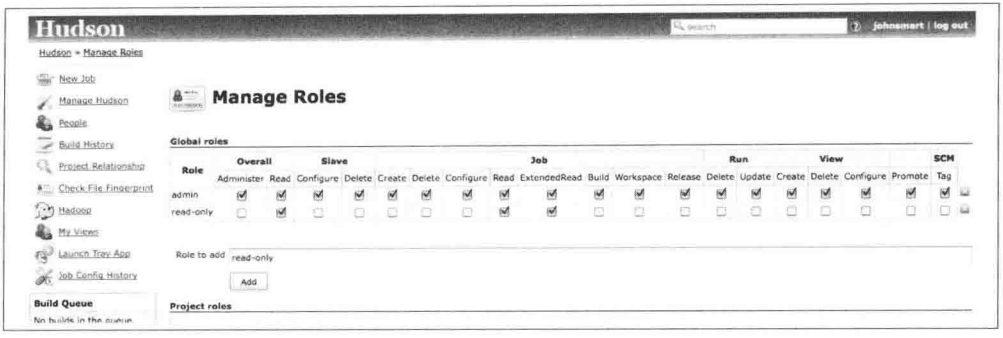


图7-25　管理全局角色

项目角色则稍微复杂些。一个项目角色重组权限集是适用于一个或多个（可能相关）项目的。你使用正则表达式来定义相关项目，因此它有助于形成清晰且统一的项目名称规则（见图 7-26）。例如，你可能希望创建区分开发人员和普通用户的角色，开发人员拥有全部的配置权力，而普通用户则只能启动构建及构建结果；或是开发人员可以配置特定的自动化部署构建作业，但生产团队仅可以执行那些作业。

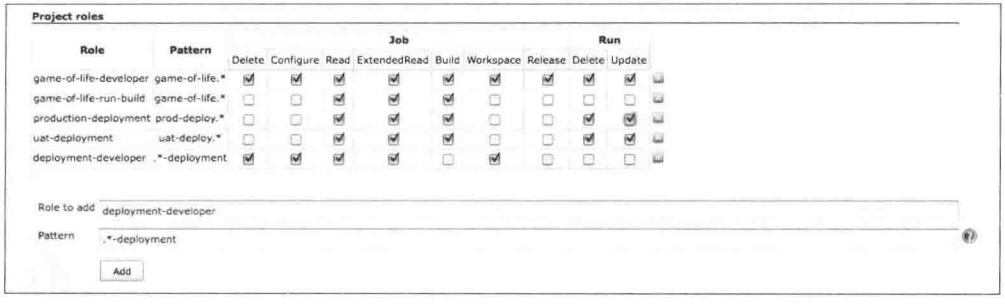


图7-26　管理项目角色

当你定义完这些角色时，可以去 Assign Roles（角色分派）界面为每个用户及用户组设置这些角色（见图 7-27）。

角色级策略是 Jenkins 中相对新的特性，但是它是一个很好的方法来简化大型团队、多团队、多项目机构的管理任务。

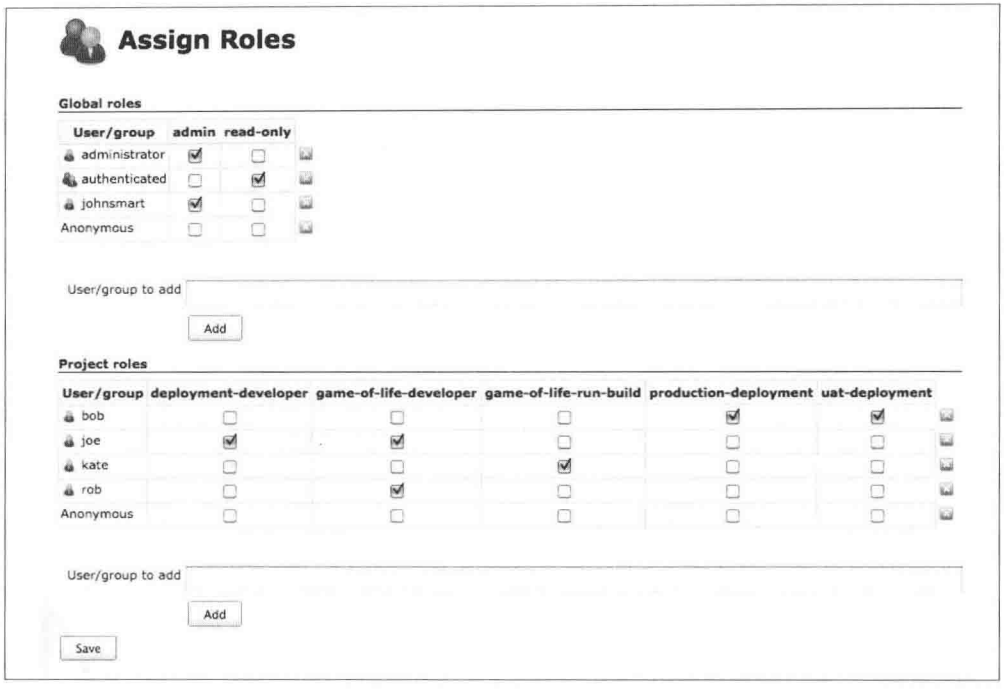


图7-27　为用户分派角色

审计——跟踪用户行为

除了配置用户账号和访问权限，也可以用于跟踪个人用户行为：换句话说，谁对你的服务器配置做了什么。在许多机构中都需要这样的跟踪机制。

有两个 Jenkins 插件可以做这件事。审计跟踪（Audit Trail）插件会在指定的日志文件中保存用户变更的记录。并且 JobConfigHistory 插件可以保存一份 Jenkins 使用的前版本各种系统和作业配置文件副本。

审计跟踪插件跟踪的主要用户行为是一组滚动的日志文件。要设置它，需到 Plugin Manager 页面并选择有效插件列表里的审计跟踪插件，然后像往常一样，等插件下载完后单击安装并且重启 Jenkins。

你可以在 Jenkins 主配置页面的 Audit Trail 部分进行配置（见图 7-28）。主要的填写项是日志路径（Log Location），用来指明日志文件要写到哪个路径下。审计跟踪插件的目的是生成系统样式日志文件，通常被放到一个指定的系统目录，如 */var/log*。你也可以配置维护的日志文件数量，及每个文件的最大物理大小（近似）。最简单的方式是提供一个

绝对路径（如 */var/log/hudson.log*），这种情况下，Jenkins 写的日志文件名字像 */var/log/hudson.log.1*、*/var/log/hudson.log.2* 等这样。当然，你需要确保用户运行的 Jenkins 实例具备往目录里进行写操作的权限。

你还可以使用 Java 日志 FileHandler（*http://download.oracle.com/javase/1.5.0/docs/api/java/util/logging/FileHandler.html*）类中定义的格式来控制生成的日志文件。对于这种格式，你可以插入变量，如 `%h`，用于当前用户的主目录，以及 `%t`，用于系统临时目录，来进行动态的文件路径设置。

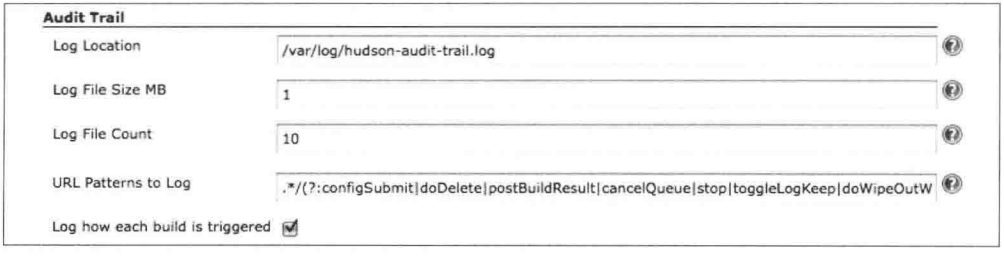


图7-28　配置Audit Trail插件

默认情况下，细节记录在审计日志是非常分散的——它们记录关键的执行操作，如创建、修改、查看、删除作业配置，以及哪个用户执行的该操作，日志还可以展示每个构建作业的起始情况。以下是一个默认日志的摘录：

```
Dec 27, 2010 9:16:08 AM /job/game-of-life/configSubmit by johnsmart
Dec 27, 2010 9:16:42 AM /view/All/createItem by johnsmart
Dec 27, 2010 9:16:57 AM /job/game-of-life-prod-deployment/doDelete by johnsmart
Dec 27, 2010 9:24:38 AM job/game-of-life/ #177 Started by user johnsmart
Dec 27, 2010 9:25:57 AM job/game-of-life-acceptance-tests/ #107 Started by upstream
    project "game-of-life" build number 177
Dec 27, 2010 9:25:58 AM job/game-of-life-functional-tests/ #7 Started by upstream
    project "game-of-life" build number 177
Dec 27, 2010 9:28:15 AM /configSubmit by johnsmart
```

这个审计跟踪是非常有用的，从系统管理员角度看更是如此。但是，它没有提供任何关于我们对 Jenkins 配置进行变更的信息。尽管如此，最重要的原因是追踪 Jenkins 中的用户行为，从而精确地监视构建作业配置被做了什么修改。当出现问题时，可以知道什么配置被修改了和是否可以撤销。JobConfigHistory 插件可以帮你做到这一点。

JobConfigHistory 插件是一个强大的工具，可以记录包括作业和系统配置的全部变更历史。通常在 Plugin Manager 中进行安装，当安装好后，你可以在 Manage Jenkins 界面进行作业历史配置的调整（见图 7-29）。

在这里，你可以配置很多非标准的有用选项。特别是你应该在“Root history folder”输入框中指定一个 Jenkins 用于存储配置历史的目录。这个目录将存储系统关联及作业关联的配置变更记录。它可以是个绝对路径（如 */var/hudson/history*），或是由 Jenkins 的主目录得出的相对路径（参见第 3 章的“Jenkins 主目录”一节）。如果你不做这些，则作业配置历史将和作业存在一起，如果你删除了作业，那么历史数据也会被删除。

192

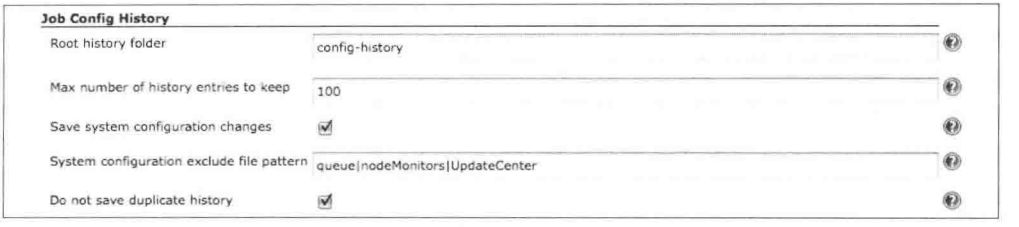


图7-29　设置Job Configuration History插件

有一些其他有用的设置项在 Advanced（高级）部分。勾选“Save system configuration changes”（保存系统配置变更）可以保持跟踪整个系统范围的配置变化，而不仅仅是具体的某个作业的相关配置变化，“Do not save duplicate history”（不保存重复历史）复选框可以避免记录没有实际变化造成的配置更新。如果不是这样的话，将记录下来一个新的配置版本（即使你只是单击了保存按钮而没有做任何修改）。Jenkins 内部也会导致这种情况——例如，当主配置界面保存时，系统配置设置也做了保存（即使没有更改任何配置）。

一旦你配置了这个插件，就可以访问整个服务器的配置历史，包括系统配置更新及每个项目的配置变更。在这两种情况下，你可以通过单击作业配置历史（Job Config History）图标，在右侧界面看到这些变更。单击 Jenkins 主控面板的图标将显示所有配置历史视图，包括作业变更和系统级变更（见图 7-30）。

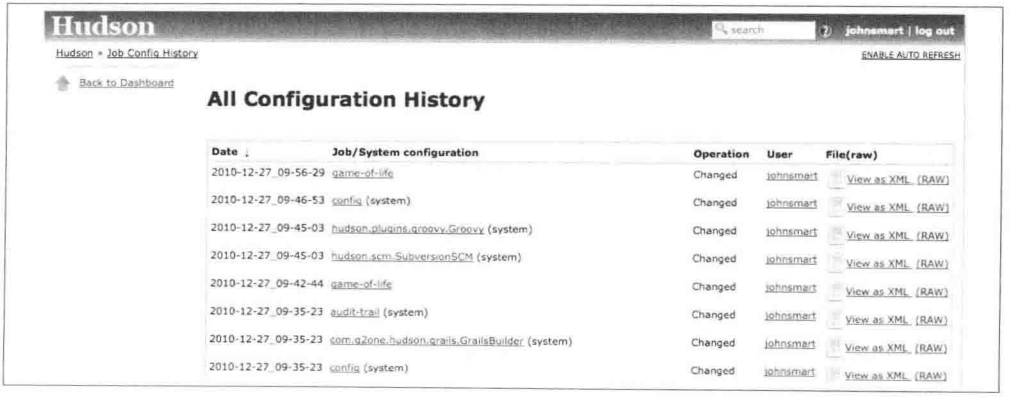


图7-30　查看配置历史

如果你单击一个系统范围的变更（“(system)”后缀列表中），Jenkins 则跳转到与其相关的所有版本列表页面，并允许你查看不同版本的差异（见图 7-31）。这些不同以 *diff* 文件进行显示，其本身的可读性并不好。然而对于小的变更，Jenkins 的大部分配置文件是可读性好的 XML 格式，很容易理解发生了什么变更。

JobConfigHistory 插件是一个强大的工具。然而在撰写本书时，它仍有些局限。正如之前提到的，这个插件只能以 raw *diff* 的格式显示不同，而且你不能存储之前版本的配置文件（在某种危险的情况下，这样做会脱离上下文，特别是对于系统范围的配置文件）。不过，它能给出一个包括构建作业和系统配置的非常清晰的变更图。

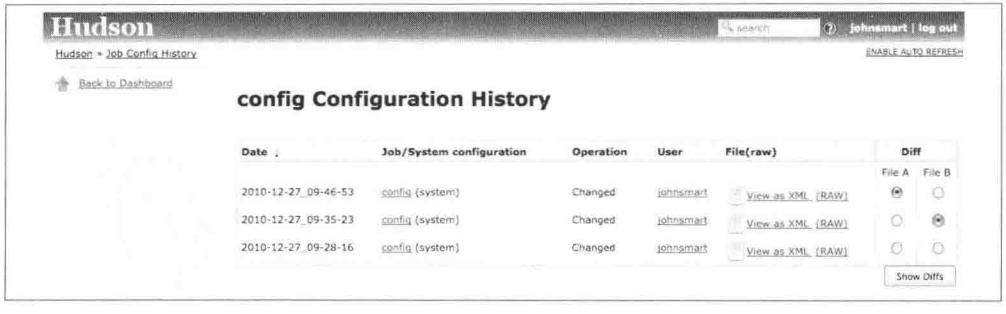


图7-31　查看配置历史差异

小结

在本章中，我们已经看到了 Jenkins 中的很多配置安全的方式。Jenkins 基于身份认证和授权的安全模型是灵活、可扩展的。对于 Jenkins 安装，你应该试着将机构与 Jenkins 安全策略作为一个整体。这可以简单地从集成本地 LDAP 仓库或使用一个成熟的如 Crown 或 CAS 的 SSO 解决方案进行搭建。在这任意一种情况下，长期看来，它均会使系统管理变得更加容易。

通知

简介

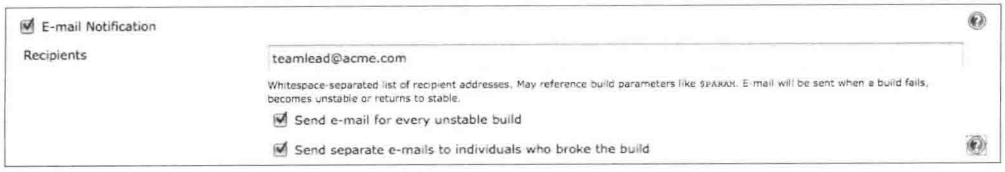
虽然使用构建服务器构建你的软件是重要的事情，但更重要的是让人们知道你的构建服务器不能做哪些。任何持续集成环境价值命题的一个关键部分是提高项目健康状况的信息流程，比如单元测试或者集成测试套件回归测试失败，又或者其他质量相关的下降问题，比如代码覆盖率或是代码质量度量。在所有情况下，CI 服务器必须尽可能快地让相关人员知晓发生的任何问题。这就是我们所说的通知。

有两种主要类别的通知策略，我称之为*被动策略*和*主动策略*（*拉* /pull 或*推* /push）。被动通知（拉）需要开发者自觉地查阅最新的构建状态，包括 RSS 源、构建消息分发器及（某些范围内的）电子邮件。主动通知（推）在构建失败时会主动提醒开发人员，并且包括诸如桌面通知、聊天和短信等方法。这两种方法各有优缺点。被动的通知策略，如构建消息分发器可以提高公众关于失败的构建的意识，并且能帮助建立将修复失败构建提升为高优先级处理的团队文化。更直接形式的通知形式可以积极鼓励开发人员通过自己的双手更快地修复中断的构建。

电子邮件通知

电子邮件通知是CI通知最明显和最常见的形式。电子邮件众所周知、无处不在、易于使用、易于配置（参见第 4 章的“配置邮件服务器”一节）。所以，当团队设置了他们的第一个持续集成环境时，它通常是团队最常用的初始通知策略。

你可以在 Jenkins 中通过勾选 E-mail Notification 复选框激活 Jenkins 电子邮件通知，并提供需要通知的人的电子邮件列表（见图 8-1）。默认情况下，Jenkins 为每个失败或者不稳定的构建发送一封电子邮件。



196

图8-1 配置电子邮件通知

请记住，在一系列失败或者不稳定构建后，首次成功的构建也将发送一封电子邮件，以表明该问题已得到修复。

正常情况下，构建不会采取太多的尝试，直到再次正常工作——开发人员应该在本地诊断和重现问题，只有在本地修复之后，才提交他们的修复到版本控制。重复的构建失败通常表明配置存在问题或者是不好的开发实践（例如，开发者不首先在本地检查就提交修改）。

你可以选择给提交变更导致构建失败的开发人员发送单独的电子邮件。这通常是个好主意，因为自上次构建之后提交过变更的开发人员自然是对生成结果最感兴趣的人。Jenkins 从当前配置的安全域中获得用户的电子邮件地址（参见第 7 章的“安全域——定义 Jenkins 用户”一节），或者你已经设置（参见第 4 章的“配置邮件服务器”一节）从 SCM 用户名派生的电子邮件地址。

如果你使用此选项，则在主要的通信组列表中包括整个团队并不是很有用。你可能想仅仅包括对监视每个构建结果（如技术主管）感兴趣的人，并让 Jenkins 直接通知相关开发人员。

假定当所做的变更导致构建失败，这是个普通（但不总是）的情况。但是，如果构建是不频繁的（例如,每晚构建,或者构建等待几个小时才开始),许多变更可能已经被提交了，很难知道哪一个真正负责构建失败。

当发送电子邮件通知时并非所有的构建都差不多。提交更改的开发人员对单元和集成测试构建的结果特别感兴趣（特别是那些由自己的更改触发的），而 BA 和测试人员可能会对自动化验收测试的状态更有兴趣并密切关注。因此，每个构建作业准确的电子邮件通知设置会不同。事实上，定义电子邮件通知策略是非常有用的。电子邮件通知策略的例子是这样的。

- 快速构建（单元 / 集成测试，运行的时间少于 5 分钟）：通知发送给团队组长以及提交更改的开发人员。

197

- 慢速构建（验收测试构建，快速构建完成之后）：通知发送送给团队组长、测试人员和提交更改的开发人员。

- 每晚构建（QA 度量、性能测试等；只有其他构建正常才运行）：所有团队成员——日常状态会议前提供了项目健康状态的快照图。

事实上，你应该基于实际的构建作业考虑什么样的通知策略是适当的，而不是所有的构建都采用一揽子策略。

更高级的电子邮件通知

默认情况下，Jenkins 电子邮件通知是一个相当直率的工具。通知消息总是发送到基本上属于同一组的人。你无法根据出的错误来判断给不同的人发送信息，或者实施任何形式的升级政策。这是有用的：例如，第一次构建中断通知给提交更改的开发人员，第二次构建失败发送不同的消息给团队领导或者整个团队。

Email-ext 插件允许你定义一个更精致的电子邮件通知策略。这个插件增加了一个 Editable Email Notification 复选框（见图 8-2），从而有效地取代了标准的 Jenkins 电子邮件通知。在这里，你可以定义默认收件人列表和微调的电子邮件内容，而且还定义一个更精确的通知策略（不同的事件用不同的消息和收件人列表）。请注意，一旦你已经为构建作业安装、配置此插件，就可以停用正常的电子邮件通知配置。

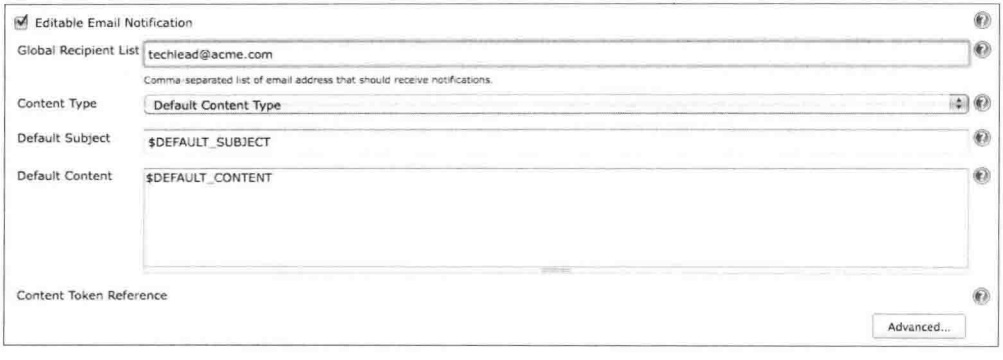


图8-2　配置高级的电子邮件通知

这个插件有两个相关但不同的功能。首先，它可以让你自定义电子邮件通知消息。你可以从大量的预定义的标签中选择创建自己的自定义邮件的标题和正文。你可以使用熟悉的美元符号（如 `${BUILD_NUMBER}` 或 `$BUILD_NUMBER`）在消息模板中包括标签。一些标签接收参数，你可以指定使用 `name=value` 格式（如 `${BUILD_LOG, maxLines=100}` 或 `${ENV, var="PATH"}`）。其中更有用的标签如下。

198

`${DEFAULT_SUBJECT}`

在 Jenkins 系统配置页面配置的默认电子邮件主题。

`${DEFAULT_CONTENT}`

Jenkins 系统配置页面配置的默认电子邮件内容。

`${PROJECT_NAME}`

项目的名称。

`${BUILD_NUMBER}`

当前构建号。

`${BUILD_STATUS}`

当前构建状态（失败、成功等）。

`${CAUSE}`

构建的原因。

`${BUILD_URL}`

Jenkins 对应构建作业页面的链接。

`${FAILED_TESTS}`

如果失败的话，显示有关失败的单元测试信息。

`${CHANGES}`

显示自上次构建以来所做的修改。

`${CHANGES_SINCE_LAST_SUCCESS}`

自从上次成功构建以来所做的所有修改。

通过单击 Context Token Reference 标签相对的 Help 图标，你可以得到可用标识的完整列表以及其可以接收参数的选项。

Advanced 按钮可以允许你基于触发器的概念（见图 8-3），定义一个更复杂的通知策略。触发器决定何时应发出电子邮件通知消息。支持的触发器包括以下内容。

Failure

任何时候构建失败。

Still Failing

任何连续构建失败。

Unstable

任何时候构建是不稳定的。

Still Unstable

任何连续的不稳定构建。

Success

任何成功的构建。

Fixed

当构建从失败或不稳定变成成功。

Before Build

每次构建开始之前发送。

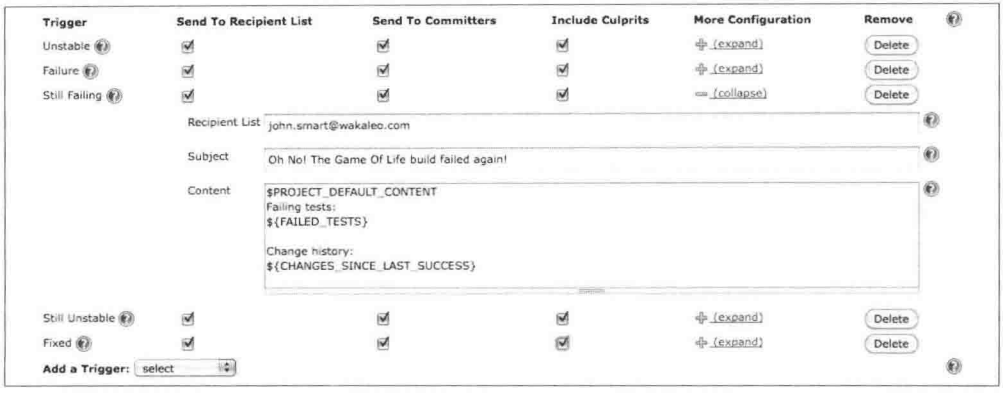


图8-3　配置电子邮件通知触发器

如你所愿，你可以设置尽可能多（或尽可能少）的触发。收件人列表和信息模板可以为每个触发器定制——例如，通过使用 Still Failing 以及 Still Unstable 触发器，你可以设置第一次构建作业失败时只通知提交更改的开发人员的通知策略；但如果第二次失

败，可以通知团队负责人。你可以选择只向提交该构建的开发者发送消息（“Send to committers”），或者也包括上次成功构建后提交修改的每个人。这有助于确保可能导致构建中断的每个人得到适当的通知。

你可以通过单击 More Configuration 选项（见图 8-3 的 Still Failing 触发器）来定制消息的内容。通过这种方式，你可以自定义不同的场合发送不同的消息。

触发器可以智能地进行交互。如果你同时配置 Failing 以及 Still Failing 触发器，则第二次构建失败只有 Still Failing 触发器起作用。

一个定制信息的实例如图 8-4 所示。

然而总的来看，作为一种通知策略，电子邮件不是没有它的缺点。一些开发人员有时候关闭电子邮件客户端以避免被人打扰。在大型机构中，每天到达的电子邮件数量会相当大，构建失败通知可能隐藏在许多其他不太重要的信息中。因此，构建失败可能并不总能得到他们需要的 CI 环境中高优先级的关注。在下面的各节中，我们将看看可用于提高团队构建失败的意识，并鼓励开发人员更快地修复一些其他通知策略。

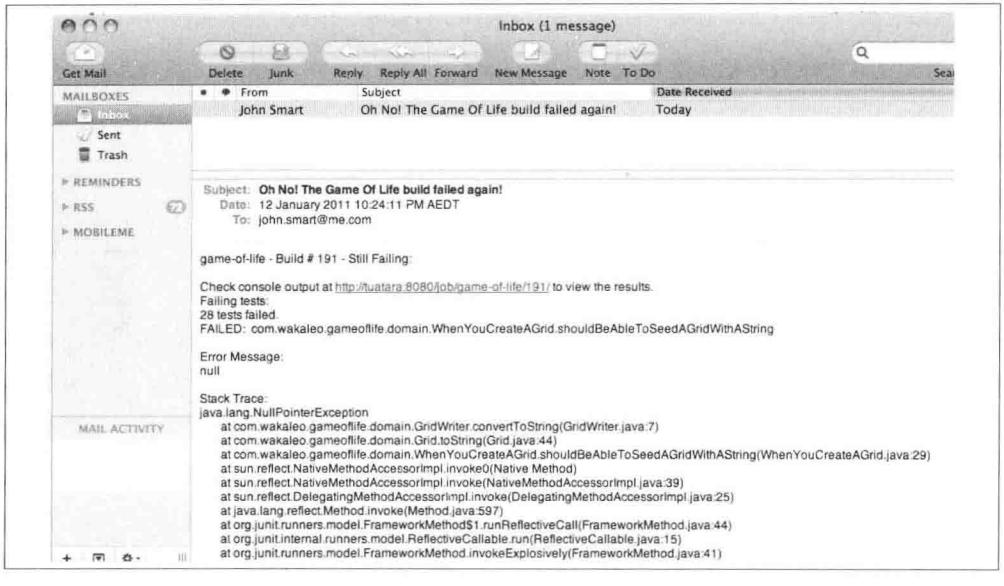


图8-4　定制通知信息

声明构建

当构建失败时，知道有人发现了这个问题并在修复它是有用的。这样避免了多个开发人员浪费时间试图解决同样的问题。

Claim 插件可以让开发人员表示他们负责修复中断构建，并正在尝试修复它。你可以以通常的方式安装此插件。一旦安装完成，开发人员可以声明失败的构建为自己所有，并选择添加注释来解释构建失败的原因、开发人员打算做什么。声明的构建在构建历史中被标记，这样其他开发人员能避免浪费时间和不必要的调查。

要为构建作业激活声明，你需要在构建作业配置页面勾选“Allow broken build claiming”选项。在这里，你可以在构建细节页面（见图 8-5）声明中断的构建。声明的构建将在构建历史中显示图标，表明已经被声明了。你可以生成构建声明“黏性”，这样所有该作业的后续构建失败将由这名开发人员自动声明，直到问题解决为止。

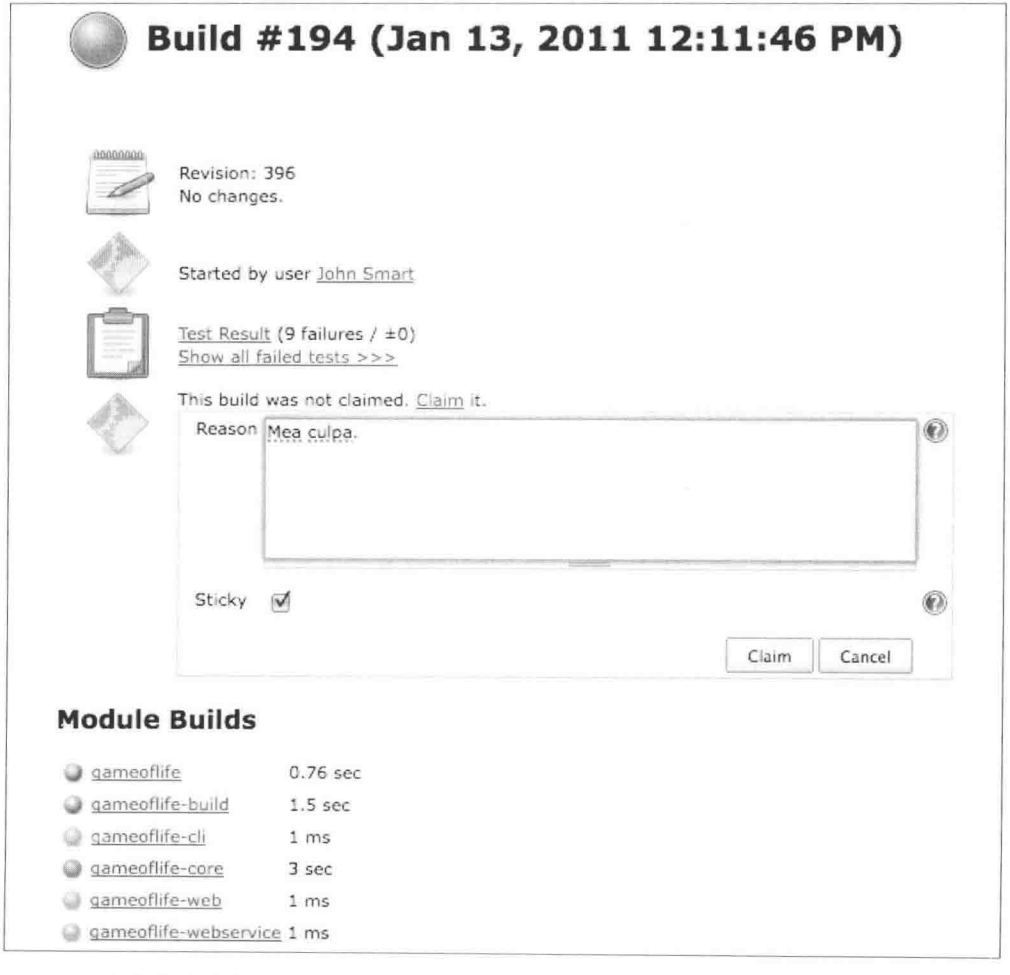


图8-5　声明失败的构建

RSS 订阅

Jenkins 还为构建结果提供了方便的 RSS 订阅，包括你的所有构建的全部构建结果（或只是针对特定视图的构建），或者指定构建的构建结果。RSS 订阅图标在构建仪表盘底部（见图 8-6），以及在每个构建作业的构建历史界面的底部，使我们可以访问所有的构建结果或者只是失败的构建结果。

RSS 订阅的 URL 很简单，支持所有 Jenkins 构建结果页面。你只需在页面 URL 后添加 */rssAll* 就能得到所有构建结果的 RSS 订阅，或者 */rssFailed* 只获取失败的构建。*/rssLatest* 为你提供只包含最新构建结果的订阅。但要获得 URL 最简单的方法是单击相应 Jenkins 屏幕上的 RSS 图标。

202



图8-6　Jenkins RSS订阅

有很多 RSS 阅读器，商业的和开源的均有，它们可用于几乎每种平台和设备，所以这是一个用于订阅构建结果的不错选择。许多常见的浏览器（特别是 Firefox 浏览器）和电子邮件客户端也支持 RSS 订阅。有些读者遇到过身份认证的麻烦；不过，如果你的 Jenkins 实例是安全的，你可能需要做一些小的额外的配置来查看构建结果。

RSS 订阅器可以成为全部构建结果的信息源，你无须连接到服务器就可以一目了然地看到构建状态。然而，大多数 RSS 阅读器是被动的工具——你可以咨询构建的状态，但是如果构建失败发生的话，通常 RSS 阅读器不能进行提示。

构建分发器

信息分发的概念通常用在敏捷的圈子里。正如敏捷大师 Alistair Cockburn 所说：

> 信息分发器通常张贴在，人们工作或者步行通过时可以看到的地方。它向读者展示他们关心的信息而无须向任何人提问。这意味着以更少的中断带来更多的沟通。

在 CI 服务器的上下文环境中，信息分发器是一个显著的设备或者显示屏，允许团队成

员和其他人容易看到是否有构建被中断。它通常显示当前所有的构建结果；或者只是失败的构建结果，显示在一个大型的位于醒目位置的壁挂式平板屏幕上。这种专门的信息分发器通常被称为*构建分发器*（build radiator）。

如果使用得当，则构建分发器是最有效的被动通知策略。它们确保每个人都知道失败的构建是非常有效的。此外，不像我们在本章后面讨论的某些极端反馈设置（Extreme Feedback Devices），构建分发器可应付许多构建作业，包括许多失败的构建作业，所以可以有效地使用在多团队上下文中。

有几个 Jenkins 构建分发器的解决方案。一个最简单的解决方案是使用 Jenkins Radiator View 插件。该插件增加了一个你可以创建新类型的作业（见图 8-7）。

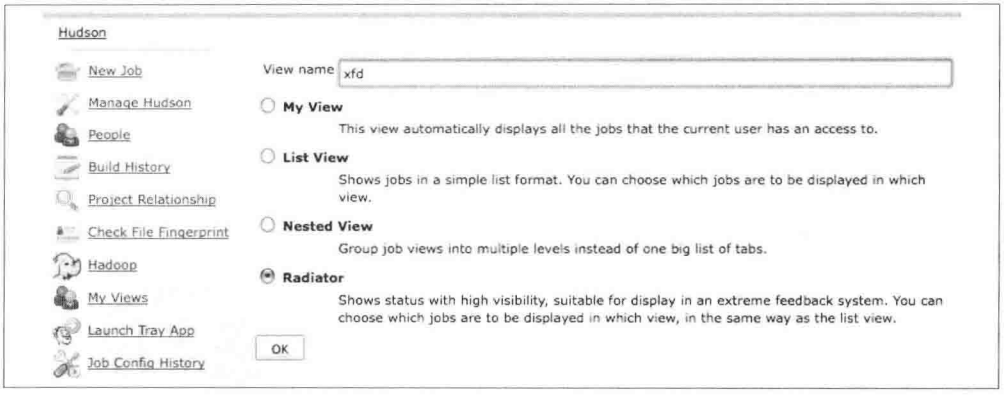


图8-7　创建构建分发器视图

配置构建分发器视图的观点类似于配置更传统的列表视图——你只需要指定想要包含在视图中的构建作业（或者单独选择它们，或者通过使用正则表达式）。

由于构建分发器视图占据了整个屏幕，因此修改或删除构建分发器有点棘手。事实上，打开视图配置屏幕唯一的途径是添加 */configure* 到视图 URL。所以如果你的构建分发器被称为“build-radiator”，你可以通过打开 *http://my.hudson.server/view/build-radiator/configure* 编辑视图配置。

构建分发器视图（见图 8-8）为每个失败或不稳定构建显示为大的红色或者黄色小框，并且突出显示构建作业名字以及一些其他细节。你可以配置构建分发器视图来显示成功以及失败的构建（它们被显示在绿色小框内）。不过，好的构建分发器应该只显示失败的构建，除非所有的构建都通过了。

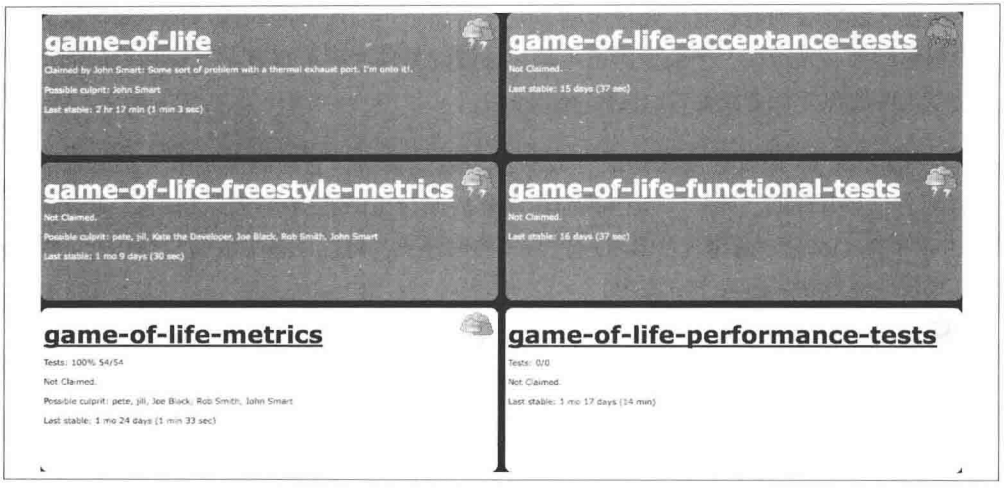


图8-8　显示构建分发器视图

即时消息

即时消息(IM)作为快速、轻便的媒介被广泛应用于专业和个人通信。即时消息是瞬间的，当想选用比电子邮件快的通知方式时，它更有优势。它也是“推”而不是“拉”。当你收到一条消息时，它会在屏幕上弹出并且需要你的关注。这使得它比简单的电子邮件信息更难被忽略或者被推迟。

Jenkins 为即时消息通知方式提供了良好的支持。即时消息插件提供了使用 IM 和 Jenkins 通信的通用支持。可以为各种 IM 协议（比如 Jabber 和 IRC）添加特定协议的插件。

用 Jabber 进行 IM 通知

今天许多即时消息服务器是基于 Jabber 的、一个开源的基于 XML 的即时消息协议。Jenkins 提供了 Jabber 即时消息良好的支持，这样开发人员可以接收构建失败的实时通知。此外，该插件运行 IM 机器人监听聊天频道，让开发人员通过聊天信息在 Jenkins 服务器上运行命令。

在 Jenkins 中设置 IM 支持是直截了当的。首先，你需要安装 Jenkins 即时通信插件和使用标准的插件管理器页面安装 Jenkins Jabber 通知插件并重启 Jenkins（见图 8-9）。

一旦做到这一点，你需要配置即时消息服务器。任何 Jabber 服务器都行。你可以使用像谷歌聊天的公共服务，或在本地设置你自己的内部消息服务器（基于 Java 的开源聊天服务器 OpenFire 是个不错的选择，参考网址：*http://www.igniterealtime.org/projects/*

openfire/index.jsp)。使用公共服务进行内部通信可能令系统管理员难以接受，你可能很难通过企业防火墙。另一方面，设置你自己的内部聊天服务，对于开发团队或者组织来说很有意义，因为它在为开发人员之间提供了技术问题或者评论的另一个沟通渠道。下面的例子将使用本地的 OpenFire 服务器，但是一般的做法将适用于任何 Jabber 兼容的服务器。

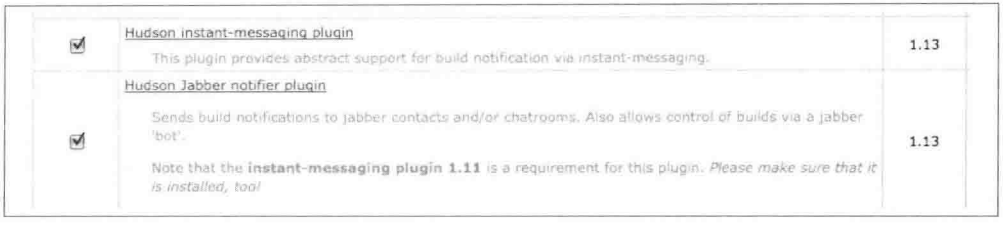


图8-9　安装Jenkins IM插件

第一步涉及在你的 Jabber 服务器上创建一个 Jenkins 专用的账号。这只是一个普通的聊天账号，但是它需要不同于你的开发人员账号（见图 8-10）。



图8-10　Jenkins需要自己专用的IM用户账号

一旦你已经设置好 IM 账号，则需要配置 Jenkins 通过该账户发送 IM 通知。进入主配置页面，勾选 Enable Jabber Notification 复选框（见图 8-11）。在这里，你提供 IM 账号的 Jabber ID 和密码。Jenkins 通常从 Jabber ID 判断 IM 服务器（如果是不同的 IM 服务器，你可以在 Advanced 选项中进行重写）。如果你正在使用群组聊天室（开发团队另一个有用的沟通策略），则也可以在这里提供这些聊天室的名称。这样，Jenkins 能够处理发布到聊天室及那些作为直接消息收到的说明。



图8-11　在Jenkins中设置基本的Jabber通知

这是你所需要的基本设置。然而，你可能需要在 Advanced 部分提供特定安装信息的额外的细节（见图 8-12）。在这里，如果不能从 Jenkins Jabber ID 派生，则可以指定 Jabber 服务器的名称和端口。你还可以提供用于 Jenkins 用户 ID 来生成相应 Jabber ID 的默认后缀。最重要的是，如果你已经加密了 Jenkins 服务器，则需要提供正确的 Jenkins 用户名和密码，以便 IM 机器人能够正确地响应指令。

一旦完成该配置，你需要为每个构建作业设置 Jabber 通知策略。打开构建作业配置页面，单击 Jabber Notification 选项。

首先，你需要定义一个收件人列表。你可以发送消息给个人（只需要使用相应的 Jabber ID，如 *joe@jabber.acme.com* ），或者你已经设置的聊天室。对于聊天室，你通常需要在聊天室 ID 的开始添加一个“*”（例如“*gameoflife@conference.jabber.acme.org”）。但是，如果聊天室 ID 包括“@conference”，Jenkins 会判断这是个聊天室并且自动追加“*”。聊天室的方法更加灵活，但是你必须要确信开发人员一直连接着聊天室，这样才能让这种方式是真正有效的。

你还需要定义通知策略。这决定了哪些构建结果将导致信息被发送出去。选项如下所示。

all

每个构建都发送通知。

failure

仅发送失败或者不稳定的构建通知。

failure and fixed

每个失败或者不稳定的构建都发送通知，并且要在失败或不稳定构建之后下一个成功的构建时发送通知。

change

任何时候构建结果改变时发送通知。

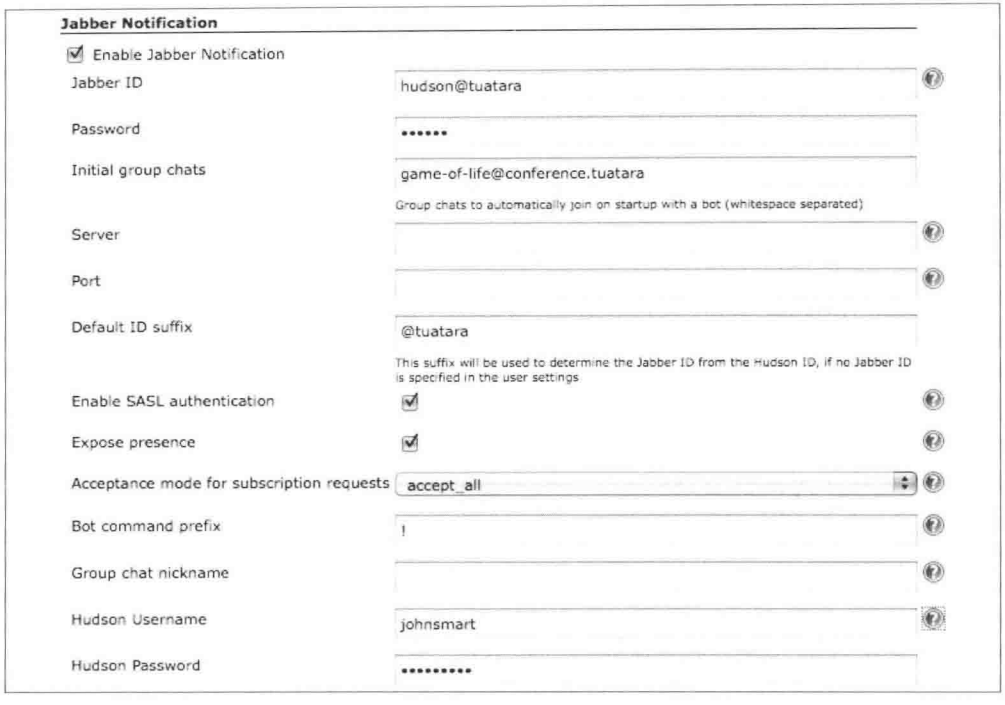


图8-12　高级的Jabber配置

如果你在使用聊天室，则也可以在构建开始时让 Jenkins 给聊天室发送通知（使用“Notify on build starts”选项）。

对于 SCM 触发的构建，Jenkins 也可以通知其他收件人，使用之前讲的默认后缀和从 SCM 用户名来生成 Jabber ID。你可以选择通知。

SCM 提交者（SCM committers）

所有为当前构建提交更改的用户，并因此怀疑可能会中断构建。

SCM 元凶（SCM culprits）

最后成功的构建之后的所有提交者。

SCM 修复者（SCM fixers）

在失败或者不稳定构建后第一个成功构建的提交者。

上游提交者（Upstream committers）

通知提交到上游和目前的构建的提交者。这自动适用于 Maven 构建作业，但是为其他类型构建则需要激活指纹识别。

208 在本书写作之际，只有一个通知策略，所以我们在本章的“更高级的电子邮件通知”一节看到的一些高级选项还可能没有 IM。

开发人员通过他们最喜爱的 IM 客户端被通知（见图 8-13）。开发人员还可以使用一组简单的命令，通过聊天会话和构建服务器交互。一些更有用的命令的示例如下所示：

- `!build game-of-life`——立即开始 *game-of-life* 构建。
- `!build game-of-life 15m`——15 分钟后开始 *game-of-life* 构建。
- `!comment game-of-life 207 'oops'`——为一个给定的构建添加构建描述。
- `!status game-of-life`——为该构建作业显示最新的构建状态。
- `!testresult game-of-life`——为最新的构建显示全部的测试结果。
- `!health game-of-life`——为最新构建的健康状态显示更加完整的总结。

你可以通过给 Jenkins 用户发送 `!help` 消息获得命令的完整列表。

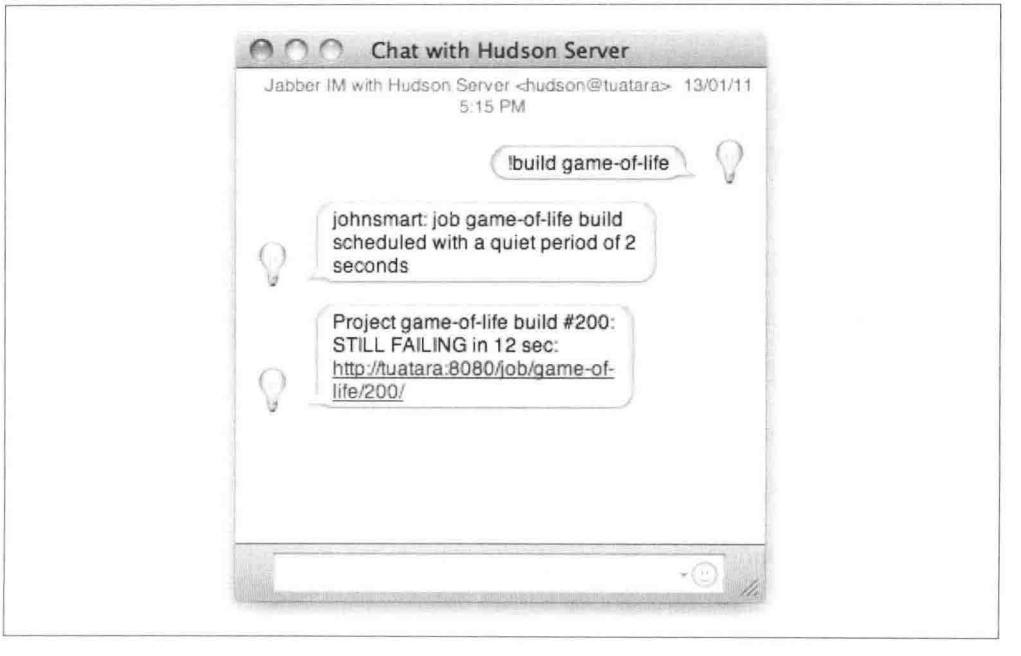


图8-13 Jenkins Jabber消息

使用 IRC 的 IM 通知

基于 Internet 的即时消息的另一种流行形式是互联网中继聊天（IRC）。IRC 传统上侧重于小组讨论（尽管也支持直接消息），尤其是在开源（Open Source）世界，它是一种非常受开发人员欢迎的交流方式。

Jenkins IRC 插件可以让你通过 IRC 频道和 Jenkins 服务器交互，既可以接收通知消息也能向服务器发出命令。像 Jabber 插件一样，你还需要安装即时消息插件让其工作。

IRC 通知

（Juven Xu 撰写）

互联网中继聊天（IRC）是一种流行的即时消息形式，主要用于频道中的小组沟通。例如，Jenkins 已在 Freenode（*http://jenkins-ci.org/content/chat*）设置了一个频道，这样用户和开发人员可以讨论 Jenkins 相关的主题。你会看到许多用户提问，大多数情况下，经验更丰富的用户会及时地提供有用的答案。

就像通过 Jabber 的即时消息一样，你可以配置 Jenkins 通过 IRC“推”通知。xchat（*http://xchat.org/*）等一些 IRC 客户端支持警报配置，这样，当消息到达时，可以闪烁托盘图标或者发出蜂鸣声。要在 Jenkins 设置对 IRC 的支持，首先需要安装 IRC 插件（*http://wiki.jenkins-ci.org/display/JENKINS/IRC+Plugin*）和 Instance Messaging 插作（*http://wiki.jenkins-ci.org/display/JENKINS/Instant+Messaging+Plugin*）。只要到标准插件管理器界面，勾选它们的复选框，然后重启 Jenkins 即可（见图 8-14）。



图8-14　安装Jenkins IRC插件

一旦完成，你需要启用 IRC 插件并对其进行配置以适应自己的环境。基本上，这涉及为你使用的 IRC 服务器提供主机名和端口号、专用的 IRC 频道，以及 IRC 插件的昵称。一个好的习惯是为 CI 通知设置专用频道，这样在其他频道聊天的人不会被打扰。你可能还需要在 Advanced 部分配置额外的细节。所有这些在 Configure System（配置系统）页面都是可用的（见图 8-15）。

除了主机名、端口、频道和我们前面提到的昵称，你还可以配置 IRC 服务器的密码或者 NIckServ 密码。如果你想通过 IRC 消息和服务器交互需要配置命令前缀。这基本上和使

用 Jabber 是相同的（请参阅本章的“即时消息”一节）。最后，你可能希望 IRC 插件使用 /notice 命令而不是默认的 /msg 命令。/notice 和 /msg 是一样的，只是消息被包含在破折号中，这能防止大多数机器人的响应。

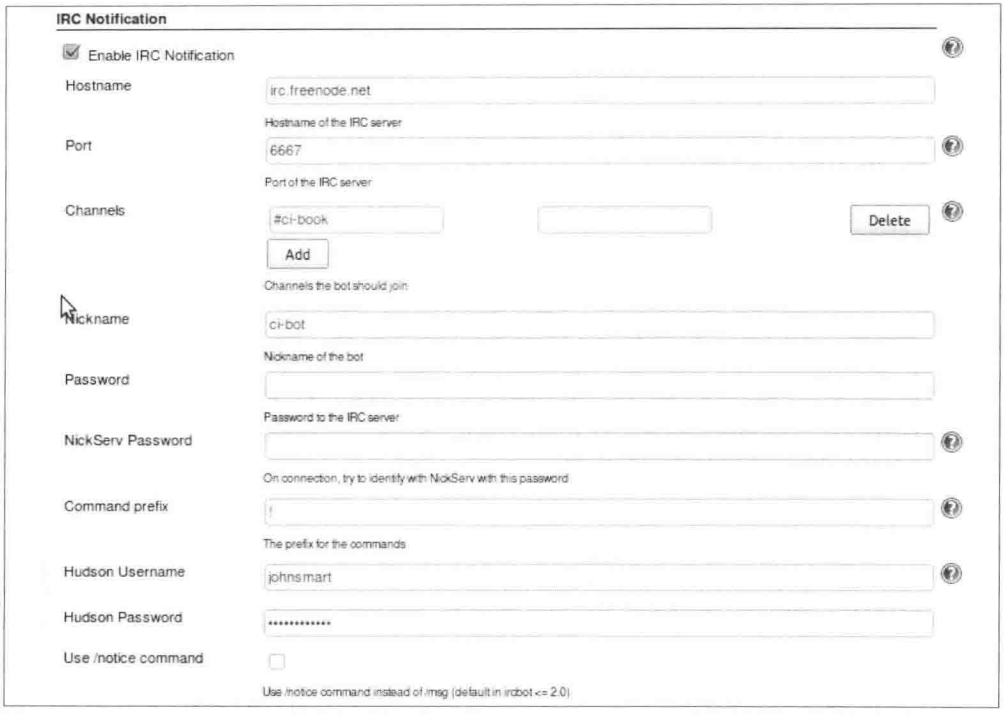


图8-15　高级的IRC通知配置

全局配置一旦准备好，你就可以为每个构建作业启用 IRC 通知并设置通知策略了。打开构建作业配置页面，进入 Post-build Actions 部分，单击 IRC Notification 选项。如果你想要设置通知策略而不是使用默认的，则单击 Advanced 按钮（见图 8-16）。

通知策略（何时发送消息通知，以及发给谁）在本章的“即时消息”一节进行了讨论。Jabber 和 IRC 插件都依赖于即时消息插件，所以它们都有一些共同的核心特征。但是，有些选项是 IRC 插件特有的。例如，你如果不喜欢全局默认值，则可以定义一个定制的频道。最后，对于频道通知消息，你可以选择在通知消息中发送什么信息。你可以选择构建总结、SCM 更改和失败的测试。

当你保存了配置后，可以继续做接下来的事情。根据你配置的内容，该插件会加入适当的 IRC 频道，并为构建作业发送通知消息。

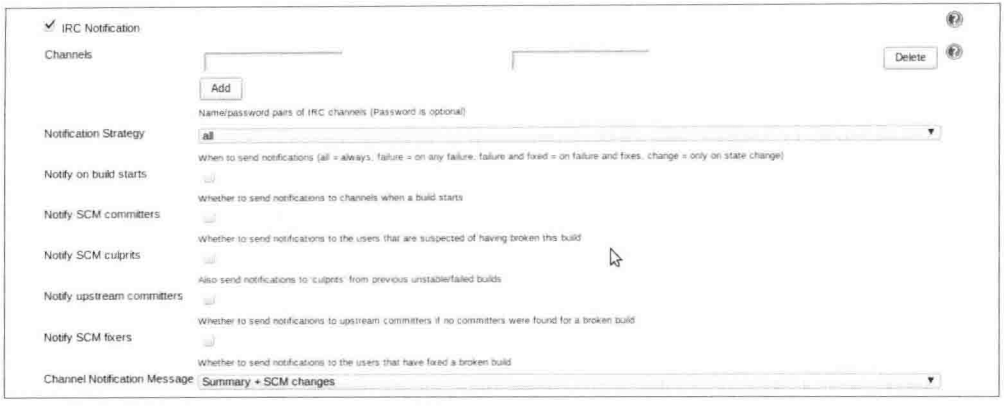


图8-16　IRC通知配置的高级构建作业

例如，在图 8-17 中，IRC 插件加入 freenode 的 #ci-book 频道。首先，用户 juven 提交了带有 scm 消息“feature x added”的变更，并且 IRC 插件通知频道中的所有人构建成功的消息。然后 juven 为 y 特征提交了其他的更改，但是这次构建失败了。John 注意到并且修复了构建错误。IRC 插件现在高兴地说：“Yippie, build fixed!”请注意该屏幕的某些行被突出显示，这是因为我以 juven 的身份登录，并且配置了 XChat IRC 客户端以突出显示包含我的昵称的消息。

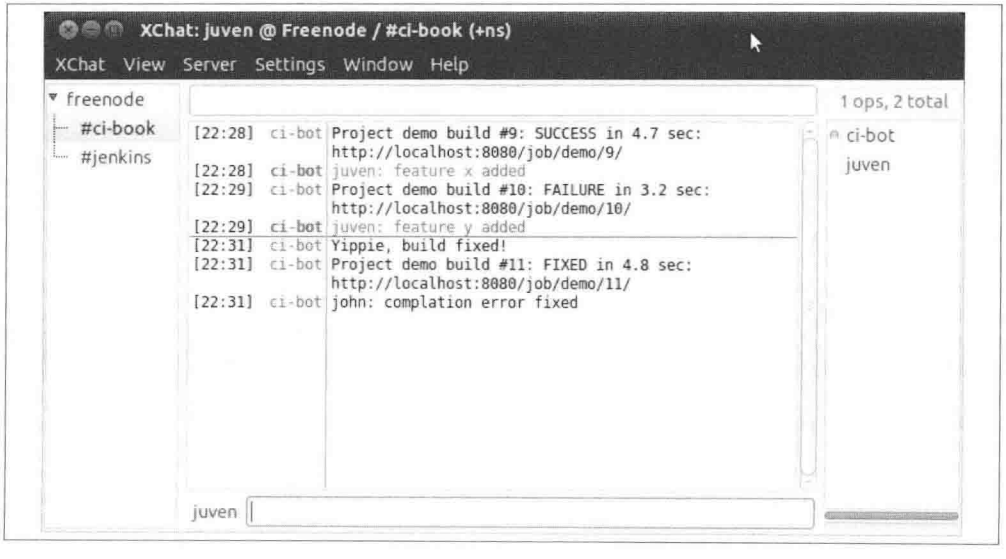


图8-17　IRC通知消息

212 桌面通知器

最佳的推送通知策略是顺利融入开发人员的日常工作环境。如果开发者在平时工作中已经养成了使用即时消息的习惯，则这也就是即时消息能成为一种有效策略的原因。

桌面通知工具也属于此类。桌面通知工具运行在本地开发人员计算机上；它或者作为独立的应用程序或窗口小组件，或者作为开发人员 IDE 的一部分。

如果你使用 Eclipse，Jenkins Eclipse 插件（*http://code.google.com/p/hudson-eclipse/*）则在 Eclipse 视图底部显示一个健康状态的图标。如果你单击该图标，则可以看到 Jenkins 项目（见图 8-18）的详细视图。在 Eclipse 首选项，你提供 Jenkins 服务器的 URL 以及任何必需的身份认证的详细信息。配置是相当简单的，但是对于给定的 Eclipse 工作区间你只能连接到一个单一的 Jenkins 实例。



图8-18 Eclipse里的Jenkins通知

Jenkins Tray Application 插件（见图 8-19）可以让你从 Jenkins 仪表盘使用 Java Web Start 启动小的 Java 客户端应用程序。

此应用程序位于系统托盘中，可让你一目了然地查看构建的当前状态，同时也带来了弹出式窗口，通知你新的构建失败（见图 8-20）。

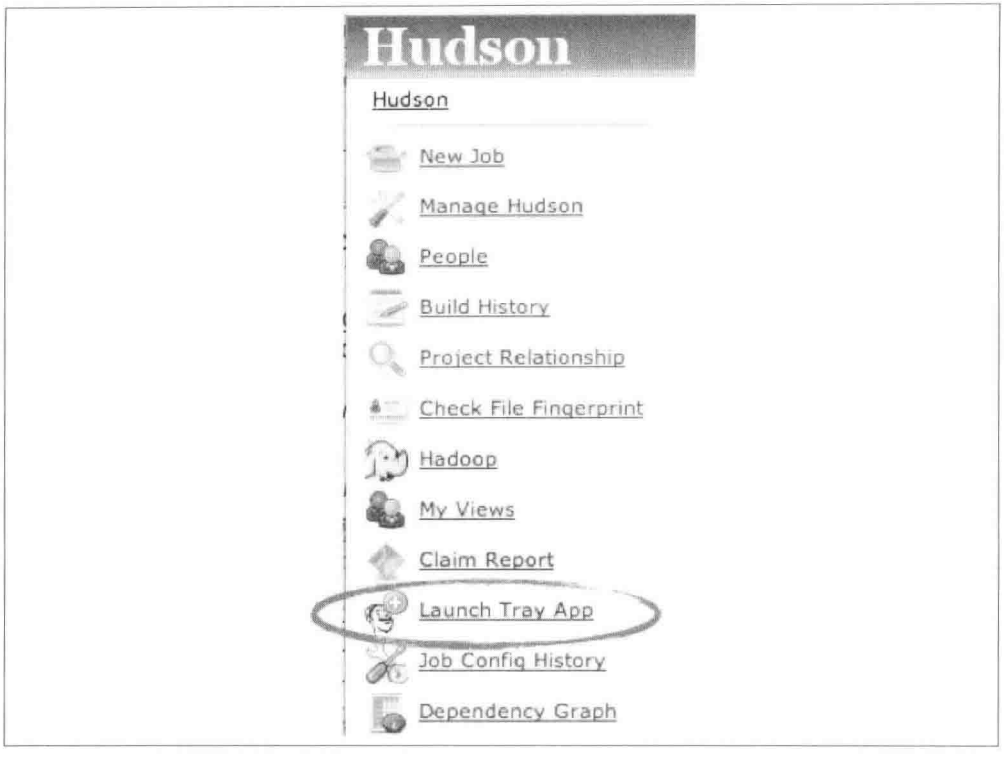


图8-19 启动Jenkins Tray Application

这当然是一个有用的应用程序，但它会受到一些限制。在本书写作之际，Jenkins Tray Application 不支持访问加密的 Jenkins 服务器。此外，开发人员需要记住每天早上重启。这看似是个小问题，但在一般情况下，当涉及通知策略时，你对开发人员要求越少越好。

Jenkins 桌面通知的最佳选择之一是使用类似 Notifo 这样的服务（参见下面的“通过 Notifo 通知”一节），它提供了桌面和移动客户端。我们将在下一节具体了解这是如何工作的。

通过 Notifo 通知

Notifo（*http://www.notifo.com*）是将实时通知发送到你的智能手机或桌面的快速、经济的服务。在 Jenkins 服务器的环境下，你可以用它设置免费或者低成本的实时通知 Jenkins 构建结果。个人账户（你需要能够接收通知）是免费的。你需要设置从 Jenkins 服务器发送通知消息的服务账户。这就是 Notifo 赚取费用的方式，但是在本书写作之际一个服务账户每月可以发送达 10 000 条免费的通知，这对于平均的 Jenkins 实例来说是

足够的。像 Notifo 这样的实时通知服务的强项之一是，通知消息可以被发送到同一用户的不同设备上（特别是智能电话和桌面客户端）。

214

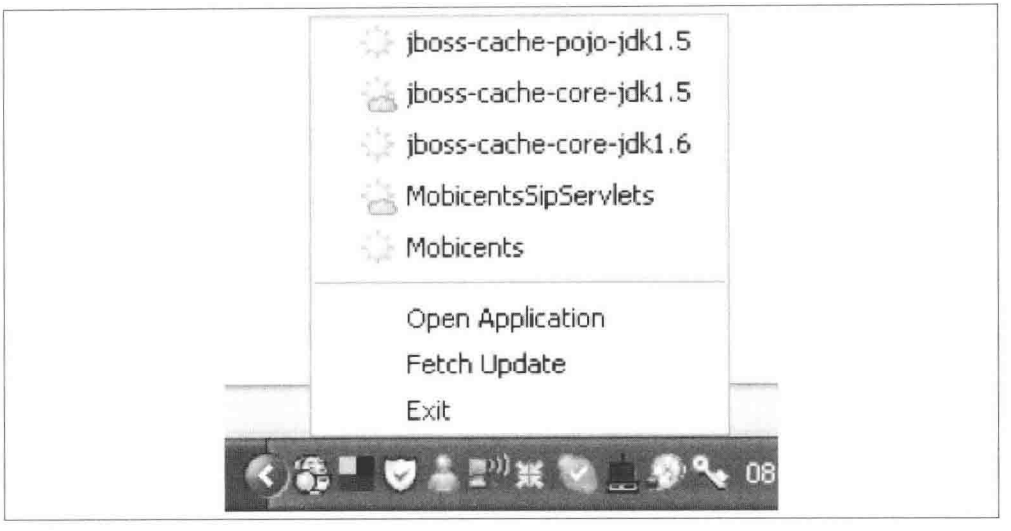


图8-20　运行Jenkins Tray Application

用 Notifo 设置 Jenkins 消息通知是相对比较简单的。首先，到 Notifo 网站并注册，以便创建一个账户。每个想要被通知的团队成员都需要自己的 Notifo 账户。他们还需要在接收通知消息的设备上安装 Notifo 客户端。在本书写作之际，Notifo 客户端可用于 Windows 和 Mac OS X 桌面、iPhone，对于 Linux 和其他智能手机的支持还在开发中。

接下来，你需要为 Jenkins 服务器设置 Notifo 服务账户。你可以用某个开发人员账户，或者为此创建新的账户。登录到 Notifo 网站，并转到 My Services 菜单。在这里，单击 Create Service（见图 8-21），并填写相应的字段。最重要的是服务用户名（Service Username），这必须是唯一的。你还可以指定网站的 URL 以及默认通知的 URL 指向你的 Jenkins 实例，以便用户通过单击此通知消息打开 Jenkins 控制台。

要从 Jenkins 服务器接收通知消息，开发人员需要订阅该服务。你可以通过发送他们订阅的请求，在服务订阅页面把开发人员添加到订阅用户列表。一旦服务已经创建，并且用户已经订阅，就可以设置你的项目发出 Notifo（见图 8-22）。你需要提供 Jenkin 服务的 API 用户名及 API 密码，可以在 Notifo 服务仪表盘看到这两个配置。

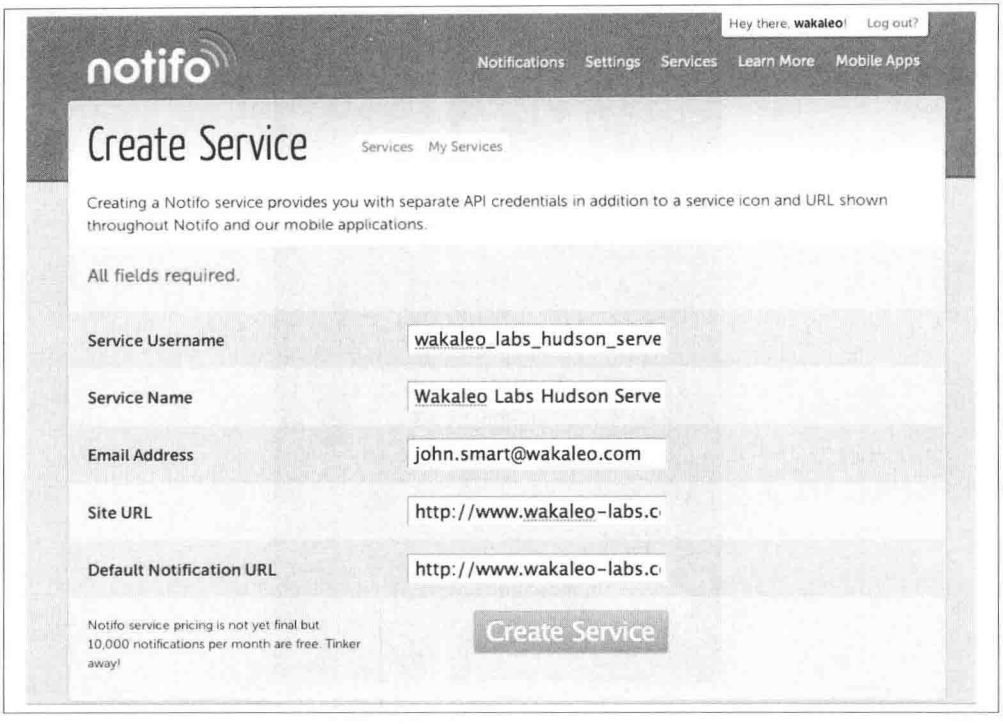


图8-21　为Jenkins实例创建Notifo服务

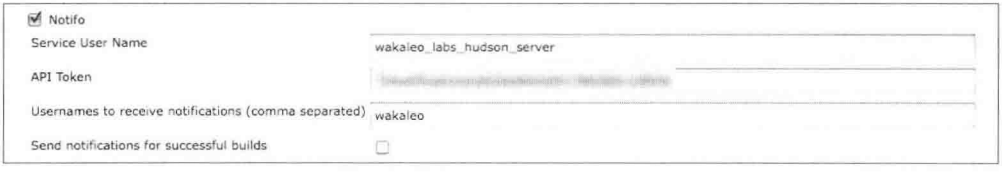


图8-22　在Jenkins构建作业配置Notifo通知

一旦设置完成，Jenkins 就发送构建失败的实时通知到运行 Notifo 客户端的开发人员那里，不管是在桌面上还是移动设备上（见图 8-23）。

在本书写作之际，复杂的通知策略还不被支持——你只需要提供要被通知的 Notifo 用户名列表。然而，这仍然是前线开发人员非常有效的通知工具。

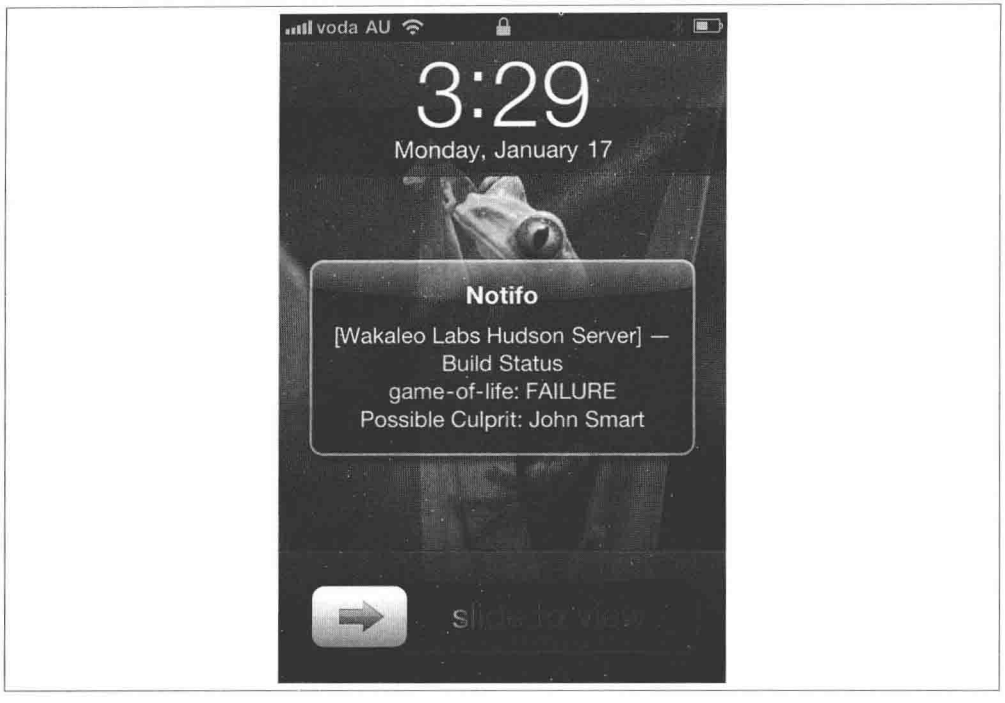


图8-23　在iPhone上接收Notifo通知

移动通知

如果 Jenkins 服务器在互联网上可见（即使你已经在 Jenkins 服务器上设置了身份认证），你也可以通过 iPhone 或者 Android 移动设备监视构建。免费的 Hudson Helper 应用程序（见图 8-24）让你列出目前的构建作业（不是服务器上的所有构建作业，就是特定视图的构建作业）。你还可以查看特定构建作业的详细信息，包括当前的状态、失败的测试和构建时间，甚至是开始和停止构建。

对于 Andriod 手机，你还可以安装 Hudson Mood 小部件，以及提供更新和关于构建失败的警告。

注意，这些移动应用程序依赖于数据连接，所以当在有移动网络的本地能正常工作时，如果相关开发人员不在国内的话，可能就无法联系到他们。

短信通知

现今，短信是另一种普遍的沟通方式，它有着即使人们不在办公室也能通知到对方的优

势。对于构建工程师，这是用于监控关键构建的优秀方式，也同样是开发人员或团队领导在非工作时间里的通知方式。



图8-24　使用Hudson Helper iPhone应用程序

SMS 网关（*http://en.wikipedia.org/wiki/SMS_gateway*）是让你通过特殊格式的电子邮件地址发送短信通知的服务（例如，*123456789@mysmsgateway.com* 可以发送 SMS 消息到 123456789）。许多手机厂商和第三方服务商提供这种服务。在 Jenkins 中没有内置支持 SMS 网关，但是这些网关的基本功能使得集成相对容易：只需将特殊的电子邮件地址添加到正常的通知列表。或者，使用高级的电子邮件配置，你可以设置只包含 SMS 电子邮件地址的单独规则（见图 8-25）。这样做可以使它更容易地微调消息内容以适应 SMS 消息格式。

一旦做到这一点，你的用户会收到短信格式的构建结果的通知（见图 8-26）。这种方法的主要缺点无疑是它不是免费的，并且需要第三方商业服务。也就是说，如果开发人员没有网络或者没有数据功能的智能手机，这是能够到达开发人员唯一的通知技术。事实上，这种技术受到系统管理员的欢迎，并且对于某些关键构建作业是非常有用的。



图8-25　通过SMS网关服务发送短信通知

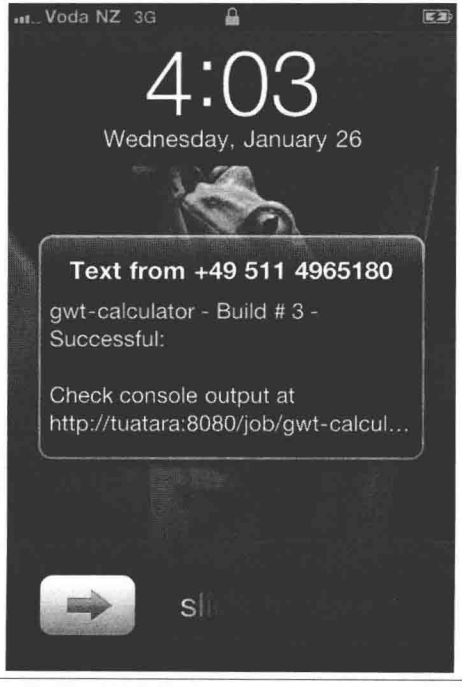


图8-26　通过短信接收通知

制造噪声

如果你有 Jenkins 实例运行在物理位置接近开发团队的机器上，则你可能还需要把声音加到通知策略的组合。这是小的合作团队的有效策略，但是如果构建服务器设置在虚拟机或其他地方就变得复杂了。 219

在 Jenkins 中有两种方法把音频反馈集成到你的构建过程中：Jenkins Sounds 插件和 Speaks 插件。这两个插件可以按常规的方式通过插件管理器页面安装。

Jenkins Sounds 插件是两者中最灵活的。它允许你基于最新构建结果和（可选）以前的构建结果构建详细的通知策略（见图 8-27）。例如，你可以配置 Jenkins 在第一次构建失败时播放一种声音，如果第二次构建失败则播放不同的声音，如果构建修复就播放另一种声音。

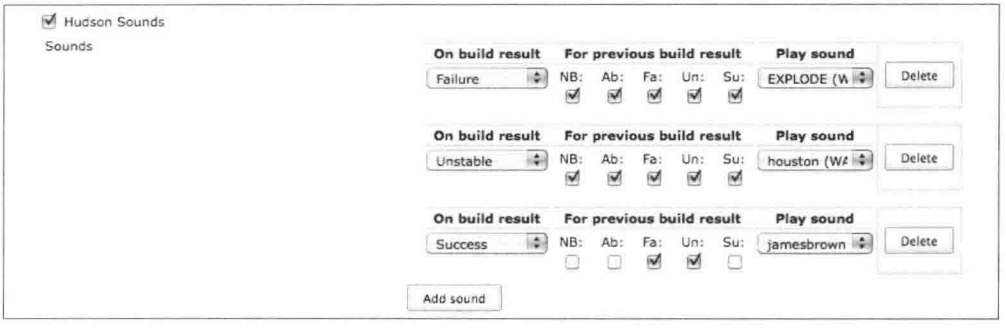


图8-27　在构建作业配置Jenkins声音规则

要进行此设置，可在构建作业配置页面的 Post-build Actions 部分勾选 Jenkins Sounds 复选框。你可以添加任意多的声音配置规则。添加一条规则是很简单的。首先，你需要选择哪个构建结果触发声音。你还需要指定以前的构建结果适用于哪个规则：不建（NB），中止（Ab），失败（Fa），不成功（Un）或成功（Su）。

Jenkins Sounds 插件提供了一连串预定义的声音，即使为最挑剔的构建管理员也提供了充足的选择。不过如果你想自定义声音，还可以直接添加。声音以包含声音文件的 ZIP 或者 JAR 文件存储在扁平的目录结构（没有子目录）中。插件提供的声音列表只是文件名，不包括扩展名。插件支持 AIFF、AU 以及 WAV 文件。 220

在系统配置（System Configuration）页面，你可以为 Jenkins 提供设置新的声音归档文件，如果声音归档文件在本地 Web 服务器就使用 *http://* 访问，如果在本地则使用 *file://* 访问（见图 8-28）。一旦你已保存配置，就可以在 Advanced 部分通过 Test sound 按钮测试声音归档的声音。

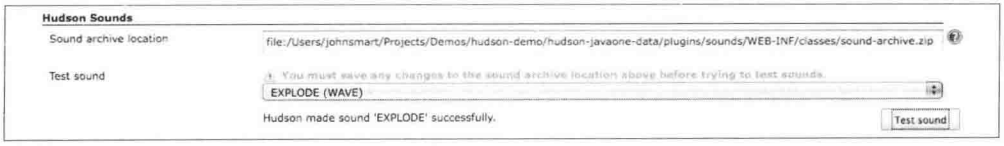


图8-28　配置Jenkins Sounds

如果你想补充传统的通知技术，则 Jenkins Sounds 插件是个很好的选择。简短、好识别的声音是一种吸引开发人员注意并且让团队知道一些东西需要修复的好方式。开发人员会更容易接受带有更多细节信息的通知。

另一种选择是 Jenkins Speaks 插件。有了这个插件，当构建失败时，你可以让 Jenkins 播放定制的通知（非常机械的声音，见图 8-29）。你可以使用 Jelly 配置确切信息。Jelly 是在较低水平的 Jenkins 广泛使用的基于 XML 的脚本语言。

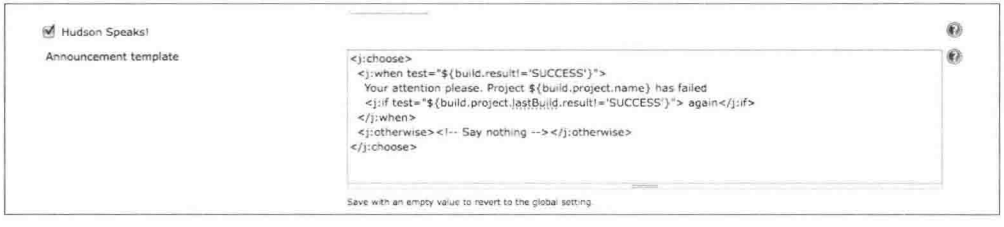


图8-29　配置Jenkins Speaks

该方法的优点在于它的精度：因为你可以在 Jelly 脚本中使用 Jenkins 变量，你可以让 Jenkins 汇报想要了解的构建状态的任何东西。这里是个简单的例子：

```
<j:choose>
  <j:when test="${build.result!='SUCCESS'}">
    Your attention please. Project ${build.project.name} has failed
    <j:if test="${build.project.lastBuild.result!='SUCCESS'}"> again</j:if>
  </j:when>
  <j:otherwise><!-- Say nothing --></j:otherwise>
</j:choose>
```

221

如果你将此字段留空，插件将使用默认模板，你可以在系统配置页面配置。事实上，做到这一点通常是个不错的选择，只有真的需要时才使用特定项目的脚本。

其缺点是机器人的声音有点难以理解。为此，先用常见的短语比如“敬请注意”（Your attention please）开始你的公告是个不错的主意；或者和 Jenkins Sounds 插件结合使用，这样你可以在实际信息广播前吸引开发人员的注意。在项目名称中使用连字符（例如 *game-of-life* 而不是 *gamoflife*）也有助于插件知道项目的名称怎么发音。

这两种方法可用于小的团队，但是较大的团队就有了局限性（尤其是当服务器物理上没有放在开发团队附近时）。将来的版本可能支持在一个单独的机器上播放声音，但是在本书写作之际此功能还不可用。

极端反馈设备

还有更多有想象力的通知工具和策略；如果你愿意尝试电子设备，则还有很多即兴发挥的余地。这包括 Ambient Orbs、Lava Lamp、交通灯或其他更奇异的 USB 控制装置等设备。当你想投射到足够大的屏幕上时，构建分发器（参见本章的“构建分发器”一节）也属于这一类。

与 Jenkins 很好地集成在一起的一台设备是 Nabaztag。Nabaztag（见图 8-30）是一种 WiFi 启用的流行的机器人兔子，可以闪烁彩灯，播放音乐，甚至说话。Nabaztag 的一个优势是，因为它通过 WiFi 工作，不局限于靠近构建服务器，所以即使你的 Jenkins 实例在服务器机房或者虚拟机上，也能很好地工作。只要极端反馈设备存在，这些小东西就能起到作用。

222



图8-30　Nabaztag

还有最重要的是，Jenkins 提供了可用于 Nabaztag 的插件。一旦安装了 Nabaztag 插件

并且重启Jenkins，就很容易配置了。在Jenkins主配置（Configuration）页面，来到Global Nabaztag Settings部分，输入电子兔子的序列号以及令牌密码（见图8-31）。你还可以提供关于构建兔子应该如何应对构建状态的变化（例如报告开始和成功的构建），以及当构建失败、成功、修复、再次失败时应该发出什么消息的默认信息。然后，为特定构建作业激活Nabaztag通知，你需要在构建作业配置页面勾选Nabaztag Publisher选项。根据你的环境，你可能希望也可能不希望为所有的构建发送通知到自己的Nabaztag上。

构建分发器明显的例外是，许多设备也有类似于Jenkins Speaks和Sounds插件（参见本章的“制造噪声”一节）的局限性——它们最适合于小型的合作团队，以及有限数目的项目。然而，当它们运行时，可以帮助你补充一般通知策略之外的一些策略。



图8-31　配置Nabaztag

223

小结

通知是整体CI策略的重要组成部分。毕竟，如果没有人倾听，则失败的构建没有多大用处。通知也不是“一刀切”的事情。你需要考虑自己的机构情况，并调整自身策略以适合当地的企业文化和主要工具集。

事实上，定义和执行适合自己环境的深思熟虑的通知策略是很重要的。例如，电子邮件是无处不在的，所以形成了许多通知策略的中坚力量；但是如果你在一个更大的团队或者从事繁忙的技术领导工作，你可能需要考虑基于高级电子邮件选项设置升级策略（参见本章的“更高级的电子邮件通知”一节）。但是，你应该用更积极的方针，比如即时消息或者桌面通知器。如果你的团队已经在使用聊天工具或者IRC频道进行沟通，最好

试着将其集成到你的通知策略中。对于特别关键的构建作业，短信通知是最优秀的策略。

你还应该确保有被动和主动（或拉和推）通知策略。一个主要的构建分发器或者极端反馈设备能够将强有力的信息发送到团队，即修复构建是一项优先任务，并能帮助形成更敏捷的团队文化。

代码质量

简介

很少有人会否认编写高质量代码的重要性。高质量的代码包括更少的 bug，更易于理解和易于维护。然而，代码质量的精确度定义却是主观的，在公司、团队甚至一个团队的内部个体之间都有不同的定义。

这就是编码标准发挥作用的地方。编码标准就是一些规则，有时是相对任意的，它定义了一些能够在一个团队或者公司内被考虑为可接受的编码风格和约束。在许多情况下，同意一系列的标准或者应用它们，要比这些标准本身更重要。确实，质量代码最重要的方面之一就是易于读懂和理解。假如一个团队内的开发人员都应用了同样的编码标准和实践，这些代码就会更易于阅读（至少对于这个团队内的成员来说是这样的）。并且假如这些标准普遍地在这个行业内使用，则这些代码对于新加入这个团队的开发人员来说也会易于阅读。

代码标准包括美学方面，比如代码布局和格式化、命名规则等；也包括潜在的不良行为，比如 Java 中在一个条件语句后面忘了加大括号。一致的编码风格能降低维护成本，使代码更清晰和更易于阅读，还能方便你对其他团队成员写的代码进行修改。

只有一个非常有经验的开发人员才能真正地从各个方面判断出代码质量。这就是代码评审的作用；除此之外，还有类似结对编程的行为。特别地，只有人的眼睛可以决定一段代码是否写得很好以及它实际上能不能完成需求里要求它做的工作。不管怎样，代码质量度量工具可以帮很大的忙。事实上，不使用这些工具就想加强代码质量标准的话是不切实际的。

这些工具分析应用程序源代码或者字节码，并且检查这些代码是否遵守了一定的规则。

代码质量度量可以包含代码质量的许多方面，从编码标准和最佳实践到代码覆盖率，几乎包含从编译警告到 TODO 注释之间的所有东西。某些度量专注于你的代码库的计量特性，比如代码的行数（NLOC）、代码平均复杂度，或者每个类的行数。其他的一些度量专注于更复杂的静态分析，或者寻找潜在的 bug 或在代码中的不好行为。

对于 Jenkins 来说，有很多不同的代码质量报告插件可以使用。许多插件用于 Java 静态分析工具，比如 Checkstyle、PMD、FindBugs、Cobertura 和 JDepend。其他的一些插件，比如 fxcop 和 NCover，它们专注于 .NET 应用程序。

要使用这些工具，需要在 Jenkins 生产报告之前配置你的构建作业去生成代码质量度量数据。

值得注意的一个例外就是 Sonar。Sonar 可以从任意一个 Maven 项目中提取代码质量度量，而不需要在你的 Maven 项目的需求中做额外的配置。这在当有大量存在的 Maven 项目需要集成到 Jenkins 里的时候以及想要为你的所有项目配置一致的代码质量报告时就显得非常好了。

下面我们会看到如何在你的 Jenkins 构建中设置代码质量报告，也会看到你如何使用它作为构建过程里的一个有效部分。

构建过程中的代码质量

在我们考虑如何在 Jenkins 里报告代码质量度量之前，先回退一步看看最开始介绍的代码质量。代码质量度量在隔离的情况下是没有多大价值的——它们需要成为一个更广泛的过程改进策略的一部分。

代码质量集成的第一个等级就是 IDE。现在的 IDE 能非常好地支持许多代码质量工具——Checkstyle、PMD 和 FindBugs，所有这些插件都可用于 Eclipse、NetBeans 和 IntelliJ，它们都能在代码质量问题上为开发人员提供快速反馈。这是一种更快、更有效的方式来为单个的开发人员提供反馈，也教会了开发人员这些组织结构或项目编码标准。

第二级别就是你的构建服务器。除了你那些标准的单元和集成测试构建作业外，需要配置一个专用的代码质量构建，而且它要运行在标准的构建和测试之后。这个过程的目的就是生产覆盖整个项目的代码质量度量，来密切关注这个项目是怎样作为一个整体进行工作的，以及从更高级别去定位任意一个问题。这些报告的有效性可以通过每周的代码质量评审进行提升，在这个评审中的代码质量问题和动态会在团队内部进行讨论。

分开运行这个作业是非常重要的，因为代码覆盖率分析和许多静态分析工具会运行得非

常慢。保持代码覆盖率测试远离构建也非常重要，因为代码覆盖率运行过程中生成的装配代码永远都不应该被部署到生产环境的仓库中。

默认情况下，代码质量报告就是一个相对被动的过程。没有人会知道一个项目的状态（假如他们不在构建服务器上找出一些信息的话）。不过这总比什么都没有好；如果你对代码质量要求非常严格的话，有一个更好的办法可以用。不是简单地报告代码质量，而是配置一个专用的运行在标准的构建和测试之后的代码质量构建，而且这个构建在代码质量度量处于一个不可接受的级别时配置为失败。你可以在 Jenkins 里或者在你的构建脚本里这么做；而把它配置在你的构建脚本之外，有一个好处就是可以更容易地更改代码质量构建失败的标准而不必更改项目的源代码。

最后，记住编码标准只是指南和建议，而不是绝对的规则。使用失败的代码质量构建和代码质量报告作为一个可能做些改进的地方的指示，而不是一个绝对值的测量。

受欢迎的 Java 和 Groovy 代码质量分析工具

有许多开源代码工具可以帮忙验证差劲的编码行为。

在 Java 世界里，三个静态分析工具经受住了时间的考验，以非常互补的方式在被广泛使用。Checkstyle 擅长检查编码标准和约定，编码行为，以及其他一些质量指标（比如代码复杂度）。PMD 是一款类似于 Checkstyle 的静态分析工具，它更专注于编码和设计实践。FindBugs 是一个革新的工具，它是为了解决 Bill Pugh 的一个正在研究的工作中的一些问题而演变来的，Bill Pugh 在 Maryland 大学的团队专注于识别潜在危险和错误的代码。如果你正在使用 Groovy 或者 Grails，则可以使用 CodeNarc，它可以检查 Groovy 编码行为和约定。

所有这些工具都可以很容易地集成到你的构建过程中。接下来我们将会看看怎样配置这些工具来生成 Jenkins 可以使用的 XML 报告。

Checkstyle

Checkstyle（*http://checkstyle.sourceforge.net*）是一款用于 Java 的静态分析工具。最初的设计是为了加强一系列高度配置的编码标准，Checkstyle 现在也能让你检查差劲的编码行为，以及过度复杂和重复的代码。Checkstle 是一款通用的、灵活的工具，在任何一个基于 Java 的代码质量分析策略中都应有它的一席之地。

Checkstyle 支持非常大量的规则，包括与命名约定、注释、javadoc 注释、类和方法大小、代码复杂度度量、差劲的编码行为以及很多其他部分相关的规则。

代码的重复是另外一个非常重要的代码质量问题——重复的或者近似重复的代码是不易于维护和调试的。Checkstyle 提供了一些能检测重复代码的支持，但是一些更专业的工具比如 CPD 在这个领域内能做得更好。

关于 Checkstyle 的一件非常好的事情就是它非常易于配置。你可以使用 Sun 的编码约定开始，努力使它们能够适用于你的需求，或者没有任何约定就开始写。使用 Eclipse 插件（或者直接在 XML 里），你可以从数以百计的不同规则中进行挑选，然后调整你选择了的规则的设置（见图 9-1）。这是非常重要的，因为不同的公司、团队甚至项目都会对编码标准有不同的需求和偏爱；更好的做法是能有一个可以持续使用的精确的规则集，而不是一个会被忽略的大概的规则集。这在牵扯到大型的遗留代码库的时候显得特别重要——在这些情况下，通常更好的做法就是使用一组更有限的规则而不是淹没在大量的相对更小的格式化问题上。

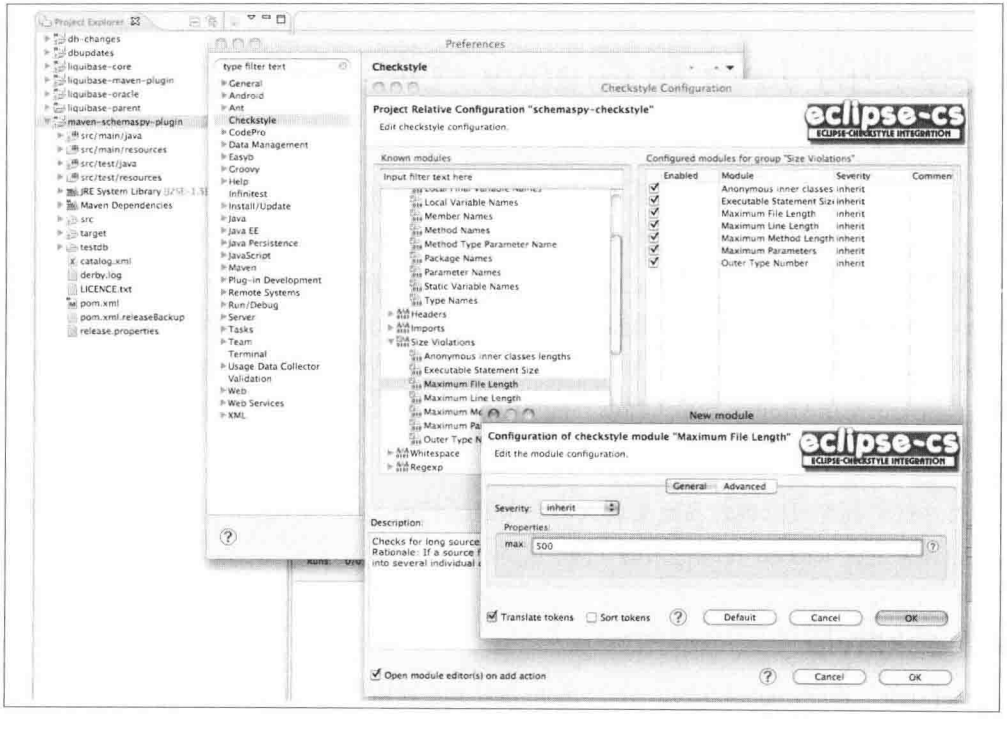


图 9-1　在Eclipse里易于配置Checkstyle

在你的构建里配置 Checkstyle 通常都是非常简单的。假如你正在使用 Ant，你需要从官网（*http://checkstyle.sourceforge.net*）上下载 Checkstyle 的 JAR 文件，并且使它对于 Ant 来说是可用的。你可以把它放置在你的 Ant *lib* 目录下；不过这样就意味着在你的构建服务器（或者任一个从节点）定制化了 Ant 安装，所以这不是一个轻便的解决方

案。一个更好的方法就是把 Checkstyle 的 JAR 文件放在你项目中的一个目录里，或者使用 Ivy 或 Maven Ant Task 库去声明 Checkstyle 上的依赖。假如你选择了把 Checkstyle 的 JAR 文件放在项目的目录里，可以如下面这样来声明 Checkstyle 任务：


```
<taskdef resource="checkstyletask.properties"
     classpath= "lib/checkstyle-5.3-all.jar"/>
```

接着，生成一个 Jenkins 可直接使用的 XML 格式的报告，你可以像下面这样做：

```
<target name="checkstyle">
  <checkstyle config="src/main/config/company-checks.xml">
    <fileset dir="src/main/java" includes="**/*.java"/>
    <formatter type="plain"/>
    <formatter type="xml"/>
  </checkstyle>
</target>
```

现在，只需调用这个任务（比如 `ant checkstyle`）就可以生成 Checkstyle 报告。

在 Maven 2 里，你可以像下面这样往 `<reporting>` 部分添加一些东西：

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      <version>2.4</version>
      <configuration>
        <configLocation>
          src/main/config/company-checks.xml
        </configLocation>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

对于一个 Maven 3 项目来说，你需要添加插件到 *maven-site-plugin* 的 `<configuration>` 部分的 `<reportPlugins>` 元素：

```
<project>
  <properties>
    <sonar.url>http://buildserver.acme.org:9000</sonar.url>
  </properties>
  <build>
    ...
    <plugins>
```

```
      ...
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-site-plugin</artifactId>
        <version>3.0-beta-2</version>
        <configuration>
          <reportPlugins>
            <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-checkstyle-plugin</artifactId>
              <version>2.4</version>
              <configuration>
                <configLocation>
                  ${sonar.url}/rules_configuration/export/java/My_Rules/checkstyle.xml
                </configLocation>
              </configuration>
            </plugin>
          </reportPlugins>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```


现在，运行 `mvn checkstyle:checkstyle` 或者 `mvn site` 就会分析你的源代码并且生成 Jenkins 可以使用的 XML 报告。

需要注意的是，在最后一个例子中，我们使用了一个已经上传到 Sonar 服务器（通过 `${sonar.url}` 属性定义）的 Checkstyle 规则集。这种策略使得 Eclipse、Maven、Jenkins 和 Sonar 使用同样的 Checkstyle 集变得简单了。

Gradle 的最新版本也提供一些集成的 Checkstyle 支持。你可以像下面这样为构建设置 Checkstyle：

```
apply plugin:'code-quality'
```

这样会默认在 *config/checkstyle/checkstyle.xml* 里的 Checkstyle 规则集。你可以用 `checkstyleCinfigFileName` 属性来覆盖它：不管怎样，在本书写作之际，你还不能让 Gradle 代码质量插件从一个 URL 获得 Checkstyle 规则。

你可以通过运行 `gradle checkstyleMain` 或者 `gradle check` 来生成 Checkstyle 报告。

PMD/CPD

PMD（*http://pmd.sourceforge.net*）是另一款非常受欢迎的静态分析工具。它专注于一些

潜在的编码问题，比如未使用或者次优化的代码，代码大小和复杂性，以及好的编码行为。一些典型的规则包括“空的 if 语句”（Empty If Statement）、“断空检查”（Broken Null Check）、“避免 if 语句的深度嵌套”（Avoid Deeply Nested If Statement）、“switch 语句应该有 default”（Switch Statement Should Have Default）以及“Logger 不是静态的”（Logger Is Not Static Final）。这些 Checkstyle 规则中有大量的重叠（尽管 PMD 确实有些更有技术性的规则，以及一些更特殊的规则比如和 JSF、Andriod 相关的规则）。

PMD 也附带了 CPD，它是一个强健的探测重复的或者近似重复代码的开源工具。

PMD 相对于 Checkstyle 显得不那么灵活，尽管你仍然可以挑选自己想要在 Eclipse 中使用的规则，然后导出它们为 XML 文件（见图 9-2）。之后你可以把这些规则集导入其他 Eclipse 项目或 Sonar，或者在你的 Ant 或 Maven 构建中使用它。

231

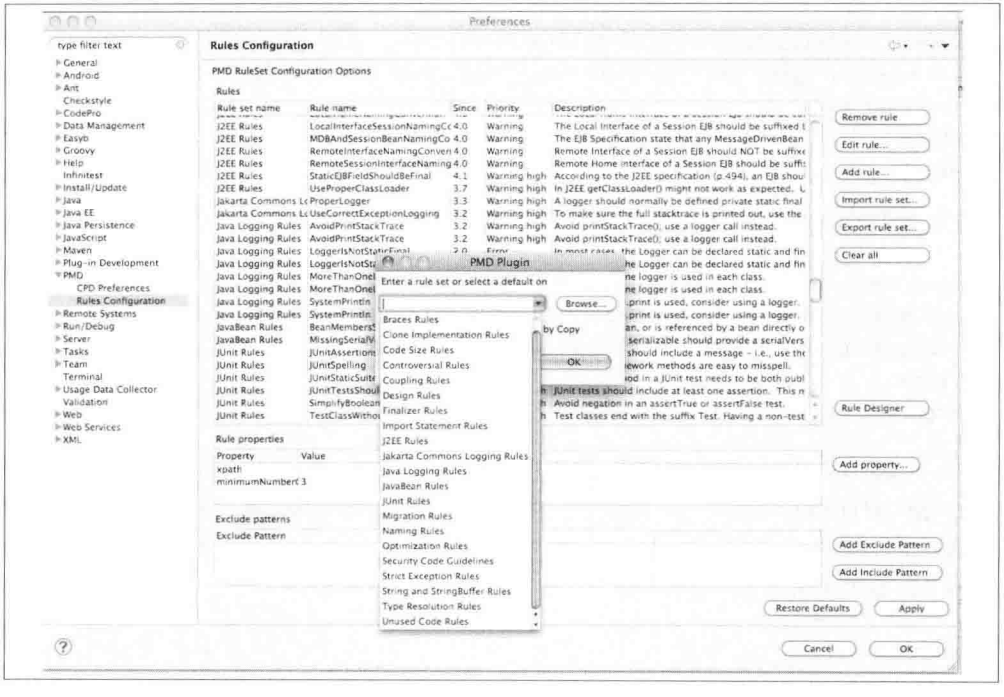


图9-2　在Eclipse中配置PMD规则

PMD 附带了一个你可以用来生成 PMD 和 CPD 报告的 Ant 任务。首先，你需要定义这些任务，如下所示：

```
<path id="pmd.classpath">
    <pathelement location="${build}"/>
    <fileset dir="lib/pmd">
```

```
      <include name="*.jar"/>
    </fileset>
</path>

<taskdef name="pmd" classname="net.sourceforge.pmd.ant.PMDTask"
        classpathref="pmd.classpath"/>

<taskdef name="cpd" classname="net.sourceforge.pmd.cpd.CPDTask"
        classpathref="pmd.classpath"/>
```

232 接下来，你可以通过调用这里举例说明的 PMD 任务来生成 PMD XML 报告：

```
<target name="pmd">
  <taskdef name="pmd" classname="net.sourceforge.pmd.ant.PMDTask"
           classpathref="pmd.classpath"/>

  <pmd rulesetfiles="basic" shortFilenames="true">
    <formatter type="xml" toFile="target/pmd.xml" />
    <fileset dir="src/main/java" includes="**/*.java"/>
  </pmd>
</target>
```

为了生成 CPD XML 报告，你应该像下面这样做：

```
<target name="cpd">
  <cpd minimumTokenCount="100" format="xml" outputFile="/target/cpd.xml">
    <fileset dir="src/main/java" includes="**/*.java"/>
  </cpd>
</target>
```

你可以把这个 XML 规则集放到项目的 classpath（例如，对于一个 Maven 项目，是在 `src/main/resources`）下，或者在一个单独的模块（假如你想要在项目之间共享配置的话）下。以下是怎样使用导出的 XML 规则集配置 Maven 2 去生成 PMD 和 CPD 报告的例子，如下所示：

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <version>2.5</version>
      <configuration>
        <!-- PMD options -->
        <targetJdk>1.6</targetJdk>
        <aggregate>true</aggregate>
        <format>xml</format>
```

```
        <rulesets>
          <ruleset>/pmd-rules.xml</ruleset>
        </rulesets>

        <!-- CPD options -->
        <minimumTokens>20</minimumTokens>
        <ignoreIdentifiers>true</ignoreIdentifiers>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

如果你正在使用 Maven 3 的话，可以把插件定义放在 `<maven-site-plugin>` 配置部分。这个例子也展示了如何使用处于另外一个依赖的情况下的规则集（在这个例子中是 *pmd-rules.jar* 文件）：

```
<project>
  ...
  <build>
  ...
    <plugins>
      ...
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-site-plugin</artifactId>
        <version>3.0-beta-2</version>
        <configuration>
          <reportPlugins>
            <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-pmd-plugin</artifactId>
              <version>2.5</version>
              <configuration>
                <!-- PMD options -->
                <targetJdk>1.6</targetJdk>
                <aggregate>true</aggregate>
                <format>xml</format>
                <rulesets>
                  <ruleset>/pmd-rules.xml</ruleset>
                </rulesets>

                <!-- CPD options -->
                <minimumTokens>50</minimumTokens>
                <ignoreIdentifiers>true</ignoreIdentifiers>
              </configuration>
```

```
            </plugin>
          </reportPlugins>
        </configuration>
        <dependencies>
          <dependency>
            <groupId>com.wakaleo.code-quality</groupId>
            <artifactId>pmd-rules</artifactId>
            <version>1.0.1</version>
          </dependency>
        </dependencies>
      </plugin>
    </plugins>
  </build>
</project>
```

现在，你就可以运行 `mvn site` 或者 `mvn pmd:pmd pmd:cpd` 来生成 PMD 和 CPD 报告。

不走运的是，目前还没有内建的 Gradle 来支持 PMD 或者 CPD，所以你不得不退而求其次地直接调用 PMD Ant 插件，如下所示：

```
configurations {
    pmdConf
}

dependencies {
    pmdConf 'pmd:pmd:4.2.5'
}

task pmd << {
    println 'Running PMD static code analysis'
    ant {
        taskdef(name:'pmd', classname:'net.sourceforge.pmd.ant.PMDTask',
                classpath: configurations.pmdConf.asPath)

        taskdef(name:'cpd', classname:'net.sourceforge.pmd.cpd.CPDTask',
                classpath: configurations.pmdConf.asPath)

        pmd(shortFilenames:'true', failonruleviolation:'false',
            rulesetfiles:'conf/pmd-rules.xml') {
            formatter(type:'xml', toFile:'build/pmd.xml')
            fileset(dir: "src/main/java") {
                include(name: '**/*.java')
            }
            fileset(dir: "src/test/java") {
                include(name: '**/*.java')
            }
```

```
        }

        cpd(minimumTokenCount:'50', format: 'xml',
            ignoreIdentifiers: 'true',
            outputFile:'build/cpd.xml') {
            fileset(dir: "src/main/java") {
                include(name: '**/*.java')
            }
            fileset(dir: "src/test/java") {
                include(name: '**/*.java')
            }
        }
    }
}
```

这个配置文件会使用到在 *src/config* 目录下的 PMD 规则集，并且在 *build* 目录下生成一个名为 *pmd.xml* 的 PMD XML 报告。它也会运行 CPD 并在 *build* 目录下生成一个名为 *cpd.xml* 的 CPD XML 报告。

FindBugs

FindBugs 是一款功能强大的代码质量分析工具，它检查应用程序的字节码来找出一些潜在的 bug、性能问题或者差的编码行为。FindBugs 是 Maryland 大学里 Bill Pugh 领导的一个研究项目的结果，它研究了来自大型的真正的项目中 bug 的字节码模式，比如 JDK、Eclipse 和来自 Google 应用程序的源代码。FindBugs 可以检测出一些相对有意义的问题，比如空指针异常、无限循环以及无意中访问到了一个对象的内部状态的问题。与许多其他静态分析工具不同的是，FindBugs 倾向于找到更少量的问题；但是这些找到的问题，大部分都是非常严重的。

FindBugs 比其他我们见过的工具都要易于配置，不过在实践中你一般无须像我们之前讨论过的其他工具一样调整这些规则。

你可以列出自己想要应用的每条规则，但是你不能配置 Maven 构建和 IDE 之间共享一个 XML 文件。 235

FindBugs 与一个 Ant 任务捆绑出现。你可以像下面展示的那样在 Ant 中定义 FindBugs 任务。FindBugs 需要涉及 FindBugs 的主目录，那就是 FindBugs 的二进制发行版解压出来的地方。为了让构建更便携，我们把 FindBugs 的安装文件存放在项目目录结构中的 *tools/findbugs* 目录下：

```
<property name="findbugs.home" value="tools/findbugs" />
<taskdef name="findbugs" classname="edu.umd.cs.findbugs.anttask.FindBugsTask" >
```

```
  <classpath>
    <fileset dir="${findbugs.home}/lib" includes="**/*.jar"/>
  </classpath>
</taskdef>
```

然后，要运行 FindBugs，你可以像在接下来的例子中展示的一样设置一个“findbugs”目标。需要注意的是 FindBugs 针对应用程序字节码运行，而不是源代码，所以你需要先编译源代码：

```
<target name="findbugs" depends="compile">
  <findbugs home="${findbugs.home}" output="xml" outputFile="target/findbugs.xml">
    <class location="${classes.dir}" />
    <auxClasspath refId="dependency.classpath" />
    <sourcePath path="src/main/java" />
  </findbugs>
</target>
```

如果你使用了 Maven 2，则不需要保留 FindBugs 安装文件的本地副本。你只需要在报告部分配置 FindBugs 为如下所示的样子：

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>findbugs-maven-plugin</artifactId>
      <version>2.3.1</version>
      <configuration>
        <effort>Max</effort>
        <xmlOutput>true</xmlOutput>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

或者对于 Maven 3 项目来说：

```
<project>
  ...
  <build>
    ...
      <plugins>
        ...
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-site-plugin</artifactId>
          <version>3.0-beta-2</version>
```

236

```
          <configuration>
            <reportPlugins>
              <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>findbugs-maven-plugin</artifactId>
                <version>2.3.1</version>
                <configuration>
                  <effort>Max</effort>
                  <xmlOutput>true</xmlOutput>
                </configuration>
              </plugin>
            </reportPlugins>
          </configuration>
        </plugin>
      </plugins>
    </build>
</project>
```

在这两种情况下，你可以通过运行 `mvn site` 或者 `mvn find bugs:findbugs` 来生成 XML 报告。XML 报告会被放在 *target* 目录下。

在本书写作之际，在 Gradle 中还没有对 FindBugs 的特别支持，所以你需要调用 FindBugs Ant 插件。

CodeNarc

CodeNarc 是一个用于 Groovy 代码的静态分析工具，类似于用于 Java 的 PMD。它检查 Groovy 源代码看是否有潜在的缺陷、差的编码行为和风格、过度复杂的代码，等等。典型的规则包括“常量 if 表达式”（Constant If Expression）、“空的 else 块”（Empty Else Block）、“GString 作为 Map 的键”（GString As Map Key）和“Grails 无状态服务”（Grails Stateless Service）。

对于基于 Ant 和 Maven 的项目来说，CodeNarc Ant 插件是最简单的选项（写这本书的时候 Maven 插件正在开发当中）。一个典型的和 Jenkins 结合使用的 Ant 配置如下所示：

```
<taskdef name="codenarc" classname="org.codenarc.ant.CodeNarcTask"/>
<target name="runCodeNarc">
    <codenarc ruleSetFiles="rulesets/basic.xml,rulesets/imports.xml"
              maxPriority1Violations="0">

        <report type="xml">
          <option name="outputFile" value="reports/CodeNarc.xml" />
        </report>
```

```
        <fileset dir="src">
          <include name="**/*.groovy"/>
        </fileset>
    </codenarc>
</target>
```

237 你可以简单地通过安装 CodeNarc 插件来把 CodeNarc 集成到一个 Grails 项目中：

```
$ grails install-plugin codenarc
```

这样就会配置 CodeNarc 去分析 Grails 应用程序代码中以及 *src/groovy* 和 *test* 目录下的 Groovy 文件。

Gradle 0.8 在代码质量插件中也为 CodeNarc 提供支持，你可以如下面这样在构建中做配置：

```
apply plugin:'code-quality'
```

这样就会默认使用 *config/codenarc/codenarc.xml* 中的 CodeNarc 配置文件。你可以重写 `codeNarcConfigFileName` 属性的值进行覆盖。

在这儿，你可以通过运行 `gradle codenarcMain` 或者更简单的命令 `gradle check` 来生成 CodeNarc 报告。

使用 Violations 插件报告代码质量问题

对于 Jenkins 最有用的代码质量插件之一就是 Violations 插件。这个插件不会分析项目源代码（你需要配置自己的构建来做这些），但它却在报告那些为每个构建和随着时间变化的趋势图所生成的代码质量度量方面做得非常好。

这个插件适合报告来自大范围的静态分析工具的代码质量度量，如下所示。

对于 Java

Checkstyle、CPD、PDM、FindBugs 和 jcreport

对于 Groovy

codenarc

对于 JavaScript

Jslint

对于 .NET

gendarme 和 stylecop

安装插件是非常直截了当的。你只需要到 Plugin Manager 界面然后选择 Jenkins Violations 插件即可。一旦你安装了插件并且重启了 Jenkins，就能在项目中使用它了。

Violations 插件本身并不生成代码质量度量数据——你需要配置自己的构建来做这些，就像先前的章节展示的一样。一个为 Maven 构建作业做了这些工作的例子说明参见图 9-3。需要注意的是在这儿我们直接调用 Maven 插件。我们也可以只运行 `mvn site`；但是假如我们只是对代码质量度量感兴趣，而不是对 Maven 生成的站点的其他元素感兴趣，则直接调用这些插件的结果就是更快的构建。

238

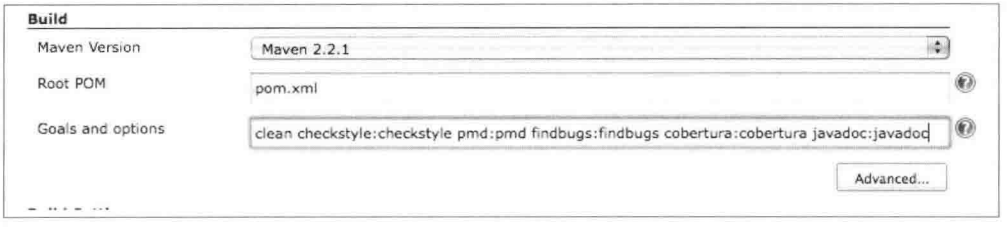


图 9-3　在Maven构建里生成代码质量报告

只要你把这个配置起来，就可以配置 Violations 插件去生成报告了；如果要求了的话，还可以基于报告结果生成触发通知。只需要去 Post-build Actions 界面，然后勾选 Report Violations 复选框即可。配置的细节会依赖于项目类型的不同而有差别。让我们先看看自由风格下的构建作业。

与自由风格下的构建作业一起工作

自由风格下的构建作业允许你灵活地做大多数的配置，同时也是你对于非 Java 项目的唯一可选项。

当你在一个自由风格的构建作业中使用 Violations 插件时，需要为你使用着的静态分析工具（见图 9-4）所生成的每一个 XML 报告指定路径。这个插件可以迎合来自同一个工具的几个报告，这对于 Maven 多模块项目来说是非常有用的——只需使用一个通配符表达式来验证你想要的那些报告（比如，`**/target/checkstyle.xml`）。

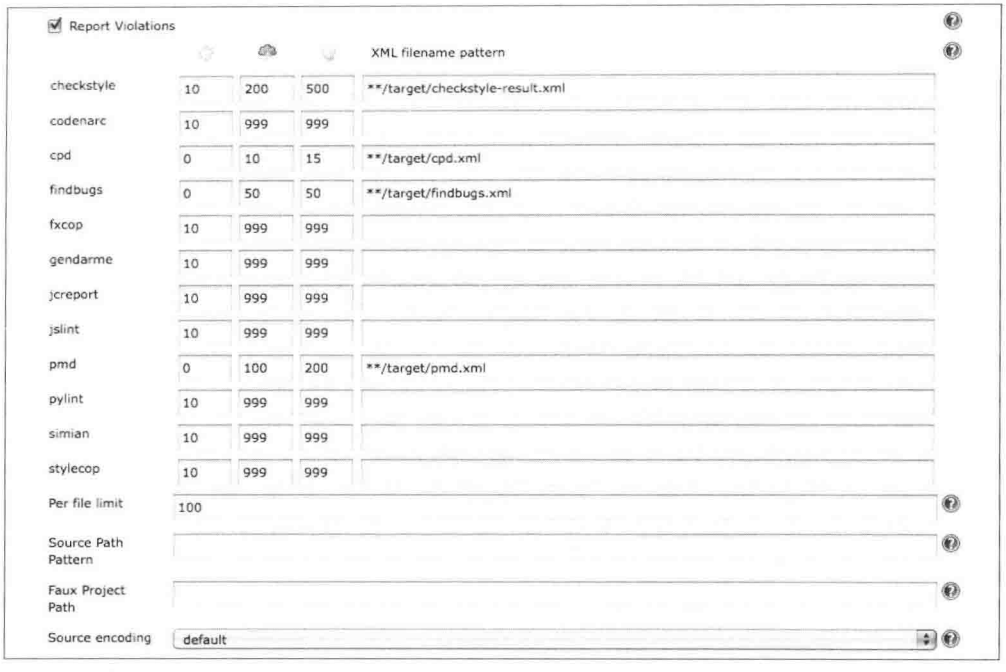


图 9-4　在一个自由风格的项目中配置Violations插件

Violations 插件会生成一个图来随着时间跟踪每一个类型的问题的数量（见图 9-5）。这个图会根据你正在跟踪的违规行为（Violations）的类型而显示不同的颜色线，以及一个最新结果的总结。

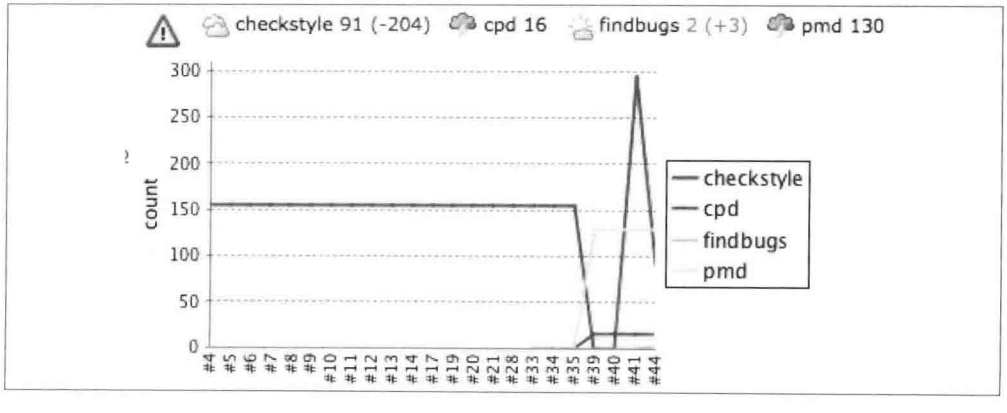


图 9-5　随着时间推移Violations的变化

你也可以单击这个图去深入到一个特殊的构建里。在这儿，你可以看到那个特殊构建产生的问题的数量（见图 9-6），有各种各样不合规类型的故障、严重程度和相关文件。

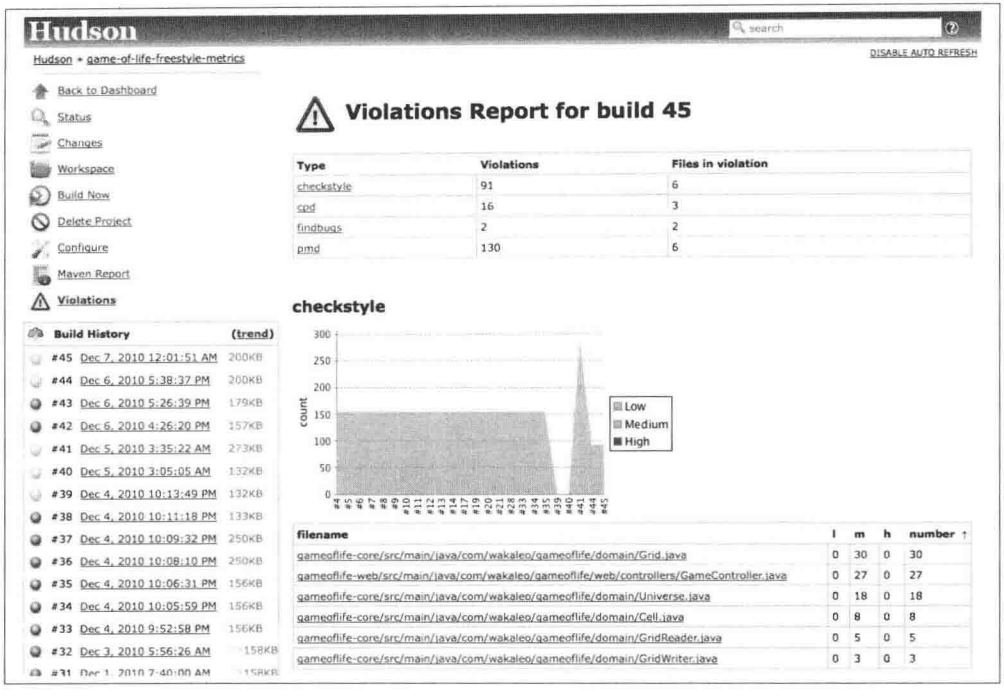


图9-6 一个给定构建的Violations

最后，你可以深入到一个特殊的类当中，可以显示出问题列表的详细信息，以及它们出现在源代码中的什么地方。

Violations 插件也支持关于代码质量的一个更前瞻性的管理。你可以使用代码质量分析报告的结果去影响在 Jenkins 仪表盘上的天气图标。这个天气图标通常跟先前那五个构建中失败了的构建数量有关，但是 Jenkins 也会考虑其他的因素，比如代码质量结果。在仪表盘上为一个项目显示一个多云的或者暴风雨的天气图标是一种更好的做法，这样能够引起关于那些简单依赖于在构建作业主页上的图和报告的代码质量问题的关注。

要想设置它，你需要回退到在 Post-build Actions 的 Report Violations 部分。在图 9-4 中的前三列展示了一个晴天图标、一个暴风雨图标和一个黄颜色的球。有晴天图标的那个就是在仪表盘主页上保持一个晴天图标不合规数量的上限。第二列，也就是暴风雨图标，是会导致一个暴风雨图标显示在仪表盘上的不合规数量。如果你有一个介于两者之间的值，就会得到一个多云的图标。

你可以为不同的工具设置不同的值。确切的阈值在各个团队、各个项目以及各个工具之间都多种多样。比如，Checkstyle 通常会比 FindBugs 或者 CPD 产生更多的问题，PMD

介于它们两者之间。你需要调整这些值来让基于你的代码运行的工具反馈你所期望的运行结果。

你可以借此进一步到第三列（有黄颜色球的那列）。这一列能让你设置可以引起构建宣称不稳定的 Violations 数量。记住，当构建变得不稳定的时候，Jenkins 会发出通知消息，所以这是一个更加积极主动的策略。

比如，在图 9-4 中，我们已经配置 Checkstyle Violations 的最小数量为 10，这就意味着晴朗的天气图标只会在有 10 个或者更少的 Checkstyle Violations 的时候才会出现。假如有 10 个以上，天气就会逐渐降级，直到 200 个 Violations 标识的时候，它会变成暴风雨天气。并且如果有 500 个或者更多的 Checkstyle Violations 的时候，项目就被标记为不稳定。

现在，让我们看一下和 PMD 一起工作的重复代码检测器——CPD 的配置。在这个项目中，我们对于重复代码采取了零容忍的策略，所以晴朗天气图标的值被设定为 0。暴风雨天气图标被设定为 10，所以如果有 10 个或者更多的连续的 Violations 的话，这个项目的天气指示器就会显示为暴风雨天气。如果这个项目有 15 个或者更多连续的 Violations 的话，它就会被声明为不稳定。

241 现在，在仪表盘主页，这个项目就会显示暴风雨天气图标和不稳定，尽管没有任何的测试错误出现（见图 9-7）。这个特殊构建之所以不稳定，是由于有 16 个 CPD Violations。另外，如果你把鼠标放在天气图标上，Jenkins 就会显示一些有关怎样计算这个特殊状态的更多细节。

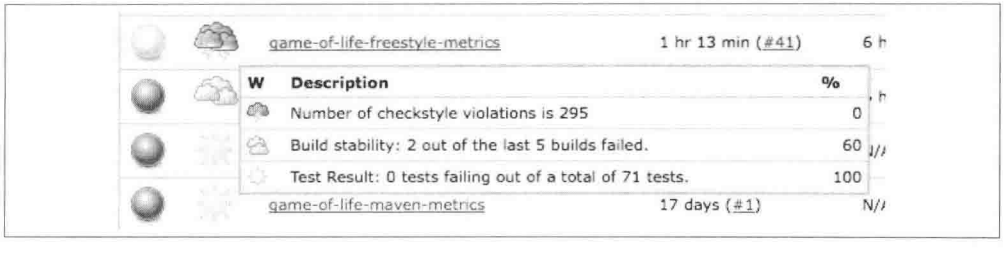


图 9-7 为一个自由风格的项目配置Violations插件

使用 Maven 构建作业

在 Jenkins 中的 Maven 构建作业使用项目中的 *pom.xml* 文件里的 Maven 约定和信息来使得它的配置更容易和更加轻量。当你在一个 Maven 构建作业中使用 Violations 插件的时候，Jenkins 使用这些约定来减少你为了配置这个插件所需要做的事情。你不需要告诉 Jenkins 哪里可以找到提供给静态分析工具（比如 Checkstyle、PDM、FindBugs 和 CPD）

使用的 XML 报告，因为 Jenkins 可以基于 Maven 约定和插件配置找到这些（见图 9-8）。假如你需要覆盖这些约定，可以选择使用在“XML filename pattern”下拉式列表中的 Pattern 选项，然后像你为自由风格构建作业所做的那样输入一个路径。

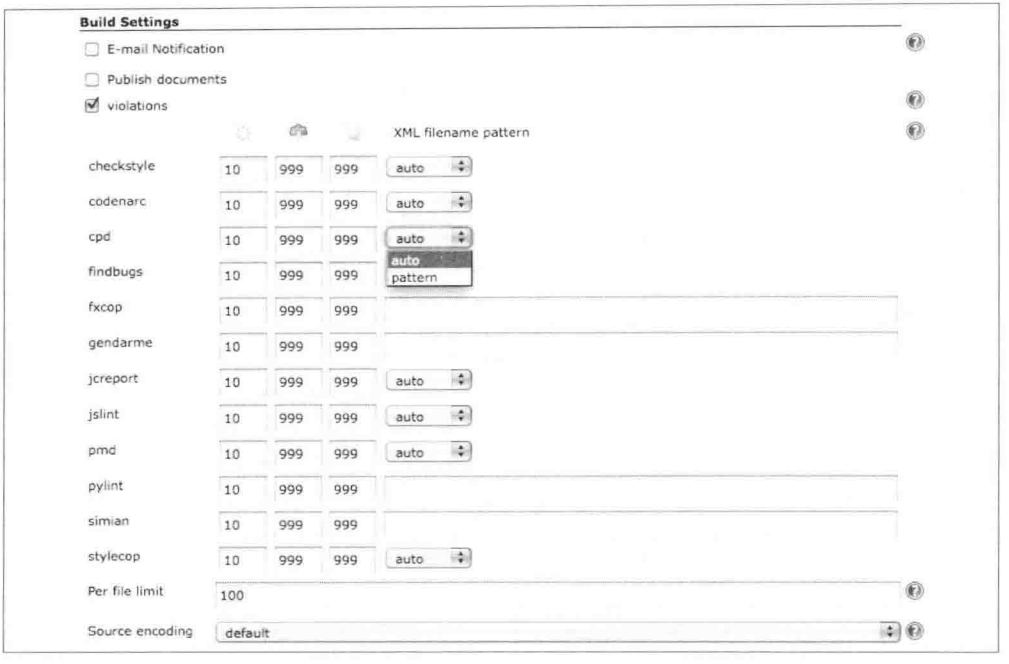


图 9-8　为一个Maven项目配置Violations插件

Violations 插件在多模块的 Maven 项目里也能很好地工作，但是在本书写作之际它需要做一点调整以获得最好的结果。Maven 构建作业能够读懂多模块项目的结构（见图 9-9）；此外，你可以深入到任何一个模块并且获得这个构建作业的构建结果的详细视图。



图9-9　Jenkins Maven构建作业能够读懂Maven的多模块结构

这是一个非常有用的功能，但它也意味着你需要做一点额外的工作来获得用于单个模块的 Violations 插件的好处。默认情况下，Violations 插件会显示代码质量度量的一个汇总视图，就像图 9-5 中的那样。你也可以单击 Violations 图，查看每一个模块的详细报告。

不过，为了让它能正确地工作，你需要为除了主项目之外的每一个模块单独地激活 Violations 插件。要做到这一点，你需要在 Modules 界面单击你想要配置的模块，然后单击“Configure”菜单。在这儿，你会看到一部分有用的配置选项（见图 9-10）。在这儿，你只需激活 Violations 选项，如果需要的话再配置一下阈值。从积极的一面来看，这意味着你可以为不同的模块定义不同的阈值。

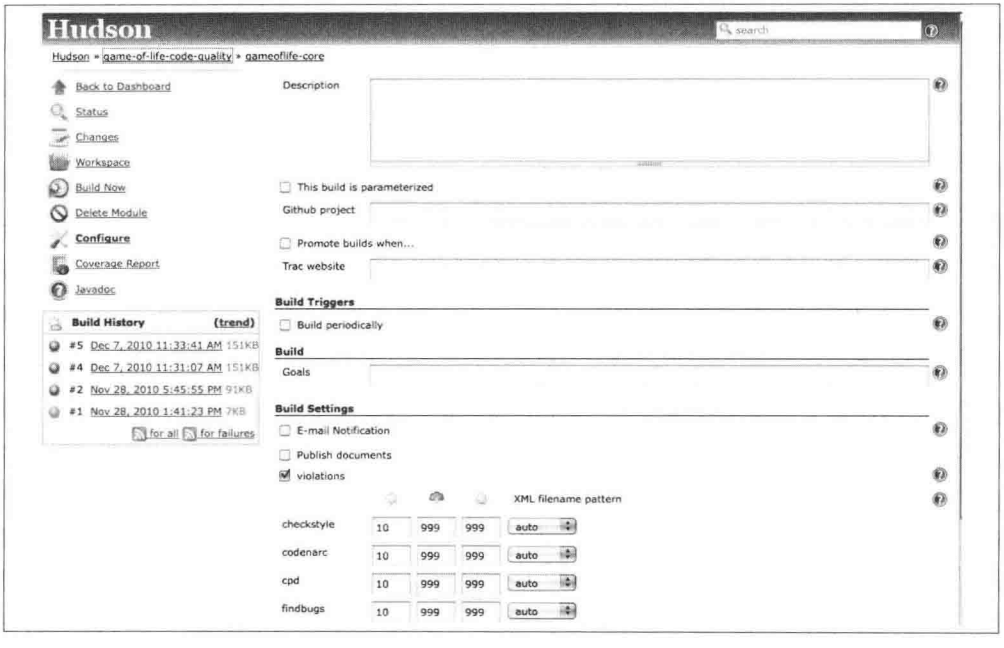


图 9-10　为每个单独的模块激活Violations插件

只要你做完了这些工作，当你单击位于项目构建作业主页面上的 Violations 汇总图时，Jenkins 就会列出每一个模块所对应的 Violations 图。

使用 Checkstyle、PMD 和 FindBugs 报告

你也可以单独地报告来自 Checkstyle、PMD 和 FindBugs 的结果。除了 Violations 插件，也有一些 Jenkins 插件会分别单独为这些工具生成趋势图和详细报告。我们接下来会看一下如何为 Checkstyle 做这些事情，但是同样的方法也适用于 PMD 和 FindBugs。你甚

至可以使用 Analysis Collector Plugin 在一个图中显示汇总的结果，它跟 Violations 插件生成的图很相似。

你可以以一种通用的方式通过 Plugin Manager 来安装这些插件。正如你所料，这些讨论中的插件分别叫作 Checkstyle 插件、PMD 插件和 FindBugs 插件。所有这些插件都使用 Static Analysis Utilities 插件，这个插件你也需要安装（图 9-11）。

244

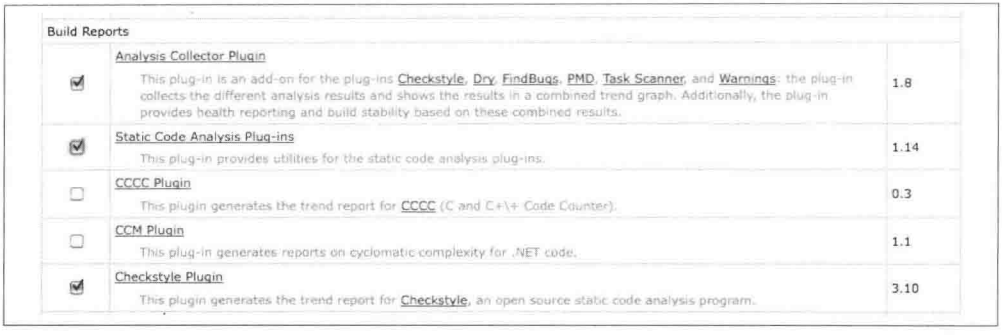


图 9-11　安装Checkstyle插件和Static Analysis Utilities插件

一旦你安装完这些插件，就可以在自己项目的配置当中设置做报告。勾选“Publish Checkstyle analysis results”复选框。在一个自由风格的构建中，你需要指定一个路径模式以便于找到 Checkstyle 的 XML 报告；在 Maven 2 构建中，Jenkins 会自己指出哪里可以找到它们。

这样就会提供基本的 Checkstyle 报告了，但是像往常一样，你可以通过单击 Advanced 按钮来更进一步地调整一些东西。在一个 Maven 2 构建当中，你可以配置健康阈值（多少数量的违规行为会导致这个构建从晴朗天气变为暴风雨天气），也可以过滤一些你想要包含在这个计算当中的有优先级的违规行为。例如，你可能只想为天气图标状态考虑那些有高优先级别的问题。

自由风格的构建还有更多的一些选项可供配置：特别地，如果违规行为的数量多于给定的数值或者新的违规行为数量多于给定的数值的话，则可以引起这个构建变得不稳定（黄球）或者失败（红球）（见图 9-12）。所以，在所举例子的配置当中，如果在一个构建中有超过 50 个任何优先级别的新的违反 Checkstyle 规则的行为时，这个构建就会被标记为不稳定。这对于 Checkstyle 来说当然有它的用处；它和 FindBugs 一起使用起来也很方便，在 FindBugs 里，高优先级的问题通常代表危险的和可能显示停止的 bug。



图 9-12　配置Checkstyle插件

现在，当构建运行的时候，Jenkins 会为 Checkstyle 中的违规行为生成一个趋势图以及详细的报告（见图 9-13）。从这儿开始，你就可以深入地看到每个优先级的、每个分类的、每个运行类型的、每个包的违规行为（Violations）等。

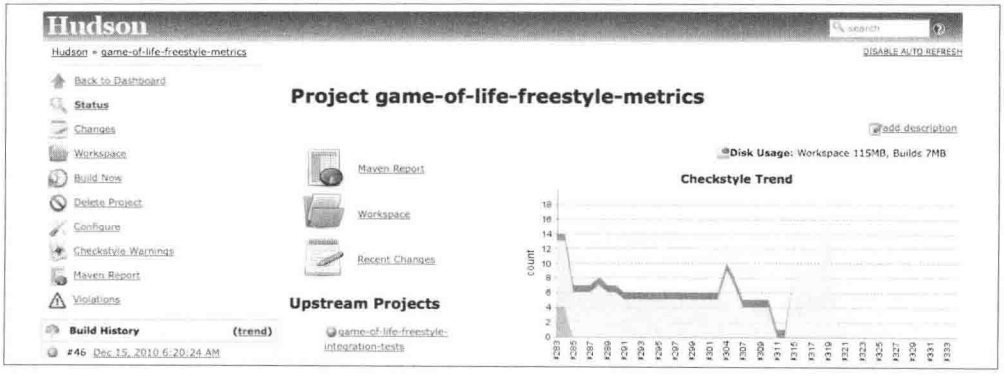


图 9-13　显示Checkstyle趋势图

就像我们之前提过的，同样的方法也适用于 PMD 插件和 FindBugs 插件。这些插件可以为特定工具的结果提供更有针对性的报告，也可以让你更多地控制这些违规行为对构建产物产生的影响。

报告代码复杂度

代码复杂度是代码质量的另一个非常重要的方面。代码复杂度可以用很多种方式来测量，但是一个常用的（易于理解的）复杂度度量就是 Cyclometric Complexity，它涉及通过一种方法来测量不同路径的数量。使用这个度量标准，复杂的代码通常会有大量的嵌套条件语句以及循环语句，这使得代码难以理解和调试。

还有一个代码质量理论把代码复杂度和代码覆盖率关联起来，给出代码可靠性的通用观点。这基于（非常好理解的）复杂且不充分地测试了的代码要比简单充分测试了的代码包含更多 bug 的观点。 246

Coverage Complexity Scatter Plot 就是被设计来让你在 Jenkins 构建中把此信息直观化（见图 9-14）。危险的复杂度和 / 或者未经测试的方法会出现在图的高处，而那些更精心编写的和经过充分测试的方法会出现在低处。

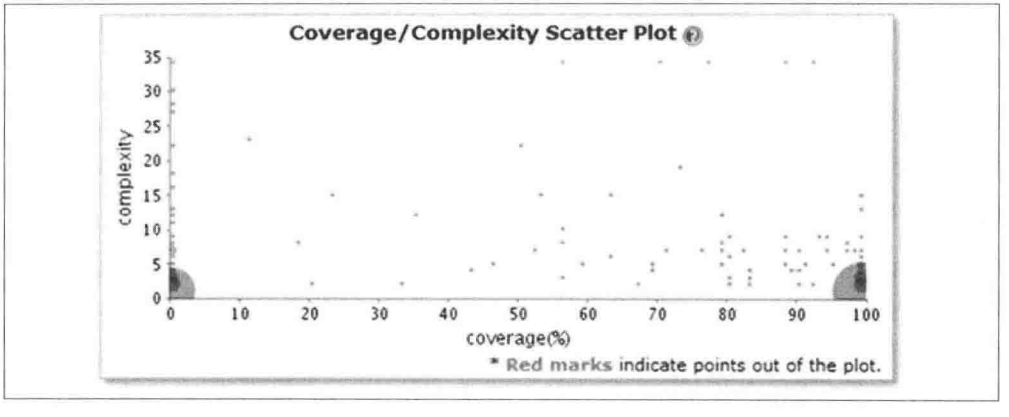


图 9-14　一个覆盖率/复杂度的散点图

这个散点图给了你一个依据于复杂度和测试覆盖率的代码状态的概览，但你也可以更进一步深入研究。如果你单击了这个图中的任意一点，就会看到相应的方法和它们的覆盖率以及复杂度（见图 9-15）。

247

Coverage / Complexity Scatter Plot

2 method(s) in the range of coverage (90%~100%) and complexity (5~9)

Method	Complexity	Coverage(%)	Total	Covered
createNextGeneration() : void	7	100	19	19
cellIsOutsideBorders(int,int) : boolean	5	100	3	3

图 9-15　可以单击图中的任意一点来进行进一步的研究

在本书写作之际，这个插件还需要 Clover，所以你的构建需要生成一个 Clover XML 覆盖率报告，而且你也应该已经安装和配置了 Clover Jenkins 插件（参见第 6 章的“使用 Clover 测量代码覆盖率”一节）。不过，对 Cobertura 和其他工具的支持正在计划当中。

报告未完成的任务

当提到代码质量的时候，静态分析并不是你可以使用的唯一工具。另一个关于你项目的常规健康状况的指标可以是 `FIXME`、`TODO`、`@deprecated` 和分散在源代码中的类似标签的数量。 如果这些东西很多的话，那就给出一个说明你的代码库还有很多未完成的工作的信号，而且不在一个非常确定的状态内。

Jenkins Task Scanners 插件能够让你在源代码中保持对这些种类的标签跟踪；而且如果有太多未完成的任务时，能视情况而定要不要在仪表盘上使用坏天气图标标记一个构建。

要配置这个，你需要安装 Static Analysis Utilities 插件和 Task Scanner 插件。一旦安装完成，就可以通过检查位于项目作业配置的 Build Settings 部分的“Scan workspace for open tasks”复选框来激活你项目中的这个插件。

配置 Task Scanner 插件是一件非常简单明了的事情（见图 9-16）。你只需简单地输入自己想要跟踪的标签；如果你觉得某些标签要比其他标签更重要的话，可以设置不同的优先级。在默认情况下，这个插件会扫描项目中的所有 Java 源代码，你也可以通过在扫描区域输入文件来重新定义这个行为。例如，在图 9-16 中，我们也检查 XML 和 JSP 文件的标签。

Advanced 按钮使你能够访问到一些更复杂的选项。可能对你来说最有用的就是 Health 阈值，它可以让你定义你所能容忍的问题的最大数量，超过这个数值，构建就不再被认为是“晴天”（sunny）了；还有“暴风雨天气”（stormy weather）状态的最小的数值。

这个插件生成一个图，按优先级展示标签趋势（见图 9-17）。如果你单击了 Open Tasks 报告，则也能看到按照 Maven 模块、包或者文件显示的任务分解，甚至列出了未完成的任务。

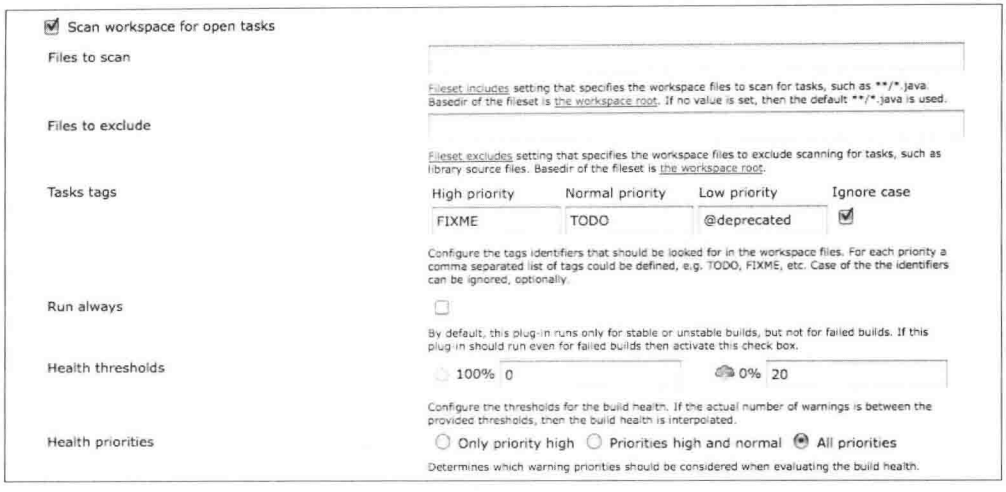


图 9-16　配置Task Scanner插件是非常简单明了的

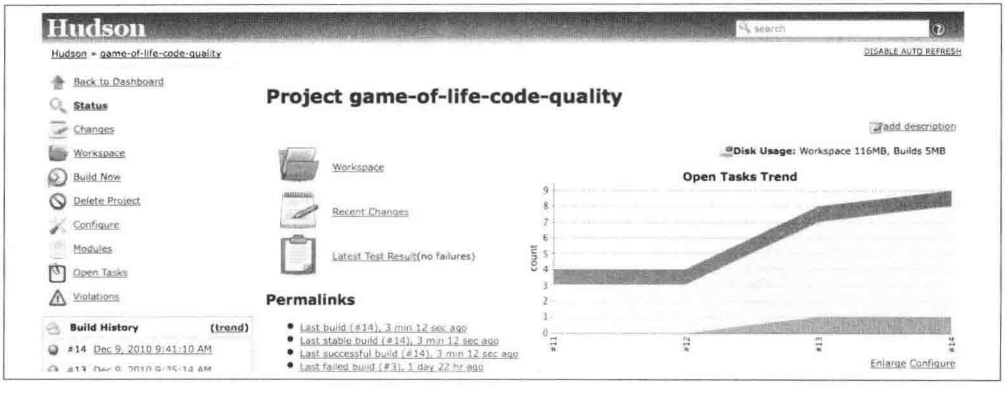


图 9-17　Open Tasks Trend图

集成 Sonar

Sonar（*http://www.sonarsource.org*）是一个可以把一系列的代码质量度量集中到一个单一的网站上的工具（见图 9-18）。它使用了几个 Maven 插件（Checkstyle、PMD、FindBugs、Cobertura 或者 Clover 等）来分析 Maven 项目，并生成一系列全面的代码质量度量报告。Sonar 报告代码覆盖率、规则一致性和文档编制，也会报告更高级别的度量标准（比如复杂度、可维护性，甚至技术债务）。你可以使用插件来扩展它的功能并且为其他语言提供支持（比如为 Groovy 源代码支持 CodeNarc）。各种各样的工具所使用的规则在 Sonar 的网站上集中管理和配置，被分析的 Maven 项目无须任何特别的配置。

这使得 Sonar 非常适合为 Maven 项目工作，因为 Maven 项目中你对 pom 文件的控制权限有限制。

Sonar 最常见的用途之一就是，Sonar 可以自动化地运行与你的 Maven 项目的代码质量相关的插件，然后把结果存储在一个关系数据库中。独立地运行着的 Sonar 服务器会接着分析和显示结果，如图 9-18 所示。

249

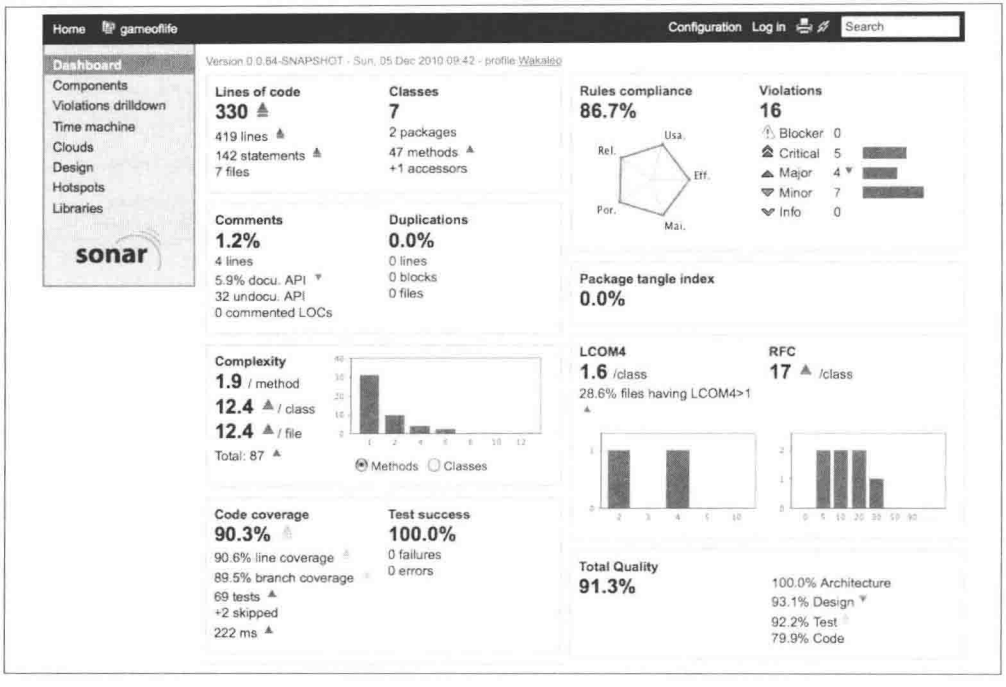


图9-18　通过Sonar进行代码质量报告

Jenkins 能很好地和 Sonar 集成。Jenkins Sonar Plugin 可以让你为自己的所有项目定义 Sonar 实例，然后在特定的构建里激活 Sonar。对于 Jenkins 实例来说，你可以在不同的或者同一台机器上运行 Sonar 服务器。唯一的约束就是 Jenkins 实例必须有访问 Sonar 数据库的 JDBC 权限，因为它会直接把代码质量度量注入数据库，而无须经过 Sonar 的网站（见图 9-19）。

Sonar 还提供给非 Maven 用户的一个 Ant 引导程序（在本书写作之际，市场上有一个 Gradle 引导程序）。

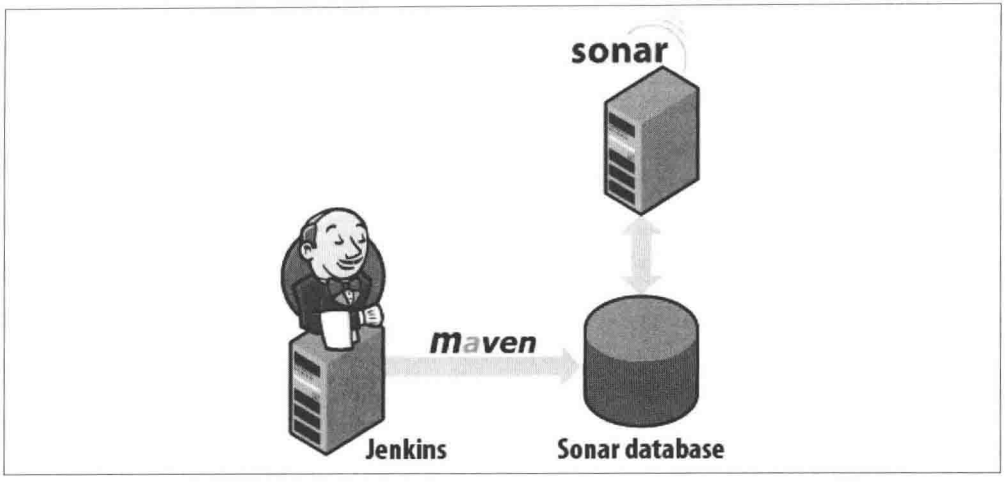


图 9-19　Jenkins和Sonar

你可以通过 Plugin Manager 以一种通用的方式安装这个插件。一旦安装完成，就可以在 Sonar 部分的 Configure System（配置系统）界面上配置 Jenkins Sonar 插件。这涉及定义 Sonar 实例——你需要多少 Sonar 实例，就可以配置多少。默认的配置是假设你正在运行一个带有默认的内嵌数据库的本地 Sonar 实例。这对于测试目的来说是很有用的，但并不是可扩展的。对于一个生产环境来说，通常你需要把 Sonar 运行在一个诸如 MySQL 或者 Postgres 这样的真实数据库上，而且你需要在 Jenkins 里配置 JDBC 连接到生产环境中的 Sonar 数据库。要做到这一点，你可以单击 Advanced 按钮，然后填入合适的域（见图 9-20）。

另外一件你需要配置的事情就是一个 Sonar 构建作业中什么时候开始 Sonar 构建。你通常需要配置 Sonar 和一个长时间运行的 Jenkins 构建作业一起运行，比如代码质量度量构建。一天内运行 Sonar 多于一次并没有用，因为 Sonar 以 24 小时为单位存储一次度量标准。默认配置是，当定期计划的构建或者手工构建触发一个作业的时候，就会在这个 Sonar 可用的构建作业中开始一个 Sonar 构建。

想要使用系统级别的配置选项激活你构建作业中的 Sonar 的话，只需要在 Post-build Actions 里检查 Sonar 选项（见图 9-21）。每当你的构建被前面定义的那些触发机制触发的时候，Sonar 就会运行起来。

通常你需要配置 Sonar 能定期运行，比如每个晚上跑一次或者每周跑一次。所以你可以在标准单元 / 集成测试构建作业上激活 Sonar，只需简单地添加一个日程表即可（见图 9-22）。这样就避免了作业之间重复的配置细节。或者，如果你已经有一个安排好了的以适当频率运行的构建作业（比如一个专用的代码质量度量构建），则你可以在这个构建

作业上激活 Sonar。

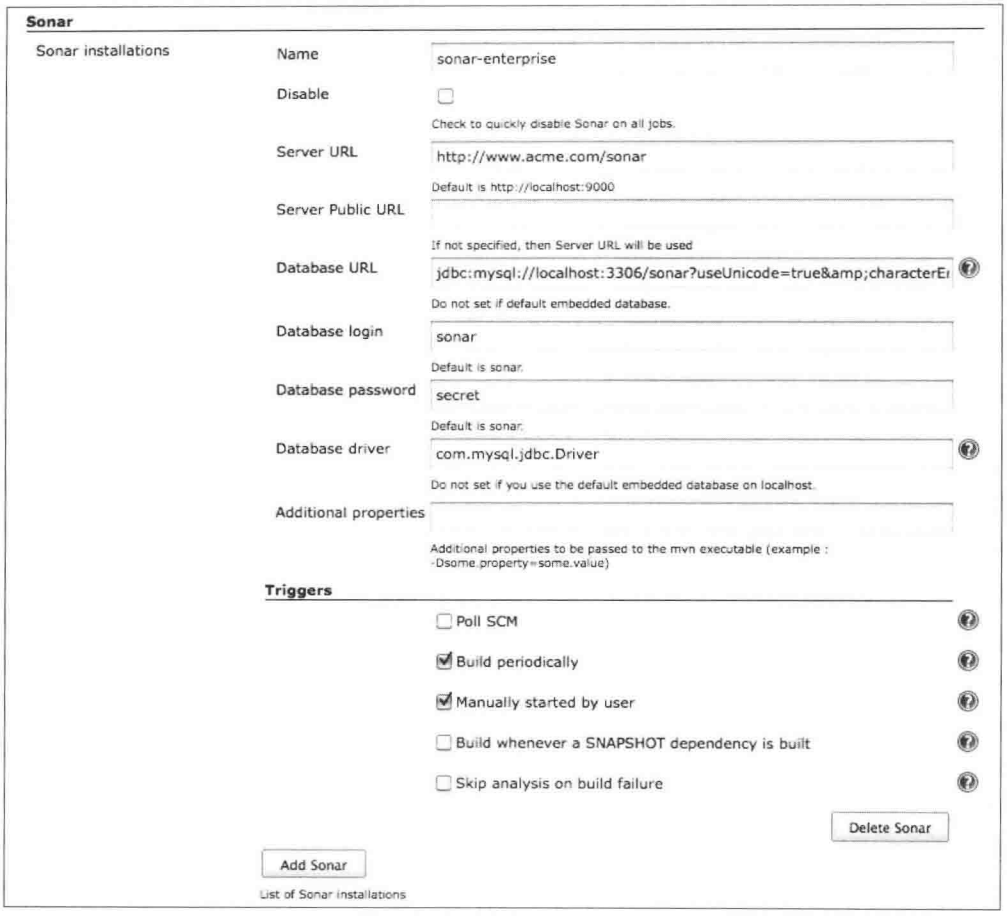


图 9-20　在Jenkins中配置Sonar

如果你单击 Advanced 按钮，则可以指定其他更复杂的选项，比如在一个独立的分支上运行 Sonar 构建，传递给 Maven 额外的命令行选项（比如额外的内存），或者覆盖默认的触发配置。

默认情况下，Sonar 即使在正常的构建失败时也会继续运行。这通常也是你所需要的，因为 Sonar 会记录构建和测试错误以及成功的结果。不过，如果需要的话，你可以在 Advanced 选项里停用这个选项。

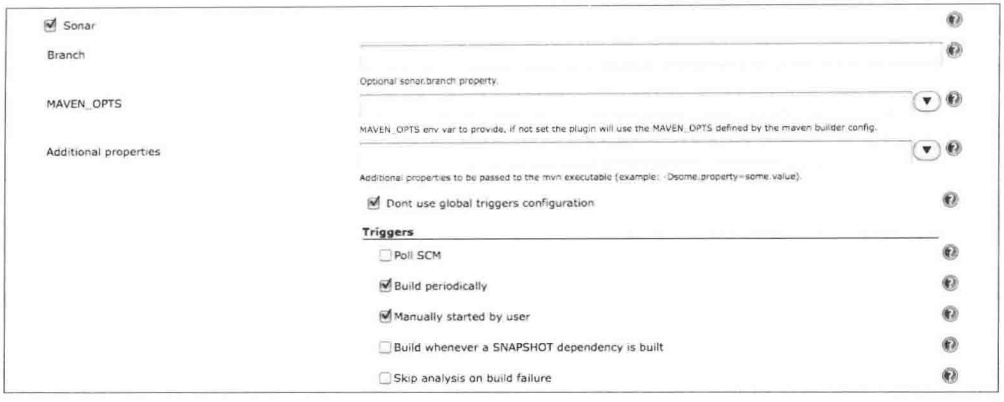


图9-21　在构建作业中配置Sonar

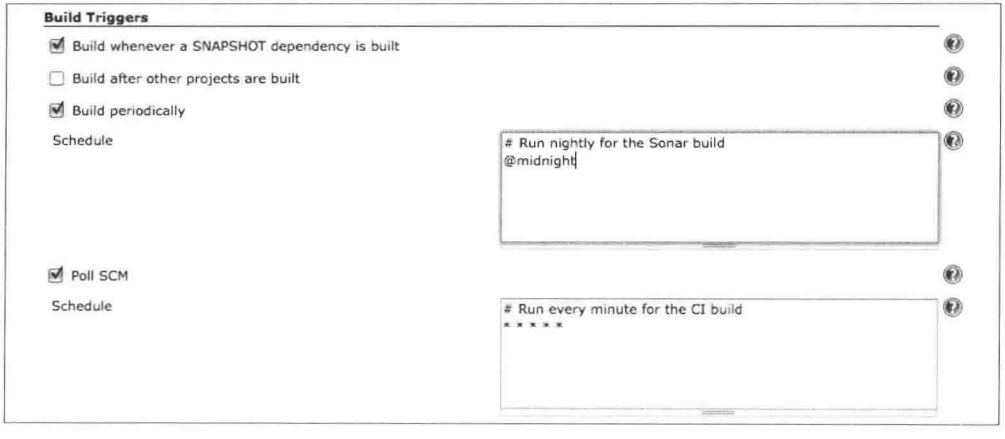


图 9-22　按时间安排Sonar构建

小结

代码质量是构建过程中非常重要的一部分，除了这里讲到的这些工具之外，Jenkins 还为一系列广泛的代码质量相关的工具提供支持。因此，Jenkins 会是你代码质量策略中关键的一部分。

第 10 章

高级构建

简介

在本章中，我们将关注一些更高级的构建作业配置。我们将讨论参数化构建，Jenkins 为用户提示额外的参数，可被传递到构建作业和多配置构建作业中；其虽然有大量的变化，但也可以运行一个单一的构建作业。我们将看一下如何并行执行构建作业和构建作业等待一到多个其他构建作业完成后继续执行，以及如何实现构建提示策略和构建管道，这样 Jenkins 不仅仅可用作构建服务器，还可以作为部署服务器。

参数化构建作业

参数化构建是很强大的，你可以在构建作业中添加各种其他维度。

Parameterized Build 插件可以让你为构建作业配置参数，可以当构建作业启动时由用户输入，或者（之后我们会看到）从其他构建作业中传入。

例如，你可能有一个部署构建作业，你想在启动构建作业时可以在一个下拉列表中选择目的环境；或你可能想指定要部署的应用版本号；抑或是当运行一个涉及 Web 测试的构建作业时，你可能想指定用于运行 Selenium 和 WebDriver 测试用例的浏览器；甚至是上传用于构建作业的文件。

注意，正确地分析和处理参数值是构建脚本的工作。Jenkins 只是为用户输入参数值和将这些参数写入构建脚本中提供了一个用户界面。

254 创建一个参数化构建作业

通过 Plugin Manager 界面，与以往一样安装 Parameterized Build 插件。当你安装完后，配置一个参数化的构建作业是非常简单的。只需勾选“This build is parameterized”选项并单击 Add Parameter 来增加一个新构建作业参数即可（见图 10-1）。你可以为任何类型构建添加参数，并且还可以根据需要为给定构建作业添加众多参数。

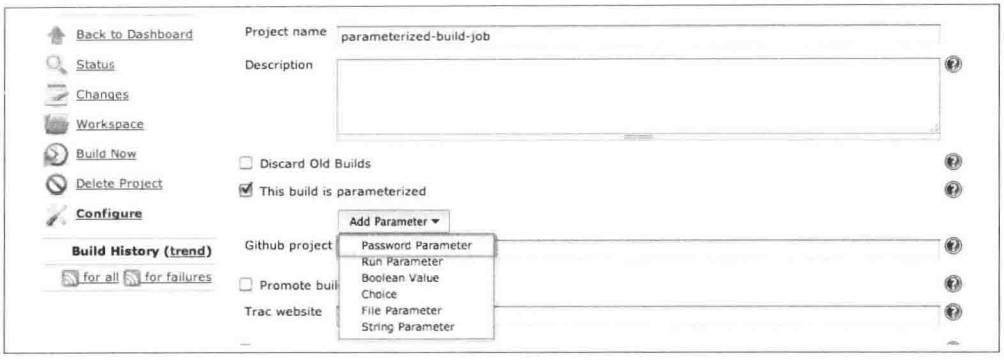


图10-1 创建一个参数化构建作业

将一个参数添加到构建作业中，只需在下拉列表中选择参数类型即可。然后配置参数细节（见图 10-2）。你有几种不同的参数类型可供选择，如字符串（Strings）、布尔（Booleans）和下拉列表（dropdown lists）。基于所选择的类型，需要输入的配置值略有不同，但其基本流程一样。所有参数类型（除了文件 File 参数外）均有一个名字和一个描述，而且一般情况下都有一个默认值。

255

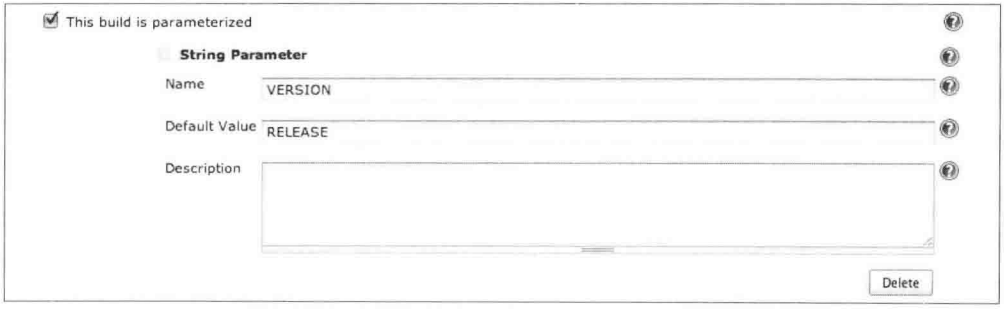


图10-2 在构建作业中添加一个参数

例如，在图 10-3 中，我们为一个部署构建作业添加一个叫作 `version` 的参数。Jenkins 会为用户显示默认值（RELEASE）；如果用户不做任何修改，这个值将会被使用。

当用户启动一个参数化构建作业（参数化构建作业经常是手动启动的）时，Jenkins 将会为用户显示一个可以为每个构建作业参数输入值的页面（见图 10-3）。

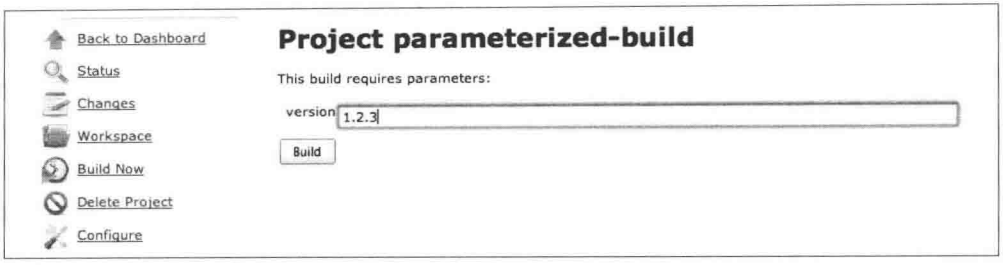


图10-3　在构建作业中添加一个参数

为构建适配参数化构建脚本

当你添加完参数时，可在构建脚本里配置使用它。选择一个好的参数名字是重要的，因为这也是一个变量名字，当 Jenkins 运行构建作业时会将它当作一个环境变量。为了说明这一点，考虑一下图 10-4 中最基本的构建作业配置，在这里我们只是简单地将构建参数打印到控制台。

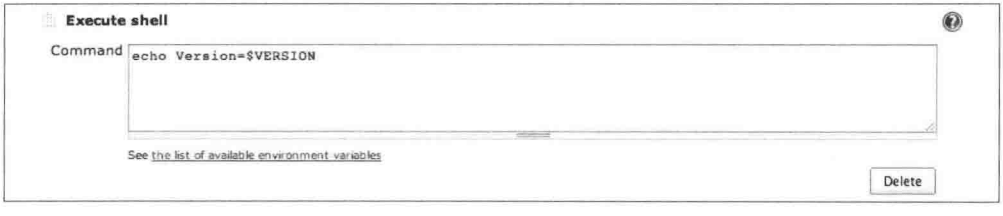


图10-4　展示一个构建参数

注意，为了使环境变量更便捷地跨操作系统使用，将它们全部大写是一个很好的做法。当运行这个脚本时，我们将在控制台得到以下输出结果：

```
Started by user anonymous
Building on master
[workspace] $ /bin/sh -xe /var/folders/y+/y+a+wZ-jG6WKHEm9KwnSvE+++TI/-Tmp-/
jenkins5862957776458050998.sh
+ echo Version=1.2.3
Version=1.2.3
Notifying upstream projects of job completion
Finished: SUCCESS
```

你还可以在构建脚本里使用这些环境变量。例如，在一个 Ant 或者 Maven 构建中，你可以使用特定的 env 属性来获取当前环境变量： 256

```
<target name="printversion">
  <property environment="env" />
  <echo message="${env.VERSION}"/>
</target>
```

另外一个方法是将参数作为属性值传递到构建脚本中。下面是一个有关 Maven POM 文件的比较复杂的例子。在这个例子中，Maven 的配置是部署一个指定 WAR 文件。我们在 `target.version` 属性中提供了用于部署的 WAR 文件的版本号，这将用于依赖声明，如下所示：

```
...
<dependencies>
  <dependency>
    <groupId>com.wakaleo.gameoflife</groupId>
    <artifactId>gameoflife-web</artifactId>
    <type>war</type>
    <version>${target.version}</version>
  </dependency>
</dependencies>
<properties>
  <target.version>RELEASE</target.version>
  ...
</properties>
```

当我们调用 Maven 时，参数作为一个构建属性传入（见图 10-5）。我们可以使用一个像 Cargo 这样的工具来做真正的部署——Maven 会从本地企业仓库管理器中下载 WAR 文件需要的版本，并将其部署到一个应用服务器上。

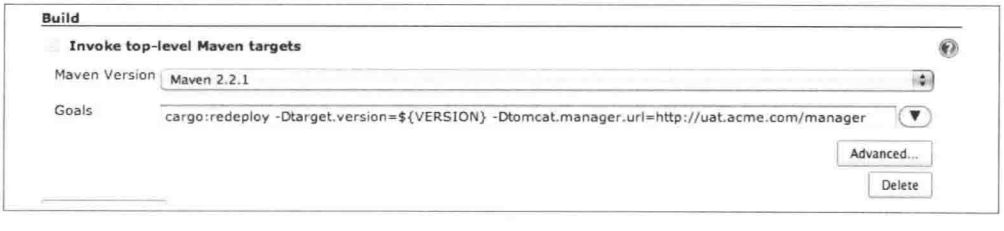


图10-5　在Maven构建作业中添加一个参数

简而言之就是，你如何将构建作业参数整合进构建中。除了普通字符串（String）参数，还有一些更复杂的参数类型，我们将在下面讨论（见图 10-6）。

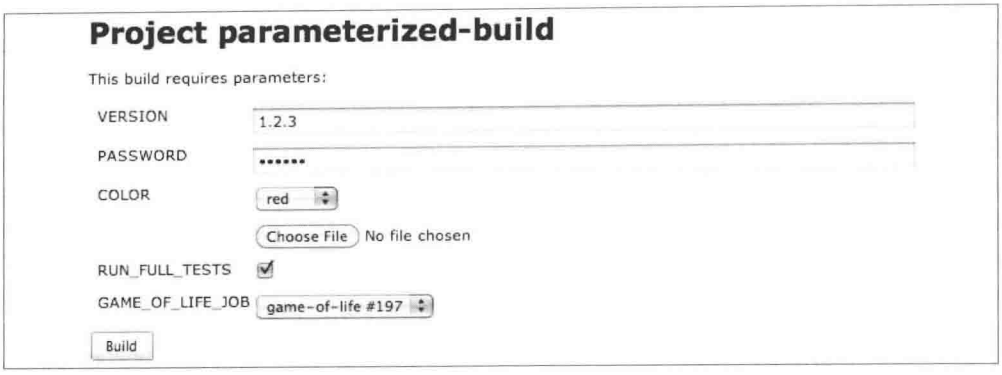


图10-6 多种有效参数类型

高级参数类型

Password（密码）参数和你想象的一样，它类似于字符串参数（除了它们是以密码字段进行显示外）。

在很多情况下，你要提出一组有限的参数选项。在一个部署构建中，你可能想让用户选择目标服务器的数量。或是你可能想展示一个支持验收测试的浏览器列表。选择(Choice)参数可以让你定义一组作为下拉列表显示的值（见图 10-7）。你需要提供一个包含值的列表，每行一个，起始值作为默认值。



图10-7 配置一个Choice参数

布尔（Boolean）参数和你想象的一样，是只有 `true` 或 `false` 值的参数。它们以复选框形式显示。

运行时（Run）参数和文件（File）参数是两个不同的参数类型，彼此行为有些不同。

运行时参数让你为给定的构建作业选择一个特定运行（或构建），如图 10-8 所示。用户从列表中选择构建运行数值。相应运行的构建 URL 存储于特定的参数中。

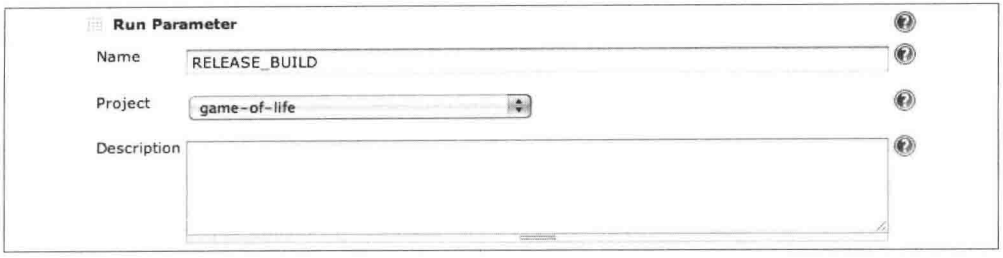


图10-8　配置Run Parameter

URL（类似于 *http://jenkins.myorg.com/job/game-of-life/197/*）可以用来获取信息或者是获取构建运行的构建产物。例如，你可以从之前的构建中获取 JAR 或 WAR 归档文件，并在一个单独的构建作业中进一步测试特定的二进制文件。例如，访问一个多模块 Maven 项目中以前构建的 WAR 文件，URL 会类似于这样：

```
http://buildserver/job/game-of-life/197/artifact/gameoflife-web/target/
      gameoflife.war
```

所以，使用图 10-8 中配置的参数，你可以用以下表达式访问 WAR 文件：

```
${RELEASE_BUILD}gameoflife-web/target/gameoflife.war
```

文件参数允许你上传一个文件到构建作业工作空间中，所以它可以用于构建脚本中（见图 10-9）。Jenkins 将把文件保存到项目的工作空间中的一个特定位置，你可以在自己的构建脚本中访问它。你可以使用 `WORKSAPCE` 变量来获取当前 Jenkins 工作目录，所以你可以通过表达式 `${WORK SPACE}/deploy/app.war` 使文件上传到图 10-9 中的位置。

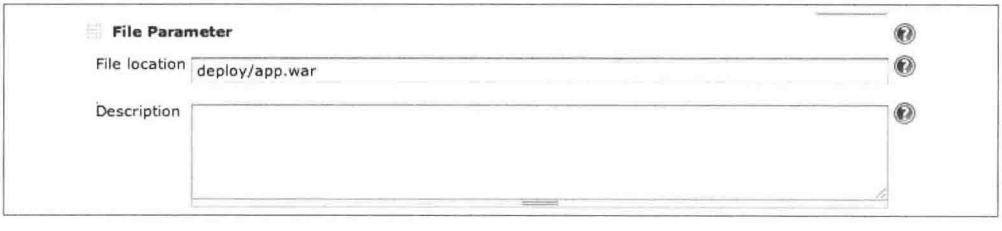


图10-9　配置File Parameter

259

使用 Subversion 标签构建

参数化触发器对于 Subversion 有额外的支持，允许你通过特定的 Subversion 标签构建。如果你想使用前一个构建作业生成的标签来运行发布版构建，则这是很有用的。例如，一个上游构建作业可能标记了一个特殊修订。或者你可能使用 Maven 标准版本发布流程

（参见本章的“通过 M2Release 插件管理 Maven 版本发布”一节）来发布新的版本。在这种情况下，带有 Maven 版本号的标签将在 Subversion 中自动生成。

这种方法对于被部署到指定平台之前需要被部分或完全重构是很有用的。例如，你可能需要使用不同的属性或者配置文件为不同的平台运行 Ant 或者 Maven 构建，所以各平台的特定配置文件可以嵌入到部署 WAR 或 EAR 文件中。

你可以通过使用“List Subversion Tag”参数类型中的标签来配置 Jenkins 构建（见图 10-10），只需提供指向项目的标签目录的 Subversion 仓库 URL 即可。



图10-10　为构建添加一个来自Subversion 标签的参数

当你运行这个构建时，Jenkins 会显示可供选择的标签列表（见图 10-11）。



图10-11　使用Subversion标签进行构建

使用 Git 标签构建

260

使用 Git 标签进行构建没有使用 Subversion 标签构建的方式简单（虽然你还可以使用一个参数来指定使用哪个标签）。事实上，由于 Git 的本质，当 Jenkins 从 Git 上获取到一份源码的拷贝时，它克隆了 Git 仓库，包括所有的标签。当在 Jenkins 服务器上已经有最新的版本时，你之后可以继续使用 `Git checkout <tagname>` 来验证标签版本。

在 Jenkins 中进行此设置，需要先为你的构建作业添加一个字符串参数（在这个例子中叫作 RELEASE——见图 10-12）。不像对于 Subversion 的支持，在此没有方法在下拉列表中列出生效的 Git 标签，所以用户需要知道他们发布用的标签名字。

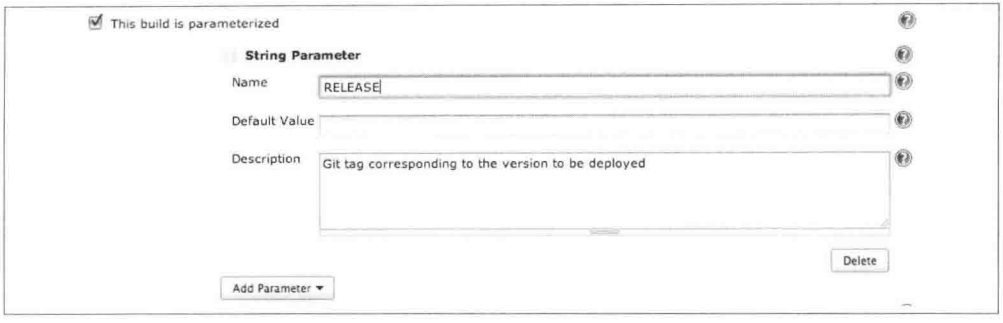


图10-12　为Git标签配置一个参数

当你添加完这个参数时，需要在仓库克隆到本地后校验标签的正确性。如果你有一个自由式构建，构建第一步是用命令行去调用 Git 检出 RELEASE 参数引用的标签（见图 10-13）。当然，更便捷的方式是写一个简单的 Ant 或 Groovy 脚本来做这些与操作系统无关的重复工作。

261



图10-13　使用Git标签进行构建

远程启动参数化构建作业

你还可以通过构建作业的 URL 地址远程启动参数化构建。参数化构建作业 URL 地址形式说明：

```
http://jenkins.acme.org/job/myjob/buildWithParameters?PARAMETER=Value
```

所以，上面这个例子，你可以这样触发构建：

```
http://jenkins.acme.org/job/parameterized-build/buildWithParameters?VERSION=1.2.3
```

当你使用一个 URL 来启动构建作业时，需要注意参数名字是大小写敏感的，并且值需要转义（就像任何的 HTTP 参数一样）。如果你正使用运行时参数，则你需要提供构建作业的名字和执行序列号（例如 `game-of-life#197`），而不只是执行序列号。

参数化构建作业历史

最后，记录一个特定参数化构建用到的参数是不可或缺的。例如，对于一个自动化部署构建作业，确切地知道部署的是哪个版本是很重要的。幸运的是，Jenkins 在构建历史中存储了这些值（见图 10-14），你可以在构建历史中看到。

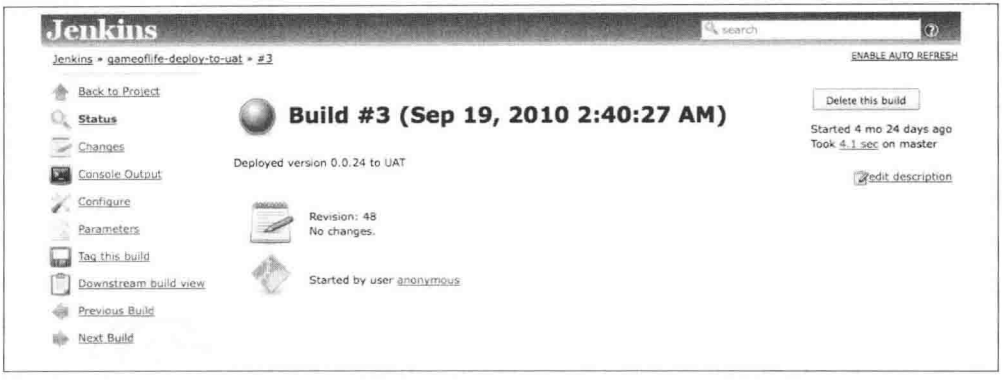


图10-14　Jenkins保存了每个构建使用的参数

参数化触发

262

当你从一个参数化构建作业中调用另一个构建作业时，能将当前构建作业的参数传递到新构建作业中是非常有用的。假设，你有一个应用需要在不同的数据库上进行测试。像我们之前看到的那样，你可以通过将目标数据库作为参数来设置参数化构建作业。你可能想设置一系列的构架，所有构建都需要这个参数。

如果你尝试在 Post-build Actions 部分使用常规的“Build other projects”选项，它不会正常运行。事实上，你不能用这种方式触发一个参数化构建。

然而，你可以用 Jenkins Parameterized Trigger 插件来实现它。通过这个插件可以配置构建作业，既可以触发参数化构建，还可以给这些构建传递任意参数。

当你安装完这个插件后，将发现构建作业配置界面有“Triggering parameterized builds on other projects”选项（见图 10-16）。这里有多种启动其他构建作业的方法。特别是，它支持启动一个后续构建作业，并传递当前构建作业的参数，而常规触发构建是不可能做到这一点的。看其是如何执行的最好方式是通过看具体例子。

在图 10-15 中，我们已经建了一个构建作业。这个构建有一个参数：DATABASE，用来在测试中指定对应的数据库。如我们看到的那样，用户会在构建启动时看到这项值的输入界面。

现在假设我们想在第一个构建作业完成后触发第二个构建作业以执行更全面的集成测试。然而我们需要第二个构建使用和第一个构建一样的数据库，可以通过第二个构建作业的参数化触发器进行配置（见图 10-16）。



图10-15　Jenkins存储了每个构建的参数

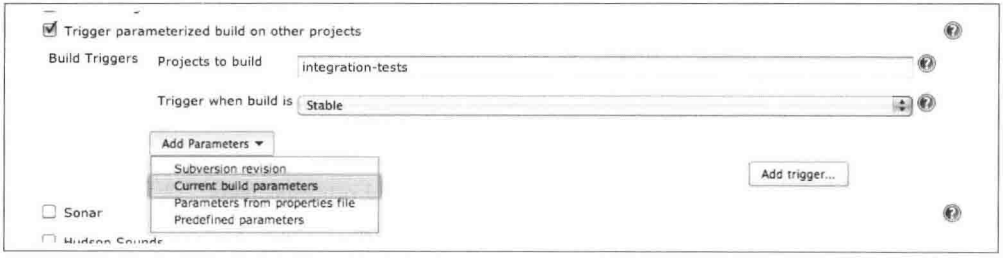


图10-16　为构建作业添加参数化触发器

在这种情况下，我们简单地传递了当前构建参数。第二个构建作业将在第一个构建完成后自动启动，并接收了由用户设置的 DATABASE 参数。你还可以调整触发规则，在 Jenkins 中设置什么时候触发构建。通常，你只是在一个构建成功地执行完后触发一个下游构建，但是通过 Parameterized Trigger，你还可以配置当一个构建不成功后触发其他构

建，当构建失败或无论第一个构建执行是否成功都触发后续构建。你甚至还可以为一个构建作业配置多个触发器。

你触发的构建作业必须也是参数化构建作业（如图 10-17 所示），并且你必须将它需要的所有参数传递过去。

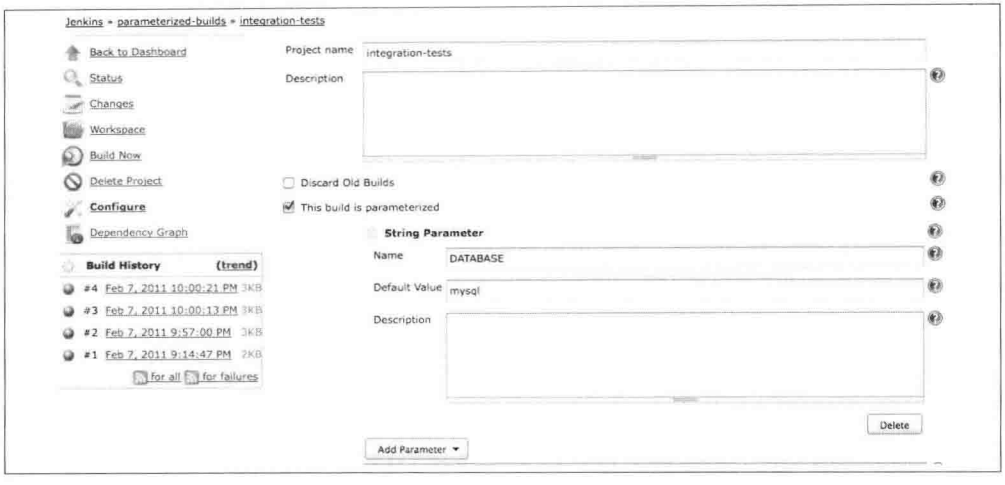


图10-17　触发的构建作业必须也是一个参数化构建作业

这个特性事实上还有更广泛的应用，而不仅是传递当前构建参数。你可以触发一个具有任意参数设置的参数化构建作业或使用被传递到当前构建的参数以及自己额外的参数。如果你有很多参数，则可以通过属性文件进行加载。在图 10-18 中，我们传递当前构建作业参数（示例中的 DATABASE 属性）和一个叫作 TARGET_PLATFORM 的额外参数。

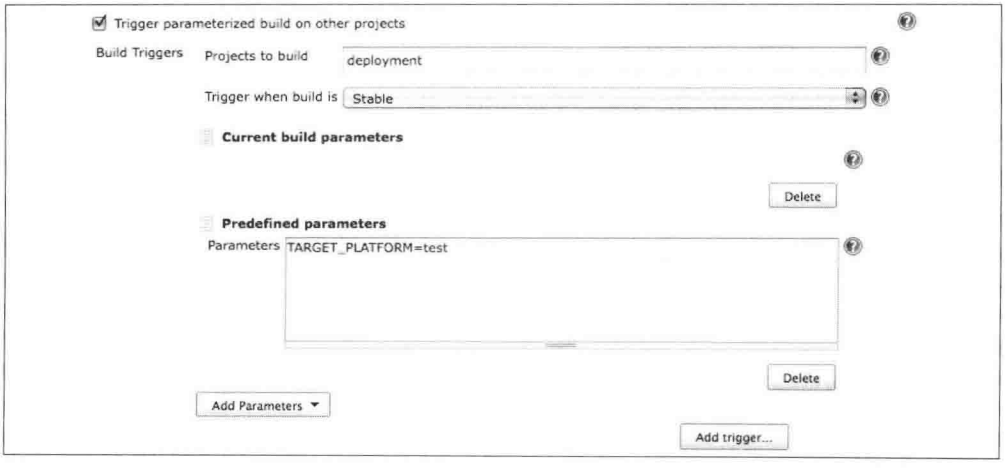


图10-18　为参数化构建作业传递预定义参数

多重结构的构建作业

多重结构的构建作业是 Jenkins 中非常强大的功能。一个多重结构的构建作业可以被理解为可以自动地运行所有可能的参数组合的一个参数化构建作业。它对于测试是非常有用处的，你可以在宽泛多样的条件下（浏览器、数据库等）使用一个单独的构建作业来测试自己的应用。

搭建多重结构构建

若想创建一个新的多重结构的构建作业，则只需在新建构建作业（New Job）页面选择该选项（见图 10-19）。

265

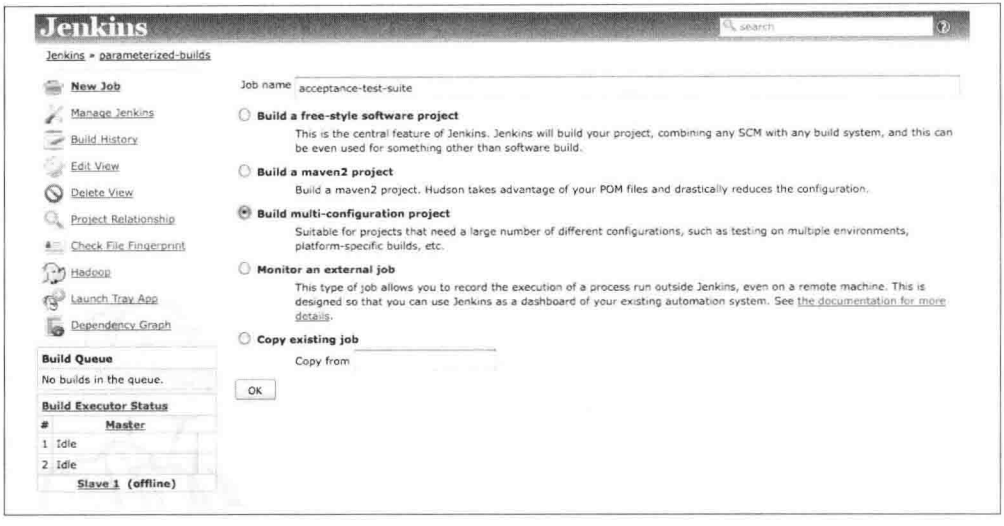


图10-19　创建一个多重结构的构建作业

多重结构的构建作业和其他构建作业一样，但是多了一个非常重要的元素：矩阵配置（Configuration Matrix，见图 10-20）。在这里可以定义不同的配置，用来执行你的构建。

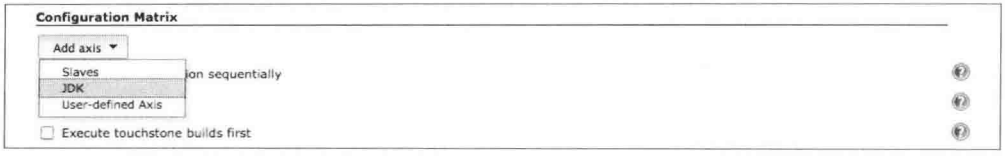


图10-20　为多重结构的构建作业添加坐标轴

你可以定义不同的配置选项轴，包括运行构建作业的各个从节点、各 JDK 版本，或用于

构建的自定义属性。例如，对于之前讨论的构建作业，我们可能想在不同的数据库和不同的操作系统上测试自己的应用。我们可以定义一个轴，轴定义了运行构建所需的各类操作系统的各从节点机器，其他轴定义了所有数据库。Jenkins 将在每个可能的数据库和操作系统上执行构建作业。

让我们看看可以定义的轴类型。

配置从节点轴

第一个选项是在不同从节点机器上并行执行的配置项（见第 11 章）。当然，通常在有一组从节点机器的情况下，你可以在其中任何一台上运行构建作业。但是如果有多种选 <266> 择，这样的情况也是合理的。例如，你可能想让测试用例运行在 Windows、Mac OS X 和 Linux 上。在这种情况下，你为从节点创建一个新的轴，如图 10-21 所示，可以有两种方式选择你想用的从节点：通过标签和独立节点。使用标签定义构建节点目录（例如，Windows 机器），无须将构建绑定到任意一台机器上。这是一个更复杂的选项，可以简单地根据需求扩展构建性能。然而，有时你可能想在一个特定机器上运行构建。在这种情况下，你可以使用“独立节点”（Individual nodes）选项并选择列表中的机器。

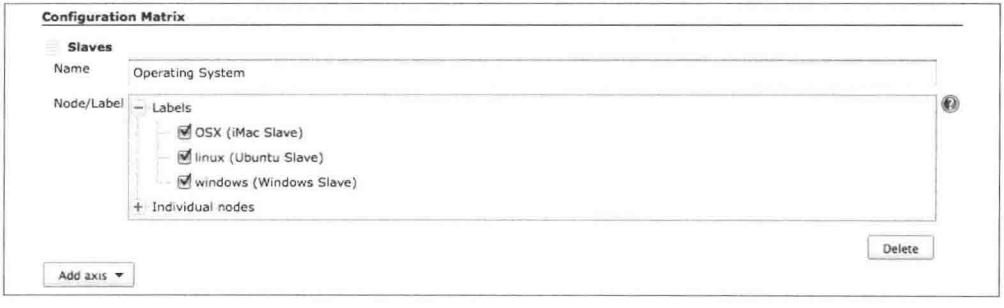


图10-21　定义从节点轴

如果需要更加灵活，则可以使用标签表达式（Label Expression），它可以在特定轴上定义哪个从节点用于构建，通过使用布尔表达式和逻辑运算符来组合的标签。例如，假如你已经根据操作系统（“windows”，“linux”）为从节点机器定义了标签，并安装了数据库（“oracle”，“mysql”，“db2”）。定义一个轴只在安装了 MySQL 的 Windows 机器运行测试用例，你可以使用 `windows && mysql` 这样的表达式。

我们将在第 11 章详细讨论从节点与分布式构建。

配置 JDK 轴

如果你的应用是部署到广泛的客户端，而只有有限的目标环境权限时，你可能需要用不同的 JDK 版本来测试应用。在这种情况下，在多结构构建中设置 JDK 轴是非常有用的。当你添加一个 JDK 轴时，Jenkins 将自动展示 JDK 版本列表（见图 10-22）。如果你需要额外添加 JDK，则只需到 Jenkins 配置界面进行添加。

267

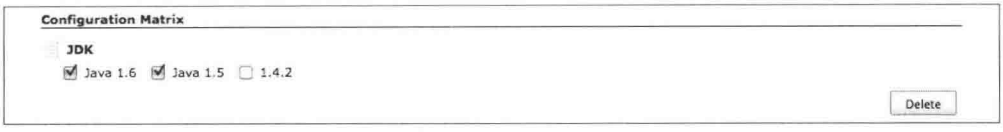


图10-22　定义JDK版本轴

自定义轴

第三种类型轴可以定义运行构建作业的不同方式（基于你定义的任意变量）。例如，你可能提供了需要做测试用的数据库列表，或用于 Web 测试的浏览器列表。这些就像是参数对于一个参数化构建作业的必要性一样，期望你提供一个所有可能值的完整列表，而不是提示你去输入一个值，Jenkins 将使用你提供的所有值来运行构建，见图 10-23）。

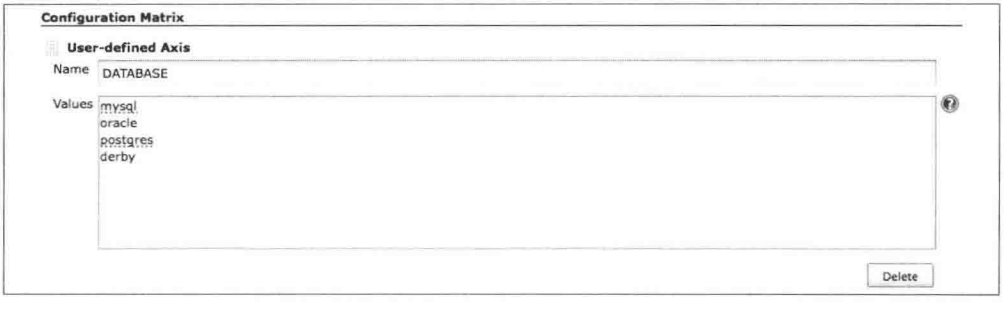


图10-23　用户自定义轴

执行多重结构构建

当你设置完轴时，可以像运行其他构建一样运行多重结构构建。只不过 Jenkins 将每种变量组合拆分为一个构建作业。Jenkins 将所有的组合结果显示到表格中（见图 10-24）。如果你单击任意一个球形图标，Jenkins 将显示对应构建的详细结果。

默认情况下，Jenkins 并行运行构建作业。然而有些情况并不适用。例如，很多 Java Web 应用使用 Selenium 或 WebDriver 针对本地由构建作业自动启动的 Jetty 实例测试。构建脚本需要进行特殊设置以便能在同一台机器上并行执行，以便避免端口冲突。在测

试运行时，如果测试用例中没有设计并行情况，并行数据访问会是另一个问题。如果你的构建没有做并行设计，通过勾选 Configuration Matrix（矩阵配置）底部的 Run each configuration sequentially 复选框选项，你可以强制 Jenkins 顺序地执行测试用例。

268

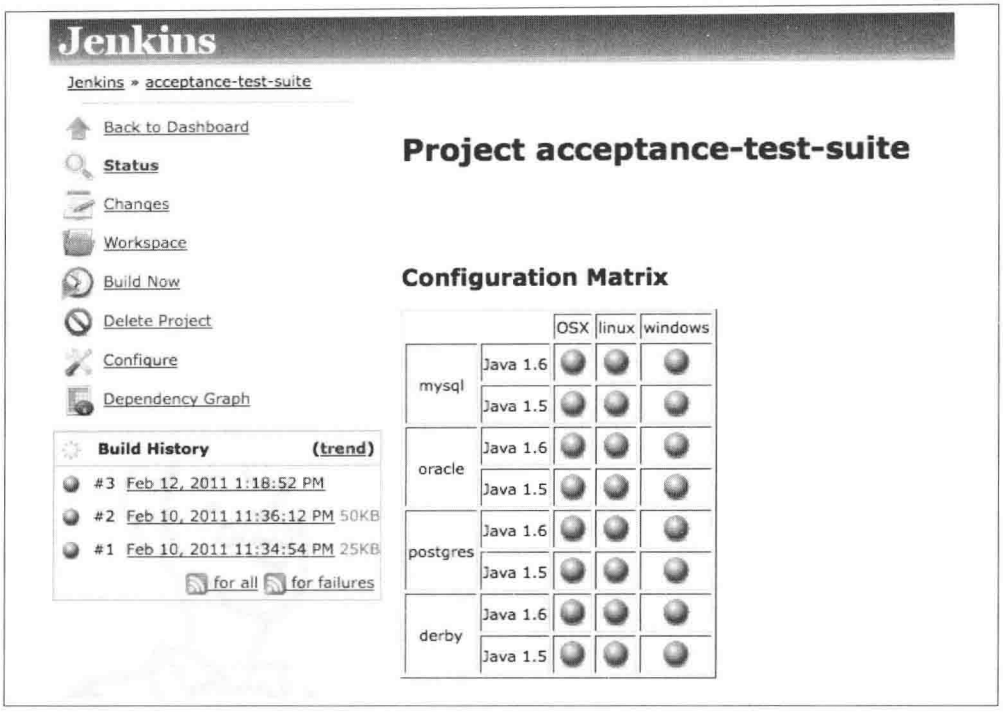


图10-24　多重构建产物

默认情况下，Jenkins 会运行所有轴上的可能组合。因此，在上面的例子中，我们有三种环境、两个 JDK 和四个数据库，一共会有 24 个构建。然而，在一些情况下，运行某些组合没有实际意义（也不可能）。例如，假设你有一个构建作业是自动化 Web 测试。如果用包括浏览器的轴（Firefox、IE、Chrome 等）和包含操作系统的轴（Linux、Windows、Mac OS），那么在 Linux 或 Mac OS 上运行 IE 测试是毫无意义的。

Combination Filter（组合过滤器）选项可以设置哪些变量组合是有效的规则。此项是可以使用你定义在各轴上的变量名字组成的 Groovy 布尔表达式。表达式为 true 时，构建就会执行。例如，假设你有一个在不同浏览器和不同操作系统运行的 Web 测试构建作业（见图 10-25）。这项测试需要在 Windows、Mac OS X 和 Linux 上运行 Firefox、IE 和 Chrome，但是 IE 只运行在 Windows 上，Chrome 不在 Linux 上运行。



图10-25　配置组合过滤器

设置组合过滤器，我们使用如下的表达式：

```
(browser=="firefox")
|| (browser=="iexplorer" && os=="windows")
|| (browser=="chrome" && os != "linux")
```

这样就会只有正确的浏览器和操作系统的组合才会被执行（见图 10-26）。执行的构建用正常颜色显示，跳过的构建用灰色显示。

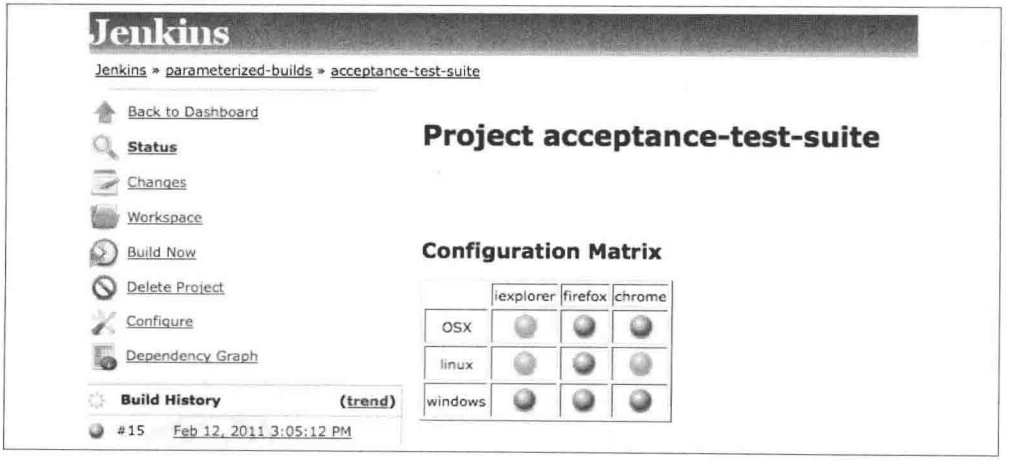


图10-26　使用组合过滤器的构建结果

另一个使用构建过滤器的原因是在一段合理的时间内，有太多的有效组合执行构建。这种情况下，最好的解决方案可能是升级你的构建服务器。次优的解决方案是可以执行一部分组合构建，在晚上执行整个组合集构建。你可以使用特殊 index 变量来实现。例如，确保两个之中只有一个真正执行了，包含的表达式是（index%2 == 0）。

你可能还想让某些构建在其他构建之前执行，作为一个状态检查。例如，你可能想首先为应用运行默认（理论上最可靠）配置，然后再继续运行更复杂的组合。要实现这一点，你可能使用“Execute touchstone builds first”选项。在这里，输入一个过滤值（如图 10-25 所示）来定义首先执行的构建。你还可以指定当那些构建执行成功后或都不成功才可以处理对应的构建作业。当这些构建都如期完成时，Jenkins 将处理其他组合。

270

自动生成 Maven 构建作业

（由 Evgeny Goldin 提供）

像前一节提到的那样，你的 Jenkins 服务器负载的构建作业数量是变化的。随着构建作业数量的增长，在 Jenkins 仪表盘中查看构建变得不是很容易了，配置它们也变得更难。想象一下对 20 ~ 50 个 Jenkins 作业进行逐个配置！此外，那些构建作业中许多可能有普通的配置元素，诸如 Maven 目标或构建内存设置，会导致重复配置和更高的维护开销。

例如，如果你决定为发布作业运行 `mvn clean install` 来代替 `mvn clean deploy` 命令并切换到备选部署方法，比如 Artifactory 提供的插件（*http://wiki.jenkins-ci.org/display/JENKINS/Artifactory+Plugin*），则只能查看所有相关作业和手动更新。

Jenkins 是一个简单便捷的工具，它将所有定义文件都存储在磁盘上。事实上你可以到构建目录 *.jenkins/jobs* 下直接更新 *config.xml*。尽管这种方法可以运行，但它还不是很理想的方法，因为它需要大量手动地对 Jenkins XML 文件进行修改、替换。

还有第三种方式来达到大规模作业更新：使用一些定义文件自动化生成配置文件。Jenkins 的 Maven 插件（*http://evgeny-goldin.com/wiki/Maven-jenkins-plugin*）就是这么做的，所有作业使用标准的 Maven 定义 XML 文件，全部保存在一个 *pom.xml* 文件中。

配置作业

271

当用 Maven Jenkins 插件配置单个作业时，你可以定义所有常用的 Jenkins 配置元素，如 Maven 目标、POM 位置、仓库 URL、电子邮件地址、日志文件保留天数等。这个插件尽可能地让你以接近于 Jenkins 常规手动配置作业的方式进行操作。

让我们看一个 Google Guice（*http://code.google.com/p/google-guice/*）构建作业：

```
<job>
        <id>google-guice-trunk</id>
        <description>Building Google Guice trunk.</description>
        <descriptionTable>
            <row>
                <key>Project Page</key>
                <value>
                    <a href="http://code.google.com/p/google-guice/">
                        <b><code>code.google.com/p/google-guice</code></b>
                    </a>
                </value>
                <escapeHTML>false</escapeHTML>
                <bottom>false</bottom>
            </row>
        </descriptionTable>
        <jdkName>jdk1.6.0</jdkName>
        <mavenName>apache-maven-3</mavenName>
        <mavenOpts>-Xmx256m -XX:MaxPermSize=128m</mavenOpts>
        <daysToKeep>5</daysToKeep>
        <useUpdate>false</useUpdate>
        <mavenGoals>-e clean install</mavenGoals>
        <trigger>
            <type>timer</type>
            <expression>0 0 * * *</expression>
        </trigger>
        <repository>
            <remote>http://google-guice.googlecode.com/svn/trunk/</remote>
        </repository>
        <mail>
            <recipients>jenkins@evgeny-goldin.org</recipients>
        </mail>
</job>
```

这个作业使用了如 `<jdkName>`、`<mavenName>` 和 `<mavenOpts>` 标准配置。代码从 Subversion 仓库（在元素 `<repository>` 定义）抓取出，还有一个每晚 0：00 定时运行的 `<trigger>`。电子邮件通知将发送到定义在 `<mail>` 中的那些人。这个配置还可在每个作业自动生成的描述表格中添加一个返回到项目页面的链接。

Jenkins 服务器生成的作业如图 10-27 所示。

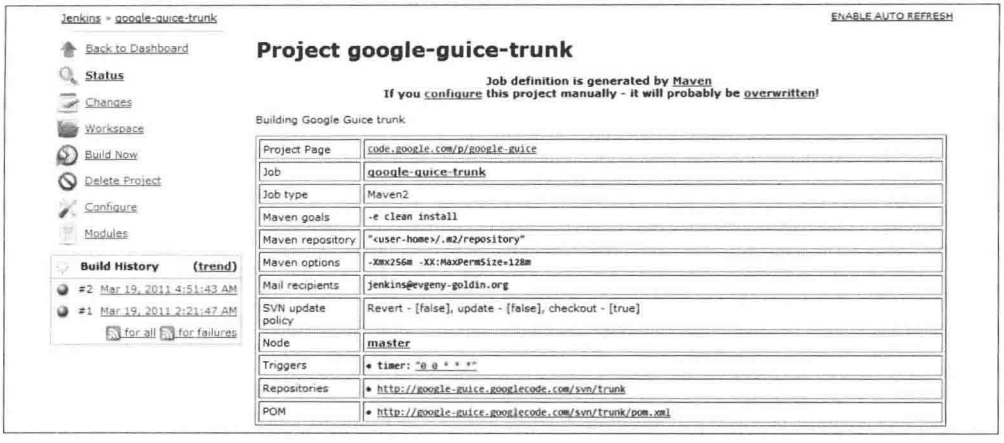


图10-27　由Maven Jenkins插件生成的作业

这是另一个在 GitHub 上构建 Jenkins 主分支的作业：

```
<job>
    <id>jenkins-master</id>
    <jdkName>jdk1.6.0</jdkName>
    <numToKeep>5</numToKeep>
    <mavenName>apache-maven-3</mavenName>
    <trigger>
        <type>timer</type>
        <expression>0 1 * * *</expression>
    </trigger>
    <scmType>git</scmType>
    <repository>
        <remote>git://github.com/jenkinsci/jenkins.git</remote>
    </repository>
    <mail>
        <recipients>jenkins@evgeny-goldin.org</recipients>
        <sendForUnstable>false</sendForUnstable>
    </mail>
</job>
```

它可以生成如图 10-28 所示的作业。

这个插件的文档（*http://evgeny-goldin.com/wiki/Maven-jenkins-plugin#.3Cjob.3E*）中有所有配置的详细参考内容。

使用继承复用作业配置

使用集中配置能生成 Jenkins 作业（如 Maven POM），这解决了一次创建、更新大量

作业的问题。所有需要你做的只是修改作业定义，使用 Manage Jenkins → “Reload Configuration from Disk” 来重新运行插件和加载定义更新。这种方法也便于将你的作业配置存储到版本控制系统中，进而更容易跟踪到构建配置的变更。

273

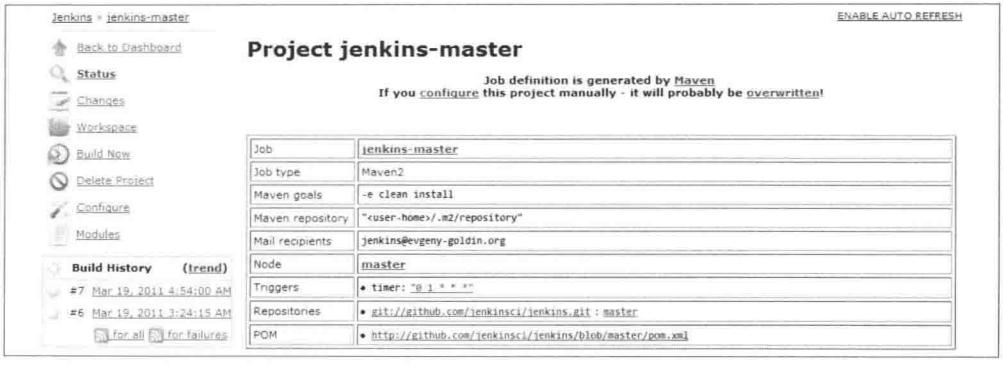


图10-28 Jenkins主作业

但是我们仍然没有解决共享大量定义文件的作业维护问题，如 Maven goals、电子邮件收件人或代码仓库 URL，为此，Maven Jenkins 插件提供了作业继承，如下例所示：

```
<jobs>
  <job>
    <id>google-guice-inheritance-base</id>
    <abstract>true</abstract>
    <jdkName>jdk1.6.0</jdkName>
    <mavenName>apache-maven-3</mavenName>
    <daysToKeep>5</daysToKeep>
    <useUpdate>true</useUpdate>
    <mavenGoals>-B -e -U clean install</mavenGoals>
    <mail><recipients>jenkins@evgeny-goldin.org</recipients></mail>
  </job>

  <job>
    <id>google-guice-inheritance-trunk</id>
    <parent>google-guice-inheritance-base</parent>
    <repository>
      <remote>http://google-guice.googlecode.com/svn/trunk/</remote>
    </repository>
  </job>

  <job>
    <id>google-guice-inheritance-3.0-rc3</id>
    <parent>google-guice-inheritance-base</parent>
    <repository>
```

```
      <remote>http://google-guice.googlecode.com/svn/tags/3.0-rc3/</remote>
    </repository>
  </job>

  <job>
    <id>google-guice-inheritance-2.0-maven</id>
    <parent>google-guice-inheritance-base</parent>
    <mavenName>apache-maven-2</mavenName>
    <repository>
      <remote>http://google-guice.googlecode.com/svn/branches/2.0-maven/
      </remote>
    </repository>
  </job>
</jobs>
```


在这个配置中，*google-guice-inheritance-base* 是一个抽象父作业，持有所有通用属性：JDK 名字、Maven 名字、日子保存天数、SVN 更新规则、Maven goals 和邮件收件人。接下来的三个作业是非常短的，仅仅指定其继承的 `<parent>` 作业并添加任何缺少的配置（示例中是仓库 URL）。当生成后，其自动继承所有父作业的属性。

所有继承的属性可以被重写，如 *google-guice-inheritance-2.0-maven* 作业，Maven 2 被 Maven 3 代替。如果你想“撤销”一个继承的属性，就需要用空值重写它。

作业继承是非常强大的，可以让作业根据各种目的、各种类别形成继承组。你可以分组 CI，用夜间或是发布作业的方式，将共享执行触发器、Maven goals 或邮件收件人集中到父作业中。这种借鉴于面向对象思想的方法可解决维护作业间共享的大量属性维护工作的问题。

插件支持

除了配置作业和重用它的定义外，你还可以为 Jenkins 插件应用特殊支持。现在，提供了简化的 Parameterized Trigger 和 Artifactory 插件，计划在未来的版本中支持其他流行的插件。

下面是一个使用 Parameterized Trigger 插件调用作业的示例。如果你已经安装过这个插件，使用此配置：

```
<job>
    <id>google-guice-inheritance-trunk</id>
    ...
    <invoke>
        <jobs>
```

```
            google-guice-inheritance-3.0-rc3,
            google-guice-inheritance-2.0-maven
        </jobs>
    </invoke>
</job>

<job>
    <id>google-guice-inheritance-3.0-rc3</id>
    ...
</job>

<job>
    <id>google-guice-inheritance-2.0-maven</id>
    ...
</job>
```


<invoke>元素可以在当前作业成功执行后调用其他作业。你可以用这种方式创建一个作业的管道，来确保管道内的每个作业都是逐个调用的。注意，如果在调用时有多个Jenkins有效执行器，指定的作业将并行启动运行。对于串行执行，你需要用<invoke>连接每个上游作业和下游作业。

默认情况下只当前作业运行结果是稳定状态时会调用。这可以修改，如下面例子所示：

```
<invoke>
    <jobs>jobA, jobB, jobC</jobs>
    <always>true</always>
</invoke>

<invoke>
    <jobs>jobA, jobB, jobC</jobs>
    <unstable>true</unstable>
</invoke>

<invoke>
    <jobs>jobA, jobB, jobC</jobs>
    <stable>false</stable>
    <unstable>false</unstable>
    <failed>true</failed>
</invoke>
```

在上面的示例中第一个调用总是调用下游作业。即使它们中的一些测试失败，管道中的作业也会被执行。

在上面的示例中，第二个调用是即使上游作业不稳定也会调用下游作业：无论什么样的测试结果调用都会发生。它可以用于对于测试和失败不太敏感的管道作业。

第三个调用是当一个上游作业只有在执行失败时调用下游作业，而不是执行结果为稳定和不稳定情况。你会发现当一个作业失败后，需要执行除常规电子邮件通知外的其他额外操作的便捷之处。

Arifactory（*http://jfrog.org*）是一个二进制仓库，可以作为 Maven 仓库管理器。Jenkins Artifactory 插件（*http://wiki.jenkins-ci.org/display/JENKINS/Artifactory+Plugin*），如图 10-29 所示，为 Jenkins 构建作业提供了很多方便功能。我们已经在第 5 章的“部署到企业仓库管理器”一节看到了一部分，包括部署构建产物完成作业的能力，或与构建产物一起发送的构建环境信息（使其具有更好的追溯性）。

276

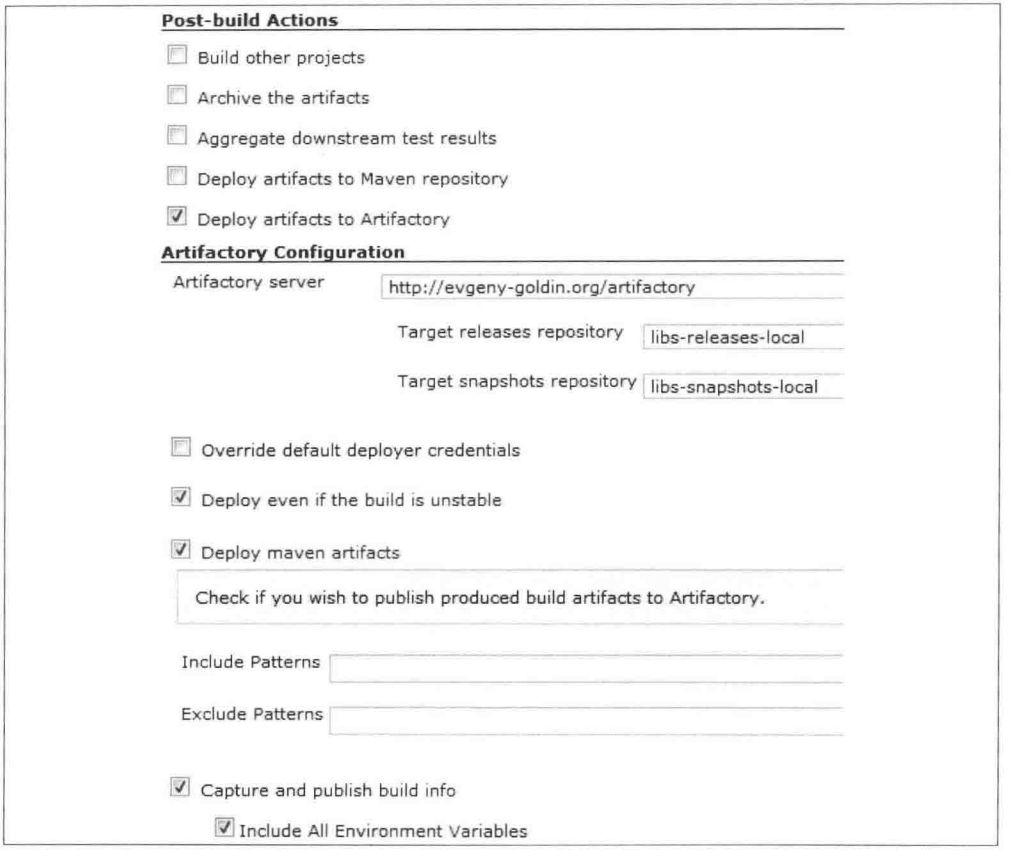


图10-29　Artifactory Jenkins插件配置

你还可以使用 Artifactory Jenkins 插件和 Maven Jenkins 插件一起部署构建产物，如下例所示：

```
<job>
```

```
    ...
    <artifactory>
        <name>http://artifactory-server/</name>
        <deployArtifacts>true</deployArtifacts>
        <includeEnvVars>true</includeEnvVars>
        <evenIfUnstable>true</evenIfUnstable>
     </artifactory>
</job>
```

277 默认部署证书是在“Manage Jenkins → Configure System”界面中配置的。也可以为每个 Jenkins 作业分别指定证书。默认 Maven 仓库是 *libs-releases-local* 和 *libs-snapshots-local*。你可以在插件文档（*http://wiki.jenkins-ci.org/display/JENKINS/Artifactory+Plugin*）中找到更详细的内容。

自由式作业

除了 Maven 作业外，Maven Jenkins 插件还可以让你配置自由式的 Jenkins 作业，如下例所示：

```
<job>
    <id>free-style</id>
    <jobType>free</jobType>
    <scmType>git</scmType>
    <repository>
        <remote>git://github.com/evgeny-goldin/maven-plugins-test.git</remote>
    </repository>
    <tasks>
        <maven>
            <mavenName>apache-maven-3</mavenName>
            <jvmOptions>-Xmx128m -XX:MaxPermSize=128m -ea</jvmOptions>
            <properties>plugins-version = 0.2.2</properties>
       </maven>
       <shell> <command>pwd; ls -al; du -hs .</command> </shell>
   </tasks>
</job>
```

自由式作业可以执行 Shell 或批处理命令，运行 Maven 或 Ant，以及调用其他作业。它们为系统脚本或其他不易获得 Jenkins 及其插件支持的作业提供了便捷的运行时环境。利用这个方法，你可以生成自由式构建作业配置文件（很像我们之前在 Maven 中看到的构建作业方式），它可以使构建环境更加统一、更易维护。

构建协作

触发下游作业是非常简单的。然而，当搭建更大、更复杂的构建作业时，有时你希望让

构建可以并行执行，或是等待某些构建作业完成后再去执行接下来的作业。在本节中，我们将着眼于哪些技术和插件可以帮你实现这一点。

在 Jenkins 中并行构建

Jenkins 内置了对并行构建的支持——当一个构建作业启动后，Jenkins 会将它分发到第一个有效构建节点上，所以你有多少个有效构建节点，就可以并行运行多少构建。

如果你需要并行运行同一个构建作业的不同情况，多重结构构建作业是一个很好的选择 <278>
（参见本章的“多重结构的构建作业”一节）。这是一种加快构建流程的方式。多重构建作业的典型应用是并行运行集成测试。一种策略是设置以不同的方式执行集成测试中不同的子集的集成测试构建作业。你可以定义分开的 Maven 属性文件，例如，使用一行命令来配置构建，用来判断运行哪个测试。当你用这种方式配置构建脚本时，它可以很容易配置一个多重结构的构建作业，用于并行运行集成测试的子集。

你还可以让 Jenkins 并行触发多个下游构建，将下游作业列表写入“Build other projects”字段中（见图 10-30）。构建作业子集会尽可能地并行执行。但是，正如我们将来会看到的那样，这可能不会一直都能符合你的需求。

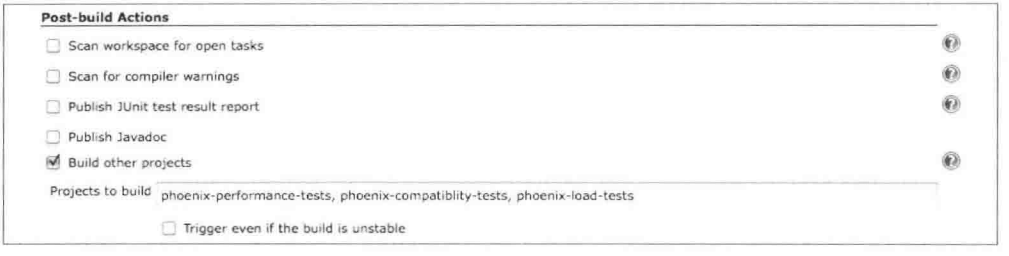


图10-30　在构建作业之后触发多个构建

依赖关系图

在我们研究并行构建的优点之前，可视化构建作业关系是很有用的。Dependency Graph View 插件可以分析你的构建并将构建之前的上下游关系以图形方式展示。这个插件使用的是 graphviz(*http://www.graphviz.org*)，如果你的服务器还没有安装，那么就需要安装它。

这个插件在主菜单中添加了一个依赖关系图的图标，它可以展示你的项目（仪表盘层级）所有构建的关系图，或所有当前构建作业（当你在一个特定项目里，见图 10-31）相关的构建作业。更重要的是，如果你单击关系图中的构建作业，Jenkins 会直接跳转到对应构建作业的项目界面。

连接

当搭建更复杂的构建管道时，经常会遇到在其他构建作业全部完成后才可以执行后续构
279 建作业的情形，但是这些上游构建作业并不需要按顺序执行。例如，在图 10-31 中，假设构建作业 *phoenix-deploy-uat* 需要在其他三个作业执行成功后才可以执行：*phoenix-compatibility-tests*、*phoenix-load-tests* 和 *phoenix-performance-tests*。

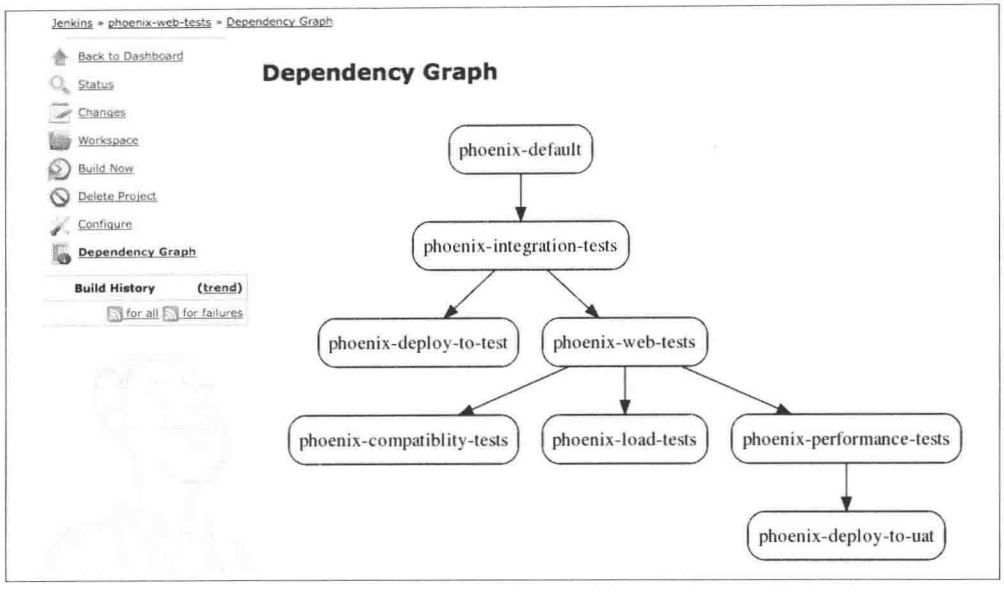


图10-31　构建作业依赖关系图

你可以通过 Joins 插件进行设置，需要在升级（Update）中心里进行安装。当安装完后，在构建作业中配置一个连接来初始连接流程（在例子中，应该是 *phoenix-web-tests*）。对于这个例子，我们需要修改构建作业 *phoenix-web-tests*，以便于首先触发 *phoenix-compatibility-tests*、*phoenix-load-tests* 和 *phoenix-performance-tests*，然后如果这三个成功执行，再执行 *phoenix-deploy-to-uat* 构建作业。

我们可以把构建作业 *phoenix-deploy-to-uat* 仅配置 Join Trigger 字段（见图 10-32）。“Build other projects”字段是不可修改的，并且会列出在本构建结束后会立即触发的构建作业。Join Trigger 字段包含了所有紧接着的下游构建作业。

因此，你不再为最后的构建作业配置原始构建触发器，因为那是多余的。

280 新的流程依赖关系图如图 10-33 所示。

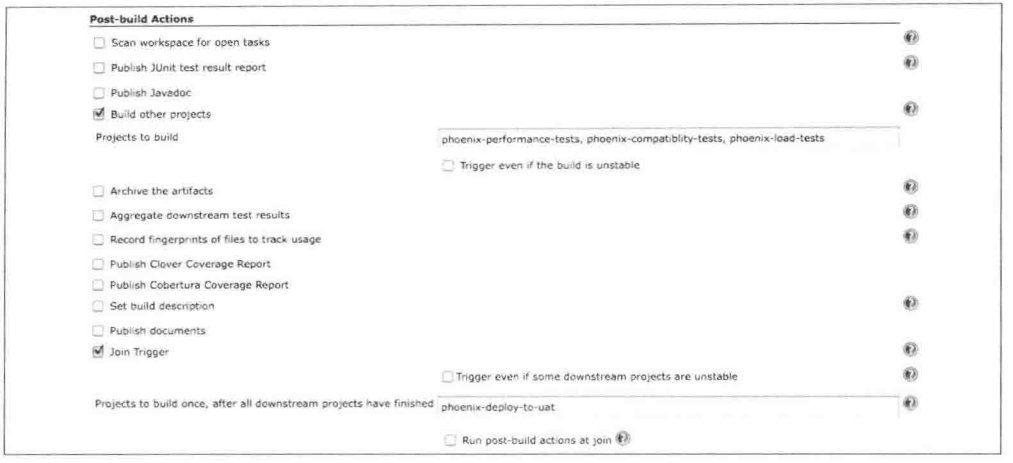


图10-32　在构建作业*phoenix-web-tests*中配置一个连接

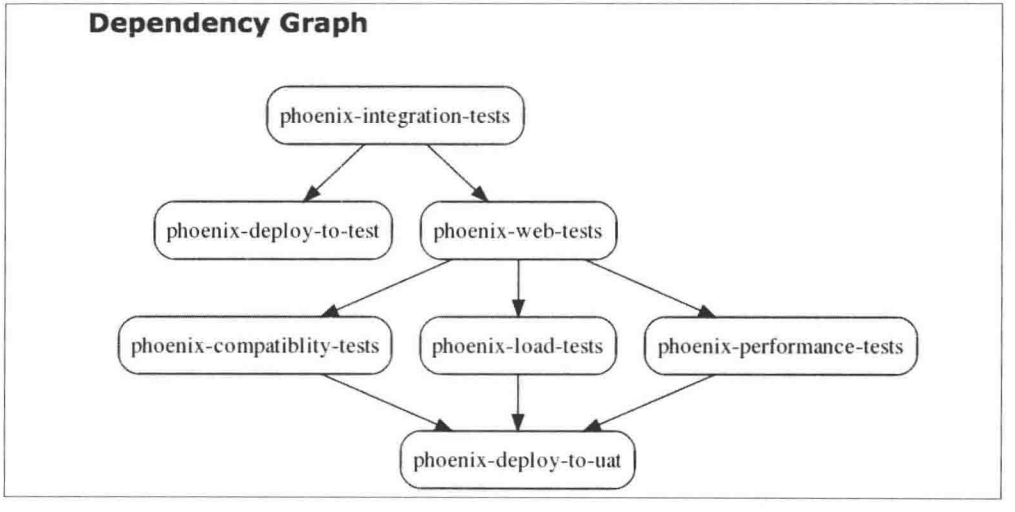


图10-33　更加复杂的构建作业依赖关系图

Locks and Latches 插件

在其他情况下，你可能在某种程度上能并行执行一系列的构建，但是某些构建作业因为并发访问资源而不能并行执行。当然，好的构建作业设计应该尽可能地将其独立出来，但是有时却很难独立出来。例如，不同构建作业可能需要访问同样的测试数据库或硬盘文件，并且并行执行可能会影响测试结果。或一个性能构建作业为了每次结果的一致性，可能需要独占访问测试服务器。

281

Locks and Latches 插件可以在某种程度上绕过这个问题。你可以用这个插件为某些资源加“锁”，这与多线程程序锁很相似。假设图 10-33 中显示的构建作业，负载测试和性能测试运行在专有服务器上，但是只有一个构建作业可以没有限制地使用这个服务器。想象其他项目的性能测试也运行在这个服务器上。

为了避免争用服务器，你可以用 Locks and Latches 插件配置一个“锁”，使同一时刻只有单独一个构建作业可以访问这个服务器。首先在系统配置(System Configuration)界面，你需要在 Locks 部分添加一个新的锁（见图 10-34）。这个锁之后会对服务上所有的构建作业启用。

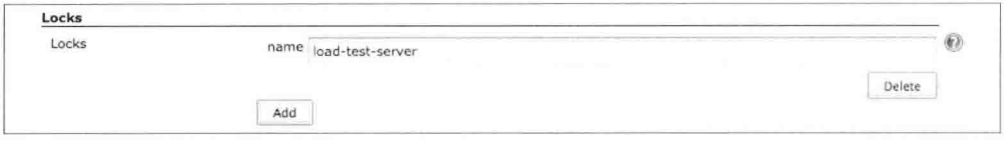


图10-34　添加一个新锁

下一步，你需要为那些争用资源的构建作业逐个进行设置。在 Build Environment 部分，你将看到一个 Locks 字段。勾选复选框并选择你创建的锁（见图 10-35）。当你为每个需要访问资源的构建作业配置完后，这些构建将只能在锁定的时间内运行。

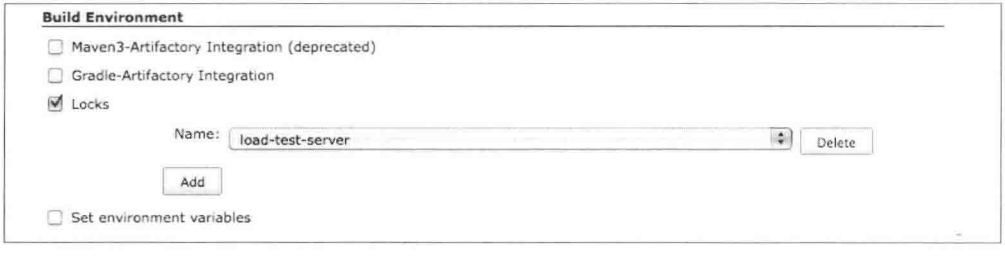


图10-35　为构建作业配置一个“锁”

构建管道及优化

持续集成不只是自动化的构建和软件测试，还有助于广泛背景下的软件产品开发和发布生命周期管理。在很多机构中，一个应用或产品的版本周期开始于开发阶段。当认为准备好时，会经由 QA 团队进行测试。如果他们认为这个版本是可接受的，会选择用户在用户验收测试 (UAT) 环境进行更多的测试。如果用户是满意的，产品就会进入生产阶段。当然，因为软件开发团队还会带来很多的变化，所以一个最基本的原则是根据某些质量相关标准，选择你的软件特定版本，之后的“升级”将放在下一个生命周期阶段。这就

是众所周知的构建优化和流程的构建管道。在本节中，我们将看看如何通过 Jenkins 构建管道来实现它。

通过 M2Release 插件管理 Maven 版本发布

282

构建管道的重要部分是定义良好的版本发布计划。包括判断在什么时间如何结束一个新的版本发布，以及如何通过唯一的标签或版本号来识别版本。如果你正在做 Maven 项目，非常推荐使用 Maven Release 插件来处理版本号。

Maven 项目使用良好的定义和良好的版本号结构。一个典型的版本号由三个数字组成（例如“1.0.1”）。开发人员使用快照版本（例如“1.0.1-SNAPSHOT”），就如名字的意思所示，这不是最终的版本。最终版本（如“1.0.1”）被一次性构建并部署到本地企业级仓库（或开源的 Maven 中心仓库），并且可以被其他项目使用。用于 Maven 打包构建产物的版本号是 Maven 的依赖关系管理系统的关键部分，并且强烈建议遵守 Maven 约定。

Maven Release 插件可自动地为你的项目更新 Maven 版本号码。简而言之，它验证、构建并测试你的应用，更新版本号码，用合适的标签更新你的版本控制系统，并将你的项目的发布版本部署到 Maven 仓库中。手工完成这项流程是比较冗长、无聊的，而 Maven Release 插件是将其自动化的非常好的方法。

然而 Maven Release 插件也是反复无常的。未提交或修改了本地文件会导致流程失败。例如，这个流程也会耗费时间和占用 CPU，尤其是对大型项目：构建应用程序、多次运行所有的单元测试和集成测试、从仓库中下载最新的源代码拷贝、向企业仓库中上传很多产品打包文件。事实上，你不希望这些东西运行在一个开发机器上。

所以在你的构建服务器上运行这些流程是很有意义的。

要实现这一点的一种方式是搭建一个特定的手动构建作业来调用 Maven Release 插件。然而，M2Rlease 插件提供了更简单的方法。使用 M2Rlease 插件，你可以添加在一个现有的构建作业中加入构建 Maven 发布版本的功能。

这个方法可以避免没必要重复的构建作业，使构建作业更易维护。

283

当你已经安装了这个插件时，可以定义任何构建作业并提供手动的 Maven Release 步骤。通过勾选 Build Environment 部分的“Maven release build”复选框（见图 10-36）来进行。在这里，你可定义你想执行触发构建的目标（`release:prepare release:perform`）。

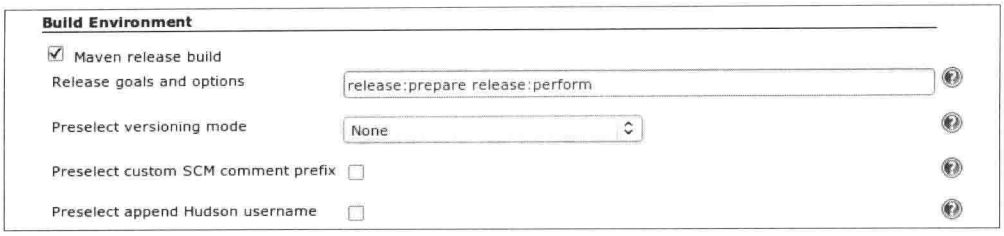


图10-36　使用M2Rlease插件配置Maven发布

当你设置完这些时，可以通过使用“Perform Maven Release”新菜单手动触发一个Maven发布（见图10-37）。

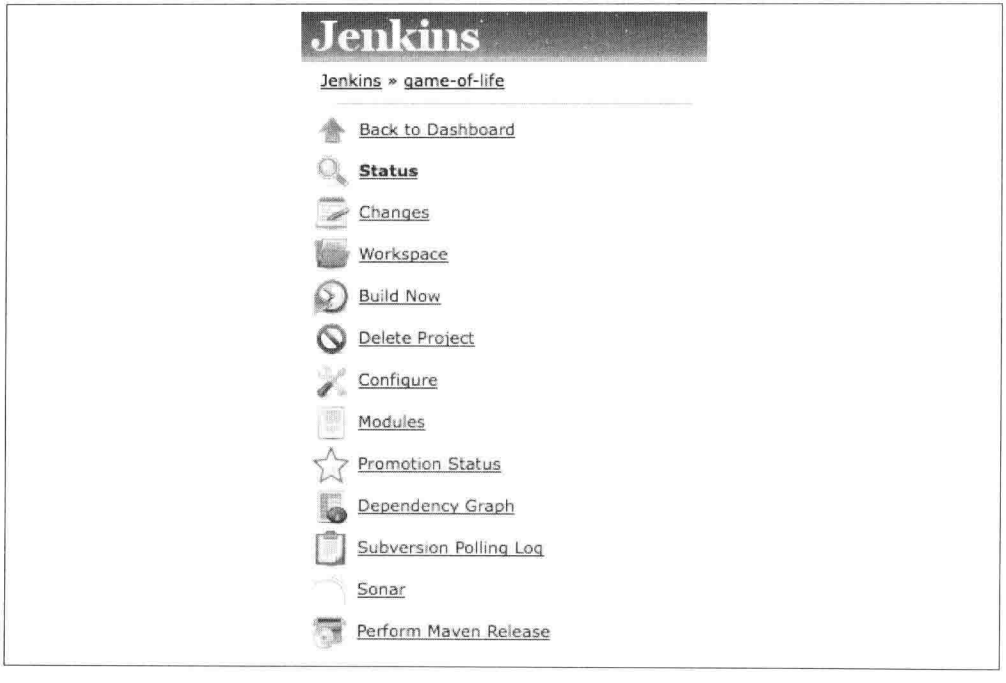


图10-37　Maven Release菜单执行项

284 这将启动一个使用你在插件配置中提供的目标的特殊构建作业（见图10-38）。Jenkins让你选择使用Maven提供的默认版本号（例如，版本1.0.1-SNAPSHOT将被作为1.0.1进行发布，并且开发版本号会更新至1.0.2-SNAPSHOT），或者提供你自定义的版本号。如果你想发布一个主版本，则可以选择手动指定1.1.0作为发布版本号，1.1.1-SNAPSHOT为下一个开发版本号。

如果你有一个多模块的Maven项目，则可以为所有模块选择一个单一的版本号配置，或

为每个模块使用不同的版本号更新。注意，在一个多模块项目中，不推荐为不同模块使用不同的版本号。

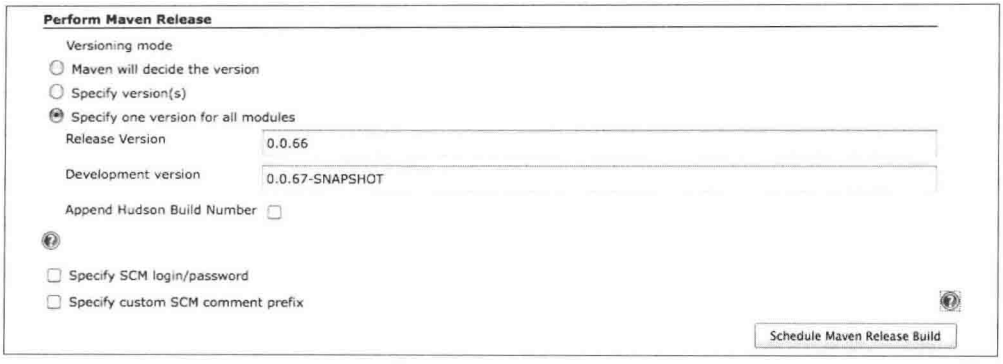


图10-38　在Jenkins中执行一个Maven发布

根据SCM配置，你可能还需提供一个有效的SCM用户名和密码，用来允许Maven在源代码仓库中创建标签。

Nexus企业仓库专业版提供了一个叫作分段仓库（Staging Repositories）的功能，在正式发布之前，将产品部署到特定阶段区域用于进一步的测试。如果你正在使用这个功能，则需要调整一下构建服务器配置。

Nexus专业版为每个独立IP地址、部署用户和HTTP用户代理创建新的阶段区域。一个给定的Jenkins构建机器总是用同样的IP地址和用户，然而，你通常会想要为每个构建分派一个单独的暂存区域。为了部署流程，配置Maven让它使用一个唯一HTTP用户代理。你可以在构建服务器上的*setting.xml*文件中加入以下内容（其中的ID必须匹配你项目的部署区域中的仓库发布ID）：

```
<server>
  <id>nexus</id>
  <username>my_login</username>
  <password>my_password</password>
  <configuration>
    <httpHeaders>
      <property>
        <name>User-Agent</name>
        <value>Maven m2Release (java:${java.vm.version}${env.BUILD_TAG }</value>
      </property>
    </httpHeaders>
  </configuration>
</server>
```

285

复制构建产物

在构建过程中涉及一些构建作业，如图 10-33 中所示，有时在一个顺序执行的构建作业中重用前一个作业的构建产物是有用的。例如，你可能想在多个独立的机器上并行执行一系列 Web 测试，使用本地应用服务器来提高性能。在这种情况下，获取前面构建的构建产物比每次重新构建更有意义；如果你正在使用 Maven，则依赖一个构建快照部署到你的企业级仓库。事实上，这两种方式都可能有构建产物不一致的风险：如果你使用一个企业级仓库的快照，例如，你将要使用最新的快照构建，但它可能未必是上游构建作业打包的那个。

Copy Artifact 插件可以从上游构建中复制构建产物，并在当前构建中进行重用。当你安装完这个插件并重启了 Jenkins 后，你将能在自由式构建作业中添加一个叫作“Copy artifacts from another project”的新类型构建步骤（见图 10-39）。

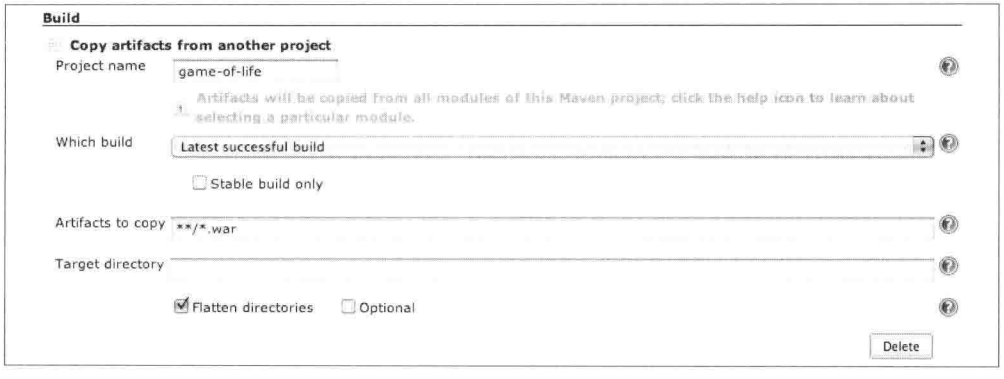


图10-39　添加“Copy artifacts from another project”构建步骤

286 这个新构建步骤可以把其他项目的构建产物复制到当前项目的工作空间。你可以指定任何其他项目，大部分是上游构建作业中的一个。当然你可以很灵活、很精确地指定自己想复制的那个构建产物。

你需要指定在其他构建作业的工作空间的哪里能查找到对应文件，以及 Jenkins 把对应文件复制到你当前项目工作空间的哪个位置。这可以是一个灵活的正则表达式（如 `**/*.war`，代表由构建作业生成的任何 WAR 文件），也可以更加精确（如 `gameoflife-web/target/gameoflife.war`）。注意默认情况下，Jenkins 会复制对应文件的目录结构，所以如果 WAR 文件是嵌套在 `gameoflife-web` 模块的 *target*（目标）目录中的，Jenkins 将它复制到你当前工作空间的 *gameoflife-web/target* 目录下。如果你不想这样做，则可以勾选“Flatten directories”选项，让 Jenkins 把所有构建产物都直接放在你指定的目录下（或默认情况，在你的项目工作空间）。

在许多情况下，你只想简单地从最后一次成功构建中获取构建产物。然而，有时你可能还想要更多的精度。“Which builds”项可以有很多方式来让你指定从哪里查找工具，包括从最新保存的构建（标记了“永久保存”的构建）、最新执行成功的构建，甚至指定的构建号。

如果你已经安装了 Build Promotion 插件（参见本章的“构建进阶”一节），则还能在特定的进阶流程中选择最新的进阶构建产物。要实现这一点，选择“Specify by permalink”，然后选择适合的构建优化流程。这是确保一致且可靠的构建管道的极好方式。例如，你可以配置一个构建进阶流程去触发一个构建，该构建把最新进阶构建生成的 WAR 文件复制并部署到特定服务器上。这可以确保即使其他构建已经发生，你也能恰到好处地部署了正确的二进制文件。

如果你正从一个多模块 Maven 构建作业复制构建产物，默认情况下，Jenkins 会复制那个构建下的所有构建产物。然而，大部分情况下你只是想要其中的某一个特定构建产物（如 Web 应用的 WAR 构建产物）。

当你需要为 Web 应用程序运行功能或性能测试时，这个插件特别有用。将这些测试放到一个单独的项目中是很有用的策略，而不是作为你的主构建流程的一部分。这使得它更容易地在不同服务器上并行运行这些测试或测试子集，所有任务均使用同样的二进制构建产物去部署测试。

例如，设想你有一个默认构建作业叫作 *gameoflife*，它会生成一个 WAR 文件，而且你想把 WAR 文件部署到本地应用服务器上并运行一系列的功能测试。此外，你希望能在多个分布式机器上并行地执行。

一种方法是创建一个专门的 Maven 项目用于在服务器上执行功能测试。然后你设置一个构建作业用于运行这些功能测试。这个构建作业会使用 Copy Artifact 插件获取最新的 WAR 文件（更加精确地说，甚至是最新优化的 WAR 文件），然后使用 Cargo 将它部署到一个本地 Tomcat 实例中。这个构建作业可以设置成可配置的（矩阵）构建作业，并行在多个机器上运行，可能还有额外的配置参数用来过滤每个构建不需要运行的测试。每个构建的执行使用原始 WAR 文件的拷贝。一个配置示例如图 10-40 所示。

287

Copy Artifact 插件不仅限于从传统构建作业中获取文件。你还可以从多重结构构建作业（参见本章的“多重结构的构建作业”一节）中复制构建产物。从每个执行配置中把构建产物复制到当前工作空间，每个构建产物都分别在各自的目录下。Jenkins 会使用多重结构构建的轴建一个目录结构。例如，设想我们需要一个对于不同目标数据库和应用程序服务器高度优化的构建产物。我们可以用图 10-41 中显示的多重结构构建作业来实现。

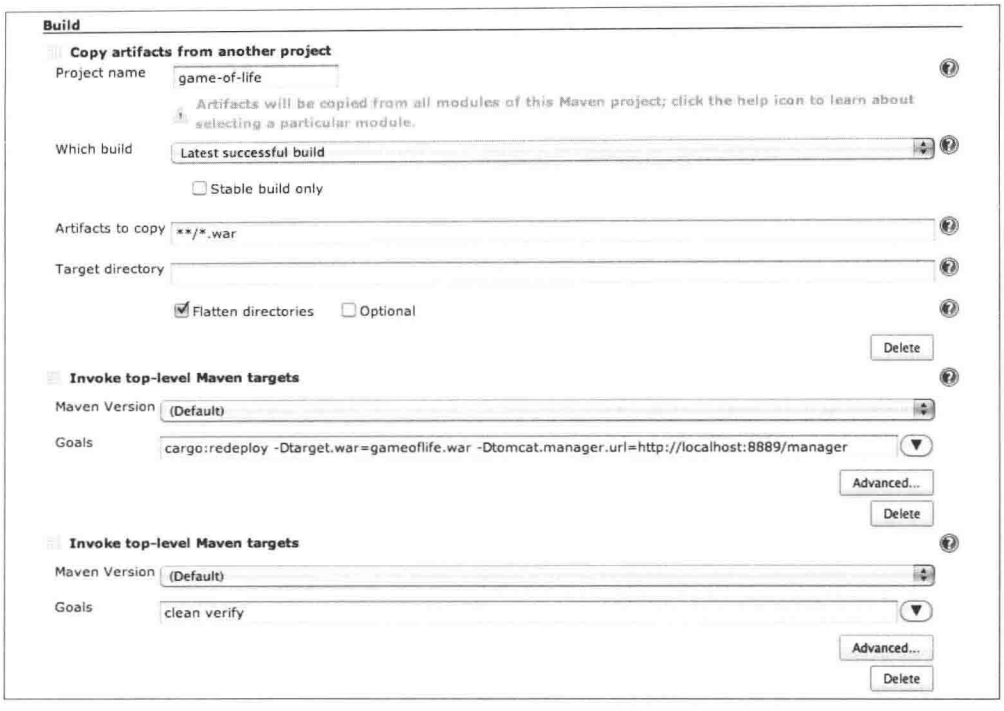


图10-40　使用WAR文件拷贝运行Web测试

288

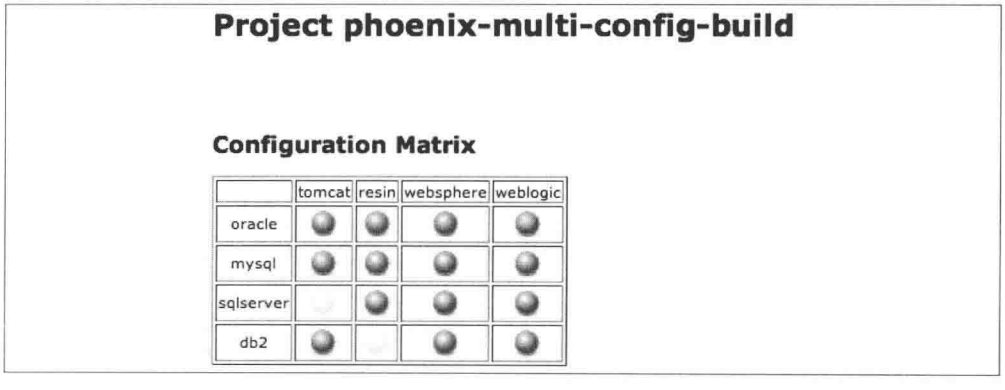


图10-41　从多重结构构建中进行复制

Copy Artifacts 插件能够复制由该构建作业生成的任何或者所有产品。如果你指定一个多重结构的构建作为构建产物源，这个插件会从多重结构的构建中复制构建并放到目标构建作业的工作空间，并使用多重构建轴目录结构。例如，如果你定义目标目录为 *multi-config-artifacts*，Jenkins 会复制构建产物到目标目录的子目录中，每个名称对应于特定的配置参数。所以使用如图 10-41 所示的构建作业，为 Tomcat、MySQL 自定义的 JAR

文件将被复制到 *$WORKSPACE/multi-config-artifacts/APP_SERVER/tomcat/DATABASE/mysql* 目录下。

构建进阶

在持续集成（Continuous Integration）世界中，所有的构建创建都是不一样的。例如你可能希望将新版本的 Web 应用程序部署到测试服务器，但必须等通过一系列的自动化功能测试和负载测试。或者你可能希望当测试人员做完测试后，能够标记某些构建为准备好了用于 UAT 部署的状态。

Promoted Builds 插件可以让你识别特定的符合额外质量标准的构建和这些构建上的触发行为。例如，你可能构建一个 Web 应用程序的构建作业，在随后的构建中运行一系列自动化 Web 测试，之后把生成的 WAR 文件部署到 UAT 服务器上用于将来的手工测试。

让我们看看这是如何工作的。在上面的项目说明中，默认构建作业（*phoenix-default*）运行单元测试和集成测试，并生成了一个 WAR 文件。这个 WAR 文件在集成测试中被广泛地重用（*phoenix-integration-tests* 构建作业），之后又经过一系列的自动化 Web 测试（*phoenix-web-test* 构建作业）。如果这个构建通过了自动化 Web 测试，我们希望将其部署到一个功能测试环境，在那里可以进行人工测试。部署到这个环境是通过构建作业 *phoenix-test-deploy* 实现的。当测试人员验证完这个版本时，它可以推送到 UAT 环境，再之后进入生产环境。整个进展过程如图 10-42 所示。

289

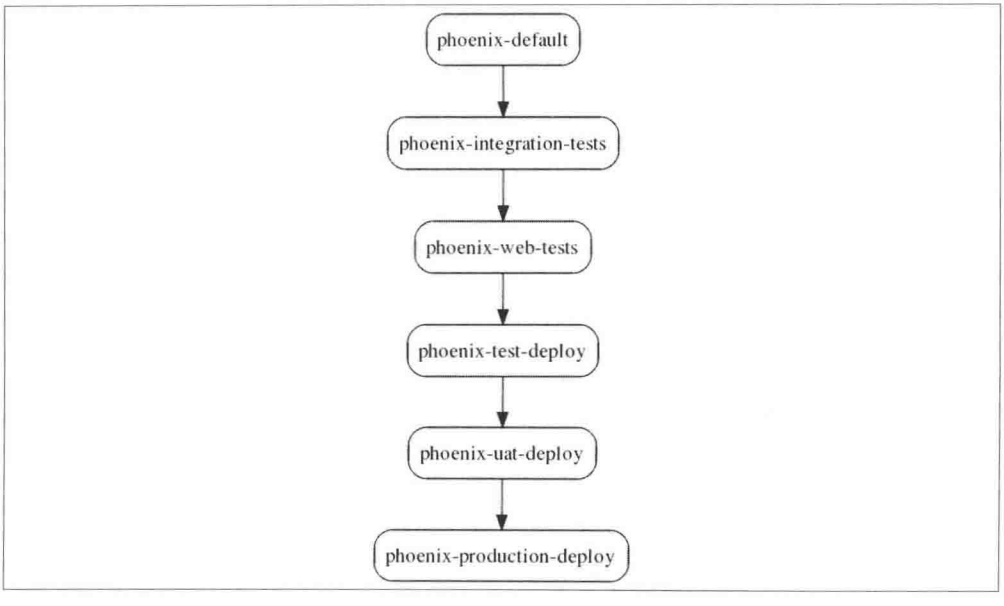


图10-42　构建作业的进阶流程

这个策略很容易通过 Promoted Builds 插件实现。当你已经安装完这个插件后，可以在作业配置界面找到一个新的“Promote builds when”复选框。你可以使用这个选项来设置构建阶段流程。在初始构建作业流程中定义一个或多个构建推进流程（例子中是 *phoenix-default*），如图 10-43 所示。一个构建作业可能有多个构建进阶流程、一些自动化及手动操作。例如在图 10-43 中，有一个叫作“*promote-to-test*”的自动构建进阶流程和一个手动的流程叫作“*promote-to-uat*”。自动的构建进阶流程根据下游构建作业结果进行触发。手动的进阶流程（勾选“Only when manually approved”复选框）只能由用户触发。

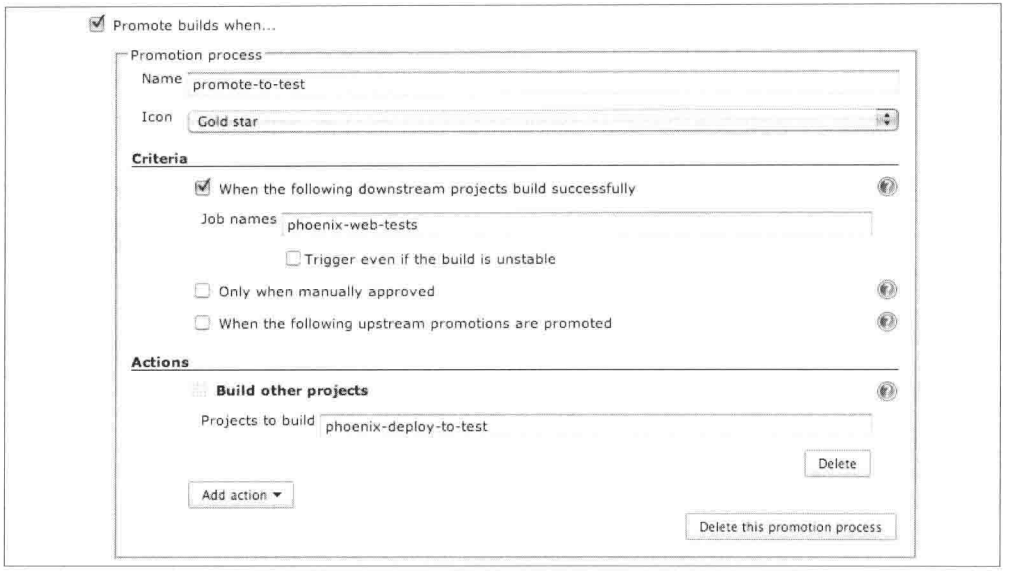


图10-43　配置构建进阶流程

让我们看一下配置自动化 *promote-to-test* 构建过程。

第一件事就是你需要定义构建进阶过程如何被触发。构建进阶可以是自动地基于下游构建工作的结果进行触发，或由用户手动触发。在图 10-43 中，构建进阶将在自动化 Web 测试（由 *phoenix-web-tests* 构建作业执行）执行成功后自动触发。

你也可以有一定的构建工作是只可以手动操作的，如图 10-44 所示。手动主要用于人为介入对构建进阶进行审批。部署到 UAT 或生产环境是一个普通的例子。另一个例子是，你想暂停自动化构建进阶，如即将发布。

手工构建，顾名思义，需要手动地批准执行。如果进阶流程触发了参数化构建作业，你还可以提供审批人在审批时需要进行输入的参数。在某些情况下，它对于指定允许某些

用户进行手动操作是非常有用的。你可以在 Approvers（审批人）列表中指定用户或用户组清单。

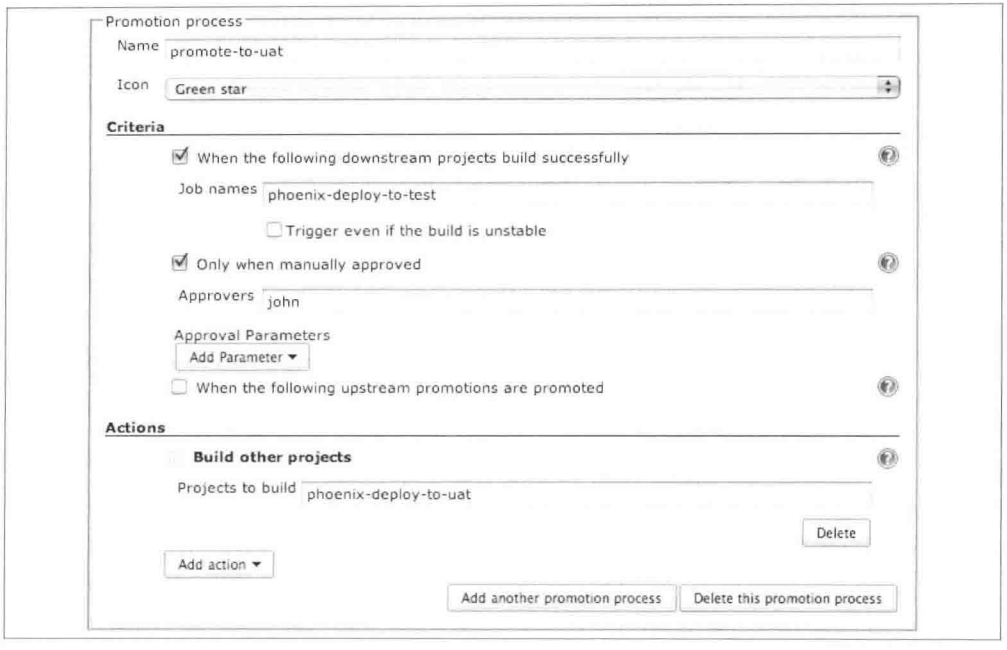


图10-44　配置一个手动构建进阶流程

有时对于审批人审批一个构建进阶加入些内容是有用的。当你设置一个手动进阶流程时，还可以定义其他必须要满足的条件，尤其是下游（或上游）构建作业必须是成功执行的（见图 10-45）。这些将会出现在“Met Qualification”（对成功的构建作业）和“Unmet Qualification”（对构建失败的或尚未执行的作业）。

接下来你需要在Jenkins中设置当构建进阶后需要做什么。就像在自由式的构建作业一样，添加相关操作。这会让构建进阶非常灵活，就像你可以在一个常规的自由式构建作业中添加任何可视化的操作，包括你的 Jenkins 实例上安装的任何有效插件的步骤。常见的操作包括调用 Maven 或 Ant 脚本，将构建产物部署到 Maven 仓库，或触发其他构建作业。

需要注意一个重要的事情，当进阶你的构建时，你不能依赖工作空间中的文件。事实上，无论自动还是手动优化构建的时候，其他构建作业可能已经将你要用到的文件删除或重写了。出于这个原因，在一个构建进阶过程中直接从工作空间获得 WAR 文件部署到应用服务器上是不明智的。更健壮的解决方法是触发一个独立的构建作业，并使用 Copy Artifacts 插件（参见本章的“复制构建产物”一节）来获取正确的文件。在这种情况下，你将复制 Jenkins 保存的构建产物，而不是直接复制工作空间中的文件。



图10-45　查看构建进阶细节

为了构建进阶的正确运行，Jenkins 需要能精确地连接上下游构建作业。更准确的实现方法是使用指纹。在 Jenkins 中，指纹是文件被构建作业处理或生成的 MD5 校验文件。通过匹配指纹，Jenkins 能够识别出引用这个文件的是哪个构建。

在构建进阶情况下，常见策略是一次性构建应用程序，然后用生成的二进制文件运行一系列下游测试构建作业。这个方法很适用于构建进阶，但需要确保 Jenkins 在构建作业之间共享或互相复制了指纹。如图 10-43 例子所示，我们需要做两件事（见图 10-46）。首先，我们需要归档生成的 WAR 文件，以便在下游项目中可以被重用。其次，我们需要记录归档构建产物的指纹。可以通过勾选“Record fingerprints of files to track usage”选项，并指定想要用的文件指纹。一个有用的快捷键是简单地提取所有的归档文件的指纹，因为这些文件通常会被下游构建作业检索重用。

这是所有需要你做的配置初始构建流程。下一步是配置 *phoenix-integration* 构建作业里的集成测试。在这里，我们使用 Copy Artifact 插件来获取 *phoenix-default* 构建作业生成的 WAR 文件（见图 10-47）。因为 *phoenix-integration* 在 *phoenix-default* 构建作业执行完成后立即触发，所以我们可以容易地获取最新构建成功的 WAR 文件。

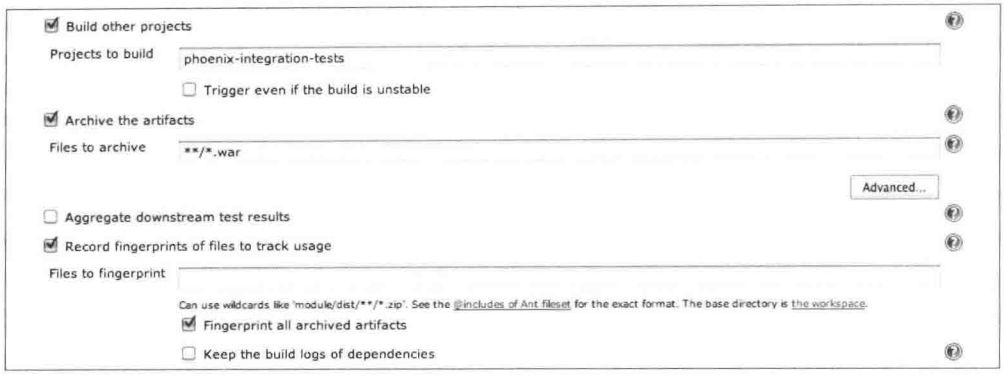


图10-46　在构建进阶流程中使用指纹

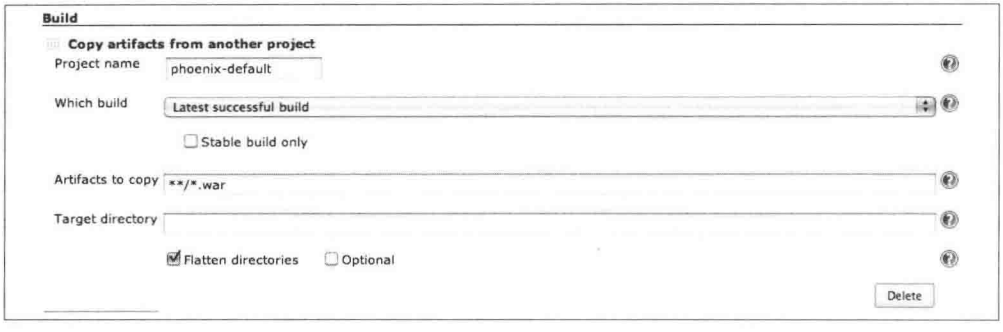


图10-47　从上游构建作业中获取WAR文件

然而，这并不完全是为集成测试要做的。*phoenix-integration* 构建作业后跟着负责自动化 Web 测试的 *phoenix-web* 构建作业。为确保构建过程的每个阶段使用的是同一个 WAR 文件，需要从上游 *phoenix-integration* 构建作业中获取它，而不是从原始 *phoenix-default* 构建工作去获取（可能同时再次被执行），所以还需要归档 *phoenix-integration* 中的 WAR 文件（参见图 10-48）。

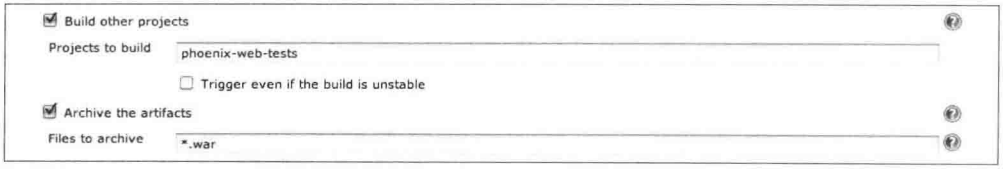


图10-48　为下游作业归档WAR文件

在 *phoenix-web* 构建作业中，使用和上面相似的配置从 *phoenix-integration* 中获取 WAR 文件（见图 10-49）。

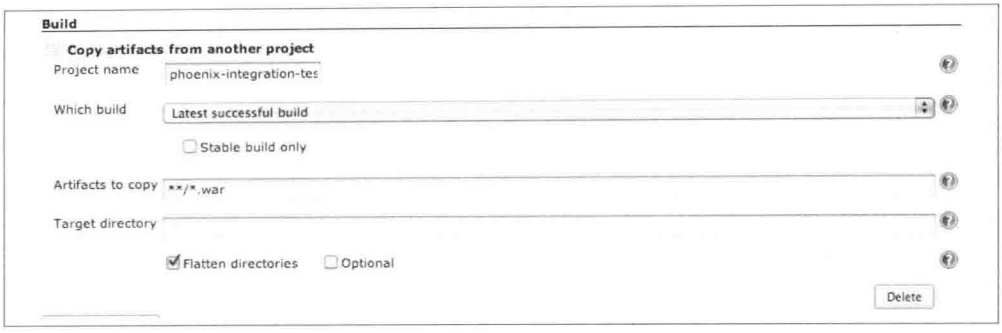


图10-49　从集成作业中获取WAR文件

为了让构建进阶过程更顺畅，还有一个更重要的事情，我们需要在 *phoenix-web* 构建作业中进行配置。正如我们前面所讨论的，Jenkins 需要确保用于测试的 WAR 文件与原始构建产生的是同一个。我们可以通过激活从 *phoenix-integration* 构建作业获取的 WAR 文件的指纹（最初是由 *phoenix-default* 构建作业生成的）来进行。因为我们已经将这个 WAR 文件复制到工作空间，如图 10-50 所示的配置将可以正常工作。

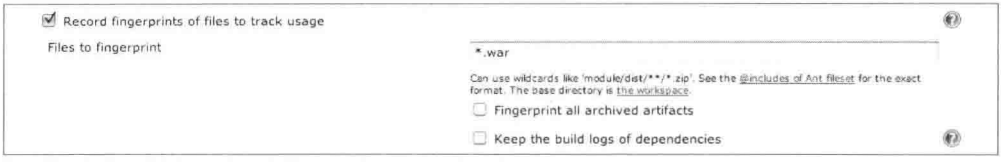


图10-50　我们需要决定使用的WAR的指纹

295 最后一步是配置 *phoenix-deploy-to-test* 构建作业最新的 WAR 文件（而不是最后执行成功的那个）。为此，我们再次使用 Copy Artifact 插件，但这次选择“Specified by permalink”选项。Jenkins 将建议你从构建进阶流程配置的构建作业中进行复制。在图 10-51 中，从 *phoenix-default* 中获取 WAR 文件，这正是我们想要的。

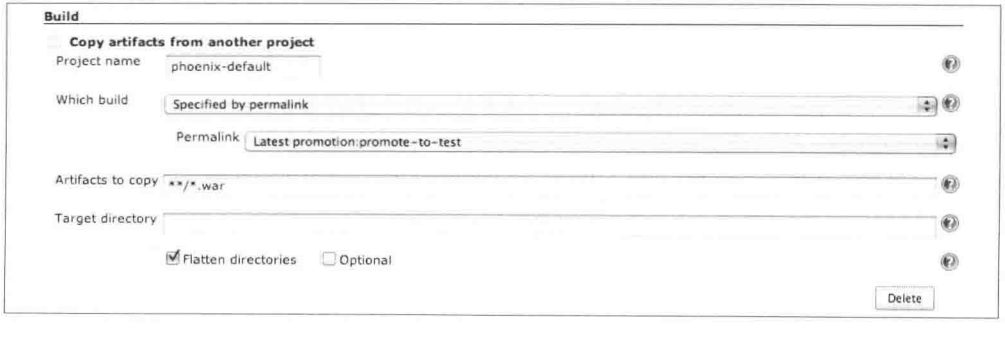


图10-51　获取最新的WAR文件

我们的进阶流程现在准备就绪。当 Web 自动化测试成功时，原始构建作业将进阶，相应的 WAR 文件也将部署到测试环境。在构建历史中进阶由星星图标表示（见图 10-52）。默认情况下，星星是黄色的，但你可以在构建进阶设置中修改星星的颜色。



图10-52　在构建历史中进阶构建以星星图标显示

你还可以使用“Promotion Status”菜单项（或单击构建历史的彩色星星）查看相应构建进阶细节，甚至重新手动运行（见图 10-45）。任何构建进阶可以手动触发，这通过单击“强制进阶”（Force Promotion，如果这个构建作业没被进阶）或“重新执行进阶”（Re-execute promotion，如果有）进行。

汇总测试结果

当在不同的构建作业间分发不同的类型测试时，很容易丢失全局的总体测试结果。测试结果分散在不同的构建作业，没有一个集中查看测试执行失败的总数据的地方。

避免这个问题的一个好方法是使用 Jenkins 汇总测试结果（Aggregated Test Results）功能。这可以获取所有下游作业的测试结果记录，并在上游构建作业中进行汇总 。你可以在初始（上游）构建作业中勾选“Aggregate downstream test results”选项（见图 10-53）进行配置。

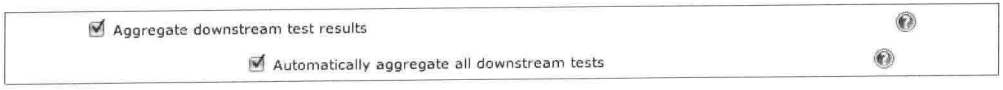


图10-53　汇总测试结果报告

汇总的测试结果可以在构建详细界面进行查看（见图 10-54）。不幸的是，这些汇总的测试结果没有出现在这个界面，但你可以单击构建页面上的 Aggregated Test Results 链接来查看全部测试执行的清单。

297

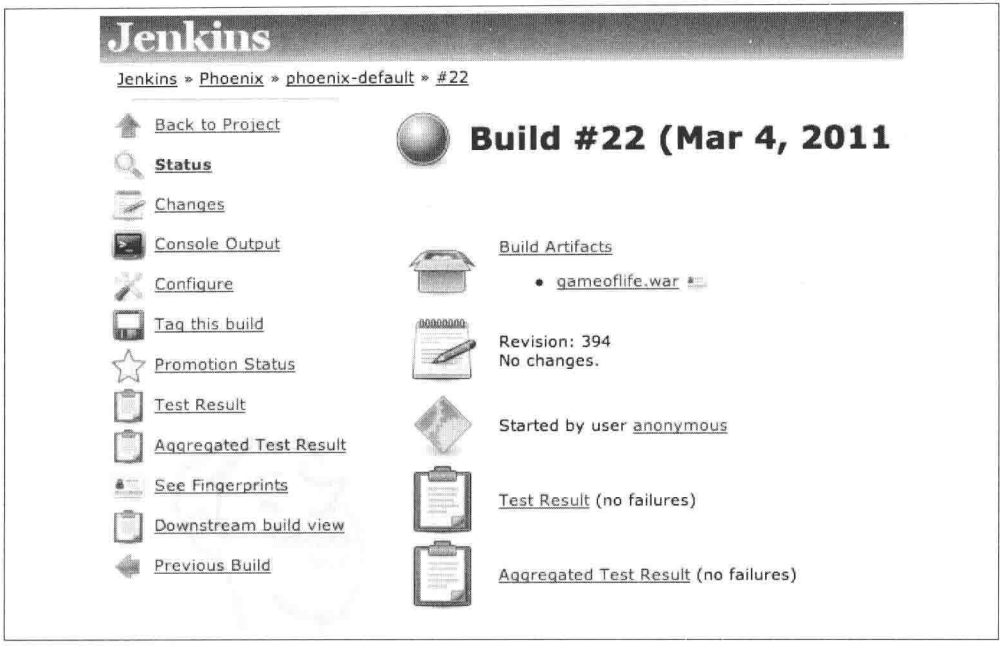


图10-54　查看测试结果汇总

为了能正确执行，你需要确保在每个阶段配置了指纹识别的二进制文件。Jenkins 只会汇总包含具有相同指纹构建产物的下游构建测试结果。

构建管道

下面介绍本节中的最后一个插件 Build Pipeline 插件。Build Pipeline 插件意味着进一步构建进阶，并帮助你设计和监控部署管道。部署管道让你的构建通过一系列质量检验，在每一个阶段有自动或手动的审批，最终部署到生产环境中。

Build Pipeline 插件提供了另一种定义下游构建作业的方法。构建管道，不像传统的下游依赖关系，它是一个线性的流程，构建作业按顺序执行。

使用这个插件，首先为每个管道中的构建作业配置下游的构建作业，像往常一样勾选使用“Build other projects”项。Build Pipeline 插件使用标准的上下游构建配置，你需要做的是自动化步骤。然而，Build Pipeline 插件也支持手动构建，用户必须手动审批下一步。对于手动步骤，需要配置你的上游构建作业的后续构建操作动作：只需勾选“Build Pipeline Plugin -> Specify Downstream Project”框，在你的项目中选择下一步，勾选“Require manual build executor”选项（见图 10-55）。

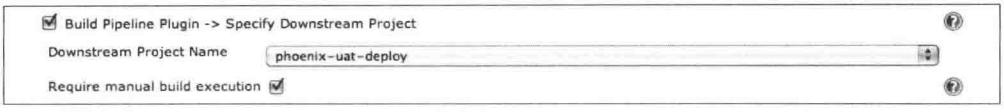


图10-55　在构建管道中配置一个手动步骤

当你设置了满意的构建流程时，可以配置构建管道视图，就像其他视图一样进行创建（见图 10-56）。 298

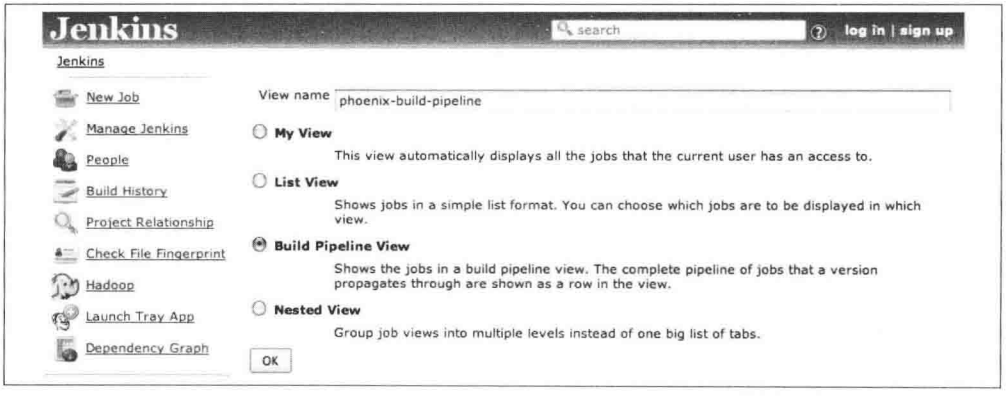


图10-56　创建一个构建管道视图

然而，配置视图有一个技巧。在本书写作之际，还没有直接配置视图的菜单项或按钮。实际上你需要手动输入 URL。幸运的是，这并不难：只需对这个视图的 URL 结尾进行添加 / 配置。例如，如果你的视图名字是“phoenix-build-pipeline”，如这里所示配置这个视图的 URL 将为 *http://my_jenkins_server/view/phoenix-build-pipeline*（见图 10-57）。

最重要的事情是这个界面中初始作业的配置。这是你的构建管道起点。你可以定义多个构建管道视图，每个视图的起始作业都不一样。你还可以配置构建序列的最大值并显示在界面上。

当配置完起点时，你可以返回到视图界面查看构建管道的当前状态。Jenkins 横向显示成功的构建作业，使用颜色来表示每个构建的执行结果（见图 10-58）。在管道中每个构建

作业都有一列。当初始构建作业启动时，在这个页面就会出现新的一行。随着构建通过构建管道中相继的构建作业，Jenkins 会在成功列里增加一个彩色盒子标志，显示各个阶段的结果。你可以单击盒子查看对应构建的详细结果。最后，如果需要手动执行，用户可以通过界面显示的操作按钮触发工作。

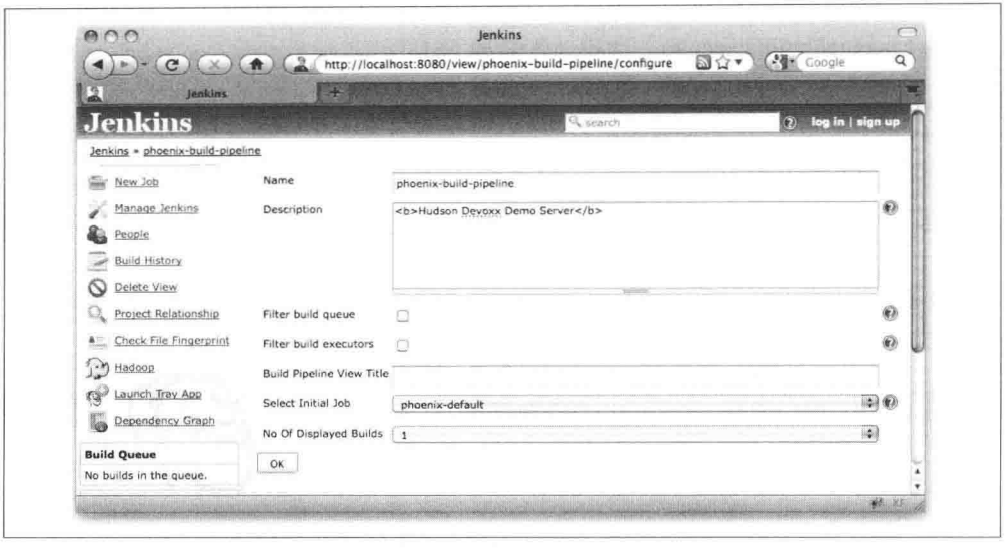


图 10-57　配置一个构建管道视图

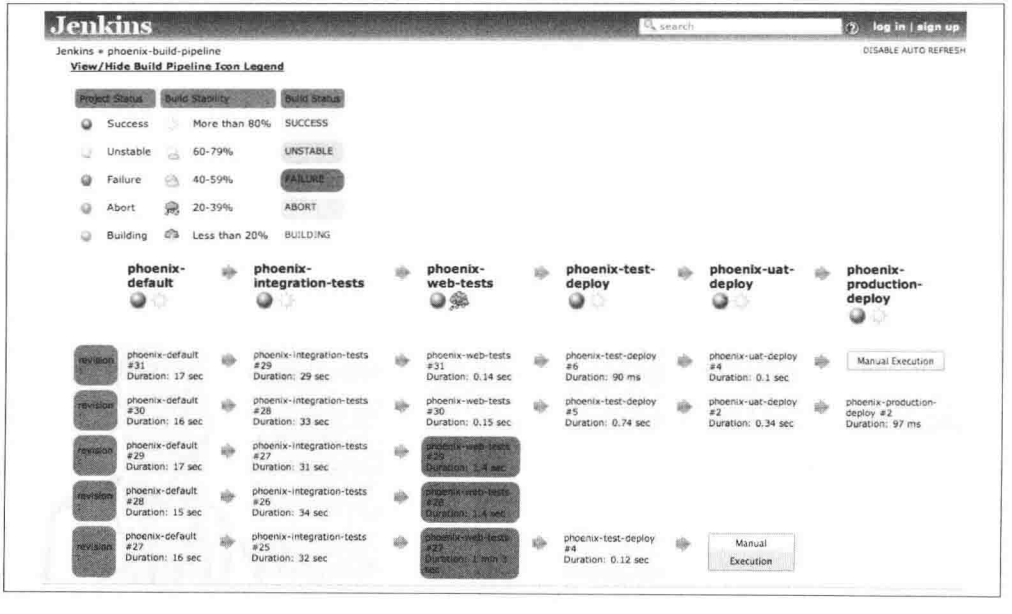


图10-58　构建管道实例

这个插件是相对较新的，没有与我们看到的其他插件进行集成。特别是，它是专为线性构建管道设计的，而且不能很好地应对分支或并行构建工作。然而，它确实给出了一个优秀的构建管道视角。

小结

300

持续集成构建作业不仅仅是简单地安排构建脚本的执行。在本章中介绍了许多工具和技术，使你能够常规地构建作业，整合并使其成为更大流程的一部分。我们已经看到如何参数化和为多重结构构建作业添加一个元素的灵活性，可以根据不同的参数运行相同的构建作业。其他工具帮助协调和安排相关组的构建作业。Joins 和 Locks and Latches 插件帮助协调并行运行的构建作业。Build Promotions 和 Build Pipelines 插件，以及 Copy Artifacts 插件，使为项目设计和配置复杂的构建进阶策略变得更加容易。

第 11 章

分布式构建

简介

可以说 Jenkins 更强大的功能之一是它能够跨大量的机器调度构建作业。设置很多构建服务器是很容易的（或者跨多台计算机共享负载，或者在不同的环境中运行构建作业）。这是能够潜在增加你的 CI 基础架构能力的一个非常有效的策略。

分布式构建通常用来吸收额外的负载，例如通过动态添加额外的机器应对构建作业中的高峰期，或者在特定操作系统或环境运行专门的构建作业。例如，你可能需要在一个特定机器或者操作系统上运行特定构建作业。例如，如果你需要使用 Internet Explorer 运行 Web 测试，则需要使用 Windows 机器。或者你的构建作业之一可能消耗大量资源，并且需要在自己专用的机器上运行，而不会影响你其他的构建作业。

构建服务器的需求随着时间的推移也会波动。如果你在产品发布周期，则在周期快结束时你可能需要运行更多的构建作业，比如更多的综合功能测试和回归测试套件可能会更加频繁。

在本章中，我们将讨论如何使用 Jenkins 建立和管理大量构建服务器。

Jenkins 分布式构建架构

Jenkins 使用主 / 从架构来管理分布式构建。你的主要 Jenkins 服务器（直到目前我们一直使用的）是主节点。简单地说，主节点的工作是处理调度构建作业，把构建分发到从节点来进行实际执行，监视从节点（必要时可能让它们上线或者离线），并且记录和发布构建产物。即使在一个分布式架构中，Jenkins 主实例也可以直接执行构建作业。

302 从节点按照被告知的工作，这涉及执行主节点分派的构建作业。你可以配置一个项目总是在特定的从节点运行，或者在某个特定类型的从节点运行，抑或让 Jenkins 挑选下一个可用的从节点。

从节点在远程机器上运行小的 Java 可执行文件，以及监听来自 Jenkins 主实例的请求。从节点可能（而且通常）在多种操作系统上运行。从实例可以以许多不同的方式来启动，这取决于操作系统和网络结构。一旦从实例在运行，它就通过 TCP/IP 连接与主实例进行通信。我们在本章的其余部分将看看不同的设置。

Jenkins 主 / 从策略

用 Jenkins 你可以配置许多不同的方式来设置分布式构建，这取决于你的操作系统和网络架构。事实是在所有情况下，构建作业运行在从节点，而且从节点怎么被管理对于终端用户来说是透明的：构建结果和构建产物最后总是会在主服务器上。

创建新的 Jenkins 从节点是一个简单的过程。首先，进入 Manage Jenkins 界面，单击 Manage Nodes。此屏幕显示从代理（更加正式的术语被称为“节点”）的列表，如图 11-1 所示。在这里，你可以通过单击 New Node 按钮建立新的节点。你还可以配置一些与你的分布式构建设置相关的参数（参见本章的“节点监控”一节）。

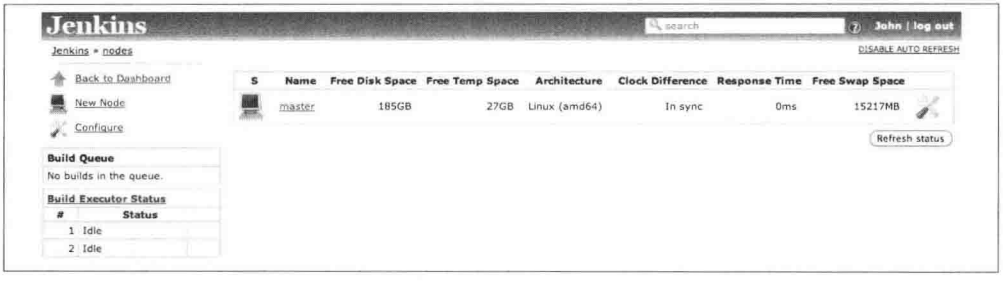


图11-1 管理从节点

当谈到管理 Jenkins 从节点时，根据你的目标系统和其他架构考虑的时候有几个不同的策略。这些策略会影响你配置从节点的方式，所以我们需要分别考虑。在下面的各节中，我们着眼于安装和配置 Jenkins 从节点最常用的方法：

303
- 主节点通过 SSH 启动从节点代理。
- 使用 Java Web Start 手工启动从节点代理。
- 把从节点代理安装为 Windows 服务。
- 直接在从节点命令行启动从节点代理。

这些策略各有其用途、优点和不便。下面逐一介绍。

主节点使用 SSH 启动从节点代理

如果你在 UNIX 环境中工作，最方便的方式无疑是使用 SSH 启动 Jenkins 从节点。Jenkins 有自己内置的 SSH 客户端，几乎所有 UNIX 环境开箱即用地支持 SSH（通常 `sshd`）。

要创建一个基于 UNIX 的从节点，请单击我们前面提到的 New Node 按钮。这会提示你输入从节点的名字和它的类型（见图 11-2）。在本书写作之际，只支持“dumb slaves”（哑巴从节点）；“哑巴”节点是被动的，只是简单响应来自主节点构建作业的请求。这是建立分布式构建架构最常用的方式，并且是在默认安装中唯一可用的选项。

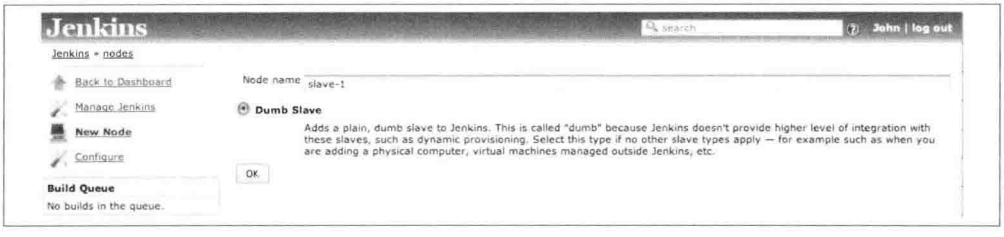


图11-2　创建新的从节点

在此屏幕中，你只需要提供从节点的名字。当你单击 OK 按钮时，Jenkins 让你提供关于从节点机器（见图 11-3）更具体的细节。

Name（名称）字段是简单地标识从节点机器的独特方式。它可以是任何东西，但是通过名称提醒你的物理机可能会有所帮助。如果名称是文件系统和友好的 URL 也会有帮助。它也支持空格，但是不使用空格会更好。所以”Slave-1”比“Slave 1”好。

Description（描述）字段也纯粹为了方便，可以被用来说明为什么使用这个从节点而不是另一个从节点。

像主要的 Jenkins 配置屏幕那样，“执行者的数量”字段允许你定义该节点可以执行多少并发构建作业。

每个 Jenkins 从节点还需要一个地方，可以是主目录，或者更确切地说，从代理可以用来运行构建作业的从节点机器的一个专用目录。你可以在 Remote FS root 字段定义此目录。你需要提供本地、特定操作系统的路径，比如对于 UNIX 机器的 */var/jenkins*，或者 Windows 的 *C:\jenkins*。没有什么关键的任务存储在此目录——一旦构建完成，每个重要的东西都被传回到主节点机器。所以你通常不需要像主节点那样，关心备份这些目录。

304

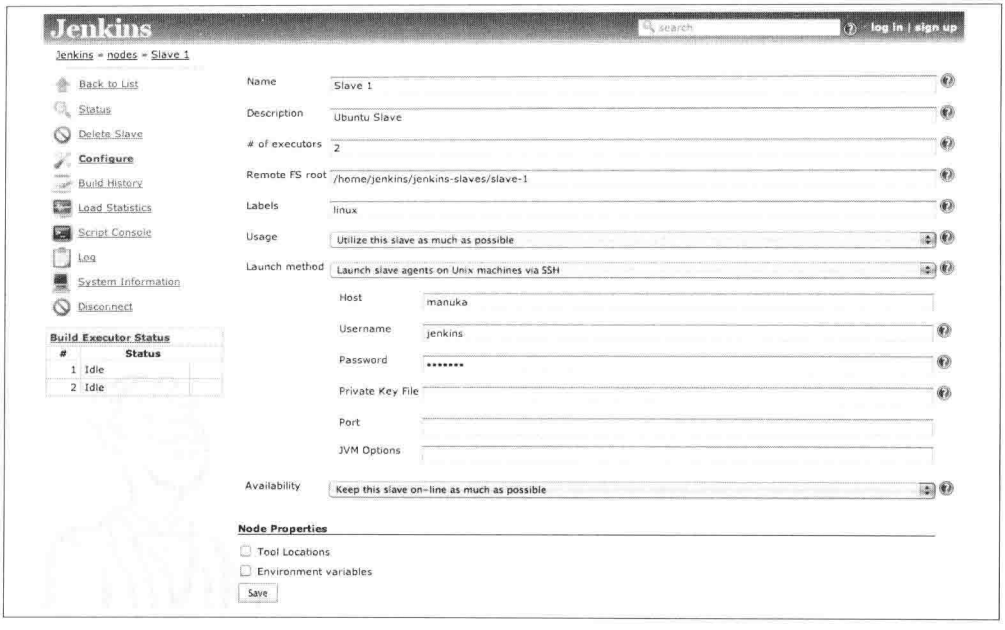


图11-3　创建UNIX从节点

当你的分布式构建架构开始规模地增长时，标签（Labels）是一个特别有用的概念。你可以为每个构建节点定义标签或标记，然后配置构建作业只运行在某个特定标签的从节点上。标签可能会涉及操作系统（unix、windows、macosx 等）、环境（staging、UAT、development 等），或是对你有用的任何标准。例如，你可以配置使用 Internet Explorer 运行自动化 WebDriver/Selenium 测试，但是仅在“windows”标签从节点运行。

Usage（使用率）字段允许你配置 Jenkins 使用该从节点的程度。你有三个选项可供选择：尽可能多地使用，为专门的构建作业保留，或根据需要使其在线。

第一个选项“Utilize this slave as much as possible”，只要该节点变得可用，就告诉 Jenkins 自由使用这个从节点运行任何构建作业。这是到目前为止最常用的一个，而且通常是你想要的。

然而，有时候第二个选项就派上用场了。在项目配置中，你可以把构建作业关联到特定
305 的节点——当特定的任务如自动化部署或者性能测试套件必须在特定的机器上运行时就有用了。在这种情况下，“Leave this machine for tied jobs only”选项能满足需求。你可以进一步把执行的最大数量设置为 1。在这种情况下，该节点不仅保留为特定类型的作业，而且任何时候它将永远只能运行这些构建作业的其中之一。这对于性能和负载测试是非常有用的配置，以便它可以在不被干扰的时候执行测试。

第三个选项是“Take this slave on-line when in demand and off-line when idle”（见图 11-4）。 顾名思义，该选项告诉 Jenkin 当需求很高时让从节点上线，并且当需求下降时让其离线。这允许你保留一些构建从节点以备大量使用，而无须维护一直运行在从节点的从节点代理。当你选择此选项时，还需要提供一些额外的细节。在“In demand delay”指出该从节点联机之前，作业必须在队列中等待多少分钟。闲置的延时表明 Jenkins 将其离线之前从节点需要闲置多久。

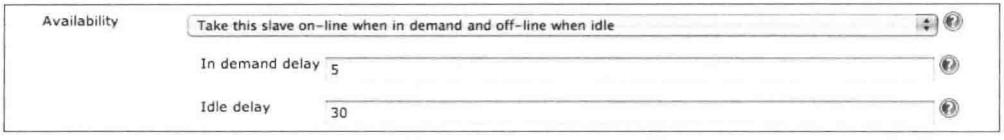


图11-4 当闲置时让从节点离线

在 Launch method（启动方式）字段可选择 Jenkins 如何启动节点的方法，正如我们前面提到的那样。我们这里讨论的配置，你会选择“Launch slave agents on Unix machines via SSH”。Advanced 按钮允许你输入 Jenkins 需要连接到 UNIX 从节点机器的额外细节：主机名、登录名、密码和端口号。你也可以在主节点机器提供 SSH 私钥文件的路径（如 `id_dsa` 或 `id_rsa`）来使用“无密码”的公 / 私钥身份认证。

你还可以配置 Jenkins 启动和停止从节点的时间。默认情况下，Jenkins 只需保持运行的从节点并且在需要时使用它（“Keep this slave on-line as much as possible”选项）。如果 Jenkins 注意到从节点已经离线（例如由于服务器重启），则若可以，它会尝试重启它。另外，Jenkins 可以更加节约你的系统资源，而且当它不需要时会让从节点离线。要做到这一点，只需选择“Take this slave on-line when in demand and off-line when idle”选项。如果你有定期的峰值和间歇的构建活动，则这是有用的，因为未使用的从节点可以离线以节省系统资源用于其他任务，当需要的时候该从节点再重新上线。

Jenkins 还需要知道在从节点机器哪里能找到构建作业需要的构建工具。这包括 JDK 以及 Maven、Ant 和 Gradle 等构建工具。如果已经配置了自动安装构建工具，你通常不需要为从节点机器做额外的配置；Jenkins 会根据需要下载并安装工具。另一方面，如果你的构建工具安装在本地从节点机器，则你需要告诉 Jenkins 在哪里可以找到它们。你可以勾选 Tool Locations 复选框，并为构建作业需要的工具提供每个工具的本地路径（见图 11-5）。

306

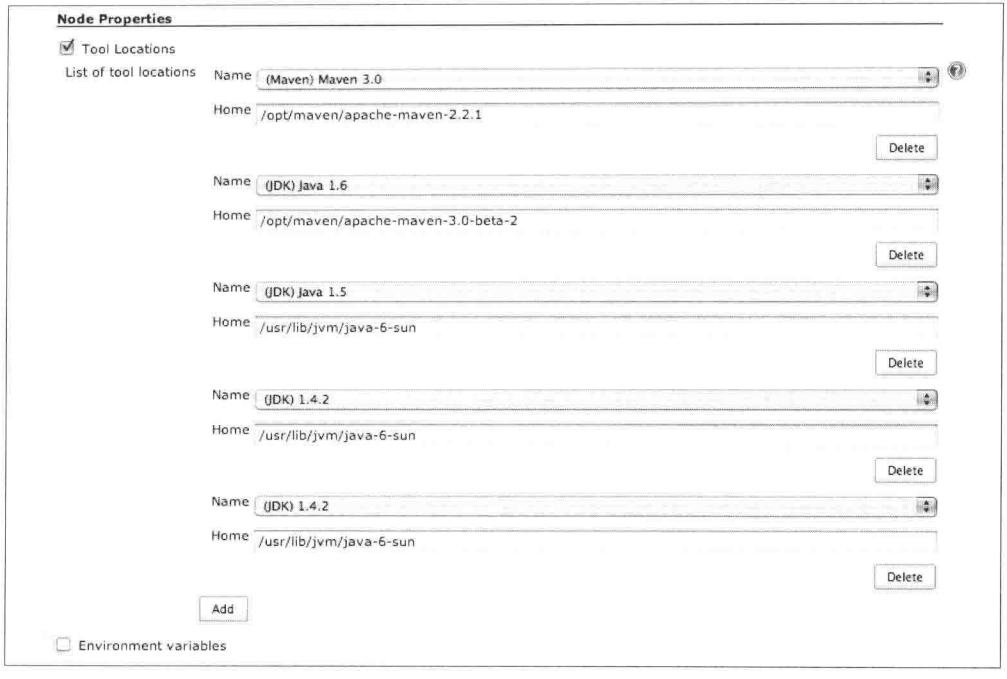


图11-5　配置工具位置

你还可以指定环境变量。这些将传递到你的构建作业中，并可以很好地允许你的构建作业根据在哪里执行而表现为不同的方式。

一旦你这样做，则新的从节点将出现在 Jenkins Nodes 页面上的计算机列表中（见图 11-6）。

307

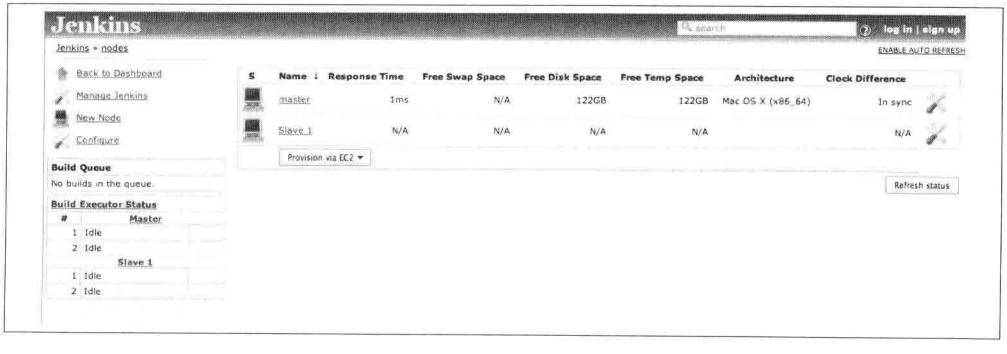


图11-6　新的从节点

使用 Java Web Start 手工启动从代理

另一种选择是使用 Java Web Start（JNLP）从从节点机器本身启动从代理。如果服务器不能连到从节点，则这种方法是有用的，例如从节点机器运行在防火墙的另一边 。不论在什么操作系统，它都能运行，然而较常用的是 Windows 从节点。它有几大主要的缺点：从节点不能被 Jenkins 自动启动或者重启。因此，如果从节点发生故障，则主实例无法重启它。

当你在 Windows 机器上执行此操作时，你需要至少进行一次手工启动 Jenkins 从节点。这涉及在机器上打开浏览器，在 Jenkins 主节点打开从节点页面，以及使用 JNLP 图标启动从节点。然而，一旦你已经启动了从节点，就可以把它安装为 Windows 服务。

有时候在 UNIX 环境中，需要从命令行执行该操作。因为防火墙或其他网络问题，或者因为 SSH 在你的环境不可用，所以你可能需要这样做。

让我们讲解这些过程。

在任何情况下，你需要做的第一件事情是创建一个新的从节点。 至于任何其他的从节点，你可以在 Nodes 屏幕单击 New Node 菜单项做到这一点。当输入有关你的从节点的详细信息时，确保在 Launch method 字段选择“Launch slave agents via JNLP”（见图 11-7）。还要记住如果有 Windows 从节点，Remote FS root 必须是 Windows 路径（例如 *C:\jenkins-slave*）。该目录并非必须存在：如果没有的话，Jenkins 会自动创建。

一旦保存了此配置，接下来登录到从节点机器，在浏览器中打开从节点屏幕，如图 11-8 所示。你会看到一个大的橙色 Launch 按钮，单击这个按钮，就可以在浏览器内直接启动从节点代理。

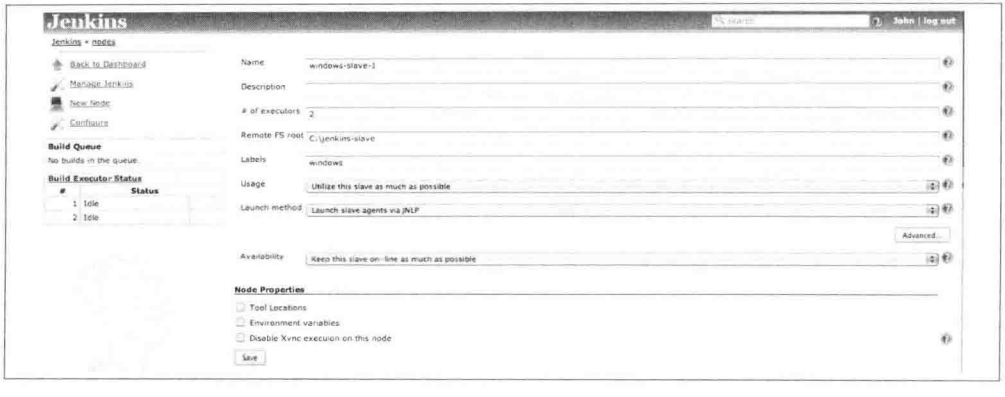


图11-7　为JNLP创建从节点



图 11-8　通过Java Web Start启动从节点

如果一切顺利，这会打开一个小窗口，提示你的从代理现在正在运行（见图 11-9）。

309

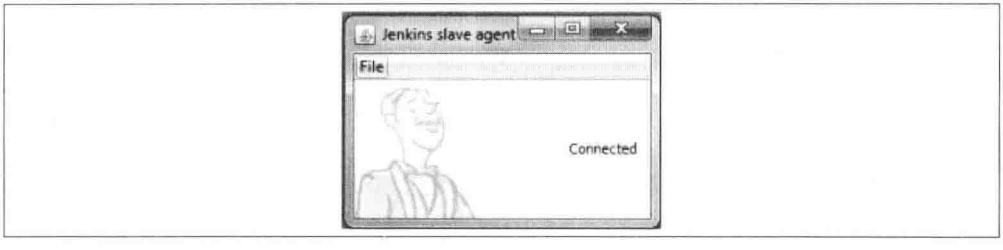


图11-9　运行中的Jenkins从代理

然而，浏览器是变化的，Java Web Start 并不总是易于使用。通常 Firefox 支持得最好（虽然你必须预装 Java JRE 以使 Firefox 支持 Java）。用 Internet Explorer 使用 JNLP 需要一些操作，将 **.jnlp* 文件与 Java Web Start 可执行文件 *javaws* 相关联，你可以在 Java *bin* 目录下找到该文件。事实上，从如下所述的命令行启动也许更容易。

一个虽然低级但更加可靠的方法是从命令行启动从节点。要做到这一点，只需从命令行窗口调用 *javaws* 可执行文件，如下所示：

```
C:> javaws http://build.myorg.com/jenkins/computer/windows-slave-1/slave-agent.jnlp
```

你需要执行的确切命令，包括正确的 URL，方便地显示在 JNLP 启动按钮下面的 Jenkins 从节点窗口（见图 11-8）。

如果你的 Jenkins 服务器上激活了安全规则，Jenkins 将在特定的非标准端口与从节点通信。如果由于某种原因该端口无法访问，则从节点将无法启动，并显示一条类似于如图 11-10 所示的错误消息。



图11-10　Jenkins从节点不能连接到主节点

这通常是防火墙在阻止端口的标志。默认情况下，Jenkins 随机使用一个 TCP 端口来与从节点通信。然而，你如果需要防火墙授权的特定端口，则可以强制 Jenkins 使用固定端口（在系统配置页面，“TCP port for JNLP slave agents”选项选择 Fixed，如图 11-11 所示）。



310

图 11-11　配置Jenkins从节点端口

把 Jenkins 从节点安装为 Windows 服务

一旦你在 Windows 机器上启动并运行从节点服务，为了避免每次重启机器都要手工重启它，你可以把它安装为 Windows 服务。要做到这一点，在从代理窗口的 File 菜单选择“Install as Windows Service”选项（见图 11-12）。



图11-12　把Jenkins从节点安装为Windows服务

一旦做到这一点，每当机器启动时，Jenkins 从节点会自动启动，并且可以像管理其他任何 Windows 服务一样（见图 11-13）。

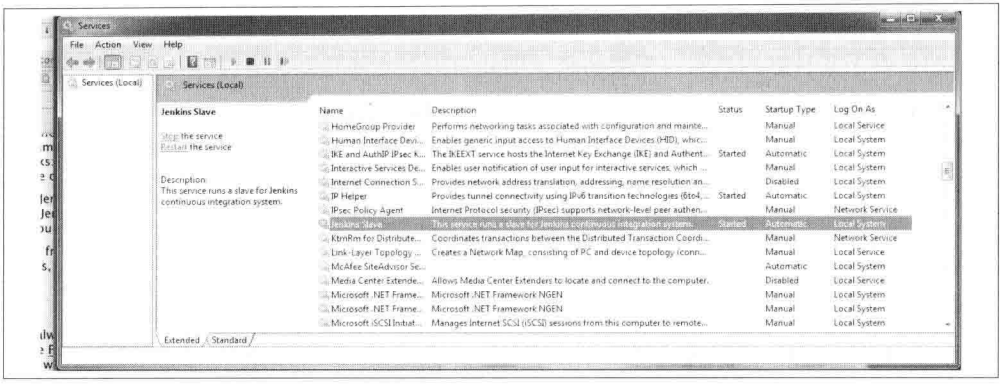


图11-13 管理Jenkins Windows服务

311

无主节点模式启动从节点

你也可以直接从命令行，在无头模式下启动从代理。如果你没有可用的用户界面，例如你从 UNIX 机器上启动 JNLP 从节点，则这是有用的。如果你正在使用 UNIX 机器工作，使用 SSH 连接通常更容易、更灵活，但是有时候网络或架构的限制会阻止你使用 SSH。在这样的情况下，从命令行启动从节点仍然是可能的。

要以这种方式启动从节点，你需要使用 Jenkins 的 *slave.jar* 文件。你可以在 *JENKINS_HOME/war/WEB-INF/slave.jar* 找到它。一旦你找到该文件，把它复制到 Windows 从节点机器上，并按以下方式运行：

```
java -jar slave.jar \
 -jnlpUrl  http://build.myorg.com/jenkins/computer/windows-slave-1/slave-agent.jnlp
```

如果你的 Jenkins 服务器需要认证，传入 `-auth username:pass word` 选项：

```
java -jar slave.jar \
 -jnlpUrl http://build.myorg.com/jenkins/computer/windows-slave-1/slave-agent.jnlp
 -auth scott:tiger
```

一旦启动了从代理，请务必将其安装为 Windows 服务，正如前面讨论到的那样。

以远程服务方式启动 Windows 从节点

Jenkins 还可以把远程 Windows 从节点作为 Windows 服务进行管理，这使用 Windows 2000 及其以后的版本开箱安装的 Windows Management Instrumentation（WMI，见图 11-14）。

当你选择这个选项时，你只需要提供 Windows 用户名和密码。节点的名字必须是从节点的主机名。

因为它不要求你物理连接到 Windows 机器进行设置，所以非常方便。然而，它有一定的局限性——尤其是不能运行需要图形界面的应用程序，因此你不能以这种方式配置从节点进行 Web 测试。实际上，这可能有点棘手，因为你可能需要配置 Windows 防火墙打开适当的服务和端口。如果你遇到麻烦，则确保你的网络配置允许 TCP 连接到端口 135、159、445 和 UDP 连接到端口 137 和 138（更多细节请参阅 *https://wiki.jenkins-ci.org/display/JENKINS/Windows+slaves+fail+to+start +via+DCOM* ）。



图11-14　让Jenkins作为Windows服务控制Windows从节点

把构建作业与一个或一组从节点关联

在前面的部分中，我们看到了如何给从节点分配标签。依据特性比如操作系统、目标环境、数据库类型或者和构建过程关联的其他标准，给你的从节点分组是一个方便的方式。这种做法的一个常见的应用程序是在专用的从节点上运行特定的操作系统功能测试，或者保留特定的机器进行性能测试。

一旦给你的从节点指定标签，你需要告诉 Jenkins 可以运行构建作业的位置。默认情况下，Jenkins 仅使用第一个可用的从节点，这通常会导致最佳的整体周转时间。如果需要把构建作业绑定到特定的计算机或一组计算机，你需要在构建配置页面勾选“Restrict where this project can be run”（见图 11-15）。接下来，在 Label Expression 字段输入机器的名称，或者标识一组机器的标签。Jenkins 将提供动态下拉列表显示可用的机器名称和标签。

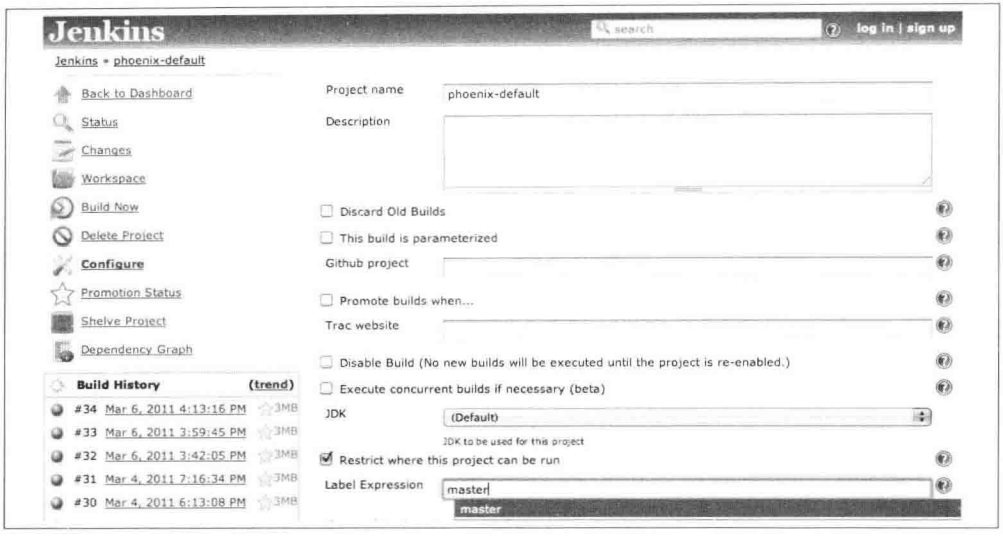


图11-15　在特定从节点运行构建作业

此字段也接受布尔表达式，允许你定义构建作业应该在何处运行的更加复杂的约束。最好通过一个例子来说明如何使用这些表达式。假设你有 Windows 和 Linux 从节点（由“windows”和“linux”标签标识），分布在三个地点（“sydney”、“sanfrancisco”以及“london”）。你的应用程序还需要对几个不同的数据库（“oracle”、“db2”、“mysql”和“postgres”）进行测试。你还可以使用标签来区分用来部署到不同环境（测试，用户验收测试，生产）的从节点。

最简单的标签表达式是确定构建作业是否可被执行。如果你的 Web 测试需要 Internet Explorer，则需要其运行在 Windows 机器上。你可以只引用相应的标签来表示。

```
windows
```

或者，你想在 Linux 机器上用 Firefox 进行测试，可以使用！否定运算符把 Windows 机器排除在候选构建节点外：

```
!windows
```

你也可以使用与（&&）和或（!!）运算符组合表达式。例如，假定 Postgres 数据库仅对 Linux 进行测试。你可以用下面的表达式告诉 Jenkins 只在安装了 Postgres 的 Linux 机器上运行特定的构建作业：

```
linux && postgres
```

或者你可以指定特定的构建作业仅会在悉尼或伦敦的用户验收测试环境运行：

```
uat && (sydney || london)
```

如果你的机器名称中包含空格，则需要用双引号将它们括起来：

```
"Windows 7" || "Windows XP"
```

你还可能发现有用的两个高级逻辑运算符。隐含运算符（=>）让你定义逻辑约束的形式："如果 A 是真的，那么 B 也必须是真的"。例如，假定你有个在任何 Linux 发行版运行的构建作业，但是如果它在 Windows 中执行，则必须是 Windows 7。可以如下表达这个约束：

314

```
windows -> "Windows 7"
```

其他逻辑运算符是 *if-and-only-if*（<=>）运算符。该操作可让你定义更加严格的约束形式"如果 A 是真的，B 必须是真的；但是如果 A 是假的，B 也必须是假的"。例如，假定 Windows 7 测试仅在用户验收测试环境中运行，而且只有 Windows 7 测试在用户验收测试环境中运行。你可以这样表达：

```
"Windows 7" <-> uat
```

节点监控

Jenkins 不仅仅只是分派构建作业到从代理：它主动监视你的从节点机器；如果认为该节点不能安全完成一个构建，就将此节点脱机。你可以在 Manage Nodes（管理节点）屏幕（见图 11-16）精确微调 Jenkins 监控的内容。Jenkins 以几种不同的方式监控从节点代理。它监控响应时间：过于缓慢的响应时间可以表示是网络问题，或者是从节点机器已经关闭。它还监视提供给 Jenkins 用户从节点可用的磁盘空间、临时目录空间以及 swap 空间，因为构建作业可能会消耗大量磁盘。它还紧密监视系统时钟，如果时钟不能正确同步，有时候会发生奇怪的错误。如果没有达到标准，Jenkins 将自动让服务器下线。

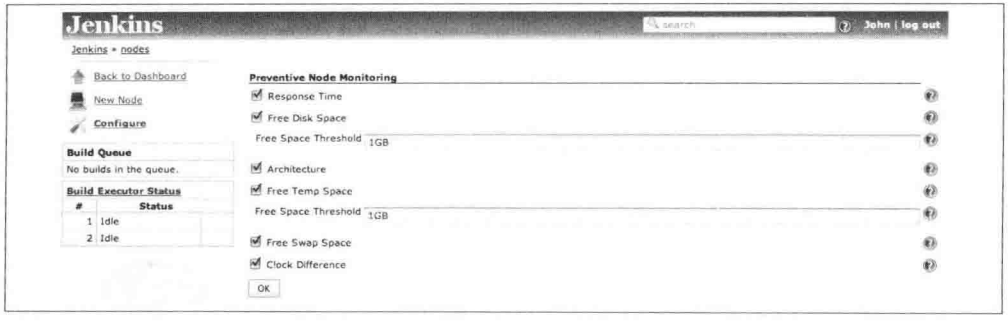


图11-16　Jenkins主动监控构建代理

云计算

云计算使用因特网上的资源扩展或者替换本地计算架构 。云计算正扩展到企业的许多领域，包括电子邮件和文件共享（Gmail 和 Google Apps 是十分著名的例子，还有许多其他的例子）、异地数据存储（如 Amazon S3），以及更多的技术服务比如源代码库（如 GitHub、Bitbucket）等。

315

当然，外部的硬件架构解决方案已经存在了很长一段时间了。云计算与更传统的服务的主要区别是其速度和灵活性，需要时该服务能够启动，不需要时可停掉。在云计算环境中，一个新的机器可以在几秒内运行并且可用。

然而，云计算在持续集成的上下文中并不总是像看起来那么简单。对于任何基于云计算的方式来工作，你的一些内部资源可能需要对外面的世界是可用的。这包括放开你的版本控制系统、你的测试数据库，以及你的构建和测试需要的其他资源的访问。当选择基于云的 CI 架构时，所有这些方面都需要仔细考虑；如果某些资源无法从互联网访问，则可能会限制你的选择。尽管如此，当涉及可扩展性时，基于云计算的 CI 有巨大的潜力。

在下面的各节中，我们将看看如何使用 Amazon EC2 云计算服务，来建立一个基于云计算的构建场。

使用 Amazon EC2

除了售书外，Amazon 还是较知名的云计算服务供应商之一。如果你愿意支付服务费用，Amazon 可以给你提供一部分持久使用的构建机器作为构建场的一部分，或者如果你现有的构建机器过载的话，则根据需要随时上线。这是优良以及合理的成本效益的方式，基于需求的基础吸收额外的构建负载，不需要维护额外的物理机器。

如果你想要一个基于云计算的 CI 架构的灵活性，但是不想使用外部硬件，另一种选择是搭建 Eucalyptus 云。Eucalyptus 是一种开源的工具，让你能够在现有硬件基础上搭建本地私有云。Eucalyptus 使用与 Amazon EC2 和 S3 兼容的 API，并且和 Jenkins 集成得很好。

设置你的 Amazon EC2 构建场

Amazon EC2 可能是最流行和最知名的商业云计算服务。若要使用这项服务，你需要创建一个 Amazon EC2 账号。这样做所需的过程记录于 Amazon 网站，我们这里就不详细陈述了。一旦你创建了自己的账号，就能够创建构成基于 EC2 的构建农场的虚拟机和镜像。

使用 Amazon EC2 时，你可以使用 Amazon Web Services（AWS）管理控制台（见图 11-17）创建虚拟机，也称为实例。你能够在这个网站管理自己正在运行的实例，并创建新的实例。你从预定义的镜像创建实例，也称为 Amazon Machine Images（AMI）。Amazon 和公共域提供了许多 AMI 镜像，可以用来作为一个起点，覆盖最流行的操作系统。一旦你创建了一个新的实例，就可以使用 SSH（UNIX 机器）或者 Windows 远程桌面。 316

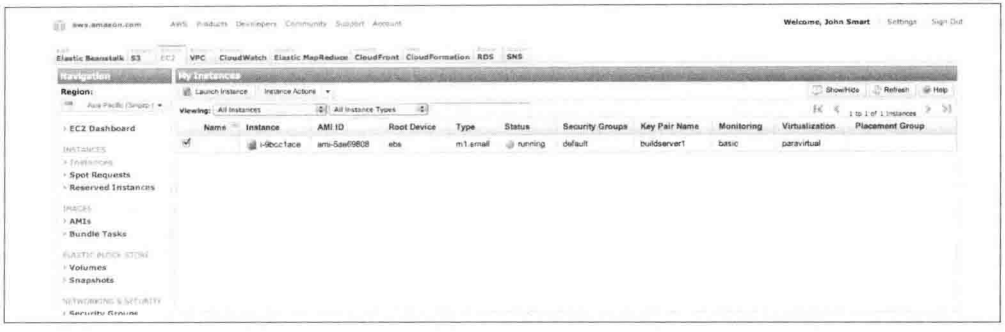


图11-17　使用Amazon AWS管理控制台管理你的EC2实例

若要配置一个构建场，可到安全构建服务器 Key Pairs 菜单配置用来访问你的 EC2 实例密钥。特别是，你需要安装 Amazon EC2 API 工具，设置适当的公钥 / 私钥，允许从你的服务器或者网络通过 SSH 连接 Amazon 实例。再次强调一下：对于所有的主流操作系统的操作在 EC2 网站有详细的参考。

你可以以两种方式使用 Amazon EC2 实例——在 Amazon EC2 上创建从节点机器并且作为远程机器使用它们，或者根据需要让 Jenkins 动态创建。或者采用两者组合的方案。两种方法各有其用途，我们将在下面分别讨论它们。

使用 EC2 实例作为构建场的一部分

创建一个新的 EC2 实例就像选择你要使用的基本镜像一样简单。你只需要提供一些有关实例的大小和能力、你想要用来访问机器的私钥的详细信息。Amazon 将基于此镜像创建新的运行的虚拟机。一旦你设置完成，EC2 实例本质上是一台机器，容易而且方便地为你的构建架构设置永久或者半永久 EC2 机器。你甚至可以选择使用 EC2 镜像作为主服务器。

设置某个现有的 EC2 实例作为 Jenkins 从节点和设置任何其他远程的从节点有点不同。如果你正在设置一个 UNIX 或者 Linux EC2 从节点，则需要参考在 AWS 管理控制台用来创建 EC2 实例的私钥文件（见图 11-18）。依据你使用的 Linux 不同，你可能还需要提供用户名。大多数发行版用 root 用户连接，但是有一些比如 Ubuntu，需要不同的用户名。 317

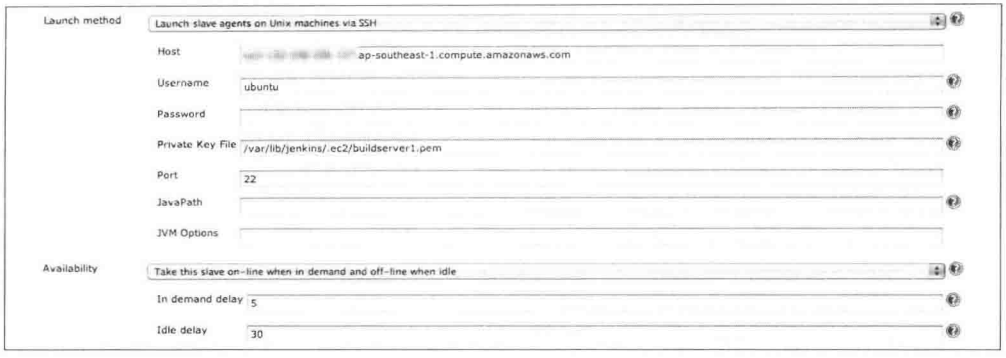


图11-18　配置Amazon EC2从节点

使用动态实例

第二种方法是根据需要动态创建新的 Amazon EC2 机器。设置专用的实例并不难，但它的扩展性不好。更好的办法是让 Jenkins 根据需求创建新的实例。要做到这一点，你将需要安装 Jenkins Amazon EC2 插件。这个插件可以让你的 Jenkins 根据需求在 EC2 云启动从节点，不需要的时候就停掉它们。该插件既能支持 Amazon EC2，也能支持 Ubuntu 企业云。在这里我们只关注 Amazon EC2。请注意在本书写作之际，Amazon EC2 插件仅支持管理 UNIX EC2 镜像。

一旦你已经安装了插件，并且重启了 Jenkins，则到 Jenkins 配置页面单击 Add a New Cloud（见图 11-19）。选择 Amazon EC2。你需要提供 Amazon Access Key ID 和 Secret Access Key，以便 Jenkins 可以与你的 Amazon EC2 账户进行通信。你可以在 EC2 仪表盘的 Key Pairs 屏幕访问这些。

你还需要提供 RSA 私钥。如果还没有的话，则到 Security Credentials 屏幕 Key Pairs 菜单创建一个。这会为你创建新的密钥对并下载私钥。把私钥放在一个安装的地方（当你想要通过 SSH 连接 EC2 实例的时候会用到）。

在高级选项中，你可以使用 Instance Cap 字段限制 Jenkins 启动的 EC2 的实例数。该限制是指积极的 EC2 实例数，而不仅仅是 Jenkins 当前正在运行的总数。这是有用的一项安全措施，因为你要为 EC2 实例活动的时间付费。

一旦配置了整体的 EC2 连接，就需要定义你要用的机器。你可以通过指定自己想要启动的 Amazon Mirror Image（AMI）服务器镜像标识符来进行。Amazon 提供一些初学者的镜像，从社区还可以找到更多镜像，但 EC2 并不支持所有的镜像。在本书写作之际，只有某些基于 32 位的 Linux 发行版镜像能正常工作。

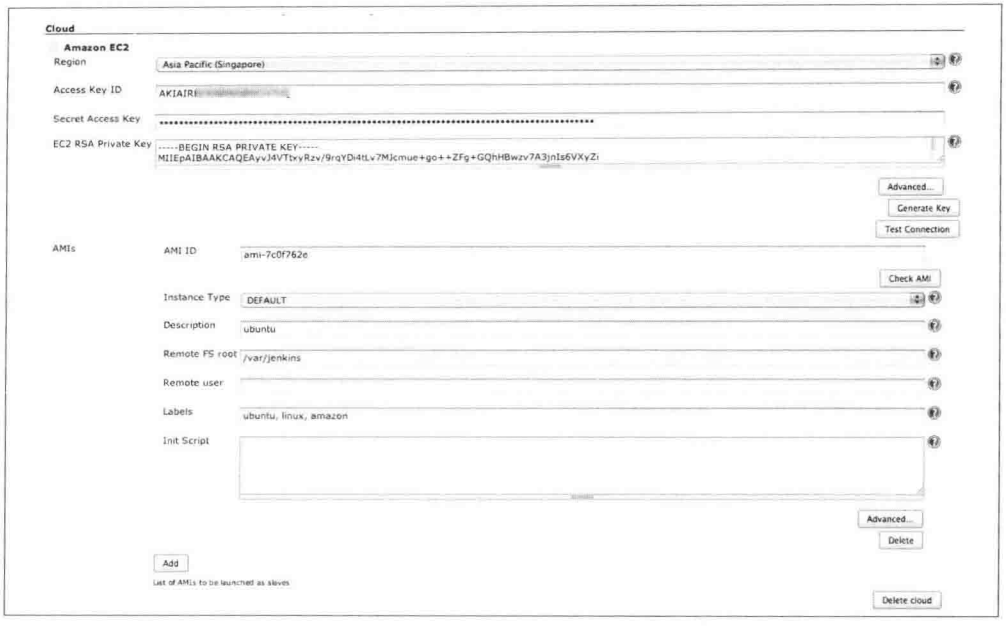


图11-19　配置Amazon EC2从节点

预定义的 Amazon 和公共 AMI 镜像对于你的永久虚拟机是有用的起始点，但是对于实施动态的基于 EC2 的云计算的目的，你需要用预装的必备的工具（Java、构建工具、SCM 配置等）定义 AMI。幸运的是，这个过程很简单：只是从通用 AMI（最好与 Jenkins EC2 插件兼容）开始，安装构建需要的所有东西。确保你使用 EBS 镜像。这样你对服务器实例做的修改保留在 EBS 卷上，以便当服务器关机的时候你也不会失去它们。然后通过在 EC2 管理控制台 Instances 屏幕选择 Create Image（见图 11-20）选项创建新的镜像。确保 Amazon EC2 默认安全组能够 SSH 到构建服务器 IP 地址。如果你不这么做，则当 Jenkins 尝试启动新的从节点时会超时。

一旦你已经准备好镜像，则可以使用它为你的 EC2 配置。

现在，在需要的时候 Jenkins 可以使用这个镜像自动创建新的 EC2 实例，一旦不再需要的时候就删除（在 Amazon，该术语叫作“终止”）实例。或者，你可以使用 Nodes 屏幕的 Provision via EC2 按钮（见图 11-21）手工让新的 EC2 从节点上线。这是测试配置的很有用的方法。

319

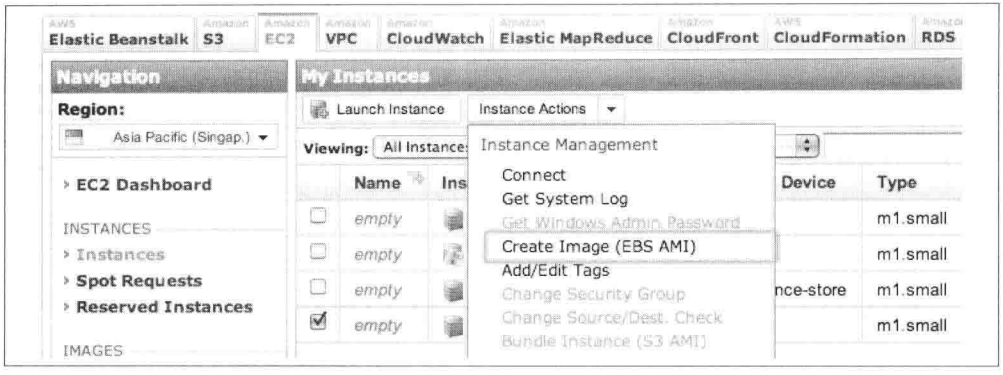


图11-20　创建新的Amazon EC2镜像

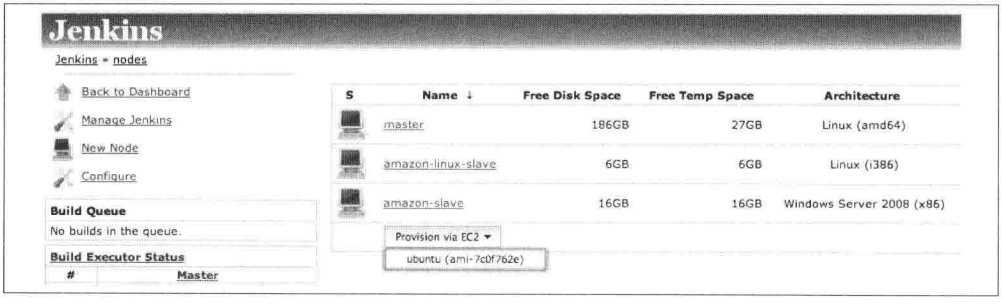


图11-21　手工让Amazon EC2从节点上线

使用 CloudBees DEV@cloud 服务

你可能需要考虑的另一种选择是使用基于专用的基于云的Jenkins架构运行Jenkins实例，比如CloudBees提供的DEV@cloud服务。CloudBees提供Jenkins服务以及各种开发服务（比如Sonar）。使用专用的Jenkins特定服务，就没有必要在你的机器上安装（或管理）Jenkins主节点或者从节点了。主节点实例自动替你配置；当你要构建一个作业时，CloudBees给你提供一个从节点，当作业完成时收回从节点。

320 这个方法和我们前面讨论的基于Amazon EC2的架构有何不同？这种方法主要的优点是管理CI架构硬件的工作大量减少。使用Amazon EC2基础架构意味着你不需要担心硬件，但是你仍然需要管理和配置自己的服务器镜像。CloudBees DEV@cloud架构是一种更高级的、以CI为中心的服务，不仅仅提供Jenkins服务器，还包括如SVN或Git仓库、用户管理和Sonar等其他相关的工具。此外，定价模型（按分钟收费）无疑比Amazon的按小时收费更适合基于云的CI架构。

基于 Amazon EC2 的服务往往（虽然并不总是）用在“混合云”环境，作业运行在云端，但是构建仍在本地。CloudBees DEV@cloud 是公有云的解决方案，整个构建都发生在云端（尽管 CloudBees 也提供了在私有云上运行的类似的解决方案）。

创建一个 CloudBees DEV@cloud 账户很简单，你可以使用免费的试用服务（注意免费的 CloudBees 服务仅有限的一组插件可用，如果使用全部插件则需要注册专业版）。要注册 CloudBees ，进入 signup 页面（*https://grandcentral.cloudbees.com/account/signup*）。你需要输入一些相关的信息，如用户名、电子邮件信息和账户名称。一旦注册成功，你就能访问 DEV@cloud 和 RUN@cloud（实际上是整个 CloudBees 平台）服务。

在这里，你不得不订阅 DEV@cloud 服务。对于我们的目的而言，你可以侥幸地只选择“免费”选项。你需要等待几分钟，因为 CloudBees 要为你部署 Jenkins 主节点。下一步是验证账户（这有助于 CloudBees 防止虚拟账号在服务上运行杂散的作业）。单击验证链接，输入你的电话号码。自动呼入的电话给你提供 pin 码，在表单上输入 pin 码。一旦做到这一点，就可以开始运行构建。

当你连接时第一个呼叫是管理控制台（称为 GrandCentral）。单击“Take me to Jenkins”按钮转到你的新的 Jenkins 主节点实例。

从这里，你与 DEV@cloud 平台的交互就像在一个独立的 Jenkins 一样。当你创建新的构建作业时，只要指向你现有的源代码仓库并且单击构建即可。 DEV@cloud 向你提供从节点（可能需要一到两分钟部署从节点）并且开始构建。

小结

321

在持续集成中，分布式构建是真正可扩展架构的关键。无论你需要添加额外的构建能力，还是构建模式受到周期性峰值需求的影响，分布式构建架构都是吸收额外的负荷极好的方式。分布式构建也能把一些特定的任务，比如特定操作系统特定的 Web 测试委托给某些专用的机器。

一旦你开始分布式构建，基于云的分布式构建场就是一个非常合乎逻辑的延伸。把构建服务器放在云端可使你在需要时更容易、更方便地扩展构建架构。

第 12 章 <323>

自动化部署和持续交付

简介

只要你的应用程序编译正确，持续集成就不应该被停止。只要你可以运行一系列的自动化测试或者为潜在的质量问题进行自动检查和审计，它同样也不应该被停止。只要你达到上述两种情况，逻辑上的下一步就是扩展你的构建自动化过程到部署阶段。这个实践就是通常被大家熟知的自动化部署或者持续交付。

以其最高级的形式来讲的话，持续部署（Continuous Deployment）是自动化测试和其他类似的验证中有任何代码或者主题的变动的话，都能立即部署到生产环境中的一个过程。它的目标是减少周期时间以及减少涉及部署过程的时间和工作量。这也会帮助部署团队减少交付个别功能或者 bug 修复所花费的时间，而且能显著地提供吞吐率。在一个传统的版本发布和部署之前能减少和消除紧张作业时期，也为过程改进和创新释放了时间和资源。这种方法与像 Kanban 这种精益流程所推进的持续改进哲学理念相差无几。

系统性地部署最新代码到生产环境中也不是所有时候都合适（不管你的自动化测试有多好）。很多公司并没有准备好每个星期都接受一个没有宣布的新版本出现；用户需要被培训，产品需要被推向市场，等等。经常在大公司中见到的，在这个主题上一个更保守的方式是：把整个部署过程自动化起来，但是只在一键式的过程中手工触发实际的部署。这就是众所周知的持续交付，它拥有持续部署的所有好处却没有它的那些弊端。持续交付上的变化也可能涉及自动化部署代码到某些环境（比如测试和质量保证），同时为其他的一些环境（比如用户验收测试和生产环境）使用一个人工操作的一键部署的方式。持 <324>
续交付最重要和明显的特征就是，所有通过了相关自动化测试和质量关口的成功的构建，就潜在地可以通过全面的自动化一键过程被部署到生产环境中，并且可以在几分钟内交

付到最终用户手里。不过，这个过程不是自动化的：这是业务，而不仅仅是IT，需要决定一个交付最新更改的最好时间。

持续部署（Continuous Deployment）和持续交付（Continuous Delivery）都被看作代表了自动化构建和SDLC实践的高级别的成熟度。这些技术的存在依赖于一个非常坚实的自动化测试集。它们的存在也依赖于一个持续集成环境和一个强健的构建管道——确实，它通常代表了构建管道的最终阶段和目标。不过，考虑到持续部署/交付能够带给公司显著的好处，这是一个值得追求的目标。在本章的剩余部分，我们将使用“持续部署”这个一般术语来指向持续部署和持续交付。确实，持续交付可以被看作由业务人员而不是开发团队，人工完成了最后一步的持续部署。

实现自动化和持续部署

以其最基本的形态来看，自动化部署可以和你要部署应用程序到一个特定的服务器所写的脚本一样简单。一个脚本的解决方案的主要好处就是简单和易于配置。不过，一个简单的脚本化的方法在你需要执行更多高级的部署活动，比如在一个机器上安装软件或者重启服务器时，会有一些限制。对于更多高级的场景，你可能需要一个更复杂的部署/配置管理解决方案，比如Puppet或者Chef。

部署脚本

一个自动化部署初始化的必要部分就是一个可脚本化的部署过程。尽管这看起来似乎是明显的，不过还是有很多公司的部署保留了一个烦琐、复杂和劳动密集型的过程，包括人工的文件复制、人工的脚本执行、手写的部署笔记等。好消息是，一般来说，我们不必使用这样的方法，而且只需要一点点工作，写一些脚本通常就能把这个过程中的大多数部分自动化。

部署脚本的复杂性随着应用程序的不同也会有着极大的变化。对于一个简单的网站来说，
325 一个部署脚本就像在目标服务器上同步一个目录那么简单。许多Java应用程序服务器可以使用Ant或者Maven插件来部署应用程序。对于一个更复杂的基础设施环境来说，部署可能会涉及在精确的协调之下部署一些应用程序和服务到多个服务器集群上。大多数的部署过程往往在这些极端条件中的某些地方倒下。

数据库更新

把应用程序部署到应用程序服务器上通常只是你解决了难题的一部分。关系型的或者其他类型的数据库，通常会在任何一个应用程序架构中都扮演主要角色。当然，在理想情

况下，你的数据库从一开始的时候就能完美运行（但这在现实情况中很少出现）。确实，当你对应用程序进行更新的时候，一般也需要更新一个或者更多的数据库。

数据库的更新通常要比应用程序的更新更难于平滑地管理，数据库的结构和内容都会被影响到。管理数据库更新是开发和部署过程中非常重要的一部分，值得好好反思和计划。

一些应用程序框架比如在 Rails 和 Hibernate 上的 Ruby，可以在一定程度上自动地管理结构数据库变更。通过使用这些框架，你通常可以指定自己是否想要在每个更新中都重新创建一个新的数据库架构，或者你在想保留已存在的数据时应该更新哪个数据库架构。尽管这些在理论上听起来很有用，但事实上它在除了非紧急的开发环境外的其他方面都有非常大的局限性。尤其是，这些工具在数据迁移方面做得不是很好。例如，假如你重命名了数据库里的一列，更新过程就会简单地创建一个新列：它不会把数据从旧的一列复制到新的一列，也不会从更新了的表里移除旧的一列。

幸运的是，这不是你唯一可以使用的方法。另一个尝试处理数据库更新这个棘手问题的工具就是 Liquibase（*http://www.liquibase.org/*）。Liquibase 是一款开源的工具，它可以使用一个高级别的方法来帮助你在数据库的版本之间对升级路径进行管理和组织。

Liquibase 的工作原理是在数据库中保留应用在表上的数据库更新动作的记录，这样就易于把任何目标数据库改为一个给定版本的应用程序的正确状态。最后，你不需要担心会两次运行同样的更新脚本——Liquibase 也有回滚更改的能力，至少对于某些类型的更改可以进行回滚。不过，因为这不会对每个更改（比如，删除了的表中的数据不能被恢复）都起作用，所以最好不要在这个特殊的功能上有过多的期望。

在 Liquibase 里，你可以把数据库更改当作“更改集”来跟踪，“更改集”中的每一个都以数据库中立的 XML 格式的形式代表一个数据库更新。更改集可以代表任何你想要在数据库里做的更改（从添加和删除表到创建或者更新列、索引和外键）：

```
<databaseChangeLog
xmlns="http://www.liquibase.org/xml/ns/dbchangelog/1.6"                    <326>
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog/1.6
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-1.6.xsd">
  <changeSet id="1" author="john">
    <createTable tableName="department">
      <column name="id" type="int">
        <constraints primaryKey="true" nullable="false"/>
      </column>
      <column name="name" type="varchar(50)">
        <constraints nullable="false"/>
      </column>
```

```
      <column name="active" type="boolean" defaultValue="1"/>
    </createTable>
  </changeSet>
</databaseChangeLog>
```

更改集也可以反映出对现有的一些表的修改操作。例如，下面这个更改集代表一个列的重命名：

```
<changeSet id="1" author="bob">
  <renameColumn tableName="person" oldColumnName="fname" newColumnName="firstName"/>
</changeSet>
```

由于这代表记录了这个更改的语义本质，Liquibase 能够正确地处理和这个更改相关的架构更新与数据迁移。

Liquibase 也可以处理对数据库内容及其结构的更新。例如，下面的更改集将一行新数据插入到一个表里：

```
<changeSet id="326" author="simon">
  <insert tableName="country">
    <column name="id" valueNumeric="1"/>
    <column name="code" value="AL"/>
    <column name="name" value="Albania"/>
  </addColumn>
</changeSet>
```

每一个更改集都有一个 ID 和一个作者，这使得它易于跟踪到底谁做了这个特定的更改，这样能减少冲突的风险。开发人员可以在他们自己的数据库结构中测试其更改集，然后在它们都准备好了之后提交到版本控制系统。下一个明显的步骤就是在任何集成测试或者应用程序部署完成之前，配置 Jenkins 构建自动运行针对合适的数据库的 Liquibase 更新，这通常会作为普通项目构建脚本的一部分。

Liquibase 能很好地集成进构建过程中——可以从命令行执行，也可以集成到一个 Ant 或者 Maven 构建脚本里。比如使用 Maven，你可如下所示配置 Maven Liquibase 插件：

```
<project>
  <build>
    <plugins>
      <plugin>
        <groupId>org.liquibase</groupId>
327     <artifactId>liquibase-plugin</artifactId>
        <version>1.9.3.0</version>
        <configuration>
        <propertyFileWillOverride>true</propertyFileWillOverride>
```

```
        <propertyFile>src/main/resources/liquibase.properties</propertyFile>
      </configuration>
    </plugin>
  </plugins>
</build>
...
</project>
```

用 Maven 和 Liquibase 一起使用的这种方法，你可以使用这个插件来更新一个给定目标数据库到当前的架构：

```
$ mvn liquibase:update
```

默认的数据库连接信息被描述在 *src/main/resources/liquibase.properties* 文件里，看起来可能像下面这样：

```
changeLogFile = changelog.xml
driver = com.mysql.jdbc.Driver
url = jdbc:mysql://localhost/ebank
username = scott
password = tiger
verbose = true
dropFirst = false
```

不过你可以从命令行覆盖所有这些属性，这使得配置一个能更新不同数据库的 Jenkins 构建变得容易。

其他类似的命令能让你生成一个 SQL 脚本（比如，你需要提交它到本地 DBA 去获取批准），或者回滚到这个架构的之前一个版本。

这当然只是一种可能的方法的一个例子。其他一些团队更喜欢手工维护一系列的 SQL 更新脚本，或者编写他们自己内部的解决方案。最重要的事情就是，在部署应用程序时，有一个你可以可靠地、可重复地使用的解决方案来更新不同的数据库到一个正确的状态。

冒烟测试

任何严肃的自动化部署都需要被一系列自动化冒烟测试跟访。自动化验收测试的一个子集就是冒烟测试的一个好的候选项。冒烟测试应该是不引人注目但却相当快的。它们运行在生产环境当中也应该是安全的，这样就限制了可以在系统里对测试用例所做修改的数量。

328

回滚更改

在配置自动化部署的时候，另外一个需要考虑的重要方面就是，如果一些地方出了问题，特别是你想要完成持续部署的时候，怎样回滚到上一个版本。的确，在需要的时候能回滚到之前的一个版本是非常关键的。

你怎么做到这一点在很大程度上依赖于你的应用程序。虽然简单明了地使用 Jenkins 重新部署应用程序的前一个版本（我们将会在本章的后面看一种能做到这个的技术），但应用程序并不是依赖的唯一方面。尤其是，你需要考虑怎样把数据库恢复到之前的状态。

我们已经看过使用 Liquibase 去管理数据库更新的可能，当然许多其他策略也有可能。不过，回滚一个数据库版本对自身提出了挑战。比如 Liquibase，但不是全部更改到数据库结构中，而只是能够让你恢复一些。不过丢失的数据（比如在删除了的表中的数据）就不是简单使用 Liquibase 就能恢复的了。

恢复数据库到先前状态的最可靠办法可能就是在升级之前对数据库做快照，然后使用这个快照来把数据库恢复到先前的状态。一种有效的方式就是在 Jenkins 中的部署构建作业上，把这个过程自动化起来，然后把数据库快照以及可用来部署的二进制文件保存为构建产物。通过这种方法，你就可以轻松地使用保存的快照恢复数据库，并且使用保存的二进制文件重新部署应用程序。我们会在本章后面看一个这种策略的例子。

部署到应用程序服务器上

Jenkins 提供了插件来帮助你把应用程序部署到一定数量的常用应用服务器上。Deploy 插件能够让你部署到 Tomcat、JBoss 和 GlassFish 上。还有 Deploy Websphere 插件试着能满足 IBM WebSphere 应用程序服务器的特殊需求。

对于其他应用服务器来说，你通常不得不把部署过程集成到构建脚本里，或者求助于定制化的脚本来部署应用程序。同样，对于其他语言，你的部署过程会多种多样，但通常会涉及一些 Shell 脚本的使用。例如，对于一个 Ruby on Rails 程序，你可能会使用一个像 Capistrano 或者 Chef 这样的工具，或者简单地使用一个 Shell 脚本。对于一个 PHP 应用程序，一个 FTP 或者 SCP 文件传输可能就能满足。

让我们首先看一些用来部署 Java 应用程序到一个应用服务器的策略。

这被叫作热部署，就是应用程序被部署到一个运行着的服务器上。这一般也是一个让你的应用程序在线上的快速有效的方法。不过，依赖于应用程序和应用服务器，很多人都知道这种方法会导致一个内存缺乏或者资源锁问题——比如老版本的 Tomcat，就因为这

个而众所周知。如果你遇到了这种问题，可能不得不在每个部署完成之后强制重启应用程序，或者在测试机上安排好应用服务器在夜间进行重启。

部署一个 Java 应用

在此我们将一起看一个怎样部署 Java Web 或者 JEE 应用程序到如 Tomcat、JBoss 或者 GlassFish 这种应用服务器上的例子。

自动化部署的一个基本原则之一就是要能够重用二进制库。在部署过程中重新构建应用程序是没效率的，也可能不可靠。的确，设想你在部署应用程序到一个测试环境中以便将来进行更多的测试之前，针对这个应用程序运行了一系列的单元和集成测试。如果你在把它部署到测试环境之前重新构建了二进制文件，源代码可能从原始版本就已经改变了，这意味着你可能无法确切地知道自己正在部署什么。

一个更有效的过程就是重用先前一个构建生成的二进制文件。比如，你可能配置一个构建作业在生成一个可部署的二进制文件（通常是一个 WAR 或者 EAR 文件）之前运行单元和集成测试。你可以使用 Copy Artifact 插件（参见第 10 章的“复制构建产物”一节）来做这件事情。这个插件能让你从另一个构建作业工作空间中复制一个构建产物到当前构建作业工作空间。这样，当和一个普通的构建触发器或者 Build Promotion 插件结合使用的时候，就能让你精确地部署在先前阶段的构建和测试生成的二进制文件。

这种方式不会在你构建应用程序的过程中加一些限制。特别地，任何特殊环境相关的配置都必须具体化到应用程序；例如，JDBC 连接或者其他这样的配置细节不应该被定义在内嵌到 WAR 文件中的配置文件里，而是应该使用 JDNI 或者被定义在一个外部化的属性文件中。如果不是这种情况，你可能就需要从一个给定的 SCM 修正版来进行构建，就像在第 10 章的“使用 Subversion 标签构建”一节中关于 Subversion 的讨论那样。

使用 Deploy 插件

如果你正在部署到一个 Tomcat、JBoss 或者 GlassFish 服务器，你倾向的最有用的工具可能就是 Deploy 插件。这个插件使得把集成这些平台上的部署到你的 Jenkins 构建过程变得相当简单明了。如果你正在部署到 IBM Websphere 上，则可以使用 Websphere Deploy 插件得到类似的结果。

通过使用在图 12-1 中简单的自动化构建和部署管道，让我们看一下在行动上如何让这个插件工作。

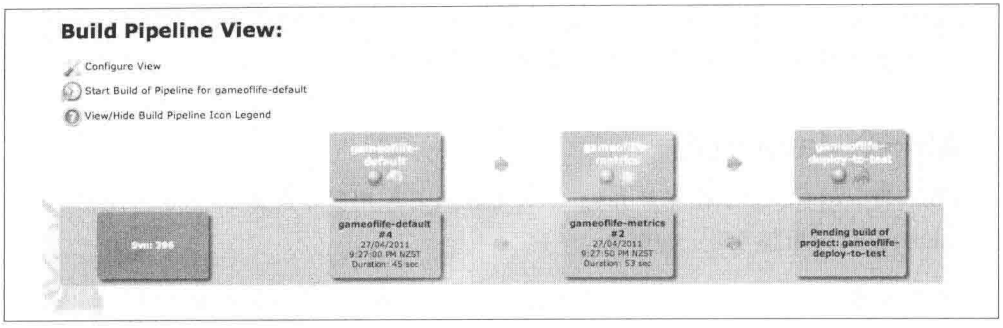


图 12-1　一个简单的自动化部署管道

在这里，默认的构建（*gameoflife-default*）运行单元和集成测试，然后构建一个 WAR 文件形式的可部署的二进制库。度量构建（*gameoflife-metrics*）运行一些关于编码标准和代码覆盖率的额外检查。如果这两个构建都成功了，这个应用程序就会被 *gameoflife-deploy-to-test* 构建作业自动化地部署到测试环境。

在 *gameoflife-deploy-to-test* 构建作业里，我们使用 Copy Artifact 插件去获取在 *gameoflife-default* 构建作业中生成的 WAR 文件并且通过 *gameoflife-deploy-to-test* 构建作业把它复制到当前构建作业的工作空间（见图 12-2）。

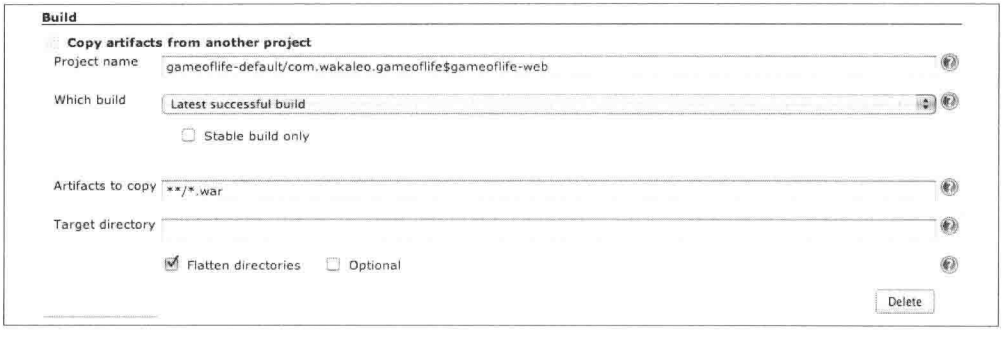


图 12-2　复制二进制的构建产物用来部署

接下来，我们使用 Deploy 插件部署这个 WAR 文件到测试服务器上。当然，一般来说它是可行的且不难去写一个手动部署脚本来把应用程序放到应用程序服务器上。在一些情况下，这或许是你唯一的选择。不过，如果有你的应用程序可用的 Jenkins 插件的话，使用它就变得相当简单了。如果你要部署到 Tomcat、JBoss 或者 GlassFish 的话，Deploy 插件对你来说就有用了。这个插件使用 Cargo 来连接应用服务器并且部署（或者重部署）应用程序。只需选择目标服务器的类型，然后指定一个带有部署权限的用户的用户名和密码的服务器 URL（见图 12-3）。

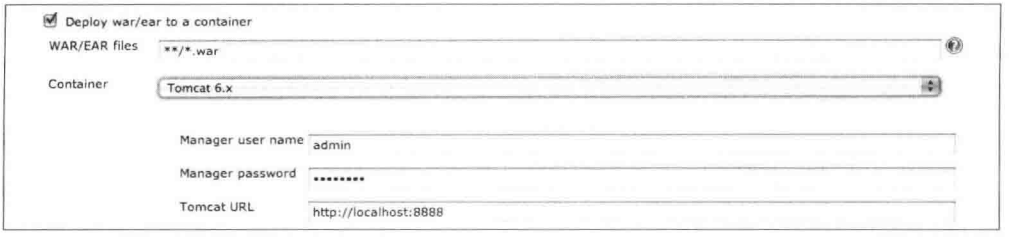


图 12-3　使用Deploy插件部署到Tomcat

这被叫作热部署，就是应用程序被部署到一个运行着的服务器上。这一般也是一个让你的应用程序在线上的快速有效的方法。不过，依赖于你的应用程序和你的应用服务器，很多人都知道这种方法会导致一个内存缺乏或者资源锁问题——比如老版本的 Tomcat，就因为这个而众所周知。如果你遇到了这种问题，你可能不得不在每个部署完成之后强制重启应用程序，或者在你的测试机上安排好你的应用服务器在夜间进行重启。

部署一个指定的版本

当你想要自动且持续地部署应用程序时，准确地识别当前的应用程序版本就变得非常重要了。有几种方法可以帮助你做到这些，从 Jenkins 在构建 / 部署架构中扮演的角色角度来看，这些方法有本质上的不同。

一些团队把 Jenkins 作为事实上的核心来使用，在这里构建产物被构建和存储以备将来参考。如果你把可部署的构建产物保存在 Jenkins 上，那么直接从你的 Jenkins 实例上部署构建产物就有完美的意义了。这并不难做到：下一节我们会看看怎样结合使用 Copy Artifacts、Deploy 和 Parameterized Trigger 插件来做到这一点。

作为另外一种选择，如果你正在使用一个企业级的仓库比如 Nexus 或者 Artifactory 来存储构建产物，那么这个仓库就应当扮演参考的中心点：Jenkins 应该构建和部署构建产物到你的中心仓库，然后从这儿部署它们。这就是如果你正在使用 Maven 作为构建工具时的典型案例；不过正在使用比如 Gradle 或者 Ivy 这些工具的团队也会使用这种方法。比如像 Nexus 和 Artifactory 这样的仓库管理者，特别是使用商业版本的，通过提供诸如构建进阶和分段仓库这些能帮助你管理构建产物发布状态的功能来使得这个阶段更易于实施。

让我们一起看看你可能会怎样使用 Jenkins 实施这些策略。

从先前的一个 Jenkins 构建部署一个版本

重新部署一个先前在 Jenkins 中部署过的构建产物是相当简单的。在本章前面的“使用 Deploy 插件”一节，我们已经看过了如何使用 Copy Artifacts 和 Deploy 插件来部署一个

先前构建作业构建的 WAR 文件到一个应用程序服务器上。现在我们需要做的就是让用户指定要部署的版本，而不是仅仅部署最新的构建。

我们可以使用 Parameterized Trigger 插件来做这件事（参见第 10 章的“参数化构建作业”一节）。首先，我们要添加一个参数到构建作业里，使用指定的“Build selector for Copy Artifact”参数类型（见图 12-4）。

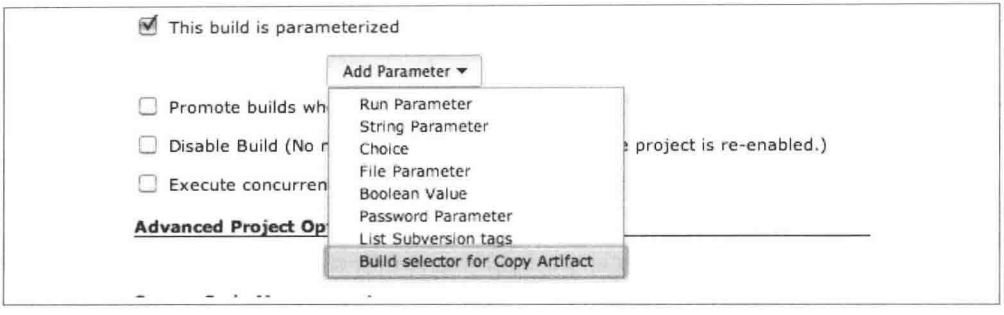


图 12-4　添加一个“Build selector for Copy Artifact”参数

这将添加一个新参数到构建作业中（见图 12-5）。在此需要输入一个名字和短描述。你提供的名字将会被用作传递给随后构建步骤的环境变量。

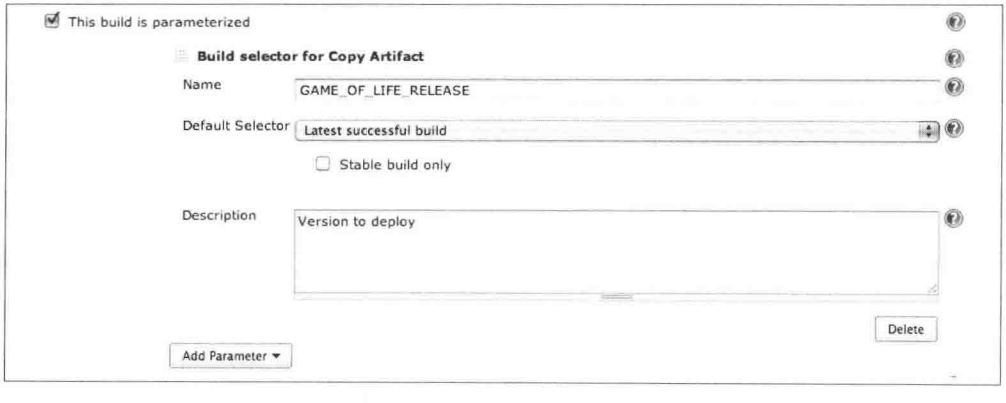


图 12-5　配置一个构建选择器参数

333 这个构建选择器参数类型可以让你用好几种方法挑选一个之前的构建，包括最近成功的构建、触发这个构建的上游构建或者一个指定的构建。所有这些选项在用户来触发一个构建的时候都是可用的。Default Selector 能让你指定这些选项中的哪一个会被默认地推荐。

当用户选择了一个特殊的构建作业时，构建号也会被存储在环境变量中，以便构建步骤中使用。这个环境变量叫作 *COPYARTIFACT_BUILD_NUMBER_MY_BUILD_JOB*，而 *MY_*

BUILD_JOB 就是原始的构建作业的名字（字母大写并且把除了 A~Z 的字母都转为下画线）。例如，假如我们从 *gameoflife-default* 的构建号为 4 的构建中复制一个构建产物，那么 COPYARTIFACT_BUILD_NUMBER_GAMEOFLIFE_DEFAULT 环境变量就会被设置为 4。

配置的第二部分就是告诉 Jenkins 要获取什么以及从哪个构建作业中获取。在项目配置的 Build 部分，我们添加一个“Copy artifacts from another project”步骤。在此可以指定构建产物构建和归档的项目（在我们的例子里是 *gameoflife-default*）。你也需要使得 Jenkins 能够使用我们先前在参数里定义了的构建。可以通过在“Which Build”选项中选择“Specified by a build parameter”来做到这一点，并且提供我们先前在构建选择器的名称字段指定的变量名称（见图 12-6）。然后，仅需配置中间构建产物为像我们在先前的例子中做的一样去复制。

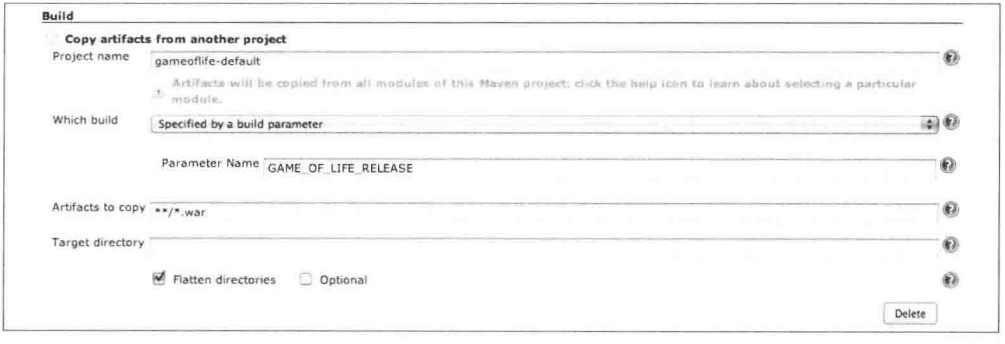
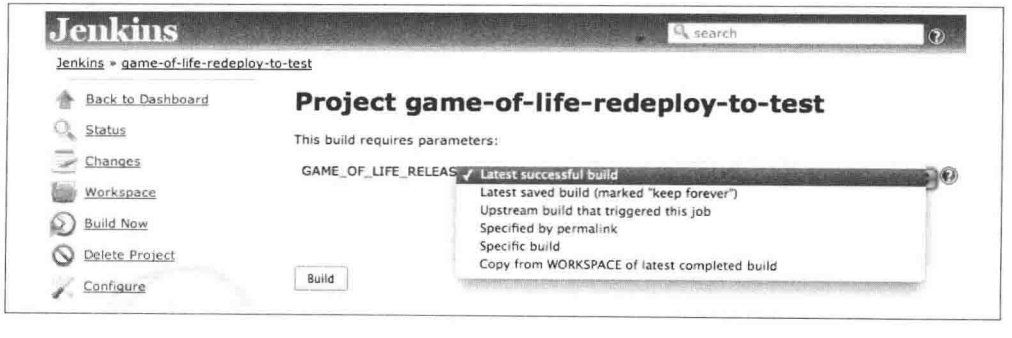


图 12-6　指定从哪里找到要被部署的构建产物

最后，我们使用 Deploy 插件来部署复制的构建产物，正如图 12-3 中说明的一样。

让我们看看在实际情况中这个构建是如何工作的。当我们手动启动一个构建的时候，Jenkins 会推荐一个选项列表来让你选择重新部署构建（见图 12-7）。

这些选项中的大多数都是一目了然的。



334

图 12-7　选择要重新部署的构建

“latest successful build”是除失败的构建之外最近的构建。所以这个选项通常就会仅再次重新部署最近的版本。假如你使用了这个选项，则可能会想要勾选“Stable builds only”复选框，这也会排除任何一个不稳定的构建。

如果你已经选择了丢弃旧的构建，则能够把某个构建作业标记为永远保留（参见第 5 章的“通用选项”一节）。在这种情况下，你可以选择部署“Latest saved build”。

对于一个位于构建管道结尾处的自动化构建作业来说，一个明智的选择就是“Upstream build that triggered this job”。用这种方法，你可以确定自己正在部署的构建产物就是先前的构建作业生成的（或者进阶），即使其他构建已经发生了。值得注意的是，尽管这种参数化的构建作业经常被用来手动部署一个指定的构建产物，但它也可以被有效地用作一个自动化构建过程的一部分。如果没有手动触发，它会简单使用你在“default selector”中定义了的任何字段值。

你也可以选择“Specified by permalink”选项（见图 12-8）。这能够让你从一些快捷键值里选择，例如 Last build 、Last stable build、Last successful build，等等。

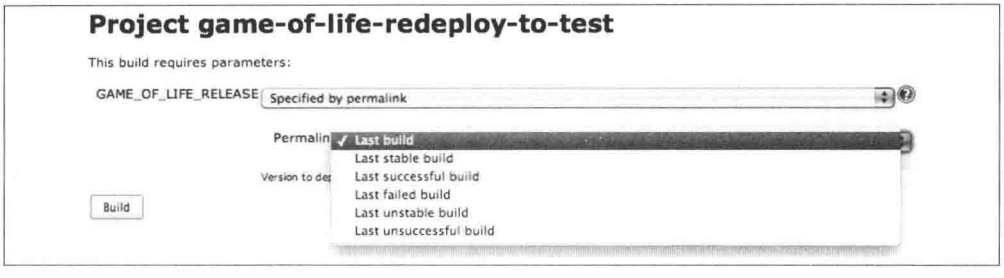


图 12-8　使用“Specified by permalink”选项

335 但是如果你想部署一个特定版本的应用程序，则一个更有用的选项是“Specific build”（见图 12-9）。这个选项能够让你提供一个具体的要部署的构建号。这是重新部署一个应用程序最灵活的方式——你只需要知道自己需要重新部署的构建的构建号，而且通常并不是太难找到（可通过查看最初的构建作业的构建历史查找）。

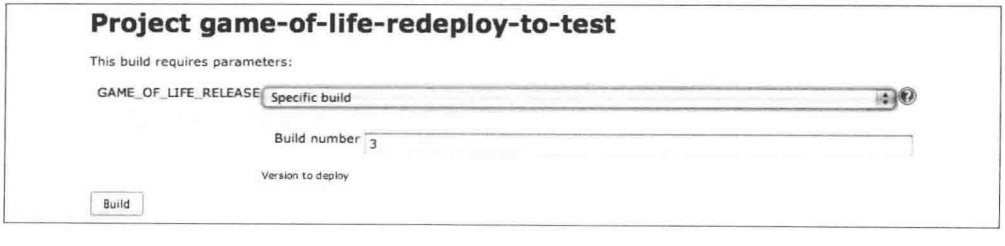


图 12-9　使用一个特定的构建

这是从先前的 Jenkins 构建作业部署或者重新部署中间构建产物的一种便捷方式。然而，在一些情况下你可能更喜欢使用一个存储在企业级仓库里的中间构建产物，比如 Nexus 或者 Artifactory。我们将在下一节介绍一个如何做到这一点的例子。

从 Maven 仓库部署一个版本

许多公司使用一个企业级的仓库管理工具比如 Nexus 和 Artifactory 来存储和共享二进制构建产物，如 JAR 文件。这种策略通常和 Maven 一起使用，但也会和其他构建工具如 Ant（Ivy 或者 Maven Ant Tasks）和 Gradle 一起使用。在一个 CI 环境中使用这种方法，快照和版本依赖关系都构建在你的 Jenkins 服务器上，然后再部署到仓库管理工具上（见图 12-10）。无论什么时候，只要一个开发人员提交源代码改动到版本控制系统，Jenkins 就会提取这些变化并且为相应的构建产物构建新的快照版本。然后 Jenkins 把这些快照构建产物部署到本地企业仓库管理器，在这儿它们可以被其他的开发人员或者公司内部的其他团队获得。我们在图 12-10 中讨论了如何让 Jenkins 自动化部署 Maven 构建产物到一个企业仓库。类似的方法也可以使用 Gradle 或者 Ivy。

336

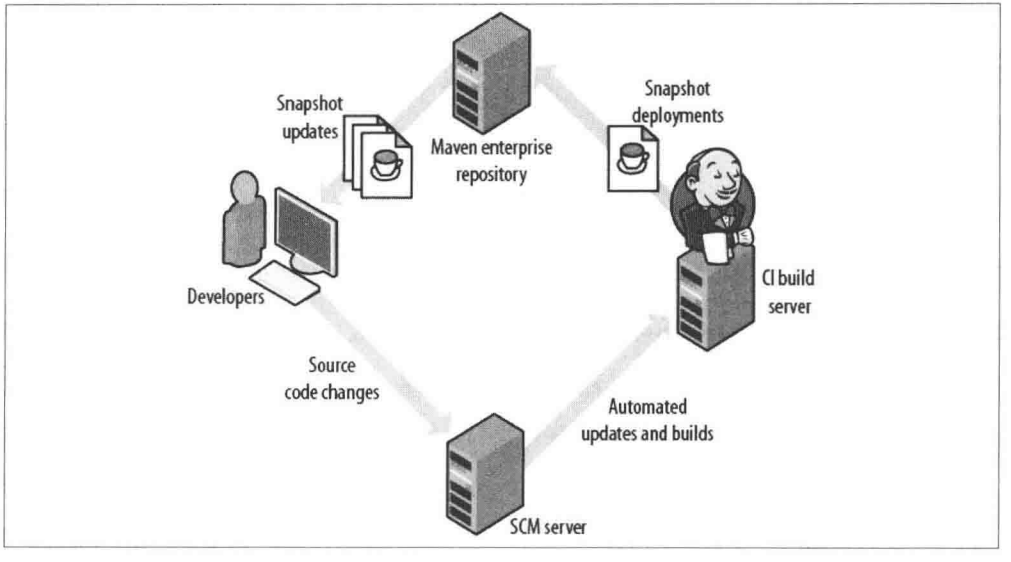


图 12-10　使用Maven企业仓库

Maven 约定使用定义良好的系统版本号，区分 SNAPSHOT 和 RELEASE 版本。SNAPSHOT 版本被认为是潜在的不稳定的最新代码库的构建，而 RELEASE 版本则是经历了更正式的发布过程的官方发布版本。通常，SNAPSHOT 构建产物留作一个开发团队内部使用的，而 RELEASE 版本被认为是准备好用作进一步测试的。

类似的方法可以用来部署构建产物，如 WAR 或者 EAR 文件——它们在 CI 服务器上被

构建和测试，然后自动部署到企业仓库，这通常会作为涉及自动化测试和质量检测的构建管道的一部分（参见第 10 章的“构建管道及优化”一节）。SNAPSHOT 版本通常会被部署到一个测试服务器上用于自动化或者手工的测试，以便决定一个版本是否准备好了正式发布。

决定什么时候创建一个发布版本以及怎样部署，在一个公司内部是多种多样的。比如，一些团队更喜欢在每次迭代或者冲刺结束后正式发布，发布的版本要有明确定义的版本号和相应的发行说明，这些要被分发给 QA 团队用于以后的测试。当一个特定版本得到 QA 团队的认可后，它才能被部署到生产环境中。其他团队使用更精益的方法，更喜欢在一个新的功能或者修复的 bug 准备好被部署的时候切割出一个新的版本。如果一个团队对于他们的自动化测试和代码质量检测特别有自信的话，那甚至可能完全自动化这个过程，定期（每晚）或者当提交了新的变化的时候生成和发布一个新的版本。

有很多方式可以完成这种策略。在本节的其余部分，我们将会看到如何使用传统的多模块 Maven 项目来做到这一点。我们的示例项目是一个名为 *gameoflife* 的网络应用，包括三部分：*gameoflife-core*、*gameoflife-services* 和 *gameoflife-web*。*gameoflife-web* 模块生成一个包括来自其他两个模块的 JAR 文件的 WAR 文件。下面就是一个我们想要部署的 WAR 文件：

```
tuatara:gameoflife johnsmart$ ls -l
total 32
drwxr-xr-x  16 johnsmart staff   544 16 May 09:58 gameoflife-core
drwxr-xr-x   8 johnsmart staff   272  4 May 18:12 gameoflife-deploy
drwxr-xr-x   8 johnsmart staff   272 16 May 09:58 gameoflife-services
drwxr-xr-x  15 johnsmart staff   510 16 May 09:58 gameoflife-web
-rw-r--r--@  1 johnsmart staff 12182  4 May 18:07 pom.xml
```

337

在本章的早前部分我们已经看到如何使用 Deploy 插件去部署一个当前构建作业生成的 WAR 文件到一个应用程序服务器上。现在我们想做的是部署一个 WAR 文件的任意版本到一个应用程序服务器上。

在第 10 章的“通过 M2Release 插件管理 Maven 版本发布”一节，我们讨论了如何配置 Jenkins 调用 Maven Release 插件生成一个应用的正式发布版本。部署过程的第一步就从这里开始，所以我们会认为这已经做了配置并且一些发布版本已经被部署到我们的企业仓库管理器。

下一步涉及创建一个专门的项目来管理部署过程。这个项目将会是一个标准的 Maven 项目。

你首先需要做的第一件事是建立一个专门的部署项目。就其最简单的形式来讲，这个项目会简单地从企业仓库中获取所请求的版本的 WAR 文件，通过 Jenkins 进行部署。在接

下来的 *pom.xml* 文件中，我们使用 *maven-war-plugin* 从企业仓库中获取一个指定版本的 *gameoflife-web WAR* 文件。我们想要的版本是在 `target.version` 属性中指定的：

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.
        w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
  maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.wakaleo.gameoflife</groupId>
  <artifactId>gameoflife-deploy-with-jenkins</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <dependencies>
    <dependency>
      <groupId>com.wakaleo.gameoflife</groupId>
      <artifactId>gameoflife-web</artifactId>
      <type>war</type>d
      <version>${target.version}</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <configuration>
          <warName>gameoflife</warName>
          <overlays>
            <overlay>
              <groupId>com.wakaleo.gameoflife</groupId>
              <artifactId>gameoflife-web</artifactId>
            </overlay>
          </overlays>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <properties>
    <target.version>RELEASE</target.version>
  </properties>
</project>
```

338

接下来，我们使用一个用户提供的属性值来配置一个 Jenkins 构建作业去调用 *pom.xml* 文件（见图 12-11）。注意我们已经设置默认值为 RELEASE，所以默认情况下，最近的发布版本会被部署。另外，用户可以提供版本的版本号来进行部署或者重新部署。

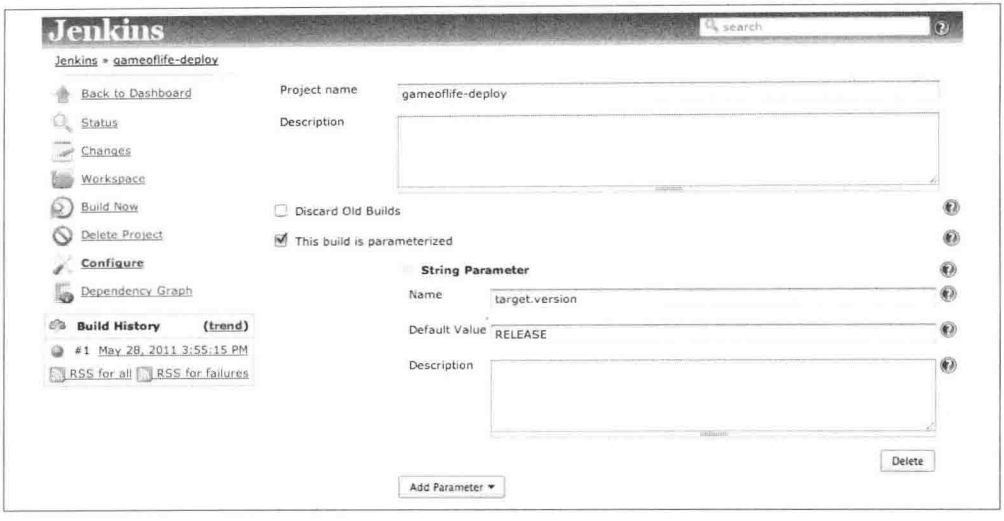


图 12-11　从Maven仓库部署一个构建产物

这个构建作业的剩余部分就是简单地检查部署项目和调用 `mvn package`，然后使用 Deploy 插件部署 WAR 文件（见图 12-12）。`target.version` 属性将会被自动传送到构建作业中以及被用于部署正确的版本。

类似的技术也可以被用于其他的项目类型。如果你正在部署到一个 Deploy 插件不支持的应用服务器，则也可以选择使用你觉得最方便的编程语言编写一个定制化脚本，并且像上面描述的那样让 Jenkins 把请求的版本号作为参数传递。

339

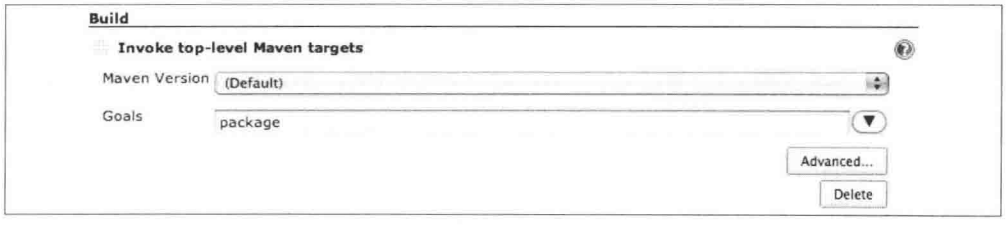


图 12-12　准备要被部署的WAR

部署像 Ruby 和 PHP 这样基于脚本的应用程序

部署使用像 PHP 和 Ruby 这样的脚本语言的项目一般要比部署 Java 应用程序简单（尽管与数据库更新相关的问题是类似的）。事实上，这些部署本质上都经常涉及复制文件到一个远程服务器上。要想在第一时间获取文件，你可以选择使用 Copy Artifacts 选项从另外一个构建作业的工作空间复制它们，或者直接从源代码库中检查源代码，如果有必

要的话可以使用详细的修正描述或者标签，就像在第 10 章的“使用 Subversion 标签构建”一节中关于 Subversion 的描述和在第 10 章的“使用 Git 标签构建”一节中关于 Git 的描述那样。之后，只要你已经把源代码放到了 Jenkins 工作目录下，则只需简单地部署它们到目标服务器上。

对于这种部署，一个有用的工具就是用于 Jenkins 的 Publish Over 系列插件（Publish Over FTP、Publish Over SSH 和 Publish Over CIFS）。这些插件提供了一种一致且灵活的方式通过一系列的协议，包括 CIFS（用于 Windows 下共享驱动器）、FTP 以及 SSH/SFTP 把你的应用程序构建产物部署到其他的服务器上。

这些插件的配置是相似的。一旦你已经安装了插件，就需要设置主机配置，它们集中在主要配置界面管理。你可以创建尽可能多的主机配置——它们会出现在一个作业配置页面的下拉列表里。

主机的配置是不言自明的（见图 12-13）。名字就是将会出现在构建作业配置中的下拉列表里的名字。你可以使用一个用户名和密码为 FTP 配置验证，或者为 SSH 配置一个 SSH 密钥或用户名和密码。你也需要提供一个在远程服务器上的已有目录作为此次配置的根目录。在 Advanced 选项中，你也可以配置 SSH 端口和超时选项。

340

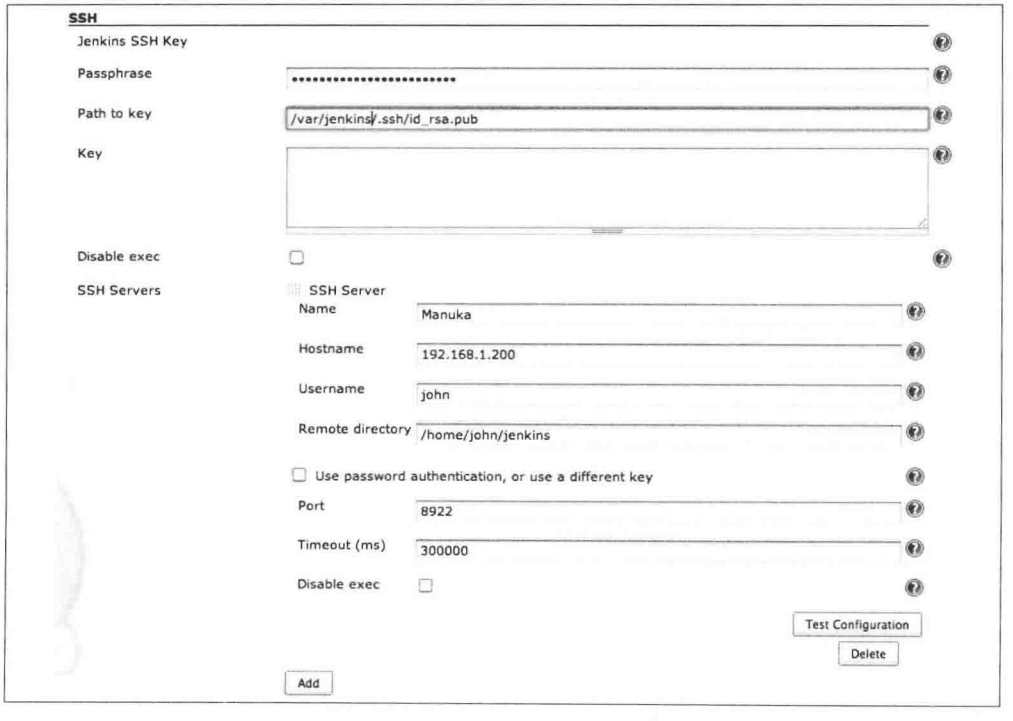


图 12-13　配置一个远程主机

只要你配置了主机，就可以建立构建作业来部署构建产物到这个主机上。可以当作一个构建步骤（见图 12-14）或一个构建后动作来做这件事（见图 12-15）。在这两种情况下，这些选项是类似的。

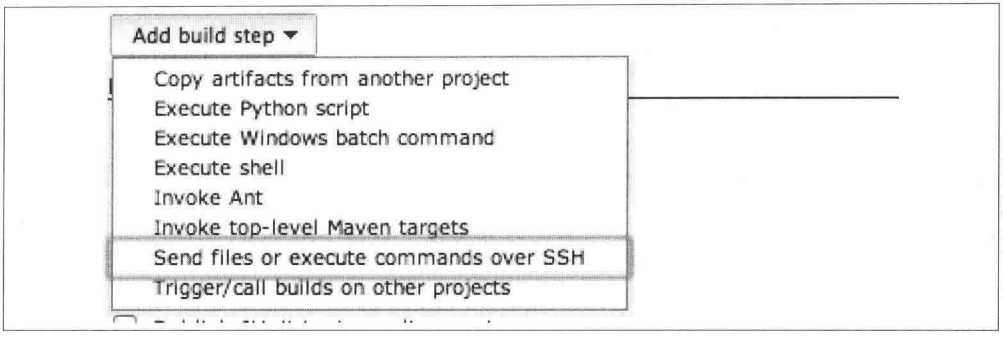


图 12-14　在构建部分部署文件到一个远程服务器上

341

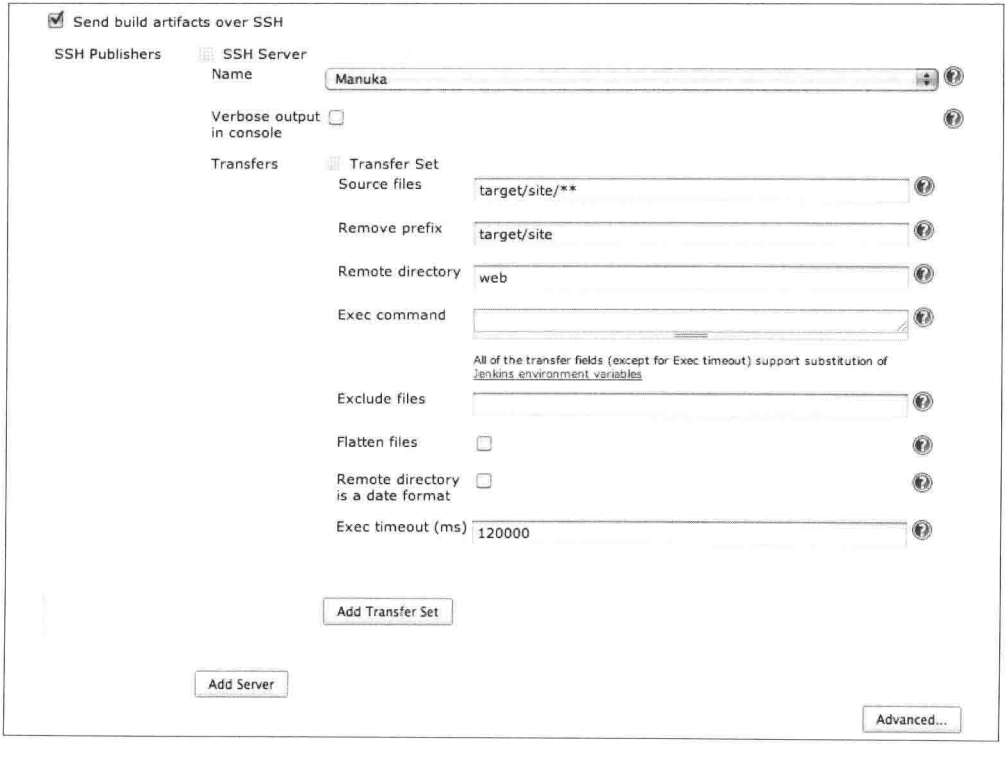


图12-15　在Post-build Actions中部署文件到远程主机中

首先，你需要从前面配置了的主机列表中选择出目标主机。接下来，需要配置你想要传

送的文件。你可以通过一个或多个“Transfer set”来做这件事。一个 Transfer set 就是一系列你部署到远程服务器上的一个指定目录的文件（通过一个 Ant 文件集表达式定义）。你也可以提供一个可以被删除的前缀——这样可以使你去掉一些不必要的、你不想它们在服务器上出现的目录（比如例子中的 *target/site* 目录路径）。你可以根据自己的需要添加尽可能多的转移集到一个远程服务器上。这个插件也提供一些选项来在远程服务器上执行命令（“Exec 命令”）或者排除某些文件，抑或使目录扁平化。

小结

自动化部署，其更高级形式为持续部署或者持续交付，被认为是现代持续集成基础设施的高峰。

在本章，我们已经浏览了一些自动化部署的技术，其大部分是围绕基于 Java 的部署。不过，这里讨论的一般原则是适用于任何一种技术的。事实上，在许多其他技术下的实际部署过程，特别是脚本语言比如 Ruby 和 PHP，比使用 Java 更简单，基本上涉及把文件复制到生产服务器上。Ruby 也受益于像 Heroku 和 Capistrano 这样的工具，以便更方便地执行任务。

在建立一个自动化部署的时候，有几个重要的方面需要考虑。首先，自动化部署是持续集成架构的最后一步：你需要定义一个构建管道从原始的编译和单元测试接管构建，还有更复杂的功能和自动化验收测试以及代码质量检测，最后部署到一个或者多个平台。你对构建管道的自信程度很大程度上依赖于你对测试的自信程度。或者，换句话说，你的测试越不可靠和不广泛，在构建过程中你就会越早落入手动测试和人工干预中。 342

最后，如果可能的话，构建你可用于部署的构建产物一次且只构建一次是非常重要的，然后在后续用于功能测试的步骤中重用它，以及部署到不同的平台上。

第 13 章 <343>

Jenkins的维护

简介

本章我们将讨论一些维护大型 Jenkins 实例的实用技巧。这其中包括如何限制和监控磁盘使用情况，如何分给 Jenkins 足够的内存，如何完成构建作业，以及如何将 Jenkins 从一台服务器迁移到另一台服务器。本书其他章节也会涉及部分上述内容，但在本章中我们将试着从系统管理员的视角讨论这些主题。

监控磁盘空间

构建的历史信息需要大量磁盘空间存储。此外，加载项目文件配置后，Jenkins 也需要花费时间分析这些构建记录。因此加载包含 1000 个归档构建的构建作业将远比加载含 50 个归档构建的构建作业耗时得多。如果你有拥有几十或几百个构建作业的一个大的构建服务器，耗时相应地增加。

或许控制磁盘占用最简单的方法是限制每个项目构建历史的存储数量，你可以在项目配置页面（参见图 13-1）通过勾选顶部的 Discard Old Builds（“清除历史构建”）复选框进行配置。如果你将保存历史构建的数量设定为 20，那么一旦构建数量超过这个数字，Jenkins 就会开始删除旧的构建。你可以通过设定数字（如不多于 20 个）或者时间（如 30 天以内）来限制历史构建的数量。Jenkins 会智能地保留最后一次成功的构建；你不必担心构建历史的自动清理会一并删除这些信息。

使用“清除历史构建”的问题在于可能使构建历史信息变得不精确。Jenkins 使用构建记录绘制构建度量和测试结果的图表。假如将保存的历史构建数量限定为 20，Jenkins 绘制的图表也将至多只包含这 20 次构建的数据。这些历史数据对于开发者而言可能十分

有用，因此在绘制图表时最好能展示所有项目相关的构建度量和结果，而不仅仅是过往一两周的数据。

344

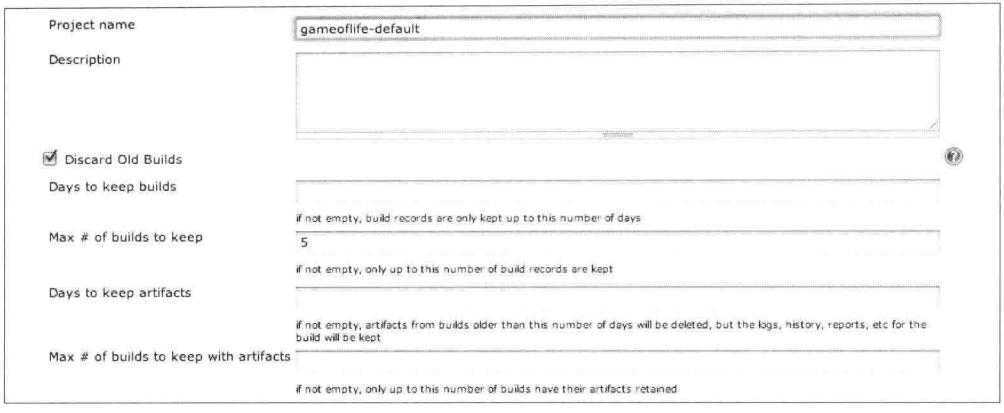


图13-1　清除历史构建

幸运的是，Jenkins 还提供了一种系统管理员和开发人员都会满意的变通方法。一般情况下，占用磁盘空间最多的是构建产物如 JAR 文件、WAR 文件等。构建历史主要存储在 XML 格式的日志文件里，并不会占用太多空间。你单击 Advanced 按钮，Jenkins 允许仅清除构建版本文件，保留构建数据。例如在图 13-2 中，可将保存构建版本的时间设定为最多 7 天。这种设定可以在很好地限制磁盘占用的同时保存开发团队所需要的全部历史构建信息。

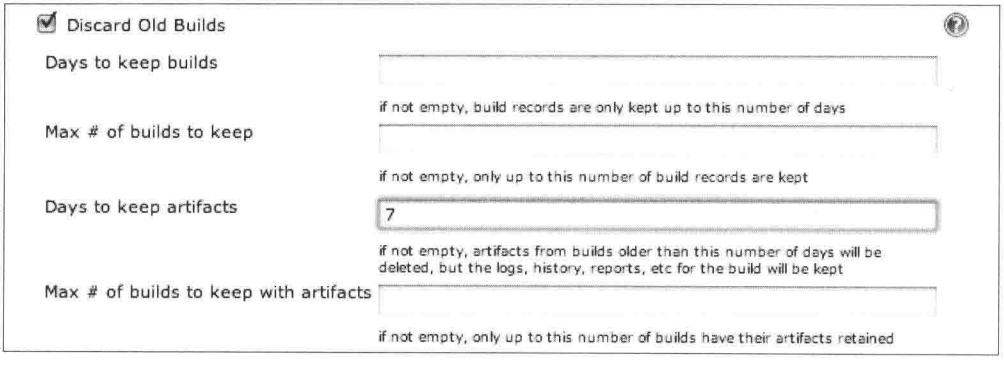


图13-2　清除历史构建——高级选项

不必担心保留最大数目构建的构建产物。无论你怎么设定，Jenkins 总会保留最新的稳定且成功的构建。因此，你总会有至少一个可用的构建版本。你还可以将某个构建版本标记为“永久保存”，以避免类似的重要构建信息被自动删除。

345

使用 Disk Usage 插件

Disk Usage（磁盘使用）插件是 Jenkins 管理工具中最为重要的插件之一。这个插件会记录和报告项目的磁盘使用空间。你可以使用它对占用大量磁盘空间的项目进行管理。

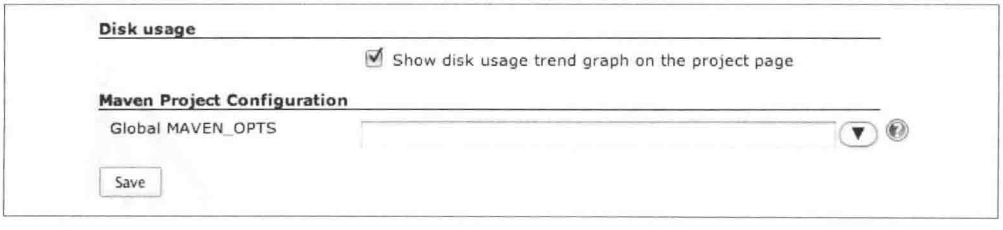
你可在 Plugin Manager 界面安装 Disk Usage 插件。当插件安装完成，且 Jenkins 重启后，Disk Usage 插件将开始记录每个项目的磁盘占用情况。你也可以在 Manage Jenkins 界面上单击“Disk Usage”链接来查看所有项目的磁盘使用情况（图 13-3）。



图13-3　磁盘使用情况

这个列表是按照总磁盘占用进行排序的，占用磁盘最多的项目排在列表顶部。表中每个项目有两列值：Builds 列的数字代表了该项目以往所有构建历史所占用的总磁盘空间，Workspace 列的数字代表构建项目所占用的空间。对正在进行中的项目，Workspace 列的数字趋近于稳定；而 Build 列的数字会随时间增长，有时增速很快。你可以通过限制项目存储的构建历史的个数、确定哪些构建版本将被保存，来控制磁盘的使用情况。

若想进一步了解磁盘空间的使用速度，你可以在系统配置（System Configuration）界面激活插件，以查看每个项目的磁盘占用随时间的变化图（图 13-4）。



346

图13-4　显示一个项目的磁盘使用情况

这将记录和显示项目磁盘占用随时间变化的情况。Disk Usage 插件所显示的磁盘使用随时间变化图参见图 13-5。你可通过它查看磁盘空间是否快速增长或者基本稳定。

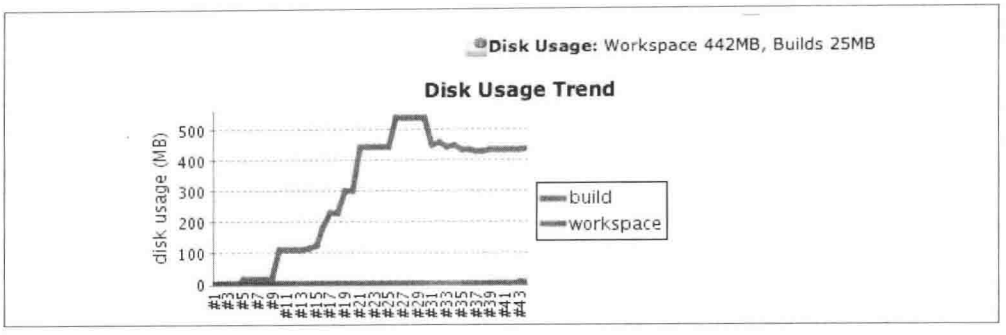


图13-5　该项目的磁盘使用随时间的变化图

磁盘使用及 Jenkins Maven 项目类型

如果你使用 Jenkins Maven 构建作业，则需了解更多的细节信息。在 Jenkins 中，Maven 构建作业会默认自动备份构建产物（build artifact）。这可能并不是你想要的。

问题是这些构建产物快照（SNAPSHOT）会大量占用磁盘空间。对于一个激活的项目，每小时可能都要运行数个构建，永久地保存所有构建的 JAR 文件将非常消耗资源。建立多模块的项目会使得上述问题进一步恶化，因为 Jenkins 将分别备份每个模块的构建产物。

事实上，如果你需要备份 Maven 构建产物快照，最好将它们直接部署到本地的 Maven 仓库管理器。比如可以配置 Nexus Pro 来实现上述目的，并配置 Artifactory 来删除旧的构建产物快照。

347 幸运的是，你也可以直接配置 Jenkins。在构建作业界面的“Build”部分单击 Advanced 按钮。这将显示额外可供设定的字段，如图 13-6 所示。

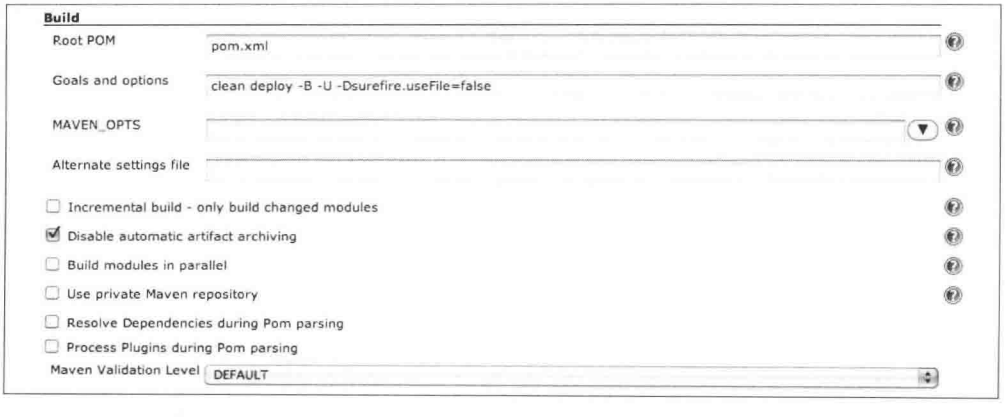


图13-6　Maven构建作业——高级设置

通过勾选 Disable automatic artifact archiving（“禁止自动构建产物备份”）复选框，Jenkins 将不再存储项目构建生成的 JAR 文件。这会让系统管理员感到满意。

但有时你仍需存储 Maven 构建产物，如帮助实现构建管道时（参见第 10 章的“构建管道及优化”一节）。这种情况下你可手动备份需要的构建产物，并设定 Discard old builds（“删除旧的构建产物”）选项来修改备份时长。

监控服务器负载

Jenkins 提供内置的服务器活动监控功能。在 Manage Jenkins 界面，单击 Load Statistics 按钮。你将看到主节点服务器负载随时间的变化图（见图 13-7）。该图将跟踪显示三个度量指标：总执行器数量、繁忙的执行器数量、队列长度。

总执行器数量（蓝线）指主、从节点上所有执行器的数量。当从节点接入或离开集群时，这个值将发生变化。这个指标可以有效地反映从节点的动态配置情况。

繁忙的执行器数量（红线）指被占用执行构建的执行器数量。你需要确保有足够的执行器来应对峰值时刻的构建作业需求。如果所有的执行器都被构建作业永久性地占用了，则需添加更多的执行器或者从节点进入集群。

348

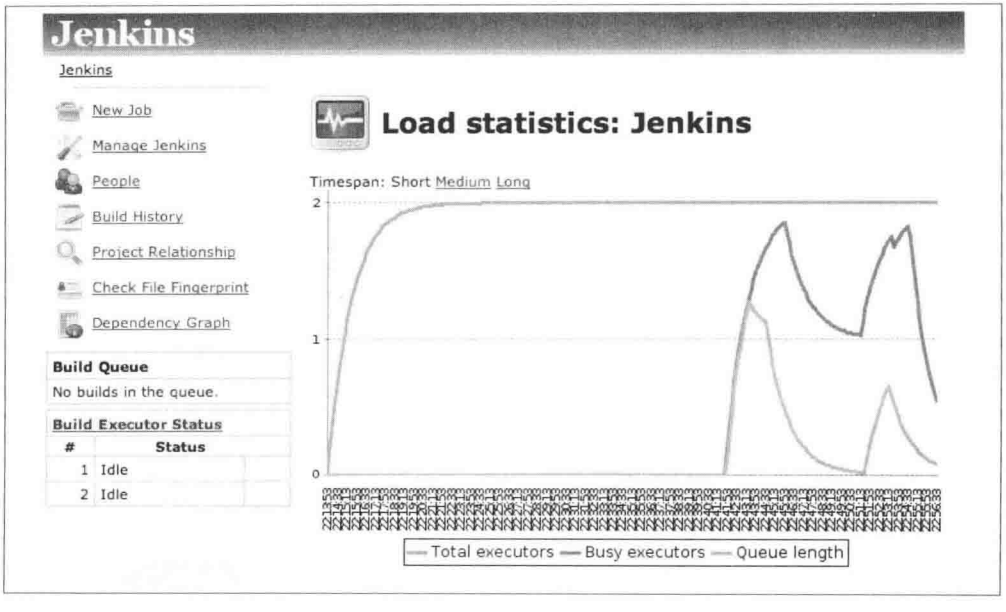


图13-7　Jenkins 加载统计量

队列长度（灰线）指等待执行的构建作业数量。如果所有执行器都被占用了，构建作业

将被加入队列中进行等待。这个指标并未计入等待上游构建执行完成的项目，所以它是衡量何时能从额外容量获益的有效指标。

在从节点界面，使用 Load Statistics 按钮可以得到类似的从节点图表。

另一个选择是安装 Monitoring（监控）插件。该插件使用 JavaMelody 生成构建服务器状态相关的完整 HTML 报告。其中包括 CPU 和系统负载、平均响应时间和内存使用（见图 13-8）。一旦安装了这个插件，你就能在 Manage Jenkins 界面使用“Monitoring of Jenkins/Jenkins master”或“Jenkins/Jenkins nodes”菜单访问 JavaMelody 生成的图表。

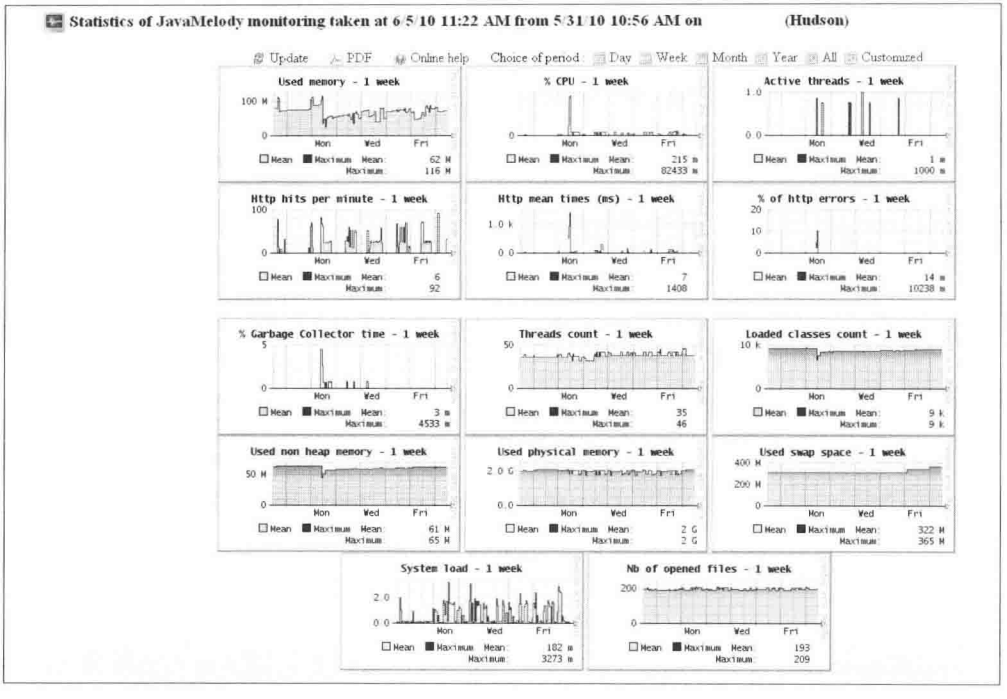


图13-8 Jenkins监控插件

备份配置

无论什么时候，备份数据信息都是重要的，备份 Jenkins 服务器配置信息也不例外。幸运的是，备份 Jenkins 相对简单。在本节中，我们将介绍一些备份的方法。

Jenkins 备份基础

最简单的备份手段是周期性地备份 *JENKINS_HOME* 目录。这个目录包含所有构建项目

相关的配置信息、从节点的配置信息以及构建历史信息。在 Jenkins 运行过程中仍可以进行备份操作——并不需要将服务器关闭。

这个手段的劣势在于 *JENKINS_HOME* 目录包含的数据可能太多（参见第 3 章的“Jenkins 主目录里有什么”一节）。你可以通过不备份以下数据来节省一些资源。这些数据可以由 Jenkins 轻松地在线重建。

$JENKINS_HOME/war。

大量的 WAR 文件。

$JENKINS_HOME/cache

已下载的 Jenkins 工具。

$JENKINS_HOME/tools

提取工具。

你也可以对构建项目数据进行选择性的备份。*$JENKINS_HOME/jobs* 目录包含每一个构建项目的作业配置、构建历史和归档文件。构建作业目录的结构如下（见图 13-9）。

350

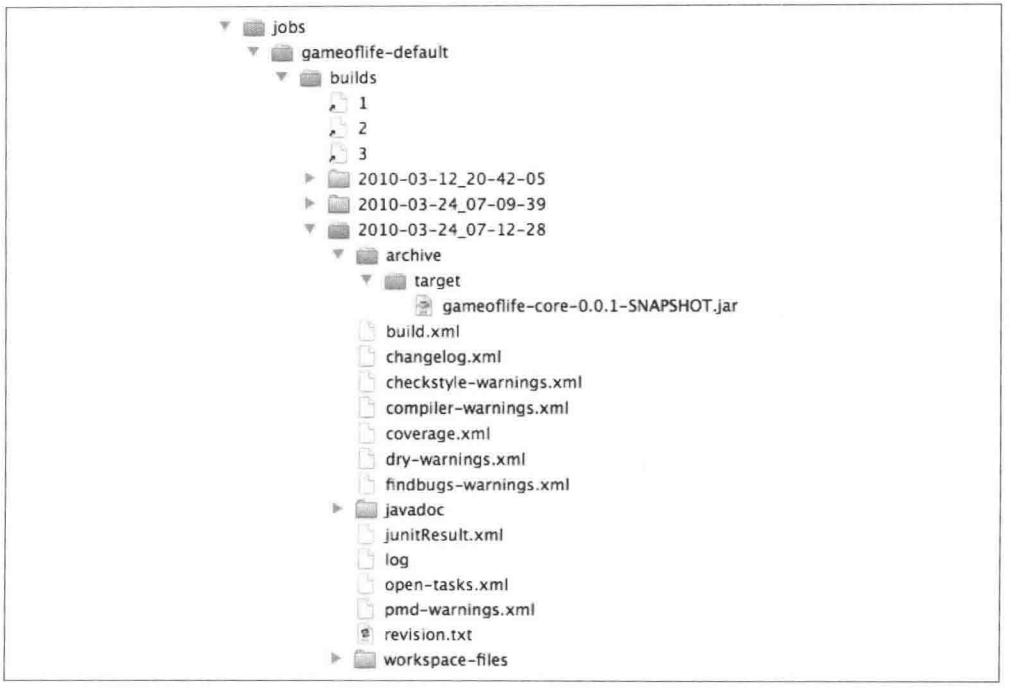


图13-9　所有构建的目录结构

你需理解构建作业目录的组织结构，以便更好地理解如何优化 Jenkins 备份。在 *jobs* 目录下每个构建作业有一个子目录。这个子目录中又包含两个子目录：*builds* 和 *workspace*。你不必备份 *workspace* 目录。在项目恢复过程中，如果 Jenkins 发现 *workspace* 目录不见了，Jenkins 会自动生成该目录。

对于 *builds* 目录，请多留意。该目录包含每个历史构建的构建产物以及生成的构建产物；按照时间戳的不同，它们被存入不同的文件目录。如果你不打算恢复构建历史和过往版本的构建产物，则无须备份这个目录。如果你需要恢复这些历史记录，请继续读下去：在每一个这样的子目录里，你可以找到构建历史信息（以 XML 格式存储，就像 JUnit 的测试结果一样）以及归档的构建产物。Jenkins 使用 XML 和文本文件生成构建作业仪表盘上的图表。如果这些数据对你十分重要，请备份它。*archive* 目录下存储了过往构建版本生成的二进制文件。你可能需要这些二进制文件，但它们很占空间。如果你把它们从备份中去除，将会省下不少空间。

351

测试备份和频繁备份同等重要。使用 Jenkins，这个过程将十分便捷。Jenkins 的主目录整体是非常灵活的，所以所有你需要为测试备份而做的就是，把你的备份提取到一个临时目录，然后启动一个 Jenkins 实例并指向这个目录。例如，假设我们已经把备份提取到一个叫作 */tmp/jenkins-backup* 的临时目录下。为了测试这个备份，首先我们要设置 *JENKINS_HOME* 目录为这个临时目录：

```
$ export JENKINS_HOME=/tmp/jenkins-backup
```

然后在另一不同端口上启动 Jenkins，并测试执行情况：

```
$ java -jar jenkins.war --httpPort=8888
```

现在 Jenkins 将在 8888 端口上启动，你可以检测它是否正常执行。

使用备份插件

上一节介绍的备份过程可以很容易地集成进整个备份流程中。但你可能想尝试更具 Jenkins 特色的备份方式。你可以使用备份插件（见图 13-10），它提供一个简洁的用户界面，供用户备份和存储 Jenkins 配置信息和数据。

该插件允许你配置及运行构建作业配置或构建历史的备份。在起始界面，你可以设定备份内容（见图 13-11）。你可以指定只备份 XML 配置文件，或者同时备份配置文件和构建历史。你也可以选择备份（或不备份）自动生成的 Maven 构建产物（在许多构建流程中，这些构建产物会保存在本地企业仓库管理器中）。你也可以指定备份作业的工作空间（一般情况下，就像我们之前所讨论的那样，你无须备份它）以及生成的所有指纹。

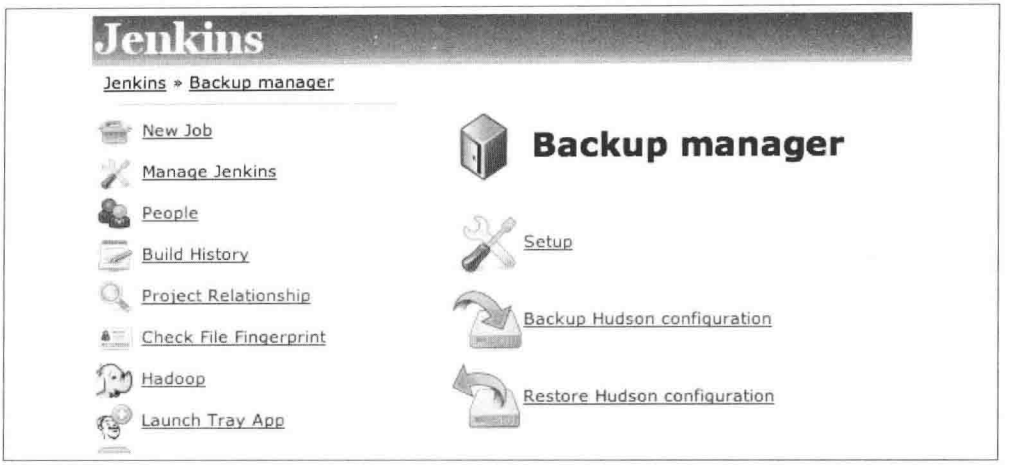


图13-10　Jenkins备份管理器插件



352

图13-11　配置Jenkins备份管理器

通过 Backup Manager 界面（可从 Manage Jenkins 界面进入），你可手动启动备份。备份操作会花费一些时间，期间会关闭 Jenkins（除非在备份配置中停用相关选项）。

生成备份期间，你无法在 Jenkins 里完成该操作，但你可从外部通过访问 URL 启动备份（如果 Jenkins 实例运行在 8080 端口上，可使用 *http://localhost:8080/backup/backup*）。在 UNIX 环境下，也可使用 `wget` 或 `curl` 触发备份。

更为轻量的自动备份

如果你只想备份构建作业的配置信息，则使用 Backup Manager（备份管理器）插件可能有些“杀鸡用牛刀”的味道。另一个选择是使用 Thin Backup（轻量备份）插件，它允许你对配置信息进行全面或增量式的备份。由于插件不会保存构建历史和构建产物，因此备份过程更为快捷，也无须关闭 Jenkins 服务器。

353 和 Backup（备份）插件类似，你可在 Jenkins 系统配置（System Configuration）界面找到该插件的按钮。单击按钮便可开始配置和安排备份，强制进行快速备份或将配置文件恢复到上一个版本。备份配置流程直接而简单（图 13-12），可使用 cron 作业语法安排全面或增量式的备份，并提供备份存储路径。

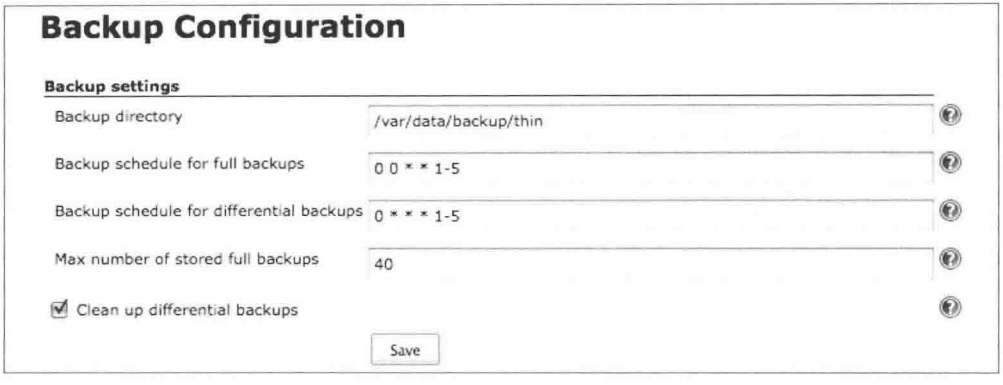


图13-12　配置轻量备份插件

如果想恢复到早先的配置，则可以在恢复页面选择想恢复到的时间（见图 13-13）。一旦配置信息被恢复到较早的版本，就需要重启 Jenkins 以加载新的配置。

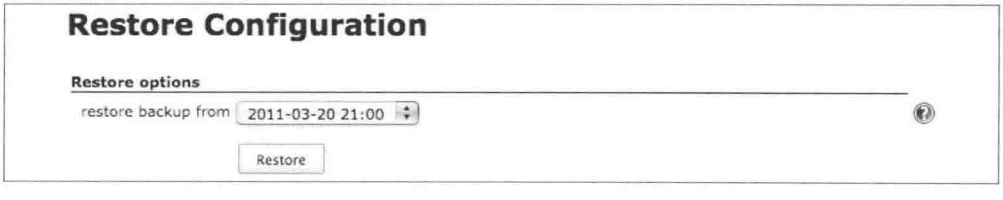


图13-13　恢复到较早的配置版本

构建作业归档

另一种解决硬盘空间问题的方法是删除或归档那些不再使用的项目。在你需要访问归档项目数据或构建产物时可以轻易地解压缩相应归档的项目。归档项目很简单：只需将构
354 建项目目录移出作业目录；当然通常情况下，你可将其压缩成 ZIP 或 tar 文件。

在下面的例子中，我们想归档 *tweeter-default* 项目。首先我们进入 Jenkins 作业目录并创建一个 *tweeter-default* 构建作业目录的 tar（压缩文档）文件：

```
$ cd $JENKINS_HOME/jobs
$ ls
gameoflife-default tweeter-default
$ tar czf tweeter-default.tgz tweeter-default
$ ls
gameoflife-default tweeter-default tweeter-default.tgz
```

因为你想归档的项目没有运行，所以可以安全地删除项目目录并移动归档文档。

```
$ rm -Rf tweeter-default
$ mv tweeter-default.tgz /data/archives/jenkins
```

当你做完这些后，可以简单地在 Manage Jenkins 界面硬盘加载配置（见图 13-14）。归档的项目将从仪表盘消失。

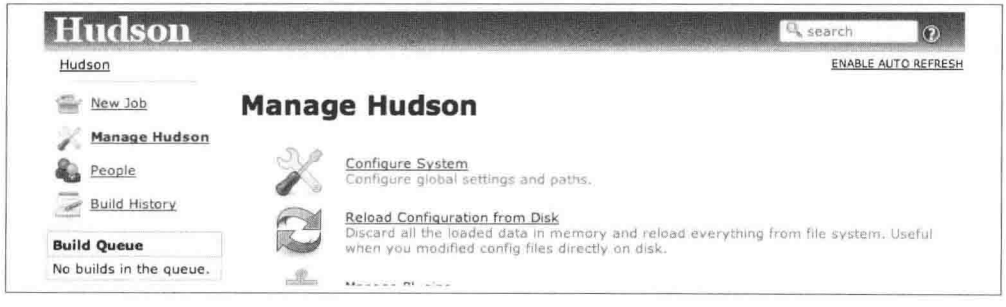


图13-14　从硬盘加载配置

在一个 Windows 机器上，你可以通过创建一个项目目录的 ZIP 文件做同样的事情。

构建迁移

有时当你需要从一个 Jenkins 实例移动或复制 Jenkins 构建作业到另外一个实例时，并不复制整个 Jenkins 配置。例如，你可能迁移构建作业及不同于原来机器的一些系统细节配置文件到 Jenkins 实例的全新分支上。或者你可能恢复归档的旧构建作业。

正如我们所看到的那样，Jenkins 把一个项目所需的所有数据存储在其主目录中 *jobs* 的子目录下。这个子目录很容易识别——它与项目的名字相同。这也是你的项目名称不能包含空格的原因，特别是当 Jenkins 在 UNIX 或 Linux 下运行时——使用更有实际意义的 UNIX 文件名也更利于管理和维护工作。 355

你可以通过在 Jenkins 实例间复制或移动构建作业目录来拷贝构建作业。项目作业目录是自给自足的——它包含全项目的配置信息和所有构建历史。将构建作业目录直接复制到运行中的 Jenkins 实例也是安全的。如要删除原有服务器上的构建作业目录，需先关闭服务器上的 Jenkins 服务。你甚至无须重启新的 Jenkins 实例就能够查看导入的结果——可在 Manage Jenkins 界面单击 Reload Configuration From Disk（重新载入磁盘中的配置）。这将载入新的作业并把它们显示在 Jenkins 仪表盘上。

还有几个陷阱请你留意。如果你想把作业迁移到一个全新的 Jenkins 配置，请记得安装或迁移原有服务器的插件。在 *plugins* 目录下，你可以简单地复制所有文件，并复制到新实例对应的目录下。

当然，可能正是因为原有服务器插件配置比较混乱，你需要将构建作业迁移到新的实例。一些插件也会常常出现问题；你可能想迁移到更为干净的安装环境下，并使用经审核过的更为知名或者好用的插件。此时，你可能需要在导入时修改一些项目配置。

原因是显而易见的。当你使用一个项目中的插件时，项目的 *config.xml* 文件中与插件相关的字段信息会被修改。如果出于某种原因，你需要将项目迁移到一个未经插件安装的 Jenkins 上，Jenkins 将无法识别这部分项目配置信息。如果两台实例上的插件版本相差太多或者插件使用的数据格式发生改变，类似的问题有时也会出现。

如果你在迁移 Jenkins 项目中使用了不同的配置信息，需花时间留意系统日志文件。无效的插件配置往往会抛出警告或异常信息。即使不是致命的，这些错误信息也经常意味着插件无法如期待般运行，或根本无法运行。

Jenkins 提供一些有用的特性，以帮助你迁移项目配置。如果 Jenkins 找到了过期或者无效的数据，它会将这些信息报告给你。在 Manage Jenkins 界面，你会看到如图 13-15 中的信息。

此时，你可以选择保持配置信息不变（例如当你回滚到 Jenkins 实例上早期的版本时）或者让 Jenkins 忽略无法读取的字段信息。如果你选择忽略，Jenkins 会弹出包含错误细节的新界面；如果需要，它也能帮你整理项目配置文件（见图 13-16）。

356

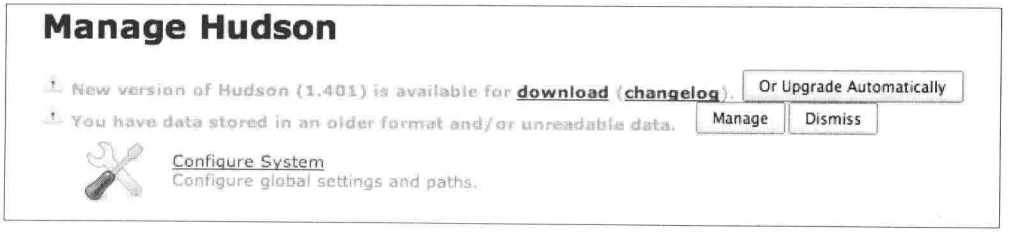
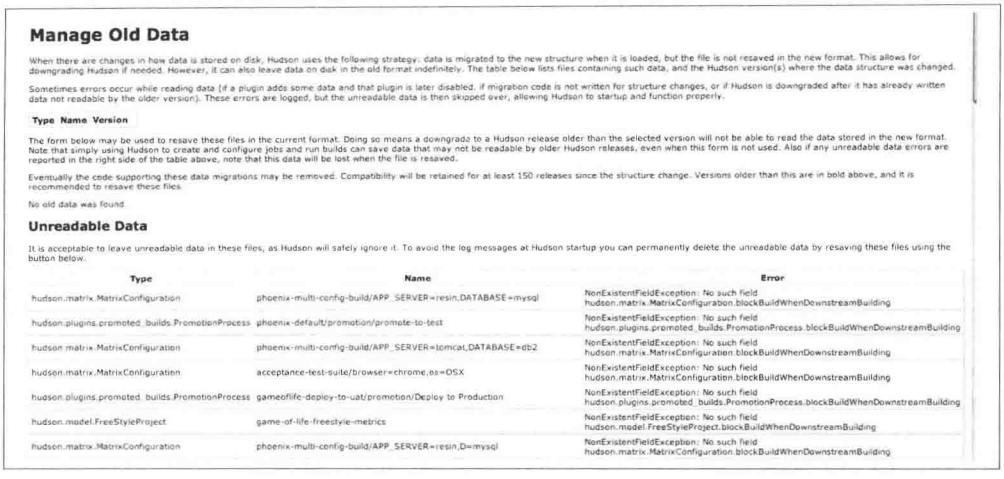


图13-15　如果你的数据和目前版本不兼容，Jenkins会通知你



Manage Old Data

When there are changes in how data is stored on disk, Hudson uses the following strategy: data is migrated to the new structure when it is loaded, but the file is not resaved in the new format. This allows for downgrading Hudson if needed. However, it can also leave data on disk in the old format indefinitely. The table below lists files containing such data, and the Hudson version(s) where the data structure was changed.

Sometimes errors occur while reading data (if a plugin adds some data and that plugin is later disabled, if migration code is not written for structure changes, or if Hudson is downgraded after it has already written data not readable by the older version). These errors are logged, but the unreadable data is then skipped over, allowing Hudson to startup and function properly.

Type	Name	Version

The form below may be used to resave these files in the current format. Doing so means a downgrade to a Hudson release older than the selected version will not be able to read the data stored in the new format. Note that simply using Hudson to create and configure jobs and run builds can save data that may not be readable by older Hudson releases, even when this form is not used. Also if any unreadable data errors are reported in the right side of the table above, note that this data will be lost when the file is resaved.

Eventually the code supporting these data migrations may be removed. Compatibility will be retained for at least 150 releases since the structure change. Versions older than this are in bold above, and it is recommended to resave these files.

No old data was found.

Unreadable Data

It is acceptable to leave unreadable data in these files, as Hudson will safely ignore it. To avoid the log messages at Hudson startup you can permanently delete the unreadable data by resaving these files using the button below.

Type	Name	Error
hudson.matrix.MatrixConfiguration	phoenix-multi-config-build/APP_SERVER=resin,DATABASE=mysql	NonExistentFieldException: No such field hudson.matrix.MatrixConfiguration.blockBuildWhenDownstreamBuilding
hudson.plugins.promoted_builds.PromotionProcess	phoenix-default/promotion/promote-to-test	NonExistentFieldException: No such field hudson.plugins.promoted_builds.PromotionProcess.blockBuildWhenDownstreamBuilding
hudson.matrix.MatrixConfiguration	phoenix-multi-config-build/APP_SERVER=tomcat,DATABASE=db2	NonExistentFieldException: No such field hudson.matrix.MatrixConfiguration.blockBuildWhenDownstreamBuilding
hudson.matrix.MatrixConfiguration	acceptance-test-suite/browser=chrome,os=OSX	NonExistentFieldException: No such field hudson.matrix.MatrixConfiguration.blockBuildWhenDownstreamBuilding
hudson.plugins.promoted_builds.PromotionProcess	gameoflife-deploy-to-uat/promotion/Deploy to Production	NonExistentFieldException: No such field hudson.plugins.promoted_builds.PromotionProcess.blockBuildWhenDownstreamBuilding
hudson.model.FreeStyleProject	game-of-life-freestyle-metrics	NonExistentFieldException: No such field hudson.model.FreeStyleProject.blockBuildWhenDownstreamBuilding
hudson.matrix.MatrixConfiguration	phoenix-multi-config-build/APP_SERVER=resin,D=mysql	NonExistentFieldException: No such field hudson.matrix.MatrixConfiguration.blockBuildWhenDownstreamBuilding

图13-16　管理过期的构建作业数据

该屏幕提供给你含有可疑数据项目的详情，以及确切的错误信息。这给了你几个选择。如果你确定自己不再需要原来创建数据的插件，通过单击 Discard Unreadable Data 按钮可以安全地删除多余的字段。或者，你还可以决定，该字段属于一个有用的插件，还没有被安装在新的 Jenkins 实例上。在这种情况下，安装插件，一切都应该很好。最后，你可以选择离开冗余数据，关注错误信息，直到某天你确定你不需要将作业迁移回旧的服务器。

然而，Jenkins 并不总是检测所有的错误或不一致性——当你迁移构建作业时，它还会检测系统日志。例如，下面这个真实的例子，Jenkins 日志显示迁移过程中发生的事情：

```
Mar 16, 2010 2:05:06 PM hudson.util.CopyOnWriteList$ConverterImpl unmarshal
WARNING: Failed to resolve class
com.thoughtworks.xstream.mapper.CannotResolveClassException: hudson.plugins.ciga
me.GamePublisher : hudson.plugins.cigame.GamePublisher
        at com.thoughtworks.xstream.mapper.DefaultMapper.realClass(DefaultMapper
.java:68)
        at com.thoughtworks.xstream.mapper.MapperWrapper.realClass(MapperWrapper
.java:38)
        at com.thoughtworks.xstream.mapper.DynamicProxyMapper.realClass(DynamicP
roxyMapper.java:71)
        at com.thoughtworks.xstream.mapper.MapperWrapper.realClass(MapperWrapper
.java:38)
```

357

该错误基本上告诉我们，不能找到一个叫作 `hudson.plugins.cigame.GamePublisher` 的类。事实上，目标安装在忽略 CI Game 插件。在这种情况下（有时

会发生），在 Manage Jenkins 页面没有出现警告信息，所以 Jenkins 无法正确地配置文件本身。

在这种情况下，最简单的解决方案是在目标服务器上安装 CI Game 插件。但是如果我们不想安装这个插件呢？我们可以忽略配置文件本身，但这可能掩盖了更重要的错误，现在整理它们会更好。

在这种情况下，我们需要手工检查和更新项目配置文件。在 UNIX 环境，我用 `grep` 来找到所有包含“cigame”的配置文件：

```
$ cd $JENKINS_HOME/jobs
$ grep cigame */config.xml
project-a/config.xml: <hudson.plugins.cigame.GamePublisher/>
project-b/config.xml: <hudson.plugins.cigame.GamePublisher/>
project-c/config.xml: <hudson.plugins.cigame.GamePublisher/>
```

在这些 *config.xml* 文件里，我发现在 `<publishers>` 第一部分引用 CI Game 插件（报告插件通常在这里配置）：

```
<maven2-moduleset>
  ...
  <publishers>
    <hudson.plugins.cigame.GamePublisher/>
    <hudson.plugins.claim.ClaimPublisher/>
  </publishers>
  ...
</maven2-moduleset>
```

要解决这个问题，我所要做的是删除违规行：

```
<maven2-moduleset>
  ...
  <publishers>
    <hudson.plugins.claim.ClaimPublisher/>
  </publishers>
  ...
</maven2-moduleset>
```

358 插件配置数据的确切位置根据插件的不同而不同，但是一般 *config.xml* 文件可读性较强，手工更新它们并不难。

因此，在 Jenkins 实例之间迁移构建作业并不难——你只需要知道典型案例的一些技巧，如果你知道在哪里能看到 Jenkins 提供的很好的工具，过程会更顺畅。

小结

在本章中，我们看了一些注意事项，你应该知道了如果你的工作是维护 Jenkins 服务器，包括如何监控磁盘和服务器使用情况，如何备份构建作业和 Jenkins 的配置文件，如何安全迁移构建作业，以及如何升级构建数据。

自动化你的单元和集成测试

用 Maven 实现你的测试自动化

Maven 是 Java 世界流行的开源构建工具，它使用声明式依赖、标准目录和构建生命周期、配置规则的做法来鼓励干净、可维护性、高级别的构建脚本。测试自动化在 Maven 得到大力支持。Maven 项目使用标准的目录结构：它会在一个名为（默认）*src/test/java* 的目录下自动寻找单元测试。不用其他的配置：只是把依赖关系添加到你的测试正在使用的测试框架，Maven 会自动在此目录结构中寻找和执行 JUnit、TestNG 甚至是 Plain Old Java Objects（POJO）测试。

在 Maven 中，你通过调用测试生命周期阶段运行单元测试，如下所示：

```
$ mvn test
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building Tweeter domain model
[INFO] task-segment: [test]
[INFO] ------------------------------------------------------------------------
...
-------------------------------------------------------
T E S T S
-------------------------------------------------------
Running com.wakaleo.training.tweeter.domain.TagTest
Tests run: 13, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.093 sec
Running com.wakaleo.training.tweeter.domain.TweeterTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.021 sec
Running com.wakaleo.training.tweeter.domain.TweeterUserTest
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.055 sec
Running com.wakaleo.training.tweeter.domain.TweetFeeRangeTest
```

```
Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.051 sec
Running com.wakaleo.training.tweeter.domain.HamcrestTest
Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.023 sec

Results :

Tests run: 38, Failures: 0, Errors: 0, Skipped: 0
```


除了执行测试，如果任何测试构建失败，Maven 还会在 *target/surefire-reports*（默认情况下）目录以 XML 和文本格式生成一套测试报告。对我们的 CI 目的而言，我们对 XML 文件感兴趣，因为 Jenkins 能够理解和分析 CI 报告的这些文件：

```
$ ls target/surefire-reports/*.xml
target/surefire-reports/TEST-com.wakaleo.training.tweeter.domain.HamcrestTest.xml
target/surefire-reports/TEST-com.wakaleo.training.tweeter.domain.TagTest.xml
target/surefire-reports/TEST-com.wakaleo.training.tweeter.domain.TweetFeeRangeTest.xm
target/surefire-reports/TEST-com.wakaleo.training.tweeter.domain.TweeterTest.xml
target/surefire-reports/TEST-com.wakaleo.training.tweeter.domain.TweeterUserTest.xml
```

Maven 定义了两个不同的测试阶段：单元测试和集成测试。单元测试应该是快速和轻量的，在尽可能短的时间内提供大量的测试反馈。集成测试是较慢的而且更麻烦，并且常常需要构建和部署应用程序到服务器上（即使是嵌入式的）来进行更完整的测试。这两种类型的测试都是重要的；对于一个精心设计的持续集成环境，能够区分它们是重要的。构建应该确保最初运行所有的单元测试——如果单元测试失败，应该很快通知开发人员。只有所有的单元测试通过，才值得进行速度更慢、更重量级的集成测试。

在 Maven 中，集成测试在集成测试生命周期阶段进行，你可以通过运行 `mvn integration-test` 或者（更简单）`mvn verify` 来调用。在这个阶段，在嵌入式 Jetty Web 服务器配置 Maven 启动你的 Web 应用程序是很容易的，或者打包并部署你的应用程序到测试服务器上。然后可以针对应用程序执行集成测试。然而最棘手的事情是告诉 Maven 如何区分单元测试和集成测试，这样只有在应用程序的运行版本存在时才执行。

有几种方法可以做到这一点，但在本书写作之际没有跨所有 Maven 项目使用的官方标准方法。一个简单的策略是使用命名规范：所有集成测试可以“Integration Test”结尾，或者被放在特定的包里面。下面的类就使用了这样的规范：

```
public class AccountIntegrationTest {

  @Test
  public void cashWithdrawalShouldDeductSumFromBalance() throws Exception {
    Account account = new Account();
    account.makeDeposit(100);
```

```
    account.makeCashWithdraw(60);
    assertThat(account.getBalance(), is(40));
  }
}
```

361

在 Maven 中，测试是通过 *maven-surefire-plugin* 插件配置的。为了确保 Maven 只在集成测试阶段运行这些测试，你可以配置以下插件，如下所示：

```
<project>
  ...
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <configuration>
          <skip>true</skip>❶
        </configuration>
        <executions>
          <execution>❷
            <id>unit-tests</id>
            <phase>test</phase>
            <goals>
              <goal>test</goal>
            </goals>
            <configuration>
              <skip>false</skip>
              <excludes>
                <exclude>**/*IntegrationTest.java</exclude>
              </excludes>
            </configuration>
          </execution>
          <execution> ❸
            <id>integration-tests</id>
            <phase>integration-test</phase>
            <goals>
              <goal>test</goal>
            </goals>
            <configuration>
              <skip>false</skip>
              <includes>
                <include>**/*IntegrationTest.java</include>
              </includes>
            </configuration>
          </execution>
        </executions>
      </plugin>
```

...

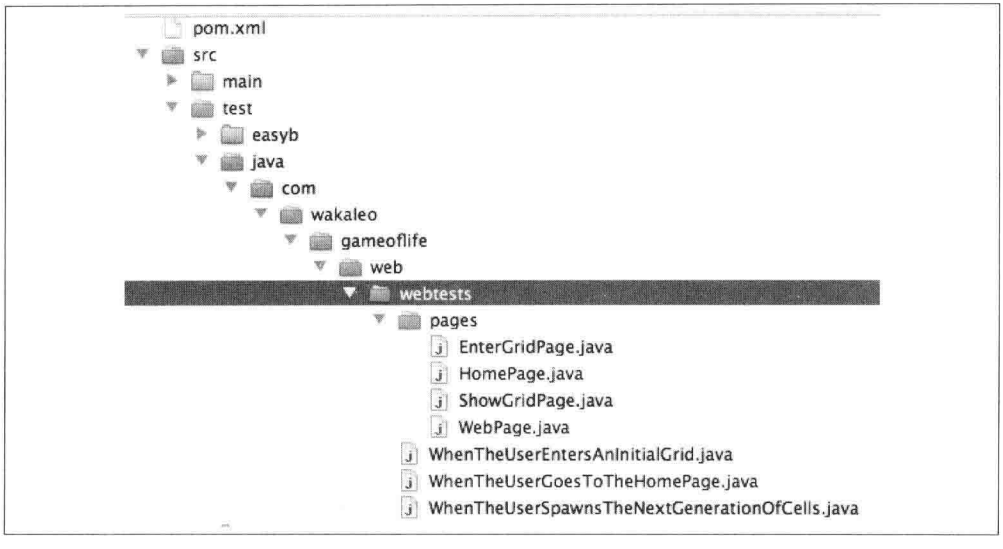
❶ 默认情况下跳过所有测试——这将停用默认 Maven 测试配置。

❷ 在单元测试阶段，运行测试但是排除集成测试。

❸ 在集成测试阶段，运行测试，但是只包括集成测试。

这将确保集成测试在单元测试阶段被跳过，只在集成测试阶段执行。

362 如果你不想对自己的测试类的名称进行不需要的限制，则可以使用软件包的名称。在图 A-1 所示的项目中，所有的功能测试都放在一个叫作 *webtests* 的包。测试的名称没有任何限制，但是我们正在使用页面对象（Page Objects）模型化应用程序的用户界面，所以我们也要确保在 *pages* 包（在 *webtests* 包下面）中的类不被测试。



图A-1 包含自由命名测试类的项目

在 Maven 中，我们可以使用下面的配置：

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <configuration>
    <skip>true</skip>
  </configuration>
  <executions>
    <execution>
      <id>unit-tests</id>
      <phase>test</phase>
```

```
      <goals>
        <goal>test</goal>
      </goals>
      <configuration>
        <skip>false</skip>
        <excludes>
          <exclude>**/webtests/*.java</exclude>
        </excludes>
      </configuration>
    </execution>
    <execution>
      <id>integration-tests</id>
      <phase>integration-test</phase>
      <goals>
        <goal>test</goal>
      </goals>
      <configuration>
       <skip>false</skip>
       <includes>
        <include>**/webtests/*.java</include>
       </includes>
       <excludes>
        <exclude>**/pages/*.java</exclude>
       </excludes>
      </configuration>
     </execution>
   </executions>
  </plugin>
```


TestNG 目前比 JUnit 对测试组的支持更加灵活。如果你在使用 TestNG，则可以用 TestNG 组标识集成测试。在 TestNG 中，测试类或者测试方法可以使用 `@Test` 注释的 `groups` 属性来标记，如下所示：

```
@Test(groups = { "integration-test" })
public void cashWithdrawalShouldDeductSumFromBalance() throws Exception {
   Account account = new Account();
   account.makeDeposit(100);
   account.makeCashWithdraw(60);
   assertThat(account.getBalance(), is(40));
}
```

在 Maven 中，使用如下的配置能确保这些测试仅在集成测试阶段运行：

```
<project>
  ...
  <build>
```

```
<plugins>
  <plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
      <skip>true</skip>
    </configuration>
    <executions>
      <execution>
        <id>unit-tests</id>
        <phase>test</phase>
        <goals>
          <goal>test</goal>
        </goals>
        <configuration>
          <skip>false</skip>
          <excludedGroups>integration-tests</excludedGroups>❶
        </configuration>
      </execution>
      <execution>
        <id>integration-tests</id>
        <phase>integration-test</phase>
        <goals>
          <goal>test</goal>
        </goals>
        <configuration>
          <skip>false</skip>
          <groups>integration-tests</groups>❷
        </configuration>
      </execution>
    </executions>
  </plugin>
  ...
```

364

❶ 不要在测试阶段运行集成测试组。

❷ 在集成测试阶段仅运行集成测试组里面的测试。

尽可能并发运行你的测试是有意义的，因为它可以显著地加快测试速度（请参见第 6 章的“请求帮助！我的测试运行得太慢了”一节）。并行测试尤其对于那些使用大量 I/O、磁盘和网络访问（如 Web 测试）而运行缓慢的测试是有用而且简便的，因为这些就是我们通常想要加快速度的一类测试。

TestNG 为并行测试提供了很好的支持。例如，使用 TestNG，可以配置测试方法在 10 个并发线程上并行运行：

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.5</version>
  <configuration>
    <parallel>methods</parallel>
    <threadCount>10</threadCount>
  </configuration>
</plugin>
```

至于 JUnit 4.7，你也可以使用类似的配置并行运行 JUnit 测试。事实上，4.7 及其以上的版本也支持如上所示的配置。

你也要设置 `<parallel>` 配置项为类（`classes`）而不是方法（`methods`），这会尝试并行运行测试类，而不是每个方法。这可能过快或者过慢（取决于你的测试类的数量），但是记住一些测试不并发可能更加安全。

产品经常变化，所以你应该尝试多个版本以得到最好的结果。

用 Ant 实现测试自动化

365

在 Ant 中设置自动测试也是相对容易的（虽然它比 Maven 需要更多配置）。尤其是，JUnit 库并没有 Ant 包，也没有对 Ant 任务提供开箱即用的支持，所以你必须亲自将它们安装在某个地方。最轻便的方法是使用如 Ivy 这样的依赖管理（Dependency Management）工具，或者将对应的 JAR 文件放在项目架构的目录里面。

若要在 Ant 中运行测试，要调用 `<junit>` 任务。典型的 Jenkins 友好的配置如本例所示：

```
<property name="build.dir" value="target" />
<property name="java.classes" value="${build.dir}/classes" />
<property name="test.classes" value="${build.dir}/test-classes" />
<property name="test.reports" value="${build.dir}/test-reports" />
<property name="lib" value="${build.dir}/lib" />

<path id="test.classpath">❶
   <pathelement location="${basedir}/tools/junit/*.jar" />
   <pathelement location="${java.classes}" />
   <pathelement location="${lib}" />
</path>

<target name="test" depends="test-compile">
   <junit haltonfailure="no" failureproperty="failed">❷
    <classpath> ❸
```

```
      <path refid="test.classpath" />
      <pathelement location="${test.classes}" />
    </classpath>
    <formatter type="xml" /> ❹
    <batchtest fork="yes" forkmode="perBatch"❺ todir="${test.reports}">
      <fileset dir="${test.src}">❻
        <include name="**/*Test*.java" />
      </fileset>
    </batchtest>
  </junit>
  <fail message="TEST FAILURE" if="failed" />❼
</target>
```

❶ 我们需要设置类路径中包含 *junit* 和 *junit-ant* 的 JAR 文件，以及应用程序类和应用需要编译和运行的任何其他依赖项。

❷ 测试本身是在这里运行的。`haltonfailure` 选项用于标识构建立即失败（如果任何测试失败）。在持续集成环境中，这不是我们想要的，因为我们需要得到后续测试的结果。所以我们将此值设为否，并使用 `failureproperty` 选项，一旦所有的测试已经完成就强制构建失败。

❸ 类路径中需要包含 JUnit 库、应用程序类和它们的依赖项以及你的编译测试类。

366

❹ JUnit Ant 任务可以生成文本和 XML 报告，但是对于 Jenkins 而言，我们只需要 XML。

❺ `fork` 选项让你在单独的 JVM 中运行测试。这通常是一个好主意，因为它可以避免与 Ant 库本身类加载问题相关的冲突。然而，JUnit Ant 任务的默认行为是为每个测试创建新的 JVM，这大大降低了测试速度。`perBatch` 选项要好一点，因为它为一批测试创建新的 JVM。

❻ 你可以在文件集元素中定义你想要运行的测试。这提供了极大的灵活性，并使它容易定义不同测试子集（集成、Web 等）的其他目标。

❼ 测试已经完成后，如果有任何测试失败，则强制标记构建为失败状态。

如果你喜欢 TestNG，Ant 在这里也得到了很好的支持。使用前面示例中的 TestNG，可以做下面的事情：

```
<property name="build.dir" value="target" />
<property name="java.classes" value="${build.dir}/classes" />
<property name="test.classes" value="${build.dir}/test-classes" />
<property name="test.reports" value="${build.dir}/test-reports" />
```

```
<property name="lib" value="${build.dir}/lib" />

<path id="test.classpath">
  <pathelement location="${java.classes}" />
  <pathelement location="${lib}" />
</path>

<taskdef resource="testngtasks" classpath="lib/testng.jar"/>

<target name="test" depends="test-compile">
  <testng classpathref="test.classpath"
        outputDir="${testng.report.dir}"
        haltonfailure="no"
        failureproperty="failed">
    <classfileset dir="${test.classes}">
     <include name="**/*Test*.class" />
    </classfileset>
  </testng>
  <fail message="TEST FAILURE" if="failed" />
</target>
```

TestNG 是一个非常灵活的测试库，TestNG 的任务有更多的选择。例如，仅运行我们前面看到的“集成测试”（integration-test）组定义的部分测试，我们可以这样做：

```
<target name="integration-test" depends="test-compile">
 <testng classpathref="test.classpath"
        groups="integration-test"
        outputDir="${testng.report.dir}"
        haltonfailure="no"
        failureproperty="failed">
     <classfileset dir="${test.classes}">
       <include name="**/*Test*.class" />
     </classfileset>
   </testng>
   <fail message="TEST FAILURE" if="failed" />
 </target>
```

367

或使用四个并发线程并行运行测试，可以这样做：

```
<target name="integration-test" depends="test-compile">
  <testng classpathref="test.classpath"
         parallel="true"
         threadCount=4
         outputDir="${testng.report.dir}"
         haltonfailure="no"
         failureproperty="failed">
```

```
    <classfileset dir="${test.classes}">
      <include name="**/*Test*.class" />
    </classfileset>
  </testng>
  <fail message="TEST FAILURE" if="failed" />
</target>
```

索引†

A

验收测试，自动化的，7，136，155-158
验收测试驱动开发，7
主动（推送）通知，195
活动目录，微软，作为安全领域，176
管理员
  用于 Jenkins 内部用户数据库，171
  用于基于矩阵的安全策略，182
聚合测试结果，295-296
Amazon EC2 云计算服务，315-319
Amazon EC2 插件，317
Amazon Machine Images（AMI），316
Amazon Web Services（AWS），316
AMI（Amazon Machine Images），316
Ant 分析（见代码覆盖率度量；
        代码质量度量；测试）74-75
  自动化测试，365-267
  使用 Cobertura 的代码覆盖率度量，
      148-150
  代码质量度量
    使用 Checkstyle，228
    使用 CodeNarc，236
    使用 FindBugs，235
    使用 PMD 和 CPD，231
  配置，74-75
  环境变量，访问，107
  自由风格下的构建步骤，104
  安装，75
ANT_OPTS 环境变量，54
应用程序服务器
  自动化部署，328-341
    Java 应用程序，329-338
    基于脚本的应用程序，339-341
  部署 Jenkins，16，53-54
  升级 Jenkins，63
二进制构建产物的归档，26
  部署到企业级仓库管理器，119-123
  禁用，118
  在自由风格的构建作业中，111-114
归档构建作业，353-354
Artifactory
  企业级仓库管理器，121，123
  Jenkins 支持，5
Artifactory 插件，275
构建产物（见二进制构建产物）
Atlassian Crowd，作为安全领域，178
Audit Trail 插件，190-191
审计用户操作，189-193
验证，169
  （见安全）
  基于矩阵的安全，181-185
  无限制的，170-171
  基于项目的安全，185-187
  基于角色的安全，188-189
自动化部署，323-328
  到应用程序服务器，328-341

†：索引所列页码为本书英文版页码，请参照用“⇨”表示的原书页码。——译者注

数据库更新，325-327
部署脚本，324
回滚更改，328
冒烟测试，327
自动化夜间构建，6
自动化测试（见测试）
AWS（Amazon Web Services），316

B

Backup 插件，351
备份，62，348-353
批量脚本，75，105-106
BDD（行为驱动开发），135
BDD（行为驱动开发），156
二进制构建产物
归档，26
部署到企业级仓库管理器，119-123
禁用，118
在自由风格下的构建作业中，111-114
在构建管道中重用，285-288
Boolean 参数，257
构建代理
为多种 JDK 版本配置，71
监控，314
构建历史
在构建目录中，60-61
有关细节，30-32
磁盘使用率，343-347
保留的构建数量，81
参数化，261
权限，184
结果摘要，26，30
构建作业中的，79
（也见于自由风格构建作业；Maven 构建作业）
归档，353-354
二进制构建产物（见二进制构建产物）
代码覆盖率度量（见代码覆盖率度量）
代码质量度量（见代码质量度量）
复制，80
创建，21-26，79-80
延迟开始，69
之间的依赖，278
分布式横跨于构建服务器，301-302
关联从节点和作业，312-314
基于云的构建区域，314-319
创建从节点，302
主 / 从架构，301-311
监控从节点，314
启动从节点，303-311
外部，监控，80
失败
声明，200
相关细节，139-141
示例，28-32
指示，27，30
通知，195，198
全局属性，69-70
历史（见构建历史）
Javadoc 的生成，33-34
加入，278-279
锁定资源，280
迁移，354-358
多种配置，263-269
组合过滤器，268
矩阵配置，267
创建，264-265
自定义轴，267
JDK 轴，266
运行，267-269
从节点轴，265-266
命名，22
参数化，253-261
用于构建脚本，254-256
创建，254
历史，261
针对 Git 标签运行，260
针对 Subversion 标签运行，259
远程启动，260-261
参数类型，254，257-258
结果报告（见报告）
运行数量，作为参数，258
并行运行，277-280
调度（见构建触发）
源代码位置，22
运行时状态，26
添加的步骤，24-26，33，36
成功的指示器，27
测试（见测试）
人工触发，26，27，102
类型，22，79

不稳定构建，139
  条件，113，152，240，244
  指示器，37
  通知，115，195，198
  触发另一个构建，98，115
Build Pipeline 插件，297
构建管道，281-299
  汇总测试结果，295-296
  部署管道，296-299
  Maven 版本号，282-284
  进阶，281，288-295
  重用构建产物，285-288
Build Promotion 插件，329
构建分发器，202-203
构建脚本（见脚本）
构建服务器，6
  安装 Jenkins，44-45
  内存需求，44
  监控负载，347-348
  多构建运行（见分布式构建）
  处理器需求，44
  升级，167
  虚拟机，45，167
构建工具，配置，72-75
构建触发器
  配置，23-24
  用于自由风格的构建作业，97-102
  人工，26，27，102
  参数化，262-263
  查询 SCM 用于版本控制变化，99
  定期，98-99
  从版本控制系统远程触发，100-102
  当另一个构建完成的时候，98
构建目录，59-61
BUILD_ID 环境变量，106
BUILD_NUMBER 环境变量，106
BUILD_TAG 环境变量，106
BUILD_URL 环境变量，107

C

CAS（中心验证服务），178
Checkstyle，227-230，242
Checkstyle 插件，244
CI（持续集成），1-3，6-8
声明失败构建，200
用于构建的云计算，167，314-319
CloudBees（赞助商），xxiii
Clover，153-155
Clover 插件，154
Cobertura，34-39，145-153
  Ant，148-150
  在构建作业中配置，150-153
  Maven，146-148
  报告，153
Cobertura 插件，150
代码复杂度，245-247
代码覆盖率度量，7，144-155
  Clover，153-155
  Covertura，34-39，145-153
  软件，145
代码示例，使用，xxiv
代码质量度量，7，225-227
  在构建作业中，226
  Checkstyle，227-230，242
  代码复杂度，245-247
  CodeNarc，236-237
  CPD，230-234
  FindBugs，234-236，242
  IDE，226
  未完成任务，247
  插件，226
  PMD，230-234，242
  软件，226，227
  Sonar，226，248-251
  Violations 插件，237-242
代码审查，225
CodeNarc，236-237
提交消息，排除触发构建作业，86
config.xml 文件，59
配置，17-21，65-68
  Ant，74-75
  构建工作，72-75
  配置系统界面，65，68-69
  CVS，76
  邮件服务器，76-77
  Git，20-21
  全局属性，69-70
  JDK，19，70-72
  加载统计界面，67
  管理节点界面，66

管理插件界面，66
Maven，18-19，73-74
通知，20
为关机界面做准备，67
代理，77-78
构建开始前的安静时期，69
从磁盘界面重加载配置，66
脚本终端，67
Subversion，76
系统信息界面，67
系统日志界面，67
在主页面的系统消息，69
版本控制系统，75-76
配置系统界面，65，68-69
本书的联系信息，xxv
持续交付，2
持续部署，2，7，323-328
到应用服务器，328-341
数据库更新，325-327
部署脚本，324
回滚变更，328
冒烟测试，327
持续集成（见 CI）
本书的贡献人，xxi
字体规范，xx
Copy Artifact 插件，285，329，332
Coverage Complexity Scatter Plot 插件，246
CPD，230-234
CppUnit，136
构建服务器需求的 CPU，44
定时任务（见外部作业）
Crowd，Atlassian，作为安全领域，178
CVS
配置，76
延迟构建作业，69，82
Jenkins 支持，20
查询，100
CVS_BRANCH 环境变量，107

D

数据库
回滚变更，328
使用自动化部署更新，325-327
用户数据库，170，171-174
Dependency Graph View 插件，278
Deploy 插件，328，329-331，332
Deploy Websphere 插件，328，329
部署（见自动化部署，持续部署）
部署管道，296-299
部署脚本，324
桌面通知，212-215
磁盘空间
构建目录，61
监控，343-347
Disk Usage 插件，345-346
分布式构建，45，301，-311
基于云的构建区域，314-319
主 / 从架构，301-311
从节点
结合构建作业，312-314
创建，302
安装为 Windows 服务，310
监控，314
作为远程服务启动，311
无主节点模式启动，311
使用 SSH 启动，303-306
使用 Java Web Start 启动，307-309
DocLinks 插件，158
文档（见 Javadoc）

E

Eclipse
Checkstyle 里的代码质量度量，228
PDM 里的代码质量度量，230
桌面通知，212
Eclipse 插件，212
邮件通知，20，173，195-199
（也见于通知）
邮件服务器，配置，76-77
Email-ext 插件，197-199
企业级仓库管理器，119-123
环境变量，106
（也见于特定环境变量）
构建参数，254
在构建步骤中使用，106-108
Eucalyptus 云，315
EXECUTOR_NUMBER 环境变量，106
Extended Read Permission 插件，187
外部作业，监控，80

F

File 参数，258
FindBugs，234-236，242
FindBugs 插件，244
指纹，292，296
指纹目录，58
自由风格构建作业，22，79，80-84
  归档二进制构建产物，111-114
  阻塞上游项目，82
  构建历史，要保留的构建号，81
  构建步骤，102-110
    Ant 构建脚本，104
    批量脚本，105-106
    环境变量，106-108
    Groovy 脚本，108-110
    Maven 构建步骤，25，103-104
    Shell 脚本，105-106
构建触发，97-102
代码质量度量，Violations，238-241
创建，22-26
延迟启动，82
描述，项目主页面，81
禁用，82
失败，140
自动生成，277
使用 Git，87-97
  构建的分支，88，92
  构建触发器，92-94
  转换到本地分支，90
  检查后清空，91
  提交更改日志中的作者信息，92
  排除来自触发器的区域，89
  排除来自触发器的用户，90
  可执行的 Git，说明，92
  在构建前合并，91
  构建后合并和推送操作，95-96
  构建前修剪分支，91
  递归更新子模块，92
  仓库地址，88
  源代码浏览器，92
  SSH 密钥，87
  工作空间位置，覆盖，91
  工作空间，构建前销毁，92
Gradle 项目中，127-129
Grails 项目中，125-126
命名，81
NAnt 构建脚本中，131
.NET 项目中，130
通知发送后，114-115
构建后操作，110-115，138
报告测试结果，110-111，138
Ruby 和 Ruby on Rails 项目，131-132
运行，115
启动其他构建作业，115
使用 Subversion，84-87
  排除来自触发的提交消息，86
  排除来自触发的区域，85
  排除来自触发的用户，86
  源代码浏览器，85
工作空间，覆盖，83
功能（回归）测试，136，137
  数量，167
  性能，167
  并行运行，168

G

Game of Life 示例应用，22-39
Gerrit Trigger 插件，94
Git，9
  要拿去构建的分支，88，92
  构建触发器，92-94
  切换到本地分支，90
  检查后清空，91
  提交更改日志中的作者信息，92
  排除来自触发器的区域，89
  排除来自触发器的用户，90
  自由风格的构建作业，87-97
  安装，10
  在构建前合并，91
  构建后合并和推送操作，95-96
  构建前修剪分支，91
  递归更新子模块，92
  仓库地址，88
  源代码浏览器，92
  SSH 密钥，87
  工作空间位置，覆盖，91
  工作空间，构建前销毁，92
Git 插件，20-21，87-88
GitHub 插件，96
GitHub 项目，4

GitHub 仓库，9，92，96
  账户，设置，11
  克隆一个本地副本，12
  创建分支，12-13
GlassFish 应用服务器，用来部署 Java 应用，329-338
GlassFish Servlet 容器，177
全局变量，69-70
Gmail，配置，77
Goldin，Evgeny（贡献人），xxi
Gradle
  内嵌，与 Jenkins 一起运行，127-129
  代码质量度量
    Checkstyle，230
    CodeNarc，237
  Jenkins 支持，5
Grails
  内嵌，与 Jenkins 一起运行，125-126
  CodeNarc 里的代码质量度量，237
Groeschke，Rene（贡献人），xxii
Groovy 脚本
  验证脚本，180-181
  CodeNarc 里的代码质量度量，236-237
  环境变量，108
  在构建作业中运行，108-110
  在脚本控制台运行，67
组
  Active Directory，176，177
  Atlassian Crowd，178
  LDAP，176
  UNIX，177

H

无主节点模式，启动从节点，311
Hibernate，数据库更新，325
Jenkins 的主目录，46-47，58-61，68
主页，16，69
热部署，328，331
HTML Publisher 插件，156-158
HTTP 代理服务器，77
Hudson，xx，3，4，5
  （也见于 Jenkins）
HUDSON_HOME 环境变量，46
HUDSON_URL 环境变量，107

I

IDE，代码质量度量，226
IM（见即时通信）
信息分发器，202-203
安装
  Ant，75
  Git，10
  JDK，71
  Jenkins，41-44
    来自二进制版本分发，43
    在构建服务器上，44-45
    在 CentOS 上，48
    在 Debian 上，47
    在 Fedora 上，48
    与 Java Web 一起启动，13-15
    在 Linux 上，43
    在 OpenSUSE 上，48-49
    在 Red Hat 上，48
    在 SUSE 上，48-49
    在 Ubuntu 上，47
    在 UNIX 上，43
    来自 WAR 文件，16，42
    在 Windows 上，42，43
    作为 Windows 服务，54-58
  JRE，10
  Maven，18-19，73
  插件，34
    （也见于特定插件）
  升级，62-63
即时通信（IM），203-209
  IRC，208-211
  Jabber 协议，204-208
集成测试，135，137
  数量，167
  性能，167
IRC（Internet Relay Chat），208-211
IRC 插件，209

J

Jabber Notifier 插件，204
Jabber 协议，204-208
Java 应用程序
  从 Maven 仓库部署，335-338
  部署到构建服务器上，329-338

重新部署一个指定版本，331
从先前的构建重新部署，332-335
测试报告，138
Java Development Kit（见 JDK）
Java Runtime Environment（JRE），安装，10
Java 版本安装，检查，41
Java Web Start
用来安装和启动 Jenkins，13-15
用来启动从节点，307-309
Javadoc，33-34
JAVA_ARGS 参数，47
JAVA_HOME 环境变量，71，107
JAVA_OPTS 环境变量，54
JBoss 应用程序服务器，部署 Java 应用程序，329，338
JDK（Java Development Kit），10
配置，19
配置多个版本，70-72
安装，71
需求，41
版本，多配置构建作业，266
JEE 应用程序（见 Java 应用程序）
Jenkins，3-4
社区，5
配置（见配置）
CVS 支持，20
专用的用户，45
环境，需求，9-13
帮助图标，18
历史，xx，4
主目录，46-47，8-61，68
主页，16，69
安装（见安装）
Java 需求，41
维护，343-358
归档构建作业，353-354
备份，62，348-353
迁移构建作业，354-358
监控磁盘空间，343-348
监控服务器负载，347-348
内存需求，44，54
作为开源项目，5
运行绑定的端口，42，44
快速发布周期，5
使用的原因，5
运行
在 Apache 服务器上，52-53
从应用程序服务器，16，53-54
从命令行，16，43
从 Java Web Start，13
作为独立的应用程序，49-52
停止，15
升级，62-63
版本控制系统支持，20，22，84
Jenkins 控制台，15
Jenkins M2 Extra Steps 插件，123
JENKINS_HOME 环境变量，46，47，68
JENKINS_JAVA_CMD 参数，48
JENKINS_JAVA_OPTIONS 参数，48
JENKINS_PORT 参数，48
JMeter，158-165
JobConfigHistory 插件，191-193
作业目录，58-61
JOB_NAME 环境变量，106
JOB_URL 环境变量，107
加入，在构建作业里，278-279
JRE（Java Runtime Environment），安装，10
JUnit 报告，136
验收测试，156
在自由风格构建作业里配置，25，138
格式化，25

K

Kawaguchi，Kohsuke（Hudson 的创建者）

L

LDAP 仓库，作为安全领域，175-176
LDAP/Active Directory，5
轻量备份，352
Linux，47
（也见于特定 Linux 平台）
升级 Jenkins，62
Liquibase，325-327
LoadStatistics 界面，67
为构建作业锁定资源，280
Locks and Latches 插件，280
LTS（Long-Team Support）发行版，4

M

M2Eclipse，5
邮件服务器，配置，76-77
维护，343-358
  归档构建作业，353-354
  备份，62，348-353
  迁移构建作业，354-358
  监控磁盘空间，343-347
  监控服务器负载，347-348
Manage Jenkins 界面，17，65-68
Manage Nodes 界面，67
Manage Plugins 界面，66
主 / 从架构用于分布式构建，301-311
矩阵构建作业（见多配置构建作业）
基于矩阵的安全，181-185
Maven，10
  自动化测试，359-364
  自由风格构建作业的构建步骤
    25，102-104
  Cobertura，146-148
  代码质量度量，
    Checkstyle，229
    CodeNarc，236
    FindBugs，235
    PMD 和 CPD，232
  配置，18-19，73-74
  环境变量，107
  Hudson 支持，5
  安装，18-19，73
  SNAPSHOT 依赖，103，116-117
  SNAPSHOT 版本，86
  版本号，282-284
Maven 构建作业，22，80，115-124
  归档二进制构建产物，禁用，118
  构建步骤，117，123
  代码质量度量，Violations，241-242
  创建，116
  部署构建产物到企业级仓库管理器，
    119-123
  磁盘使用率，346-347
  自动生成，270-277
    Artifactory 插件，275
    配置，271-272
    配置的继承，272-274
    Parameterized Trigger 插件，274
  增量构建，118
  模块，管理，123
  Post-build Actions，119
  私有仓库，118
  报告测试结果，137
  并行运行模块，118
  测试结果，139
Maven Jenkins 插件，270，277
Maven Release 插件，282
MAVEN_OPTS 环境变量，54
McCullough，Matthew（贡献人），xxii
内存，需求，44，54
度量指标（见报告）
Microsoft Active Directory，作为安全领域，
    176-177
迁移构建作业，354-358
移动设备，通知，216
MSBuild 插件，130
MSTest 插件，130
多配置构建作业，80，263-269
  组合过滤器，268
  配置矩阵，267
  创建，264-265
  定制化轴，267
  JDK 轴，266
  运行，267-269
  从节点轴，265-266

N

Nabaztag 插件，221
NAnt 构建脚本，131
NAnt 插件，131
.NET 项目，130
Nexus
  企业级仓库管理器，123，284
  Hudson 支持，5
夜间构建（见自动化夜间构建）
NODE_LABELS 环境变量，106
NODE_NAME 环境变量，106
通知，195
  主动通知，195
  构建指示器，202-203
  配置，20
  桌面通知，212-215
  邮件，173，195-199

从自由风格构建作业，114-115
即时消息，203-209
移动设备，216
使用 Nabaztag，221
被动（拉取），195
RSS 反馈，201-202
智能手机，213-216
SMS 消息，216-217
声音，218
语音，220
Notifo，213-215
NTLM 代理验证，78
NUnit，136

O

Odd-e（赞助商），xxiii
未完成任务，报告，247

P

P 环境变量，180
参数化构建作业，253-261，263
（也见于多配置构建作业）
构建脚本，254-256
创建，254
历史，261
对于 Git 标签的运行，260
对于 Subversion 标签的运行，259
远程启动，260-261
参数类型，254，257-258
Parameterized Build 插件，253
Parameterized Trigger 插件，262，274，332
参数化触发器，262-263
被动（拉取）通知，195
Password 参数，257
性能
代码覆盖率分析，145
应用程序，158-165
测试，141-142，166-168
权限（见验证）
PHP 应用程序，部署到应用服务器，
339-341
PHPUnit，136
管道（见构建管道）
插件
Active Directory，176
Amazon EC2，317
架构，Jenkins 和 Hudson 比较，5
Artifactory，121，275
Audit Trail，190-191
备份，351
构建管道，351
构建提升，329
CAS，178
Checkstyle，244
Clover，154
Cobertura，150
Copy Artifact，285，329，332
覆盖率 / 复杂度散点图，246
Crowd，Atlassian Crowd，178
依赖图视图，278
部署，328，329-331，332
部署 Websphere，328，329
磁盘使用率，345-346
DocLinks 插件，158
Eclipse，212
外部邮件，197-199
扩展读权限，187
FindBugs，244
Gerrit Trigger，94
Git，20-21，87-88
GitHub，96
HTML 发布者，156-158
安装，34-36
即时通信，204
IRC，209
Jabber 通知器，204
Jenkins M2 额外的步骤，123
JobConfigHistory，191-193
Locks and Latches，280
管理，66
Maven Jenkins，270，277
Maven Release，282
MSBuild，130
MSTest，130
Nabaztag，221
NAnt，131
Parameterized Build，253
Parameterized Trigger，262，274，332
PMD，244

提升构建，288
发布，339
Role Strategy，188
脚本安全领域，179-181
SFEE，178
声音，218
语音，220
Task Scanners，247
轻量备份，352
Tray Application 插件，212-213
升级，63
Violations，237-242
xUnit，138
插件目录，58
PMD，230-234，242
PMD 插件，244
准备关机界面，67
处理器，构建服务器需求，44
基于项目的安全，185-187
项目级别的权限，基于角色的安全，188
提升构建插件，288
进阶，281，288-295
属性
构建参数，256
全局，69-70
代理，配置，77-78
发布插件，339

Q

构建开始前的安静期，69，82

R

指示器，信息，202-203
回归测试（见功能（回归）测试）
从磁盘界面重新加载配置，66
远程服务，启动从节点，311
报告
验收测试结果，156-158
代码覆盖率度量，7，34，39
Clover，155
Cobertura，153
代码质量度量，7
Checkstyle，242
FindBugs，242
未完成任务，247
PMD，242
Violations 插件，237-242
Javadoc API 文档，33-34
性能测试结果，163-165
测试结果，30-32
汇总，295-296
配置，137-138
显示，139-142
JUnit 报告，25，110-111
RSS 反馈，201-202
角色策略插件，188
基于角色的安全，188-189
RSS 反馈，构建结果，201-202
Ruby 应用程序，131-132，339-341
Ruby on Rails 项目，131-132，325
运行参数，258

S

SCM（源代码管理），22，84-97
（也可见于版本控制系统）
脚本终端界面，67
脚本安全领域插件，179-181
基于脚本的应用程序，部署到应用服务器上，339-341
脚本，104
（也可见于 Ant；Maven）
批量脚本，75，105-106
定制化验证脚本，179
部署脚本，324
Groovy 脚本，108-110
语言支持，110
参数化，254-256
Shell 脚本，75，105-106
安全，169-171
验证，169
基于矩阵的安全，181-185
无限制，170-171
基于项目的安全，185-187
基于角色的安全，188-189
启用，169
安全领域，169
Atlassian Crowd，178
CAS，178
定制化，179-181

启用注册，170
启用用户注册，173
Jenkins 内部用户数据库，170，171-174
LDAP 仓库，175-176
Microsoft Active Directory，176-177
Servlet 容器，177
SFEE，178
Unix 用户和组，177
Servlet 容器
作为安全领域，177
独立模式运行 Jenkins，49
SFEE（Source Forge 企业版），178
Shell 脚本，75，105-106
从节点
对于分布式构建，45，301-311
对于多配置构建作业，265-266
智能手机，通知，213-215
冒烟测试，327
SMS 信息，通知，216-217
SNAPSHOT 依赖，103，116-117
SNAPSHOT 版本，86
Sonar
代码质量度量，226，248-251
构建频率，98
Sonatype 工具，4，5
Sounds 插件，218
声音，通知，218
源代码浏览器
Git，92
Subversion，85
源代码管理（见 SCM：版本控制系统）
Source Forge 企业版（见 SFEE）
Speaks 插件，220
本书的赞助商，xxiii
SSH 密钥，11，87
SSH，启动从节点，303-306
单机应用程序
运行 Jenkins，49-52
升级 Jenkins，62
开始页面（见主页）
String 参数，254
Subversion
配置，76
排除来自触发构建的提交消息，86
排除来自触发构建的区域，85
排除来自触发构建的用户，86
自由风格的构建作业，84-87
Jenkins 支持，20
源代码浏览器，85
标签，构建，259
SVN_REVISION 环境变量，107
系统信息界面，67
系统日志界面，67

T

任务扫描插件，247
TDD（测试驱动开发模式），135
测试结果趋势图，30
测试驱动开发，7
Test::Unit，136
TestNG，136，138，143
测试
验收测试，7，136，155-158
自动化，6，7，135-137
Ant，365-367
Maven，359-364
自由风格的构建作业，138
功能（回归）测试，136，137
忽略，142-144
集成测试，135，137
Maven 构建作业，137
性能，141-142，166-168
性能测试，158-165
报告，30-32
汇总，295-296
配置，137-138
显示，139-142
JUnit 报告，25，110-111
冒烟测试，327
测试驱动开发，7
单元测试，135，136
Web 测试，136，137
轻量备份插件，352
Tomcat 应用程序服务器
部署 Java 应用程序，329-338
部署 Jenkins，16
Tomcat Servlet 容器，177
Tray Application 插件，212-213

U

U 环境变量，180
Ubuntu 企业云，317
单元测试，135，136
UNIX，43
  （也见于特定 UNIX 平台）
  用户和组，作为安全领域，177
不稳定的构建，139
  标准，113，152，240，244
  指示器，37
  通知，115，195，198
  触发另一个构建作业，98，115
更新目录，58
升级，62-63
用户数据库，170，171-174
  （也可见于安全，安全领域）
userContent 目录，58
用户
  管理员
    Jenkins 内部用户数据库，171
    基于矩阵的安全，182
  审计操作，189-193
  验证（见验证）
  声明失败的构建，200
  排除触发构建，86-90
  在构建服务器上，45
users 目录，58

V

版本控制系统，9，75
  （也可见于 CVS；Subversion）
  配置，22，75-76
  查询触发构建的更改，90
  远程触发构建，100-102
  支持 Jenkins，75-76，84
版本号，Maven，282-284
Violations 插件，237-242
虚拟机，构建服务器，45，167
Visual Studio MSBuild，130

W

Wakaleo Consulting（赞助商），xxiii
war 目录，59
WAR 文件，安装 Jenkins，16
Web 测试，136，137
WebSphere 应用程序服务器，328，329
Windows
  Jenkins 安装包，42
Windows 服务，
  安装 Jenkins，54-58
  安装从节点，310
  启动从节点，311
WMI（Windows 管理说明），311
workspace 目录，59

X

XML 格式的测试报告（见 JUnit 报告）
Xu，Juven（贡献人），xxii
xUnit，136，138
xUnit 插件，138

关于作者

John Ferguson Smart 是知名的国际顾问、培训师、作者，开源和敏捷 Java 开发与测试实践者。他专门研究如何帮助开发团队用测试驱动开发（包括 BDD 和 ATDD）、自动化验收测试、持续集成、构建自动化和干净代码实践来提高技术。他提供在开源技术、SDLC 工具和敏捷开发方面的培训和指导。

John 是 Wakeleo 咨询公司（*http://www.wakaleo.com*）的总监，提供咨询、培训以及开源和敏捷 Java 开发实践的顾问服务。

封面介绍

本书封面上的动物：这是一种丽春雨蛙（Pseudacris 蛙）。这些小型两栖动物体长 1 ~ 1.5 英寸，分布于美国的北卡罗来纳州、佛罗里达州中部和路易斯安那州东部。它们喜欢没有茂密植被的浅水区域，比如池塘、路旁沟渠和沼泽地等。

丽春雨蛙的颜色取决于分布的位置，可以是黑色、白色、棕色、红色、绿色，或者一些变异色。它们都有暗条纹或从鼻孔经过眼睛贯穿到肩膀的斑点，大多数还有其他各种斑点或条纹。它们每年 11 月到次年 3 月进行繁殖。在此期间，在浅水区或其附近，还可以听到公蛙的呼叫。

丽春雨蛙的名字来自于其交配的呼叫声：Pseudacris 来自古希腊的“虚假蝗虫”。该名字在 1836 年由美国博物学家 John Edwards Holbrook 提出，他注意到快速尖锐的声音很像由臭名昭著的蝗虫发出的。

封面图片来自 *Cassell's Natural History*。