

第3章 基于 Subversion 的持续集成实战

通常，多人协作的软件研发团队中的软件配置管理（Software Configuration Management, SCM）工作必不可少，而这一工作中的 CMS（配置管理系统，比如 ClearCase、Subversion）是不可或缺的。CruiseControl 持续集成服务器支持各种主流的配置管理系统，这是成熟 CI 服务器的重要基本特性。本章内容将以其支持的 Subversion 配置管理系统为主线，来实战持续集成，全章内容自成体系。

3.1 CruiseControl 内置的 SCM 支持

作为一款成熟的 CI 服务器，开源的 CruiseControl 支持各种主流的 SCM。比如，AccuRev、AlienBrain、ClearCase、CM Synergy、CVS、Darcs、AllFusion Harvest、MKS、Perforce、Plastic SCM、PVCS、SnapshotCM、Borland StarTeam、Surround SCM、Subversion、Microsoft Visual Studio Team Foundation Server、Visual SourceSafe、Mercurial 等。由于 CruiseControl 内置插件架构的灵活性和可扩展能力，使得增加新的 SCM 支持并不困难。

通常，如果基于某种 SCM 工具实施持续集成服务器，则同时需要在部署 CruiseControl 服务器的机器上安装这一 SCM 工具的客户端，比如 CVS、Subversion 客户端支持。图 3-1 展示了这一过程。

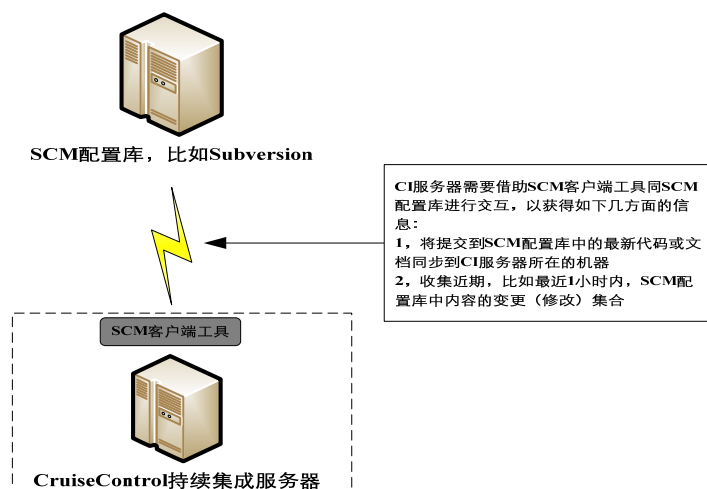


图 3-1 CruiseControl 借助于 SCM 客户端工具与配置库交互

迄今为止，对于 CruiseControl 支持的大部分 SCM 工具而言，CI 集成人员需要单独将 SCM 客户端工具安装到 CruiseControl 持续集成服务器所在的机器上。随着 CruiseControl 持续集成服务器新版本的发布，一些 SCM 客户端工具会被内置到 CruiseControl 发布版中，比如将各类 SCM 提供的 SDK、运行期支持附带在 CruiseControl 正式发布版里面。一旦 SCM 客户端工具被内置到 CruiseControl 后，持续集成的实施过程将被简化。

3.2 Subversion 的安装及配置

现在，我们来介绍 Subversion 服务器和客户端的安装及配置、使用等方面的内容，这是实施持续集成的前提。如果 CI 集成人员所在的研发团队已经存在现成的 Subversion 环境，则可以跳过这部分内容。

关于各种 SCM 工具的安装及配置使用

尽管本书给出了若干种 SCM 工具的安装、配置及使用方法，但实际企业应用采纳它们的方式、方法都会有所差别，而且这里给出的做法不一定是最好的，希望 CI 集成人员能够注意这一点。

另外，为搭建全新的 CI 环境，CI 集成人员需要准备 SCM 服务器、SCM 客户端工具、CruiseControl 服务器的安装及配置、SCM 配置库的准备、获得本地快照、准备 config.xml 配置文件等。

3.2.1 安装 Subversion

Subversion SCM 工具支持多种类型的远程客户，比如 SSH、HTTP WebDAV、基于文件路径等。这里以 HTTP WebDAV 访问方式为例，即同时需要安装 Apache HTTP 服务器和 Subversion 服务器。

首先，CI 集成人员需要去 Apache 官网（<http://httpd.apache.org/>）下载 Apache HTTP 服务器，比如 apache_2.0.61-win32-x86-openssl-0.9.7m.msi。随后，依据这一安装程序完成 HTTP 服务器的安装，这里将它安装在“D:\Program Files\Apache Group\”位置。图 3-2 给出了安装过程中的一个截图。

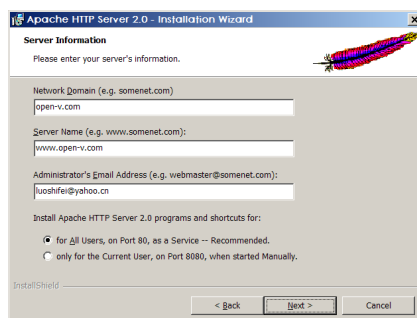


图 3-2 安装 Apache HTTP 服务器

一旦成功完成 Apache HTTP 服务器的安装，Windows Server 2003 中会多出一名字为“Apache2”的服务，而且这一服务的启动方式是“自动”。如果需要，CI 集成人员可以把它修改为“手动”。当这一服务启动后，通过浏览器便能够验证 Apache HTTP 服务器是否成功安装及启动，具体见图 3-3。

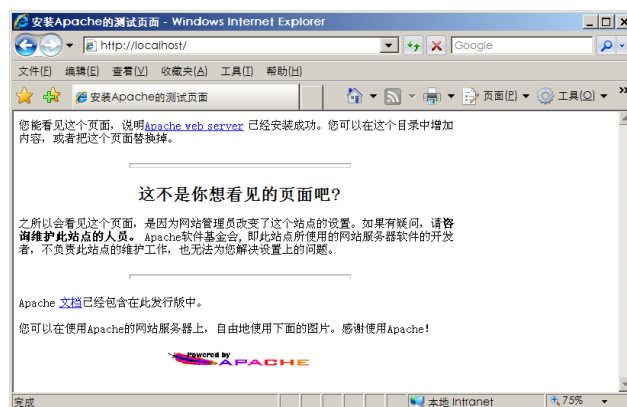


图 3-3 成功访问 Apache HTTP 服务器

CI 集成人员需要去 Subversion 官网 (http://subversion.tigris.org/project_packages.html) 下载 Subversion 服务器，比如 svn-1.4.5-setup.exe。随后，请 CI 集成人员完成它的安装，这里将它安装在“D:\Program Files\Subversion”位置。通过 Apache Monitor 能够证实 Subversion 是否安装成功，具体见图 3-4。

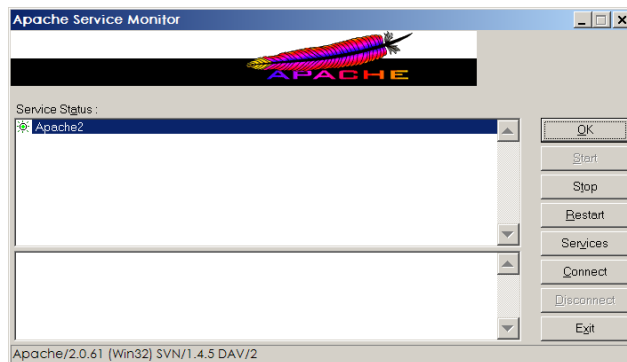


图 3-4 成功访问 Apache HTTP 服务器

另外，我们还能够通过“svn --version”命令行验证 Subversion 服务器的安装，具体如下。

```
D:\cruisecontrol-bin-2.7.2>svn --version
```

```
svn, 版本 1.4.5 (r25188)
```

```
编译于 Aug 22 2007, 20:49:04
```

```
版权所有 (C) 2000-2007 CollabNet。
```

```
Subversion 是开放源代码软件，请参阅 http://subversion.tigris.org/
```

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

```
可使用以下的仓库访问模块:
```

```
* ra_dav : 通过 WebDAV(DeltaV)协议访问仓库的模块。
```

- 处理“http”方案
- 处理“https”方案

```
* ra_svn : 使用 svn 网络协议访问仓库的模块。
```

- 处理“svn”方案

```
* ra_local : 访问本地磁盘的仓库模块。
```

- 处理“file”方案

```
D:\cruisecontrol-bin-2.7.2>
```

3.2.2 配置及初始化 Subversion 配置库

现假定把 Subversion 配置库存放在 D:\svn-repos 位置，并将 petclinic 项目存储到这一配置库中。首先，借助 svnadmin 进行配置库的初始化，操作示例如下。

```
d:\cruisecontrol-bin-2.7.2>mkdir d:\svn-repos
```

```
d:\cruisecontrol-bin-2.7.2>svnadmin create d:\svn-repos
```

```
d:\cruisecontrol-bin-2.7.2>
```

其次，借助 svn 命令完成 petclinic 项目到 Subversion 配置库的导入工作，具体如下（假定 petclinic 项目已经放在了 D:\temp 目录中）。

```
D:\temp>svn import -m "导入 Spring PetClinic 项目" file:///d:/svn-repos/petclinic/trunk
增加          petclinic
增加          petclinic/loadtest
增加          petclinic/loadtest/petclinicloadtest.jmx

.....

提交后的版本为 1。

D:\temp>
```

接下来，修改位于 D:\Program Files\Apache Group\Apache2\conf 目录的 httpd.conf 配置文件，并在最后添加如下类似内容。

```
<Location /svn-repos>
    DAV svn
    SVNPath d:/svn-repos
</Location>
```

好了，重启 Apache HTTP 服务器，便可通过浏览器访问到位于 Subversion 配置库中的 petclinic 配置项，具体见图 3-5（<http://localhost/svn-repos/petclinic/trunk/petclinic/>）。

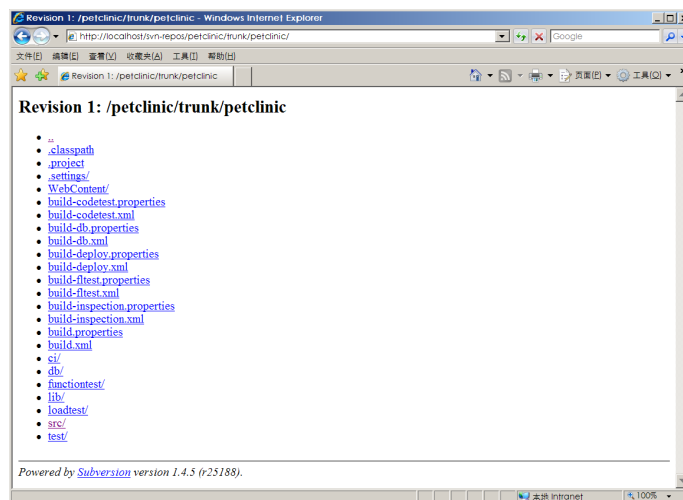


图 3-5 查看 petclinic 项目所在的 Subversion 配置库

通常，我们需要保护 Subversion 配置库，这里以保护 HTTP WebDAV 为例。首先来借助 Apache HTTP 服务器内置的 htpasswd 实用工具创建若干用户，并将用户（svnuser1/password、svnuser2/password）存储到 D:\svn-repos\conf 目录的 htpasswd 文件里面。操作示例如下。

```
D:\Program Files\Apache Group\Apache2\bin>htpasswd -c -m d:\svn-repos\conf\htpasswd svnuser1
New password: *****
Re-type new password: *****
Adding password for user svnuser1

D:\Program Files\Apache Group\Apache2\bin>htpasswd -m d:\svn-repos\conf\htpasswd svnuser2
New password: *****
Re-type new password: *****
Adding password for user svnuser2

D:\Program Files\Apache Group\Apache2\bin>
```

现在，我们需要调整 httpd.conf 配置文件，即调整上述<Location/>元素，具体如下。

```
<Location /svn-repos>
    DAV svn
    SVNPath d:\svn-repos

    AuthType Basic
    AuthName "Subversion PetClinic Repository"
    AuthUserFile d:\svn-repos\conf\htpasswd
    Require valid-user
</Location>
```

现在匿名用户再也不能访问 Subversion petclinic 配置库了，图 3-6 给出了操作示例。

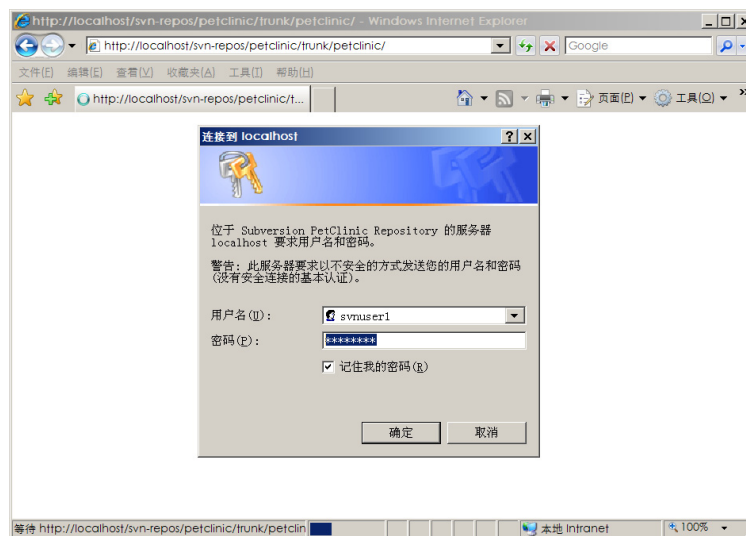


图 3-6 通过 HTTP Basic 认证访问 Subversion petclinic 配置库

如果允许匿名用户浏览 Subversion 配置库，而不向他们开放写权限，则可以根据如下配置示例调整<Location/>元素。

```
<Location /svn-repos>
```

```
DAV svn
SVNPath d:\svn-repos

AuthType Basic
AuthName "Subversion PetClinic Repository"
AuthUserFile d:\svn-repos\conf\htpasswd

<LimitExcept GET PROPFIND OPTIONS REPORT>
    Require valid-user
</LimitExcept>
</Location>
```

哪类文件应该作为配置项？

通常，SCM 配置库中不应该存储如下一些产出物：被编译出的.class，比如 petclinic WAR 应用中的 WEB-INF/classes、testbin、functiontestbin 等目录；其他类型的产出物，比如运行 Selenium 自动化功能测试产生的 selenium.txt 文件等。

那哪些文件应该作为配置项呢？比如，项目基代码（包括 JSP 文件）、单元及集成测试代码、功能测试代码及脚本、负载测试代码及脚本、SQL DDL 和 DML 脚本、第三方 Jar 包等。

从持续集成角度看，凡是 CI 服务器能够加工出的临时产出物都不用作为 SCM 配置项进行管理。即使管理这些产出物，这些产出物的时效性又没有办法进行控制和保证，因此还是建议不要将这类产出物提交到 SCM 配置库中。当然也存在一些例外，具体情况要具体分析。

3.3 Spring PetClinic 概述

全书将围绕 Spring Framework 2.5 内置的 PetClinic 示例应用展开，为适应持续集成工作的需要，我们修改并完善了它。

通过单击 petclinic 内置的 db\hsqldb\server.bat 脚本，便能够达到启动 HSQL DB 数据库的目的，具体见图 3-7。

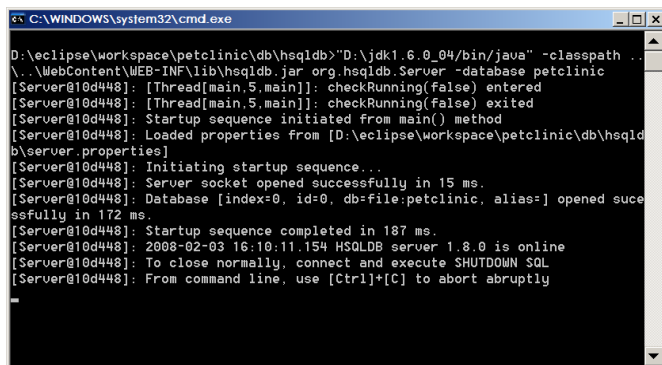


图 3-7 启动 HSQL DB 数据库

随后，将 petclinic WAR 应用部署到 Apache Tomcat 后，我们便能够访问到其主页，具体见图 3-8。



图 3-8 PetClinic 主页

整个持续集成涉及到的主要内容（比如持续数据库集成、持续单元及集成测试、持续评审、持续部署、持续功能及负载测试、持续反馈）都将通过这一示例应用进行阐述。

3.4 借助 Subversion 和 CruiseControl 实施持续集成

本节将详细介绍基于 Subversion 和 CruiseControl 的持续集成实战。

3.4.1 从 Subversion 检出 petclinic 项目

在实施持续集成工作前，我们必须把目标代码从 SCM 工具中检出。通常，各类 SCM 工具都提供了命令行或 GUI 工具，比如 svn、cvs、clear tool 等。在这些工具的帮助下，CI 集成人员能够把已存储到 SCM 配置库中的 petclinic 项目检出到本地，从而获得本地快照，即私有工作空间（private workspace）。

现在，借助于如下 svn 命令（<http://localhost/svn-repos/petclinic/trunk/petclinic>），CI 集成人员能够将 petclinic 项目检出到 D:\cruisecontrol-bin-2.7.2\projects 目录。

```
D:\cruisecontrol-bin-2.7.2\projects>svn co http://localhost/svn-repos/petclinic/trunk/petclinic

.....

A    petclinic\loadtest
A    petclinic\loadtest\petclinloadtest.jmx

.....

取出版本 1。

D:\cruisecontrol-bin-2.7.2\projects>
```

3.4.2 配置 config.xml

从前一章内容了解到，config.xml 的地位非常重要，它完整地给定了 CruiseControl 实施持续集成工作的具体细节。接下来，我们将再次深入到这一配置文件中，从而顺利完成基于 Subversion 的持续集成实战。

下面摘录了 config-svn.xml 配置文件的部分配置信息。<currentbuildstatuslistener/> 元素用于监控 ProjectEvent 事件，并将事件中内置的最新构建状态写入到某文本文件中，以供 Web 控制台读取。此时，这一 status.txt 文本文件位于 D:\cruisecontrol-bin-2.7.2\logs\petclinic 目录。

```
<listeners>
    <currentbuildstatuslistener file="logs/${project.name}/status.txt" />
</listeners>
```

```
<bootstrappers>
  <svnbootstrapper localworkingcopy="projects/${project.name}"
    username="svnuser2" password="password"/>
</bootstrappers>
```

与此同时，<svnbootstrapper/>元素能够从 Subversion 获得最新的 petclinic 项目快照。CI 集成人员借助位于 D:\Program Files\Subversion\bin 目录中的 svn 命令行工具也能够达到如下类似目的，示例操作如下。实际上，CruiseControl 内部正是借助 svn 命令行工具达到快照的更新目的。注意，这一命令的执行是在 D:\cruisecontrol-bin-2.7.2\projects\petclinic 目录中完成的。

```
svn update --non-interactive --username svnuser2 --password password
```

下面摘录了用于收集修改集合、调度构建工作的相关配置信息。可以看出，每隔 1 小时（3 600 秒），构建工作便可能会被触发 1 次。如果 1 小时内 petclinic 配置库发生了变化，则 CruiseControl 会自动完成单次构建工作，即调用 D:\cruisecontrol-bin-2.7.2\projects\petclinic 中 build.xml 的默认 Ant Target（ci），即 ci 目标。

```
<modificationset quietperiod="1">
  <svn localworkingcopy="projects/${project.name}"
    username="svnuser2" password="password"/>
</modificationset>
<schedule interval="3600">
  <ant anthome="apache-ant-1.7.0"
    buildfile="projects/${project.name}/build.xml"/>
</schedule>
```

<svn/>用于收集上次构建工作到这次即将展开的构建工作期间 Subversion petclinic 配置库所发生的变更。类似地，CruiseControl 内部也是借助 svn 工具收集修改集合的，示例操作如下。注意，这一命令的执行是在 D:\cruisecontrol-bin-2.7.2\projects\petclinic 目录中完成的。

```
svn log --non-interactive --xml -v -r "{2008-02-08T08:01:59Z}":"{2008-02-08T08:04:56Z}"
--username svnuser2 --password password
```

下面摘录了 XML 构建结果（日志信息）的自定义工作。可以看出，JUnit（/clover/report）、Checkstyle（/checkstyle/xml）、PMD（pmd/xml/）等持续集成执行结果被合并到构建日志中，比如位于 D:\cruisecontrol-bin-2.7.2\logs\petclinic 目录的 log20080208165715Lbuild.10.xml 日志文件。

```
<log>
  <merge dir="projects/${project.name}/cioutput/clover/report/">
  <merge dir="projects/${project.name}/cioutput/checkstyle/xml/">
  <merge dir="projects/${project.name}/cioutput/pmd/xml/">
</log>
```

下面摘录了持续反馈涉及到的相关配置信息。一旦构建成功，位于 cioutput 目录及子目录的各类构建结果会被分发到 D:\cruisecontrol-bin-2.7.2\artifacts\petclinic 位置。

```
<publishers>
  <onsuccess>
    <artifactspublisher dest="artifacts/${project.name}"
      file="projects/${project.name}/cioutput/${project.name}fortest.war" />
    <artifactspublisher dest="artifacts/${project.name}"
      file="projects/${project.name}/cioutput/${project.name}forproduction.war" />
    <artifactspublisher dest="artifacts/${project.name}"
      file="projects/${project.name}/cioutput/${project.name}-src.zip" />
    <artifactspublisher dest="artifacts/${project.name}"
      dir="projects/${project.name}/cioutput/apidoc"
      subdirectory="javadocs" />
    <artifactspublisher dest="artifacts/${project.name}"
      dir="projects/${project.name}/cioutput/cover/html"
      subdirectory="codecoverage" />
    <artifactspublisher dest="artifacts/${project.name}"
      dir="projects/${project.name}/cioutput/jdepend/html"
      subdirectory="jdepend" />
    <artifactspublisher dest="artifacts/${project.name}"
      dir="projects/${project.name}/cioutput/javancss"
      subdirectory="javancss" />
    <artifactspublisher dest="artifacts/${project.name}"
      dir="projects/${project.name}/cioutput/functiontest/html"
      subdirectory="functiontest" />
    <artifactspublisher dest="artifacts/${project.name}"
      dir="projects/${project.name}/cioutput/loadtest/html"
      subdirectory="loadtest" />
  </onsuccess>
  <htmlmail buildresultsurl="http://localhost:8888/cruisecontrol/buildresults/
    ${project.name}"
    mailhost="mail.open-v.com" password="password"
    username="luosf" defaultsuffix="@open-v.com"
    returnname="PetClinic Continuous Integration"
    returnaddress="luosf@open-v.com"
    charset="UTF-8"
    skipusers="true"
    xsldir="webapps/cruisecontrol/xsl"
    css="webapps/cruisecontrol/css/cruisecontrol.css">
    <always address="luosf@open-v.com" />
  </htmlmail>
</publishers>
```

与此同时，无论持续构建结果成功与否，luosf@open-v.com 邮箱始终会收到最新的单次构建情况。<htmlmail/>元素用于定义邮件发送信息。比如，mailhost 属性用于

指定 SMTP 服务器地址、username 和 password 属性用于指定发送构建结果的邮件账户。`<always/>`子元素表明，无论构建结果任何，luosf@open-v.com 邮箱一直会收到每次构建详情。

3.4.3 触发持续集成工作

好了，CI 集成人员现在可以准备启动 CruiseControl CI 服务器。为触发持续集成工作，请 CI 集成人员依如下顺序进行相关工作。

- 运行 `D:\cruisecontrol-bin-2.7.2\projects\petclinic\db\hsqldb` 目录中的 `server.bat`，以启动 HSQL DB 数据库。
- 将 Tomcat 6.0.16 安装在 `D:\apache-tomcat-6.0.16` 目录，并启动这一 Tomcat 服务器，比如于 `D:\apache-tomcat-6.0.16\bin` 目录执行“`catalina run`”。注意，请确保“没有将 petclinic WAR 应用手工部署到其中”。
- 修改 `config-svn.xml` 中的邮件服务信息，比如邮件服务器信息、邮件账户等。
- 在运行 `cruisecontrol.bat` 时，记得加上“`-configfile config-svn.xml`”参数信息，或者 CI 集成人员可以在 `cruisecontrol.bat` 中指定这一参数，或者将 `config-svn.xml` 重命名成 `config.xml`。可选地，注意将 8888 端口置为 CruiseControl 暴露的 Web 控制台入口。

如果一切顺利，则在启动 CruiseControl 服务器后的 1 分钟内，单次构建工作能够顺利完成。如果需要手工干涉这一 CI 服务器，则可以借助第 2 章内容介绍的两种办法，或者通过“`/dashboard`”Web 控制台，具体见图 3-9（通过单击“Builds”标签页后能够到达这一页面）。

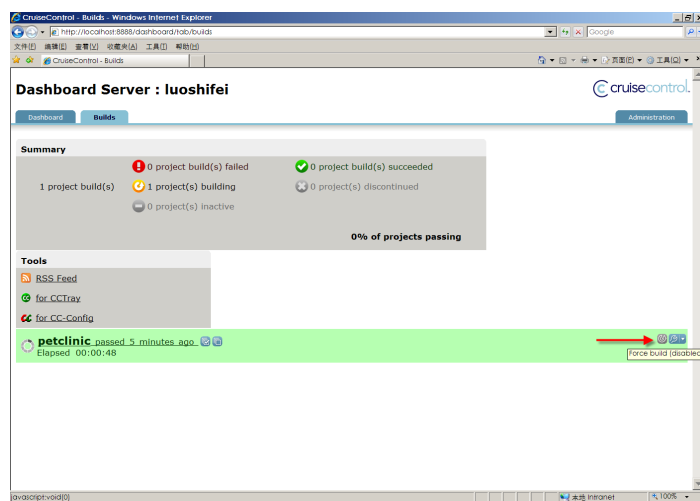


图 3-9 打乱 CruiseControl 自身的持续构建节奏（Force Build）

一旦单次构建工作结束，`D:\cruisecontrol-bin-2.7.2\artifacts\petclinic` 会新增不同的

子目录，这些子目录都是基于时间戳命名而成的，比如 20080312203105，其代表了本次构建工作最后 1 次收集修改集合的时间。这些子目录包含的具体内容见图 3-10。

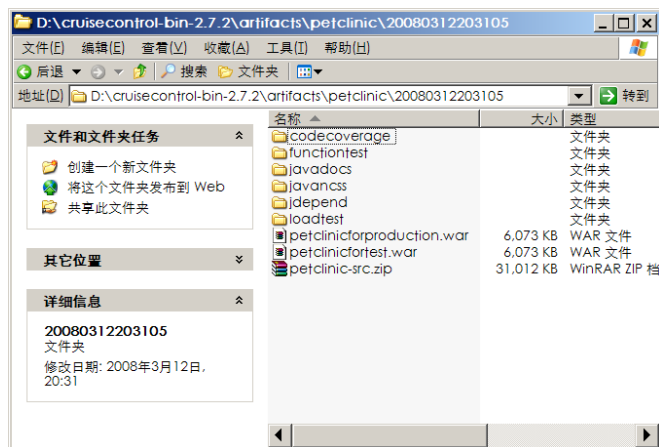


图 3-10 被自动分发的构建产出物

其中，petclinic-src.zip 压缩存档是本次构建工作使用到的完整 Eclipse petclinic 项目快照；petclinicfortest.war 和 petclinicforproduction.war WAR 包是分别用于测试和生产目的的 Web 应用存档；codecoverage 目录存储了代码覆盖度详情；functiontest 目录中摆放了功能测试的运行结果；javadocs 目录放置了针对 PetClinic 的 Javadoc（即详细设计）；javancss 和 jdepend 中分别存储了静态代码的评审结果；loadtest 目录存储了负载测试的执行情况。

借助 CruiseControl 内置的两个不同 Web 控制台（/cruisecontrol 和/dashboard），CI 集成人员（包括项目涉及人员）能够查看或下载它们。

3.4.4 查看持续构建结果

一旦单次构建工作完成后，CI 集成人员先前设定的邮箱将能够接收到如图 3-11 所示的类似邮件。这一邮件是持续反馈工作的重要组成部分，它将 Subversion petclinic 配置库中存在的问题集中反映在一起，并以“PUSH”的方式告知团队涉及人员。

图 3-11 自动接收到的持续构建邮件

根据邮件中给定的 URL 链接信息，它能够把我们带入到如图 3-12 所示的“构建结果”页面中。

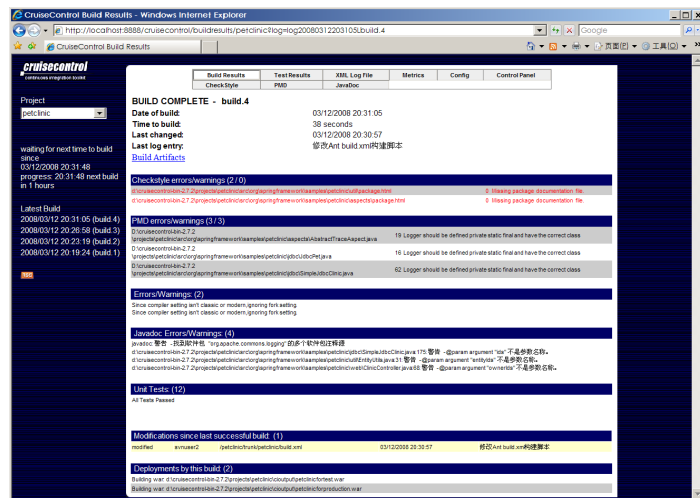


图 3-12 “Build Results” 页面

这一邮件表明，在当前的 petclinic 项目中，Checkstyle 问题存在 2 个；PMD 问题存在 3 个；Javadoc 问题存在 4 个；单元及集成测试全部通过；上次构建到这次构建工作期间，Ant build.xml 构建文件发生了修改，而且是由 svnuser2 用户完成的；本次构建工作产出了两个不同 WAR 包（petclinicfortest.war 和 petclinicforproduction.war）。通过“Build Artifacts”超链接（见图 3-13），我们还能够挖掘出当前项目存在的其他更多问题。



图 3-13 构建产出物

其中，codecoverage 目录存储了 Atlassian Clover 代码覆盖度情况，即在运行单元及集成测试代码期间，哪些基代码被测试到及具体的测试情况如何（具体见图 3-14）。测试代码的质量得到保证后，基代码的质量自然能够得到保证，最终确保了整个产品的质量。

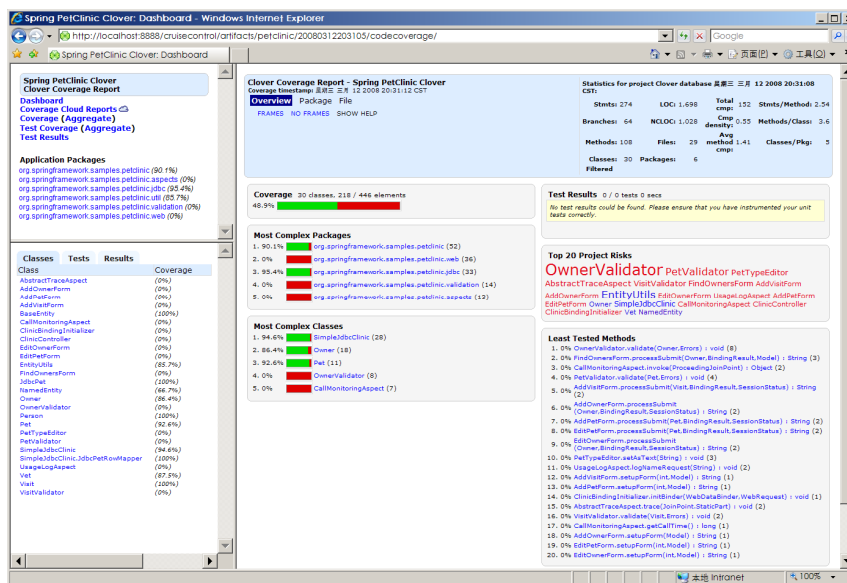


图 3-14 代码覆盖率

图 3-15 给出了 functiontest 目录存储的 Selenium 自动化功能测试结果。自动化功能测试工作消除了传统手工测试工作中的 80%重复内容，从而大大提高了功能测试人员的工作效率，并能加快产品的交付速度。

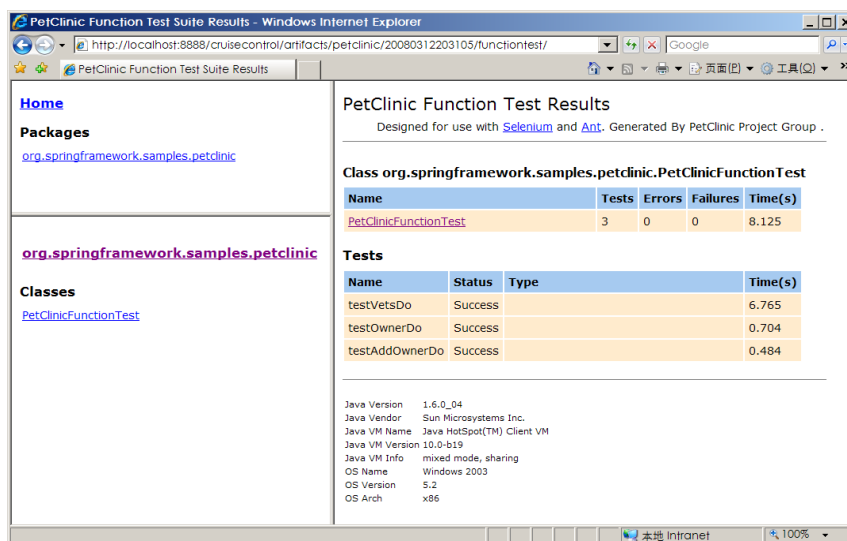


图 3-15 自动化功能测试结果

图 3-16 展示了 javadocs 目录中基于代码注释自动生成的详细设计文档。项目中的各类设计人员能够随时查看到项目的最新详细设计文档，而且这些文档还是自动生成的。

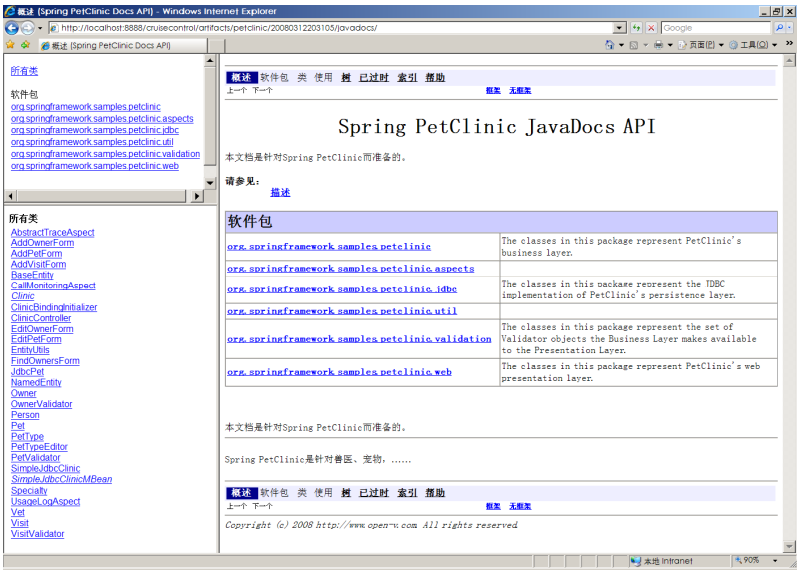


图 3-16 代码注释

图 3-17 展示了 JavaNCSS 的图形化输出结果。JavaNCSS 能够完成代码的静态评审，它是一款优秀的工具。

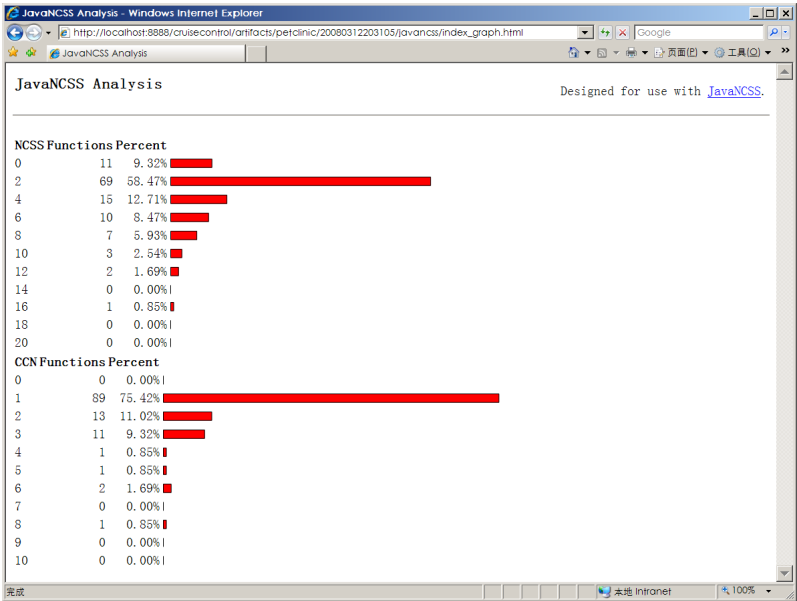
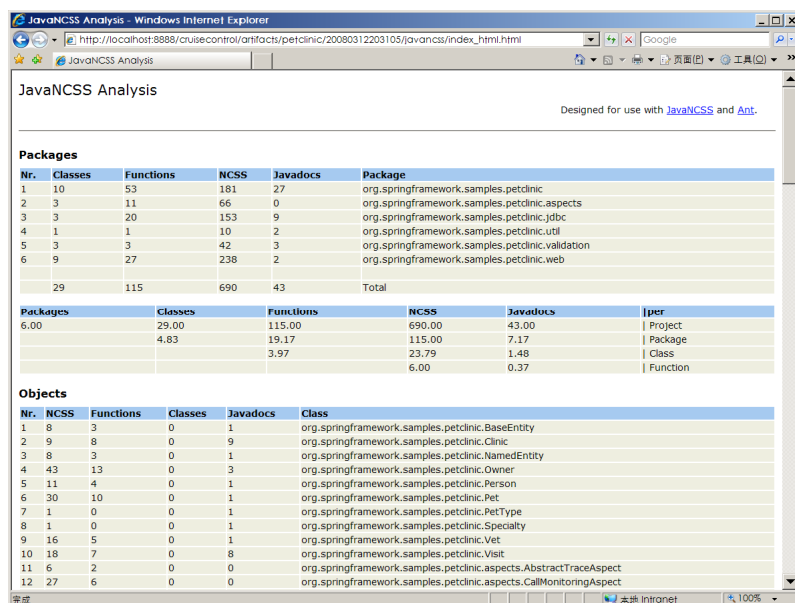


图 3-17 图形版的 JavaNCSS

图 3-18 展示了 JavaNCSS 的文字版输出结果。



JavaNCSS Analysis

Designed for use with [JavaNCSS](#) and [Ant](#).

Packages

| Nr. | Classes | Functions | NCSS | Javadocs | Package |
|-----|---------|-----------|------|----------|--|
| 1 | 10 | 53 | 181 | 27 | org.springframework.samples.petclinic |
| 2 | 3 | 11 | 66 | 0 | org.springframework.samples.petclinic.aspects |
| 3 | 3 | 20 | 153 | 9 | org.springframework.samples.petclinic.jdbc |
| 4 | 1 | 1 | 10 | 2 | org.springframework.samples.petclinic.util |
| 5 | 3 | 3 | 42 | 3 | org.springframework.samples.petclinic.validation |
| 6 | 9 | 27 | 238 | 2 | org.springframework.samples.petclinic.web |
| 29 | 115 | | 690 | 43 | Total |

Packages

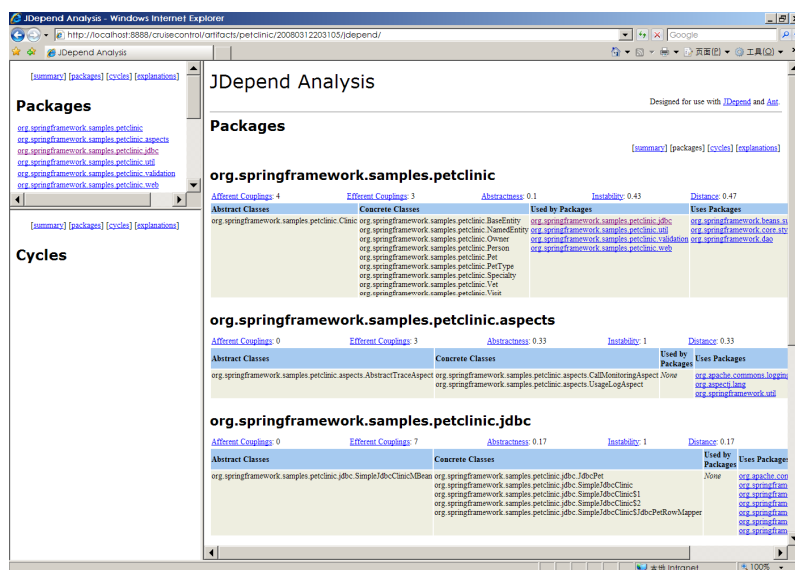
| Packages | Classes | Functions | NCSS | Javadocs | lper |
|----------|---------|-----------|--------|----------|----------|
| 6.00 | 29.00 | 115.00 | 690.00 | 43.00 | Project |
| | 4.83 | 19.17 | 115.00 | 7.17 | Package |
| | | 3.97 | 23.79 | 1.48 | Class |
| | | | 6.00 | 0.37 | Function |

Objects

| Nr. | NCSS | Functions | Classes | Javadocs | Class |
|-----|------|-----------|---------|----------|--|
| 1 | 8 | 3 | 0 | 1 | org.springframework.samples.petclinic.BaseEntity |
| 2 | 9 | 8 | 0 | 9 | org.springframework.samples.petclinic.Clinic |
| 3 | 8 | 3 | 0 | 1 | org.springframework.samples.petclinic.NamedEntity |
| 4 | 43 | 13 | 0 | 3 | org.springframework.samples.petclinic.Owner |
| 5 | 11 | 4 | 0 | 1 | org.springframework.samples.petclinic.Person |
| 6 | 30 | 10 | 0 | 1 | org.springframework.samples.petclinic.Pet |
| 7 | 1 | 0 | 0 | 1 | org.springframework.samples.petclinic.PetType |
| 8 | 1 | 0 | 0 | 1 | org.springframework.samples.petclinic.Specialty |
| 9 | 16 | 5 | 0 | 1 | org.springframework.samples.petclinic.Vet |
| 10 | 18 | 7 | 0 | 8 | org.springframework.samples.petclinic.Visit |
| 11 | 6 | 2 | 0 | 0 | org.springframework.samples.petclinic.aspects.AbstractTraceAspect |
| 12 | 27 | 6 | 0 | 0 | org.springframework.samples.petclinic.aspects.CallMonitoringAspect |

图 3-18 文字版的 JavaNCSS

图 3-19 展示了 jdepend 目录存储的 JDepend 评审结果。在静态评审代码的架构设计方面，JDepend 存在着很大的优势，比如探知已实现代码中包之间的耦合情况。



JDepend Analysis

Designed for use with [JDepend](#) and [Ant](#).

Packages

[\[summary\]](#) [\[packages\]](#) [\[cycles\]](#) [\[explanations\]](#)

org.springframework.samples.petclinic

Affected Couplings: 4 Effort Couplings: 3 Abstractness: 0.1 Instability: 0.43 Distance: 0.47

| Abstract Classes | Concrete Classes | Used by Packages | Uses Packages |
|--|---|---|---|
| org.springframework.samples.petclinic.Clinic | org.springframework.samples.petclinic.BaseEntity org.springframework.samples.petclinic.NamedEntity org.springframework.samples.petclinic.Owner org.springframework.samples.petclinic.Person org.springframework.samples.petclinic.Pet org.springframework.samples.petclinic.PetType org.springframework.samples.petclinic.Specialty org.springframework.samples.petclinic.Vet org.springframework.samples.petclinic.Visit | org.springframework.samples.petclinic.jdbc org.springframework.samples.petclinic.util org.springframework.samples.petclinic.validation org.springframework.samples.petclinic.web | org.springframework.samples.petclinic.aspects |

org.springframework.samples.petclinic.aspects

Affected Couplings: 0 Effort Couplings: 3 Abstractness: 0.33 Instability: 1 Distance: 0.33

| Abstract Classes | Concrete Classes | Used by Packages | Uses Packages |
|---|--|------------------|---|
| org.springframework.samples.petclinic.aspects.AbstractTraceAspect | org.springframework.samples.petclinic.aspects.CallMonitoringAspect | None | org.springframework.samples.petclinic.web |

org.springframework.samples.petclinic.jdbc

Affected Couplings: 0 Effort Couplings: 7 Abstractness: 0.17 Instability: 1 Distance: 0.17

| Abstract Classes | Concrete Classes | Used by Packages | Uses Packages |
|---|---|------------------|---|
| org.springframework.samples.petclinic.jdbc.SimpleDbClinicBean | org.springframework.samples.petclinic.jdbc.DbPet org.springframework.samples.petclinic.jdbc.SimpleDbClinic org.springframework.samples.petclinic.jdbc.SimpleDbClinicF1 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF2 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF3 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF4 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF5 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF6 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF7 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF8 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF9 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF10 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF11 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF12 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF13 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF14 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF15 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF16 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF17 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF18 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF19 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF20 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF21 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF22 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF23 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF24 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF25 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF26 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF27 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF28 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF29 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF30 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF31 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF32 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF33 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF34 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF35 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF36 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF37 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF38 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF39 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF40 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF41 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF42 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF43 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF44 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF45 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF46 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF47 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF48 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF49 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF50 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF51 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF52 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF53 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF54 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF55 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF56 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF57 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF58 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF59 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF60 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF61 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF62 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF63 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF64 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF65 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF66 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF67 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF68 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF69 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF70 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF71 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF72 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF73 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF74 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF75 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF76 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF77 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF78 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF79 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF80 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF81 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF82 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF83 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF84 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF85 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF86 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF87 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF88 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF89 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF90 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF91 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF92 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF93 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF94 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF95 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF96 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF97 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF98 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF99 org.springframework.samples.petclinic.jdbc.SimpleDbClinicF100 | None | org.springframework.samples.petclinic.web |

图 3-19 JDepend 自动分析出的代码结构

图 3-20 展示了 loadtest 目录存储的自动负载测试详情。JMeter 是 Apache 开发的一款用于负载测试的优秀工具，这里正是将 JMeter 集成到 CruiseControl 持续集成服务器而达到持续负载测试的目的。

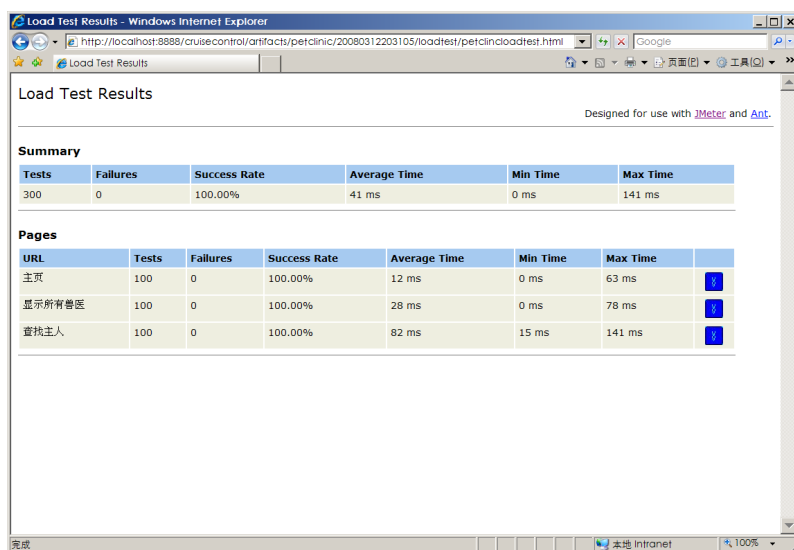


图 3-20 负载测试的执行情况

在仔细研究“Build Results”标签页后，我们再一次研究其他标签页。图 3-21 展示了“Test Results”标签页，这一标签页主要给出了单元及集成测试的执行情况，比如测试结果是否成功、执行各测试用例所花的时间、具体的测试日志。

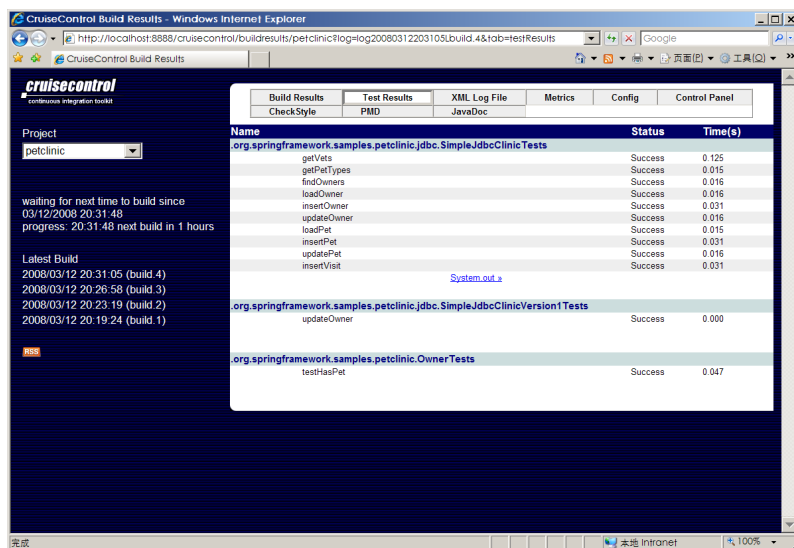


图 3-21 单元及集成测试的运行详情

图 3-22 给出了本次构建工作生成的完整日志信息，即将 log20080312203105Lbuild.4.xml 日志文件（位于 D:\cruisecontrol-bin-2.7.2\logs\petclinic\目录）输出到浏览器中。实际上，其他标签页展示的所有信息都是来自于存储在 D:\cruisecontrol-bin-2.7.2\logs\petclinic 目录中的日志文件集合。可以看出，通过“XML Log File”标签页，CI 集成人员还可以在远程下载到完整的日志文件。

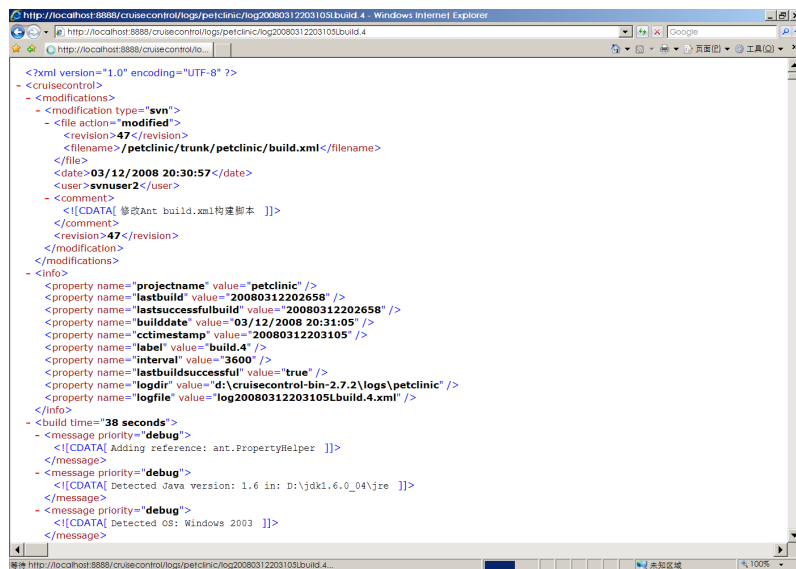


图 3-22 浏览历史日志信息

图 3-23 展示了“Metrics”标签页。它主要是在历史构建日志信息基础上进行了分析和统计而得来的，通过这一标签页，CI 集成人员能够了解到产品研发过程是否进展顺利。比如，不成功的构建比例是否很多、这些失败的构建发生在何时、Checkstyle/PMD/Javadoc 问题的分布情况等。

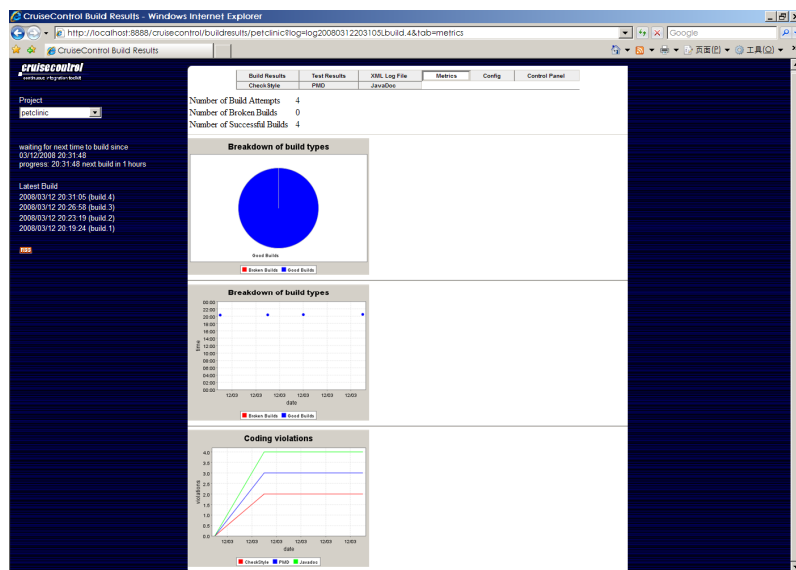


图 3-23 浏览统计信息

图 3-24 展示了“Checkstyle”标签页。不同于“Build Results”标签页所展示的 Checkstyle 问题，这里针对不同文件进行了分门别类，使得开发者修复问题更便捷。

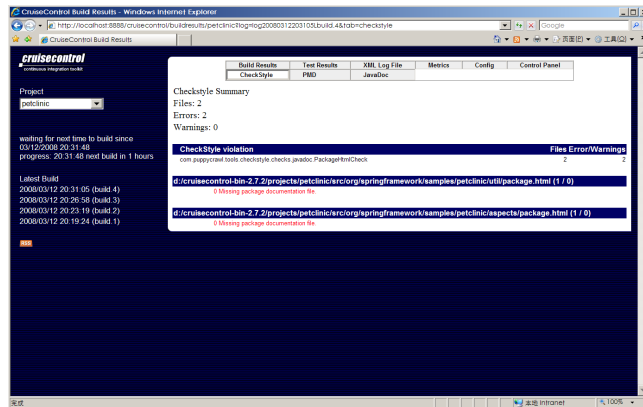


图 3-24 Checkstyle 问题细节

图 3-25 展示了“PMD”标签页。不同于“Build Results”标签页所展示的 PMD 问题，这里针对不同文件进行了分门别类，使得开发者修复问题更便捷。

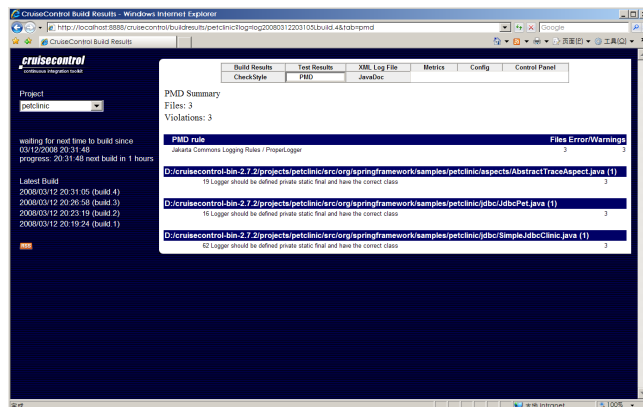


图 3-25 PMD 问题细节

图 3-26 展示了“Javadoc”标签页。

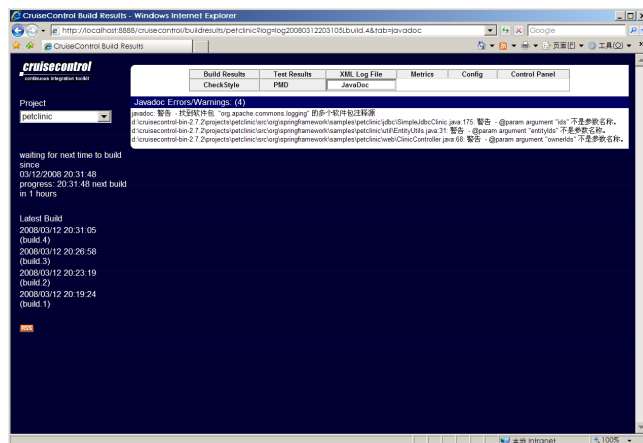


图 3-26 Javadoc 问题细节

到现在为止，CI 集成人员一定存在很多疑惑。不用着急，本书第 3 部分会系统性地研究相关内容，比如背景知识、这些结果到底代表了什么、为什么要实施它们等。到目前为止，对于 CI 集成人员而言最紧要的事情是，能够顺利获得上述各类产出物，并初步了解和理解它们。与此同时，CI 集成人员要逐步掌握 CruiseControl 持续环境的搭建，而全书第 2 部分内容都是围绕这一主题展开研究的。

3.5 Subversion 客户端支持

若 CI 集成人员更喜欢命令行方式，则 svn 便是不错的 Subversion 客户端工具。至于 GUI 类型的客户端，存在 Subclipse、TortoiseSVN、Subversive 等几十种。其中，Subclipse 和 Subversive 是基于 Eclipse 插件方式运行的。图 3-27 展示了 Subclipse 官网。

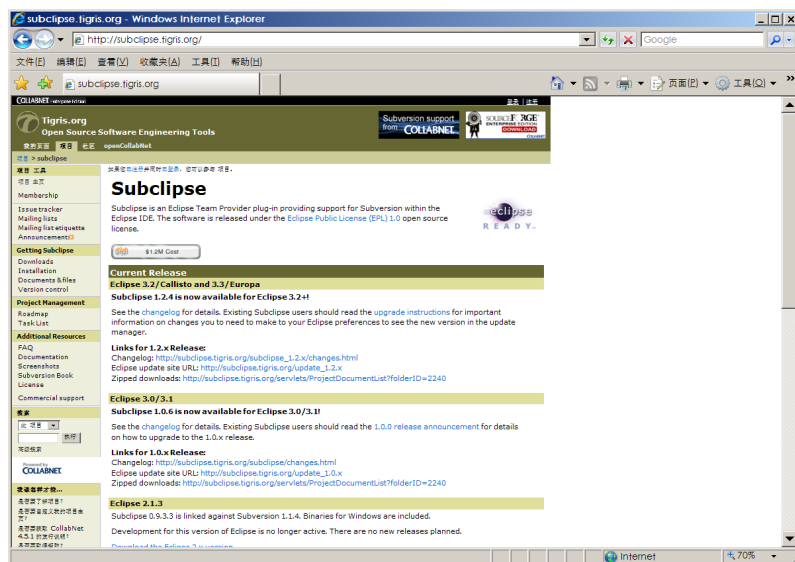


图 3-27 Subclipse 官网

图 3-28 展示了 Subclipse 插件的工作情景。CI 集成人员（包括研发人员）可以借助 Subclipse 插件将存储在 Subversion 配置库中的 petclinic 项目（<http://localhost/svn-repos/petclinic/trunk/>）同步到自身的私有工作空间中，进而体验 Subclipse 所蕴藏的强大功能。

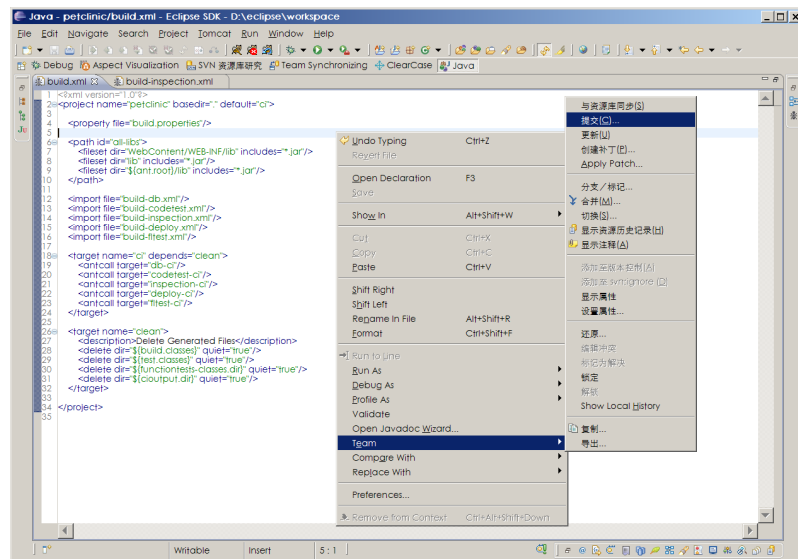


图 3-28 Subclipse 工具情景

3.6 CruiseControl 若干技巧

通过本节介绍的 CruiseControl 技巧，可以帮助广大开发人员大大提高持续集成的效率。

3.6.1 模块化 Ant 文件

CruiseControl 支持若干不同的脚本（构建）技术，以驱动持续集成工作，比如 Ant 构建技术、Maven 1.x（Maven 2.x）、NAnt、Phing。甚至，任意可执行的批处理或其他类型的程序都可以驱动持续集成工作。

随着持续集成工作的深入进行，脚本的内容会逐渐增多，比如从几千到几万行，甚至更多。如何合理组织它们呢？在实施 petclinic 持续集成工作期间，我们采纳了 Ant 构建技术，并根据持续集成内容的不同来组织相应的脚本文件集合。比如 build-db.xml 存放了同持续数据库集成相关的内容；Ant build-codetest.xml 文件管理同持续单元及集成测试相关的脚本内容；build-inspection.xml 存储了同持续评审相关的内容；build-deploy.xml 存储了同持续部署相关的脚本；build-ftest.xml 摆放了同持续功能及负载测试相关的 Ant 脚本。至于持续反馈相关的内容，则直接托付给了 CruiseControl 服务器。

下面摘录了 petclinic 内置的 Ant build.xml。它借助 Ant <import/>元素将其他 Ant 脚本集结在一起。

```

<?xml version="1.0"?>
<project name="petclinic" basedir="." default="ci">

    <property file="build.properties"/>

    <path id="all-libs">
        <fileset dir="WebContent/WEB-INF/lib" includes="*.jar"/>
        <fileset dir="lib" includes="*.jar"/>
        <fileset dir="${ant.root}/lib" includes="*.jar"/>
    </path>

    <import file="build-db.xml"/>
    <import file="build-codetest.xml"/>
    <import file="build-inspection.xml"/>
    <import file="build-deploy.xml"/>
    <import file="build-fltest.xml"/>

    <target name="ci" depends="clean">
        <antcall target="db-ci"/>
        <antcall target="codetest-ci"/>
        <antcall target="inspection-ci"/>
        <antcall target="deploy-ci"/>
        <antcall target="fltest-ci"/>
    </target>

    <target name="clean">
        <description>Delete Generated Files</description>
        <delete dir="${build.classes}" quiet="true"/>
        <delete dir="${test.classes}" quiet="true"/>
        <delete dir="${functiontests-classes.dir}" quiet="true"/>
        <delete dir="${cioutput.dir}" quiet="true"/>
    </target>

</project>

```

熟悉 Spring Framework 的 CI 集成人员都应该清楚，开发者可以也通过类似的 `<import/>` 元素引用到其他 Spring IoC 配置文件。

3.6.2 稳健应对构建失败

CI 集成人员是否注意到，`config-svn.xml` 中 `<project/>` 元素的 `buildafterfailed` 属性取值为 `false`，其取值摘录如下。我们需要合理地、有计划地控制 `buildafterfailed` 属性的取值（`true` 或 `false`，其默认取值是 `true`）。

```
<cruisecontrol>
```



```
<project name="petclinic" buildafterfailed="false">
.....
</project>
</cruisecontrol>
```

通常，在大中型研发团队中，项目涉及人员（架构师、DBA、开发者、测试人员、QA 人员、集成工程师）都会将各自的工作成果提交到统一的 SCM 配置库，比如 Subversion petclinic 配置库中。在研发期间，基代码、测试代码、SQL DDL 和 DML 脚本等会不定期提交到 SCM 配置库中。然而由于各种原因，比如 A 开发者忘记将一些修改过的基代码提交到 Subversion 中、B 功能测试人员将编译通不过的测试代码提交到 Subversion 中等，最终使得 CruiseControl 某次的构建工作失败。一旦构建工作失败，则项目各涉及人员便会收到与如图 3-29 所示类似内容的邮件。

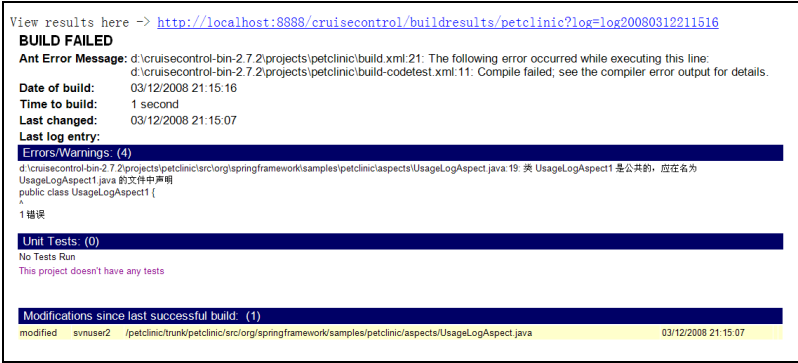


图 3-29 构建失败

当<project>插件中的 buildafterfailed 属性取值为 true（即默认值）时，加上项目涉及人员并未积极采取措施修复上述邮件反映的问题，则当下次构建时机到来后（比如，1 小时过去了），即使 Subversion petclinic 配置库中没有发生任何变化，反映编译错误的上述类似邮件还是会再次发送到项目涉及人员手中，即 CruiseControl 还是会定时触发构建工作。

请 CI 集成人员设想一下：某团队有 50 人规模。深夜 1:00 时，只有 A 工程师还在研发现场工作。不幸的是，他往 SCM 配置库提交了某编译通不过的单元测试类后就离开了办公室。第 2 天早上 9:00 上班时，这 50 人可能分别收到内容一样的 8~9 封邮件，这些邮件告知他们：昨晚 A 工程师提交的测试代码导致了持续集成工作失败。在正常情况下，我们都希望这 8~9 封邮件能够减少到 1 封。

另一方面，当 buildafterfailed 属性取值被置为 false 时，又是另一番风景。此时，即使 CruiseControl 构建时机到来（比如，自上次构建后 1 小时过去了），但 Subversion petclinic 配置库中没有任何变化，构建工作就不会再次被触发了。对于上述同样 50 人的场景，第 2 天早上 9:00 上班时，这 50 人只会收到 1 封反映编译错误的邮件。

在实际产品研发期间，对于实施持续集成工作而言，不同团队会不同地看待和控

制 `buildafterfailed` 取值，因为引起构建失败的原因很多。比如，在实施持续数据库集成期间，某数据库连接临时不可用；在实施持续部署期间，用于部署目标 `petclinic` WAR 应用的 `Apache Tomcat` 服务器不可用。在理想情况下，这类外部因素导致的构建失败不应该干扰到 `CruiseControl` 正常的持续集成节奏。

无论如何，不同团队需要区别对待 `buildafterfailed` 属性及其取值！

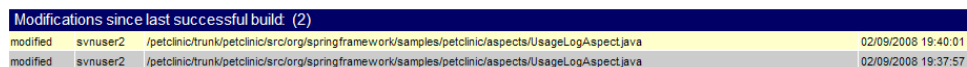
3.6.3 控制收集修改集合的策略

从附录 A 得知，`<modificationset>` 元素持有 `requiremodification`、`quietperiod`、`ignorefiles` 等属性。通常，`CruiseControl` CI 服务器会依据 `<schedule>` 元素中的 `interval` 取值设定的频率来进行构建工作，比如以 1 次/3600 秒的频率实施单次构建工作。

粗略地说，一旦 SCM 配置库中存在新的变化时，而且 `CruiseControl` 设定的持续构建时机已经来临，则单次构建工作便立即被触发。严格地说，只有 `<modificationset>` 元素内置的各子元素（比如 `<svn>`、`<cvs>`、`<pvc>`）收集到具体的修改信息时，持续构建工作才可能被触发。下面再次给出了针对 `petclinic` 项目的 `<modificationset>` 元素配置示例。

```
<modificationset quietperiod="1">
  <svn localworkingcopy="projects/${project.name}"
      username="svnuser2" password="password"
      uselocalrevision="true"/>
</modificationset>
```

可以看出，在正常情况下，只有同 `http://localhost/svn-repos/petclinic/trunk/petclinic/` 位置相匹配的 SCM 配置内容发生变更时，`<svn>` 元素才能够收集到修改集合，进而触发持续构建工作。图 3-30 给出了 `<modificationset>` 元素所收集到的修改集合示例。



| Modifications since last successful build: (2) | | | |
|--|----------|--|---------------------|
| modified | svnuser2 | /petclinic/trunk/petclinic/src/org/springframework/samples/petclinic/aspects/UsageLogAspect.java | 02/09/2008 19:40:01 |
| modified | svnuser2 | /petclinic/trunk/petclinic/src/org/springframework/samples/petclinic/aspects/UsageLogAspect.java | 02/09/2008 19:37:57 |

图 3-30 `<modificationset>` 收集到的修改集合

上述 `<svn>` 使用到 `uselocalrevision` 属性，其默认值为 `false`。一旦其取值为 `true`，修改集合的获得将基于上次构建的时间与本地快照的版本号（`revision`）统计出，而不再完全基于时间区间，下面给出对应的统计命令示例（注意，16 是版本号）。

```
svn log --non-interactive --xml -v -r "{2008-02-09T14:43:05Z}":16
--username svnuser2 --password password
```

与此同时，对于具有原子提交（Atomic Commit）能力的 SCM 工具而言，比如 `Subversion` 和 `Perforce`，我们建议这类项目中 `<modificationset>` 元素的 `quietperiod` 取值最好置为 0，而且本地快照的更新操作直接让 `<bootstrappers>` 的子元素（引导器集合）

完成即可。

最后，我们再来研究 `ignorefiles` 属性的使用。下面给出了配置示例，这一示例说明，修改集合不会统计那些后缀名为 `.xml` 和 `.properties` 的 SCM 配置内容变更。比如，SCM 配置库中的 `aftest.properties`、`applicationContext-ftest-common.xml`、`PetClinicFunctionTest.java` 文件内容发生了变更，修改集合只会承认 `PetClinicFunctionTest.java` 文件被修改了。这意味着，无论 `aftest.properties` 和 `applicationContext-ftest-common.xml` 配置项是否发生变更，它们都不会影响到持续集成工作的节奏。

```
<modificationset quietperiod="1" ignorefiles="*.xml,*.properties">
  <svn localworkingcopy="projects/${project.name}"
    username="svnuser2" password="password"/>
</modificationset>
```

如果 `ignorefiles` 取值中存在多项内容，则要用逗号将它们隔开。

从本节内容可以看出，因收集修改集合策略的不同，使得 CruiseControl 持续集成的节奏存在一定的变化。

3.6.4 自定义构建产出物的分发渠道

有关构建产出物的分发渠道，CruiseControl 支持多种类型，比如邮件服务器、FTP 服务器、Ant 构建技术、RSS、SCP、Socket、Yahoo IM 消息等。

下面给出了 `<htmlmail/>` 元素的简化版，即 `<email/>` 元素的使用。

```
<email
  buildresultsurl="http://localhost:8888/cruisecontrol/buildresults/${project.name}"
  mailhost="mail.open-v.com" password="password"
  username="luosf" defaultsuffix="@open-v.com"
  returnname="PetClinic Continuous Integration"
  returnaddress="luosf@open-v.com"
  skipusers="true"
  subjectprefix="aci">
  <always address="luosf@open-v.com" />
</email>
```

在采用 `<email/>` 元素后，邮件内容将很简单（从 HTML 邮件类型变成了纯文本类型），图 3-31 给出了示例。此时，邮件中并未包含持续集成的具体细节信息，CI 集成人员只有登录到 Web 控制台才能够了解到。是否注意到，`subjectprefix` 属性用于自定义邮件主题的前缀。

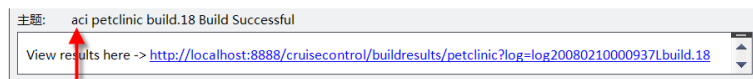


图 3-31 基于 `<email/>` 元素接收到的邮件

<artifactspublisher/>将产出物分发到特定位置，比如 D:\cruisecontrol-bin-2.7.2\artifacts 目录，从而使得通过 Web 控制台能够浏览并下载到它们。


如果需要，CruiseControl CI 集成人员可以直接借助 Ant 构建技术完成构建产出物的分发，在<antpublisher/>插件的帮助下，能够顺利完成任务，但 CI 集成人员必须提供对应的 Ant 脚本。

如果 CruiseControl CI 集成人员需要将各种<publishers/>子元素（分发器）组合在一起，则可以使用<compoundpublisher/>元素。下面给出了其使用示例。这时还使用到<onfailure/>元素，一旦构建工作失败，被包裹在其中的各分发器便会被触发，比如邮件会被发送出去。

```
<compoundpublisher>
  <onsuccess>
    <artifactspublisher dest="artifacts/${project.name}"
      file="projects/${project.name}/cioutput/${project.name}fortest.war" />
    <artifactspublisher dest="artifacts/${project.name}"
      file="projects/${project.name}/cioutput/${project.name}forproduction.war" />
  </onsuccess>
  <onfailure>
    <htmlmail
      buildresultsurl="http://localhost:8888/cruisecontrol/buildresults/${project.name}"
      mailhost="mail.open-v.com" password="password"
      username="luosf" defaultsuffix="@open-v.com"
      returnname="PetClinic Continuous Integration"
      returnaddress="luosf@open-v.com"
      charset="UTF-8"
      skipusers="true"
      xsldir="webapps/cruisecontrol/xsl"
      css="webapps/cruisecontrol/css/cruisecontrol.css">
      <always address="luosf@open-v.com" />
    </htmlmail>
  </onfailure>
</compoundpublisher>
```

3.6.5 借助 SVNLabelIncrementer 插件控制构建 Label 的生成

在默认情况下，CruiseControl 会生成类似“build.2”格式的构建 Label，以标识特定的单次构建工作，图 3-32 给出了示例。



| Project | Status (since) | Last failure | Last successful | Label | Build |
|-----------|------------------|--------------|-----------------|---------|-------|
| petclinic | waiting (下午6:09) | | 下午6:09 | build.2 | Build |

图 3-32 默认的构建 Label

在实际 CI 环境中, CI 集成人员希望构建 Label 更具内涵, 比如构建 Label 是否能够基于 SCM 配置库版本生成? 实践证明, 这是非常好的想法。CruiseControl 每次进行单次构建工作时, 都会存在对应的本地快照, 而这一快照肯定是来自 SCM 配置库的某一版本。当然, 不同 SCM 配置工具的版本策略有所不同。Subversion 的版本管理非常简单、实用, 每次往版本库中提交新的工件时, Subversion SCM 配置库的版本会自动递增。相比之下, CVS 用户可没有这么幸运。

针对不同 SCM 工具, CruiseControl 内置了不同的递增器。有关递增器的具体细节, CI 集成人员可以参考附录 A。这里将围绕 SVNLabelIncrementer 递增器展开, 为了使用它, 我们需要往 config.xml 中注册它, 具体如下。

```
<plugin name="labelincrementer"
      classname="net.sourceforge.cruisecontrol.labelincrementers.SVNLabelIncrementer"/>
```

随后, CI 集成人员需要告知 Subversion 本地快照的位置给这一插件, 具体如下。

```
<labelincrementer workingcoppath="projects/${project.name}"/>
```

下面给出了<labelincrementer/>相关配置。

```
<cruisecontrol>
  <project name="petclinic" buildafterfailed="false">
    <plugin name="labelincrementer"
          classname="net.sourceforge.cruisecontrol.labelincrementers.
            SVNLabelIncrementer"/>

    <labelincrementer workingcoppath="projects/${project.name}"/>

    <listeners>
      <currentbuildstatuslistener file="logs/${project.name}/status.txt" />
    </listeners>

    .....
  </project>
</cruisecontrol>
```

一旦启用 SVNLabelIncrementer 后, 构建 Label 的内容将类似于“svn.72M”, 这里的 72M 表示 Subversion 版本号, 图 3-33 展示了这一情况。


| <u>Project</u> | <u>Status (since)</u> | <u>Last failure</u> | <u>Last successful</u> | <u>Label</u> | |
|---------------------------|-----------------------|---------------------|------------------------|--------------|---|
| petclinic | waiting (下午6:27) | | 下午6:26 | svn.72M |  |

图 3-33 基于 SVNLabelIncrementer 生成的构建 Label

很显然, SVNLabelIncrementer 为排查持续集成暴露的各种项目问题带来了便利。比如, 据某次构建工作反馈的结果表明, 某 Java 类存在若干 PMD 问题。与此同时,

研发团队在不断地研发这一产品，而且这一 Java 类也在不断修改。因为启用了 SVNLabelIncrementer，这使得定位当时存在的 PMD 问题变得简单、高效。当没有“svn.72M”类似信息时，为找到这些 PMD 问题的根源，项目涉及人员需要想方设法找到当时使用的项目快照，这可悬了，因为这一项目快照可能早就被删除了。有了“svn.72M”后，项目涉及人员可以直接去 Subversion SCM 配置库中找到这一 Java 类。

最后再提一点，上述<labelincrementer/>递增器的注册也可以挪到<cruisecontrol/>根元素下，从而使得其他项目能够直接使用到它，而不用再次去注册这一递增器。配置示例如下。有关<plugin/>的使用细节，请参考附录 A。

```
<cruisecontrol>

  <plugin name="labelincrementer"
    classname="net.sourceforge.cruisecontrol.labelincrementers.
      SVNLabelIncrementer"/>

  <project name="petclinic" buildafterfailed="false">

    <labelincrementer workingcopypath="projects/${project.name}"/>

    <listeners>
      <currentbuildstatuslistener file="logs/${project.name}
        /status.txt" />
    </listeners>

    .....
  </project>
</cruisecontrol>
```

3.7 小结

本章内容从零开始借助于 Subversion 和 CruiseControl 实施持续集成工作。可以看出，实施持续集成并不难，期间不会涉及到太多的技术难题。尽管这里是以 Subversion SCM 工具为例展开 CruiseControl 持续集成的实施工作，但整个思路和做法适用于其他各种类型的 SCM 工具。

下章将基于“CVS+CruiseControl”实施持续集成工作。

【参考及推荐资料】

- <http://cruisecontrol.sourceforge.net/index.html>
- <http://subversion.tigris.org>
- <http://subclipse.tigris.org>
- <http://www.eclipse.org>

- <http://www.polarion.org/index.php?page=overview&project=subversive>
- <http://www.eclipse.org/subversive>
- <http://httpd.apache.org>
- 《Pragmatic.Version.Control.Using.Subversion.2nd》