

---

## 持续集成的实施

---

# 目 录

<b>1</b>	<b>Continuous Integration .....</b>	<b>1</b>
1.1	CI 概述 .....	1
1.2	CI 服务器介绍 .....	2
<b>2</b>	<b>借助 CC 实施 CI.....</b>	<b>3</b>
2.1	CruiseControl 持续集成服务器 .....	3
2.1.1	内置的 connectfour 项目 .....	4
2.2	启用定制的 CI 工具集 .....	5
2.2.1	配置新项目 .....	5
2.2.2	启动 CruiseControl.....	9
2.3	Ant build.xml 构建文件 .....	10
2.3.1	Ant 构建文件概述.....	10
2.3.2	配置 build.xml 中可复用的<property/>、<path/>等基础内容 .....	11
2.3.3	编译 target.....	14
2.3.4	打包 target.....	14
2.3.5	生成 Javadoc.....	15
2.3.6	借助 CheckStyle 和 PMD 审计代码质量 .....	16
2.3.7	借助 FindBugs 审计代码质量 .....	19
2.3.8	运行测试代码及评估测试代码覆盖度 .....	21
2.3.9	使用 Ivy 进行 jar 包管理 .....	23
2.4	在 CC 中集成 SVN .....	26
2.5	增加 CC 访问控制.....	27
<b>3</b>	<b>使用 Hudson 实施 CI.....</b>	<b>31</b>
3.1	Hudson 概述 .....	31
3.2	配置一个新项目 .....	32
3.3	插件管理 .....	36

# 1 Continuous Integration

持续集成（Continuous Integration，简称 CI）实现了持续的质量控制，在提高软件质量的同时有效降低消耗的时间成本。持续集成的突出特点就是每次完成一小步，但是动作很频繁，能够及时反馈项目的最新状态。

## 1.1 CI 概述

持续集成的最大意义在于持续和自动化的构建、评估和测试过程以及实时的信息反馈。持续集成可以在我们提交了一个文件之后就告诉我们功能、质量和对整个系统的影响情况，而不是在上线之前才发现严重的问题。

持续集成的优势：

- 当测试失败或者出现错误时，允许及时回滚代码
- 不断测试修复集成问题，避免临时抱佛脚
- 预警问题和冲突代码
- 所有的代码变化可以及时得到测试
- 及时向开发人员反馈他们编写代码的功能、质量和对系统的影响情况
- 频繁的代码检查促进开发人员写出模块化和简洁的代码
- 项目任何时候都保持在可运行状态

持续集成的劣势：

- 强大的测试套件和编写良好的测试代码才能发挥自动化测试的优势。
- 一定的硬件成本（小项目忽略不计）
- 初始部署的人力成本

持续集成的最佳实践：

- 维护一个版本库
- 自动构建
- 自动测试
- 及早提交代码
- 每次提交都被构建

- 保持快速构建
- 在模拟生产环境中测试
- 很容易获得最新的构建产物
- 每个人都可以看到构建结果
- 自动发布

## 1.2 CI 服务器介绍

主流的持续集成服务器有 CruiseControl 和 Hudson。我们会在不同项目中采纳它们。当前，主要使用到 CruiseControl。

本文档将主要围绕这一定制好的 CruiseControl 展开论述。

## 2 借助 CC 实施 CI

这一章我们将详细介绍利用 CruiseControl 实施持续集成的详细步骤，以及各种实用工具的集成方法。在实际动手操作之前，需要具备以下前提条件：

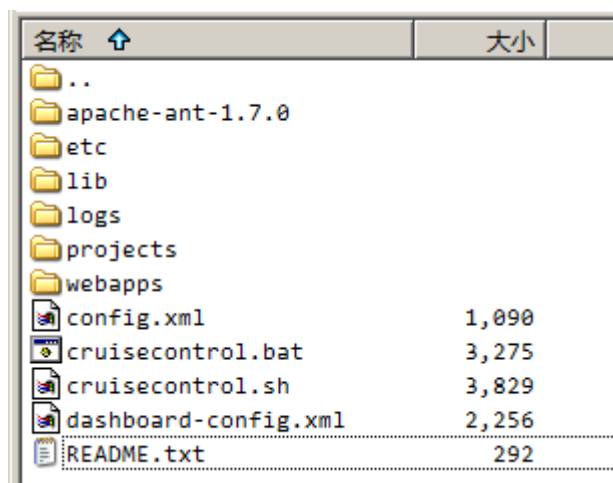
1. 安装 JRE（或者 JDK），并设置 JAVA\_HOME 和 Path 环境变量。
2. 安装 SVN 客户端，可以是 Subversion 命令行工具或者易用的 TortoiseSVN 图形化客户端，使用方法可以参考本系列文档，即《7.主流 SCM 工具的配置使用.pdf》。

### 2.1 CruiseControl 持续集成服务器

如果您只是对定制的 cruisecontrol-bin-2.8.3-cdev.zip 感兴趣，则可以跳过本节内容，而去阅读下节内容。

CruiseControl 是一个基于 Java、开源的持续集成框架，由 ThoughtWorks 公司出品。他提供了大量的插件以支持多样化的版本管理、构建工具以及大量的构建结果通知渠道。CruiseControl 只需要简单的模块化配置就可以实施。

其最新版本是 2.8.3，下载地址：<http://cruisecontrol.sourceforge.net/download.html>。下载之后解压到本地磁盘，文件结构如下：



CruiseControl 自带了 ant，projects 目录存放项目的源代码；logs 目录按项目名称存放构建过程产生的日志；webapps 存放了自带的 web 应用和文档；cruisecontrol.bat 和 cruisecontrol.sh 是启动文件，启动过程中使用的端口都在这个文件里配置；config 是我们部署 CruiseControl 过程中的核心文件，它包括了我们的全部配置信息。

## 2.1.1 内置的 connectfour 项目

官方的 CruiseControl 已经自带了一个演示项目 connectfour，代码位于 projects 目录下。首先打开一个命令终端，进入 CruiseControl 安装目录，输入 `cruisecontrol.bat` 启动（需要 Java 运行环境）：

```
D:\ci\cruisecontrol-bin-2.8.3>cruisecontrol.bat
"D:\develop\jrmc-3.1.2-1.5.0\bin\java" -Djavax.management.builder.initial=mx4j.
server.MX4JBeanServerBuilder "-Djetty.logs=logs" -jar "D:\ci\cruisecontrol-bin-
2.8.3\lib\cruisecontrol-launcher.jar" -jmxport 8000 -webport 8080 -rmiport 1099

WARNING: cc.home reset to D:\ci\cruisecontrol-bin-2.8.3
Classpath: D:\ci\cruisecontrol-bin-2.8.3\lib\cruisecontrol-launcher.jar;D:\ci\cr
uisecontrol-bin-2.8.3\lib\activation.jar;D:\ci\cruisecontrol-bin-2.8.3\lib\ant-l
auncher.jar;D:\ci\cruisecontrol-bin-2.8.3\lib\ant.jar;D:\ci\cruisecontrol-bin-2.
8.3\lib\commons.jar;D:\ci\cruisecontrol-bin-2.8.3\lib\commons-codesc=1.3.jar;D:\ci\c
```

如果出现端口冲突，请编辑 {CruiseControl 安装目录}\`cruisecontrol.bat` 修改对应的端口或者先停止其他端口冲突的程序。启动完成之后首先会启动第一次构建，并在当前目录下生成 `connectfour.ser` 文件，这个文件记录了 `cruisecontrol` 项目的相关历史构建信息。此外目录下还多出了一个 `artifacts` 文件夹，这个文件夹存放了项目构建过程中产生的应用包和报告等产出物。

打开浏览器，输入以下地址进行访问：<http://localhost:8080/cruisecontrol/>，在主页面上我们可以看到所有项目当前的构建状态。



点击项目名称，我们还可以看到更详细的信息。左侧是构建的历史信息，只要不删除日志我们可以看到所有的构建历史记录。

**历史信息**

**发布的构建结果**

**日志**

**单元测试**

**最新修改的代码**

**构建产出物**

在发布的构建结果页面，我们可以下载最新发布的应用包或者测试报告：

artifacts\connectfour\20100723142705

name	file size	modified date
<a href="#">connectfour.jar</a>	6.9K	Fri Jul 23 14:28:09 CST 2010

这是一个很简单的项目，构建过程只包括了编译、打包等项目中最基础的工作。通过简单的配置我们就可以让 CruiseControl 包揽更多的，也更有价值的工作。

## 2.2 启用定制的 CI 工具集

前面谈到，我们（cruisecontrol-bin-2.8.3-cdev.zip）对 CruiseControl 官方发布版进行了定制，使得它更适合于实施持续集成。接下来一一阐述重要细节。

### 2.2.1 配置新项目

假定有一新项目要在 CruiseControl 中配置，而且 SVN 版本库地址是：

<http://cdev-srv/svn/bimp/>。

## Revision 532

SVN ROOT



如上图所示，这个项目包括了一个构建脚本 `build.xml`，这个脚本可以在 ANT 下手动调用，执行包括编译、打包、生成 Javadoc、单元测试、静态代码检查等工作。这个文件是实施持续集成的前提，可根据需要在项目中进行完善。

首先，使用自己喜欢的方式将代码检出到本地，这里以命令行操作进行说明。比如，打开一个命令行终端，定位到 {CruiseControl 安装目录}\projects 位置，并输入以下命令检出项目。这里假定 `cruisecontrol-bin-2.8.3-cdev.zip` 解压到 `D:\ci\cruisecontrol-bin-2.8.3` 位置。

```
D:\ci\cruisecontrol-bin-2.8.3\projects>svn co http://cdev-srv/svn/bimp/
```

检查代码文件，确定是否检出成功。

打开 CruiseControl 目录下的 `config.xml` 文件，并配置好如下类似内容。

```
<cruisecontrol>

  <project name="bimp" buildafterfailed="false">
    <svnlabelincrementer
      workingcoppypath="projects/${project.name}"/>
    <listeners>
      <currentbuildstatuslistener
        file="logs/${project.name}/status.txt"/>
    </listeners>
  </project>
</cruisecontrol>
```



```

</listeners>
<bootstrappers>
    <svnbootstrapper
        localWorkingCopy="projects/${project.name}/"/>
</bootstrappers>
<modificationset quietperiod="1" ignoreFiles="">
    <svn LocalWorkingCopy="projects/${project.name}/"/>
</modificationset>
<schedule interval="3600">
    <ant anthome="apache-ant-1.7.0"
        buildfile="projects/${project.name}/build.xml"
        target="ci">
    <!--<ant antscript="ant-jdk14.bat"
        buildfile="projects/${project.name}/build.xml"
        target="ci">-->
    </ant>
</schedule>
<log>
    <merge dir="projects/${project.name}/cioutput"/>
    <gzip every="2" unit="WEEK"/>
    <deleteartifacts every="2" unit="WEEK"/>
</log>
<publishers>
    <onsuccess>
        <artifactspublisher
            dest="artifacts/${project.name}"
            file="projects/${project.name}/cioutput
                /bimp-src.zip"/>
        <artifactspublisher dest="artifacts/${project.name}"
            file="projects/${project.name}/cioutput
                /bimp.war"/>
        <artifactspublisher dest="artifacts/${project.name}"
            dir="projects/${project.name}/cioutput/apidoc"
            subdirectory="${project.name}-javadocs"/>
    </onsuccess>
</publishers>
</project>

</cruisecontrol>

```

我们可以注意到，文件的根节点是<cruisecontrol>，这个节点可以包括多个<project>节点，每个<project>节点配置一个项目。

以下对各个配置项详细说明：

- **svnlabelincrementer** 是可选的，以 **svn** 库的版本号标识每一次构建，形如 **svn.xxx**，如果不配置，则采用默认的形式 **build.xxx**，版本号从 1 开始随每次构建递增。
  - **listener**：指定状态文件的位置，CruiseControl 会向该文件写入和读取该文件的构建状态。
  - **bootstrappers**：在构建之前会优先执行 **bootstrappers**，以确定构建的条件是否满足。通常 CC 会检查上次构建以来是否有代码更新，如果没有就不启动构建。**bootstrapper** 可以定义多个，它们之间是相互独立的。
  - **svnbootstrapper**：获得 SVN 服务器的最新代码。CruiseControl 内置了主流 SCM 工具的支持。
  - **modificationset**：容器元素，它用于收集所有已注册 **SourceControl** 元素（<svn/>）的修改集合，比如上次成功构建到本次构建期间 SVN 服务器中所发生的代码变更。其中，**quietperiod** 属性表示在指定的时间内（单位是秒）无人提交代码，才会执行下一步。这一参数的目的是为了预防在更新过程中有人正在提交代码，从而导致更新了部分代码而更构建失败。
  - **svn**：指定通过 SVN 获得代码变更统计信息。
  - **schedule**：指定构建触发策略，可以使用 **interval** 指定两次构建的间隔时间，或者在子节点中通过 **time** 属性指定一个时间点。
  - **ant**：使用 **ant** 执行构建，**buildfile** 指定构建脚本文件，**target** 指定要执行的构建任务，**ant** 的构建脚本中一个 **target** 常常会依赖其他的 **target**。
  - **log**：指定对日志的处理方式，**meger** 将构建过程中产生的日志合并到主日志，默认将合并目标目录下所有 **.xml** 文件；**gzip** 将压缩旧的日志文件；**deleteartifacts** 会删除很久之前的存档文件。
  - **publishers**：在构建完成之后执行，指定将构建的结果，例如日志、报告或者源代码包，发布到结果文件夹、**ftp**、或者给指定的邮箱发邮件、甚至通过 **IM** 工具发布一条消息。**publishers** 允许按照构建结果成功或者失败指定不同的动作。
  - **onsuccess**：指定构建成功之后将 **artifactspublisher** 节点指定的报告或者代码包发布到构建结果文件夹，我们可以通过访问 CruiseControl 平台查看或者下载这些文件。
- config 文件的详细配置手册说明：<http://cruisecontrol.sourceforge.net/main/configxml.html>。

## 2.2.2 启动 CruiseControl

配置完成之后执行脚本 {CruiseControl 安装目录}\cruisecontrol.bat，启动 CruiseControl 后会首先执行第一次构建。对于有些复杂的构建任务来说可能要等上几分钟。

通过地址 <http://localhost:8888/cruisecontrol/> 访问 CruiseControl,在主页可以看到项目当前的构建状态。下图同时启用了两个不同项目。

Project	Status (since)	Last failure	Last successful	Label
<a href="#">bimp</a>	waiting ( 下午11:31)		下午11:10	svn.532 <a href="#">Build</a>
<a href="#">connectfour</a>	waiting ( 下午11:34)		下午11:33	build.2 <a href="#">Build</a>

等待第一次构建完成，点击项目名称进入详细构建信息展示页面。

### Status Page

[bimp](#)

waiting for next time to build since 2010-09-29T15:31:23 progress: 2010-09-29T15:31:23 next

### Latest Build

2010/09/29 23:10:11 (svn.532)
2010/09/29 15:52:25 (svn.520)
2010/09/29 09:58:17 (svn.502)
2010/09/29 00:36:44 (svn.494M.1)
2010/09/28 16:18:15 (svn.494M)
2010/09/28 15:15:17 (svn.493M.1)
2010/09/28 15:11:13 (svn.493M)
2010/09/28 15:04:29 (svn.492)

Build Results	Test Results	XML Log File	Metrics	Config
CheckStyle	PMD	FindBugs	EMMA Coverage	Javadoc
<b>BUILD COMPLETE - svn.532</b>				
Date of build:		2010-09-29T15:10:11		
Time to build:		2 minutes 28 seconds		
Last changed:		2010-09-29T12:32:22		
Last log entry:		测试C环境是否正常		
<a href="#">Build Artifacts</a>				
Checkstyle errors/warnings (0 / 0)				
PMD errors/warnings (0 / 0)				
FindBugs errors/warnings (4 / 4)				
com.longtop.bip.common.i18n.LanguageManager		111 Incorrect lazy initialization and update		
com.longtop.bip.common.smartfile.util.SmartUpload		368 Non-virtual method call passes null for		
com.longtop.bip.org.vo.BIPDevolveVO		259 Very confusing method names		
com.longtop.bip.org.vo.BIPExplicitreportpathVO		254 Very confusing method names		
<b>Errors/Warnings: (4)</b>				
注意：某些输入文件使用或覆盖了已过时的 API。				
注意：要了解详细信息，请使用 -Xlint:deprecation 重新编译。				
注意：某些输入文件使用了未经检查或不安全的操作。				
注意：要了解详细信息，请使用 -Xlint:unchecked 重新编译。				

在构建结果发布页面，我们可以直接下载应用包或浏览报告。

# artifacts\bimp\20100929231011

name	file size	modified date
<a href="#">bimp-src.zip</a>	42.3M	Wed Sep 29 23:12:52 CST 2010
<a href="#">bimp.war</a>	41.7M	Wed Sep 29 23:12:58 CST 2010
<a href="#">emma-coverage/</a>	0	Wed Sep 29 23:13:04 CST 2010
<a href="#">javadocs/</a>	0	Wed Sep 29 23:13:04 CST 2010

## 2.3 Ant build.xml 构建文件

通过上述过程，我们成功地实施了持续集成工作。为借助 `cruisecontrol-bin-2.8.3-cdev.zip` 实施 CI，相关人员需要准备好两方面的内容。其一，配置好 CruiseControl 要求的 `config.xml` 配置文件。其二，需要在项目中准备好相应的构建文件，比如 Ant `build.xml`。读者也可以基于其他构建技术提供构建文件，比如 Apache Maven。

此前，我们已经成功配置了 `config.xml`，而 Ant `build.xml` 需要接下来详细介绍。

### 2.3.1 Ant 构建文件概述

一个构建文件定义了一系列的 `target`，一个 `target` 就是一个任务或者动作，例如编译，打包，单元测试，生成 Javadoc 等任务通常都会定义一个 `target`。在这些 `target` 之间定义依赖关系，可以将所有要执行的任务串起来。一个简单的构建文件的结构通常如下：

```
<?xml version="1.0"?>
<project name="bimp" basedir="." default="src.zip">

  <!-- Local system paths -->
  <property name="build.webroot" location="${basedir}/WebContent"/>

  <!-- classpath -->
  <path id="runtime-lib">
    <fileset dir="${build.lib}" includes="*.jar"/>
    <fileset dir="lib" includes="*.jar"/>
  </path>

  <path id="src.dir" location="src"/>
  <path id="src.java">
```

```

        <fileset dir="src" includes="com/longtop/bip/**/*.java"/>
    </path>

    <target name="clean" description="Prepare for clean build">
        ...
    </target>

    <target name="build.srsc" depends="clean"
        description="Compile Src Java Files into Class Files">
        ...
    </target>

    <target name="dist.war" depends="build.srsc"
        description="package for deploy">
        ...
    </target>

    <target name="src.zip" depends="dist.war"
        description="package source file">
        ...
    </target>
    .
    .
    .
</project>

```

`<project>`为文件的根节点，定义项目的名称，运行脚本的基础目录，默认执行的构建任务等。

`<property>`定义一个属性，取值可以为一个字符串或者一个文件系统路径，可以通过 `${property_name}` 的方式引用，`property` 一旦定义无法改变。我们通常在文件的开始定义所有的 `property`。

`<path>`是非常常见的元素，通常配合 `fileset` 和 `pathelement` 定义一组目录或者文件的集合，通过 `id` 引用。常常用来定义 `classpath`、源文件、源文件路径。

`<target>`定义具体的任务，通过 `depends` 定义依赖的前提任务，执行脚本的时候 `Ant` 会根据依赖关系将所有要执行的任务进行排序，顺次执行。

### 2.3.2 配置 build.xml 中可复用的 `<property/>`、`<path/>` 等基础内容

从下节开始，我们将逐一揭开 `build.xml` 中可定义的各种 `Ant target`。在项目中稍加改造，读者就可以拿来使用。

这里首先给出全部会在下文 **target** 中引用到的 **property** 和 **path** 的定义，下文不再重复给出。

```
<!-- Local system paths -->
<property name="build.webroot" location="${basedir}/WebContent"/>
<property name="build.webinf" location="${build.webroot}/WEB-INF"/>
<property name="build.classes" location="${build.webinf}/classes"/>
<property name="build.lib" location="${build.webinf}/lib"/>
<property name="build.webinf.temp"
location="${build.webroot}/web-inf-temp"/>
<property name="test.classes" location="${basedir}/build/testclasses"/>
<property name="test.instr.dir" location="${basedir}/build/classes"/>
<!-- CI paths -->
<property name="ci.dir" value="${ant.home}/ci"/>
<property name="cioutput.dir" value="${basedir}/cioutput"/>
<property name="javadoc.dir" value="${cioutput.dir}/apidoc"/>
<property name="checkstyle.dir" value="${cioutput.dir}/checkstyle"/>
<property name="pmd.dir" value="${cioutput.dir}/pmd"/>
<property name="jdepend.dir" value="${cioutput.dir}/jdepend"/>
<property name="findbugs.dir" value="${cioutput.dir}/findbugs"/>
<property name="findbugs.home" value="${ci.dir}/findbugs"/>
<property name="junit.dir" value="${cioutput.dir}/junit"/>
<property name="junit.temp.dir" value="${junit.dir}/junit-temp"/>
<property name="emma.dir" location="${cioutput.dir}/emma"/>
<property name="emma.temp.dir" location="${emma.dir}/emma-temp"/>

<property name="emma.enabled" value="true"/>

<property name="bimp.war" value="${cioutput.dir}/bimp.war"/>
<property name="bimp.src" value="${cioutput.dir}/bimp-src.zip"/>
<property name="app.copyright" value="Copyright (c) 2010 LongTop. All
rights reserved."/>

<!-- javac.debug must set to true if import findbugs -->
<property name="javac.debug" value="true"/>

<!-- jdk.version take one of 1.4,1.5,1.6 -->
<condition property="jdk.version" value="${jdk.version.main}" else="1.5">
    <isset property="jdk.version.main"/>
</condition>

<!-- classpath -->

<path id="runtime-lib">
```

```
<fileset dir="${build.lib}" includes="*.jar"/>
<fileset dir="lib" includes="*.jar"/>
</path>
<path id="checkstyle-lib">
    <fileset dir="${ci.dir}/checkstyle/lib" includes="*.jar"/>
</path>
<path id="pmd-lib">
    <fileset dir="${ci.dir}/pmd/lib" includes="*.jar"/>
    <pathelement location="${ci.dir}/pmd/rules"/>
</path>
<path id="emma-lib" >
    <pathelement location="${ci.dir}/emma/lib/emma.jar"/>
    <pathelement location="${ci.dir}/emma/lib/emma_ant.jar"/>
</path>

<path id="src.dir" location="src"/>
<path id="src.java">
    <fileset dir="src" includes="com/longtop/bip/**/*.java"/>
</path>
<path id="src.conf">
    <fileset dir="src">
        <include name="**/*"/>
        <exclude name="**/*.java"/>
    </fileset>
</path>

<path id="test.dir" location="test"/>
<path id="test.conf">
    <fileset dir="test">
        <include name="**/*"/>
        <exclude name="**/*.java"/>
    </fileset>
</path>
```

其中：

- `${ci.dir}`是定制的内容，位于 CruiseControl 自带的 Ant 目录下，只要通过这个 Ant 执行脚本，就可以定位到这个目录，这里存放了我们定制的工具使用到的规则文件和 jar 包。
- `${cioutput.dir}`是所有构建产出物的存放目录。
- `<condition>`可以根据 `jdk.version.main` 的值设置 `jdk.version` 从而在不同版本的 jdk 下运行脚本。

### 2.3.3 编译 target

编译是 Java 项目最为常见的 target，主要是通过 Ant 内置的 javac 任务搜索源目录下所有的 java 文件编译存放到目标文件夹。

依赖的 clean target 是为了将之前编译的文件和构建结果文件删除，以重新开始构建。

debug 参数很重要，在某些依赖于编译结果的任务中需要将 debug 设置为 true，例如如果 FindBugs 任务必须在开启 debug 的时候才能显示问题所在的行号。

需要注意的是不要忘了将源代码目录下除了 java 之外的配置文件拷贝到目标目录。

```
<target name="clean" description="Prepare for clean build">
    <delete dir="${cioutput.dir}"/>
    <delete dir="${build.classes}"/>
    <echo message="java version is ${jdk.version}"/>
</target>

<target name="build.srcs" depends="clean"
    description="Compile Src Java Files into Class Files">
    <mkdir dir="${build.classes}"/>
    <javac destdir="${build.classes}" source="${jdk.version}"
        target="${jdk.version}" deprecation="false" optimize="false"
        debug="${javac.debug}" failonerror="true" memorymaximumsize="512m"
        fork="true">
        <classpath refid="runtime-lib"/>
        <src refid="src.dir"></src>
    </javac>
    <copy todir="${build.classes}" preservelastmodified="true"
        failonerror="false">
        <path refid="src.conf"></path>
    </copy>
</target>
```

### 2.3.4 打包 target

在应用部署之前，我们通常会将应用打包进行分发，有时候还会针对不同的操作系统、应用服务器或者数据库使用不同的配置打成各种不同的包。这里给出了 .war 包和源代码的打包脚本样例。

其中注释掉的一行脚本 `<!--<antcall target="conf.for.product"/>-->` 是为了调用一个 target，这个 target 就是为了满足不同的环境而调整配置文件，从而提供有针对性的应用包。因此在此之前我们将 WEB-INF 目录拷贝了一份，如果要打多个包，还可以拷贝多份分别进行调整。



而源代码包就比较简单了，直接使用 `zip` 指令压缩项目根目录，同时过滤掉编译的结果文件和构建产出文件，这些产出文件都存放在 `${cioutput.dir}` 目录下。

```
<target name="dist.war" depends="build.srcs">
  <mkdir dir="${cioutput.dir}"/>
  <delete dir="${build.webinf.temp}"/>
  <copy todir="${build.webinf.temp}"
        description="copy WEB-INF to a temp folder">
    <fileset dir="${build.webinf}" excludes="lib/**,classes/**"/>
  </copy>
  <!--<antcall target="conf.for.product"/>-->
  <delete file="${bimp.war}"/>
  <war webxml="${build.webinf.temp}/web.xml" destfile="${bimp.war}">
    <fileset dir="${build.webroot}"
            excludes="WEB-INF/**,web-inf-temp/**"/>
    <lib dir="${build.lib}"/>
    <webinf dir="${build.webinf.temp}"/>
    <classes dir="${build.classes}"/>
  </war>
  <delete dir="${build.webinf.temp}"/>
</target>
<target name="src.zip" depends="dist.war">
  <delete file="${bimp.src}"/>
  <zip destfile="${bimp.src}">
    <fileset dir="." includes="*"
            excludes="cioutput/**,WebContent/WEB-INF/classes/**,build/**"/>
  </zip>
</target>
```

### 2.3.5 生成 Javadoc

一个好的项目除了有编写良好的代码之外，还需要有清晰的文档对代码进行说明。而 Javadoc 是其中很重要的一部分，我们经常把 Javadoc 直接作为详细设计的提交物。

```
<target name="javadoc" description="--&gt; creates the API documentation">
  <delete dir="${javadoc.dir}"/>
  <mkdir dir="${javadoc.dir}"/>
  <javadoc packagenames="com.longtop.bip.*" source="${jdk.version}"
        sourcepathref="src.dir" defaultexcludes="yes"
        destdir="${javadoc.dir}"
        author="true" version="true" use="true" windowtitle="BIMP API"
        access="private" useexternalfile="yes" maxmemory="512m">
    <doctitle><![CDATA[<h1>BIMP API</h1>]]></doctitle>
```

```

<bottom><![CDATA[<i>${app.copyright}</i>]]></bottom>
<classpath refid="runtime-lib"/>
</javadoc>
</target>

```

CI 的构建结果 Tab 页中会将各种 Javadoc 问题暴露出来，见下图所示。

#### Errors/Warnings: (4)

注意：某些输入文件使用或覆盖了已过时的 API。

注意：要了解详细信息，请使用 `-Xlint:deprecation` 重新编译。

注意：某些输入文件使用了未经检查或不安全的操作。

注意：要了解详细信息，请使用 `-Xlint:unchecked` 重新编译。

#### Javadoc Errors/Warnings: (0)

或者，通过专门的 Javadoc Tab 页也可以了解到当前项目存在的各种问题。

Build Results	Test Results	XML Log File	Metrics	Config	Control Panel
CheckStyle	PMD	FindBugs	EMMA Coverage	Javadoc	

#### Javadoc Errors/Warnings: (0)

## 2.3.6 借助 CheckStyle 和 PMD 审计代码质量

CheckStyle 和 PMD 都是不错的源代码静态检查工具，规则灵活，支持自定义规则集合。

CheckStyle 偏重于代码风格的检查，而 PMD 则更多地对代码的语义进行检查。

两个 target 定义摘录如下。

```

<path id="checkstyle-lib">
  <fileset dir="${ci.dir}/checkstyle/lib" includes="*.jar"/>
</path>
<path id="pmd-lib">
  <fileset dir="${ci.dir}/pmd/lib" includes="*.jar"/>
  <pathelement location="${ci.dir}/pmd/rules"/>
</path>

<target name="checkstyle"
  description="Generates a report of code convention violations.">

```

```
<taskdef resource="checkstyletask.properties"
  classpathref="checkstyle-lib"/>
<delete dir="${checkstyle.dir}"/>
<mkdir dir="${checkstyle.dir}"/>
<checkstyle config="${ci.dir}/checkstyle/rules/bi_checks.xml"
  failureProperty="checkstyle.failure" failOnViolation="false">
  <formatter type="xml"
    tofile="${checkstyle.dir}/checkstyle-report.xml"/>
  <fileset dir="src" includes="com/longtop/bip/**/*.java"/>
</checkstyle>
<xslt in="${checkstyle.dir}/checkstyle-report.xml"
  style="${ci.dir}/checkstyle/xslt/checkstyle-noframes.xsl"
  out="${checkstyle.dir}/checkstyle-report.html"/>
</target>

<target name="pmd">
  <taskdef name="pmd" classname="net.sourceforge.pmd.ant.PMDTask"
    classpathref="pmd-lib"/>
  <delete dir="${pmd.dir}"/>
  <mkdir dir="${pmd.dir}"/>
  <pmd targetjdk="${jdk.version}"
    rulesetfiles="codesize.xml,clone.xml,basic.xml,logging-java.xml,
      logging-jakarta-commons.xml,braces.xml">
    <formatter type="xml" toFile="${pmd.dir}/pmd-report.xml"
      toConsole="true"/>
    <fileset dir="src" includes="com/longtop/bip/**/*.java"/>
  </pmd>
  <xslt in="${pmd.dir}/pmd-report.xml"
    style="${ci.dir}/pmd/xslt/pmd-report-per-class.xslt"
    out="${pmd.dir}/pmd-report.html"/>
</target>
```

同样地，我们能够透过 CI 服务器的对应 Tab 页了解到这些 target 反馈的代码问题。

其中，CheckStyle 的反馈结果示例如下。

Build Results	Test Results	XML Log File	Metrics	Config	Control Panel
CheckStyle	PMD				

## Checkstyle Summary

Files: 68

Errors: 346

Warnings: 346

CheckStyle violation	Files	Error/Warnings
com.puppycrawl.tools.checkstyle.checks.imports.UnusedImportsCheck	67	343
com.puppycrawl.tools.checkstyle.checks.imports.RedundantImportCheck	1	1
com.puppycrawl.tools.checkstyle.checks.imports.AvoidStarImportCheck	1	1
com.puppycrawl.tools.checkstyle.checks.imports.IllegalImportCheck	1	1

(24 / 24)

3 Unused import - java.util.ArrayList.

4 Unused import - java.util.Date.

而 PMD 的反馈结果示例如下。

Build Results	Test Results	XML Log File	Metrics	Config	Control Panel
CheckStyle	PMD				

## PMD Summary

Files: 144

Violations: 692

PMD rule	Files	Error/Warnings
Braces Rules / IfStmtsMustUseBraces	83	405
Java Logging Rules / AvoidPrintStackTrace	38	121
Java Logging Rules / SystemPrintln	20	55
Braces Rules / IfElseStmtsMustUseBraces	12	33
Java Logging Rules / LoggerIsNotStaticFinal	28	28
Basic Rules / EmptyCatchBlock	10	20
Basic Rules / EmptyIfStmt	9	13
Basic Rules / EmptyStatementNotInLoop	7	12
Code Size Rules / TooManyFields	2	2
Basic Rules / UselessOverridingMethod	1	1
Basic Rules / ClassCastExceptionWithToArray	1	1
Basic Rules / UnnecessaryConversionTemporary	1	1

(35)

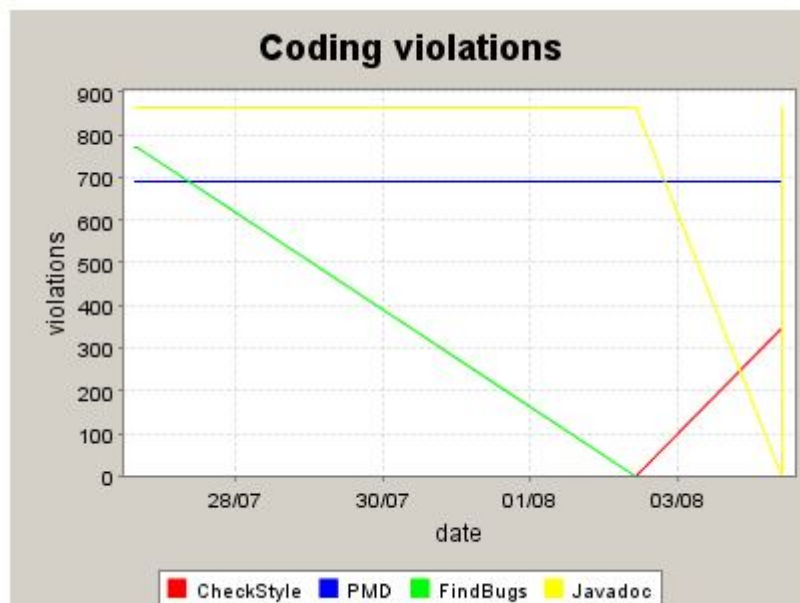
Avoid using if statements without curly braces

3

Avoid using if statements without curly braces

3

切换到 Metrics 面板, 在 Coding violations 统计报表中可以看到 CheckStyle 和 PMD 的统计信息。



### 2.3.7 借助 FindBugs 审计代码质量

FindBugs 与 CheckStyle 和 PMD 检查源代码不同, FindBugs 是对编译之后的字节码进行检查。对应的 `target` 定义如下。

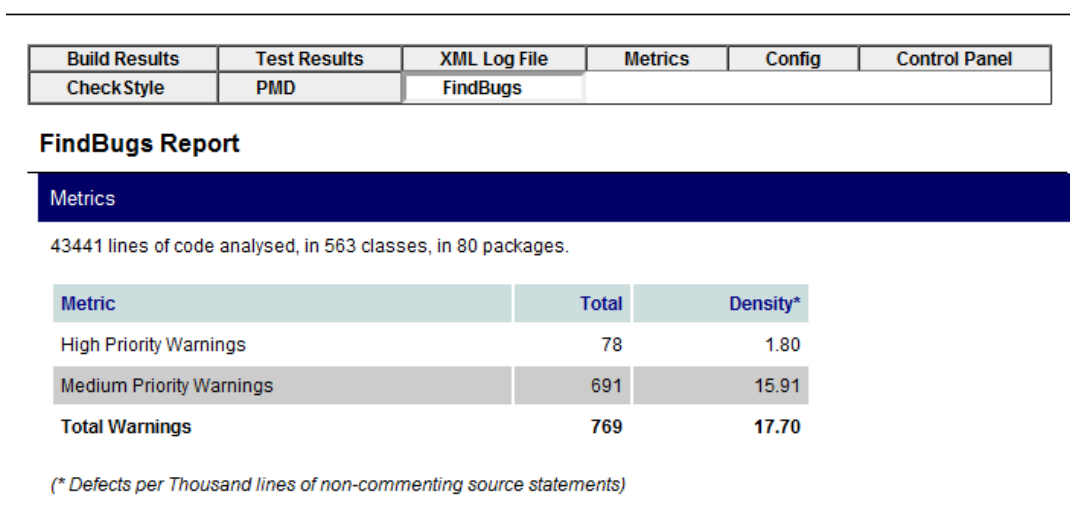
```
<target name="findbugs" depends="build.srcs">
  <delete dir="${findbugs.dir}"/>
  <mkdir dir="${findbugs.dir}"/>
  <taskdef name="findbugs"
    classname="edu.umd.cs.findbugs.anttask.FindBugsTask"
    classpath="${findbugs.home}/lib/findbugs-ant.jar"/>
  <findbugs home="${findbugs.home}" output="xml:withMessages"
    jvmargs="-Xmx256M" reportLevel="high" debug="false"
    effort="max" outputFile="${findbugs.dir}/findbugs-report.xml">
    <class location="${build.classes}"/>
    <sourcePath refid="src.dir"/>
    <auxClasspath path="${build.lib}"/>
  </findbugs>
  <xslt in="${findbugs.dir}/findbugs-report.xml"
    style="${findbugs.home}/xslt/default.xsl" force="true"
    out="${findbugs.dir}/findbugs-report.html"/>
</target>
```

注意, `findbugs` 的 `reportLevel` 可以控制输出的问题的优先级, 可取值“low”, “medium”或者“high”, 其中 low 发现的问题最多, 而 high 发现的问题最少, 往往优先级也更高。

需要注意的是 `buid.srcs` 任务中 `javac` 的 `debug` 属性必须设置为 `true`, 否则报告中将无法

显示出现的问题在源代码中的行数。

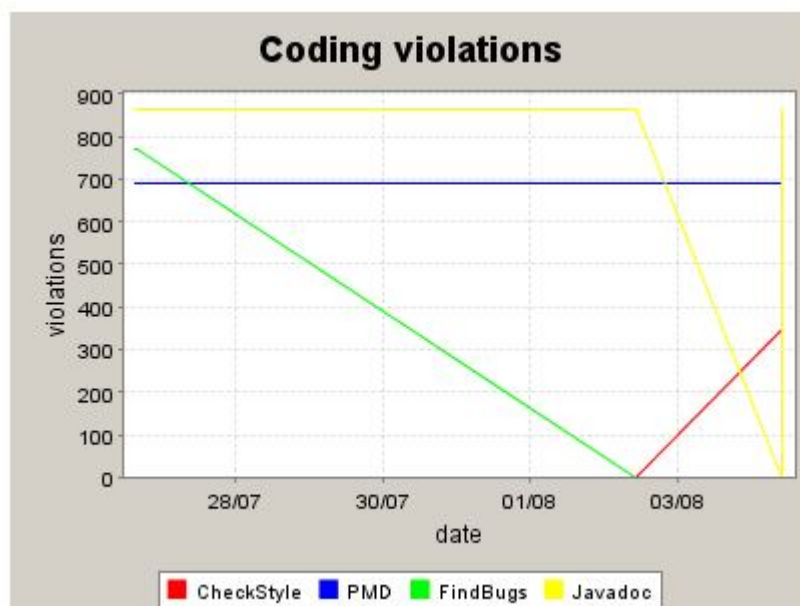
在 CI 构建结果页面可以看到 FindBugs Tab 页面，切换到该面板，可以看到详细的报告信息。



CI 的构建结果 Tab 页中也可以看到 FindBugs 的结果：

FindBugs errors/warnings (769 / 769)	
com.fenet.cdsJc.common.PermissionDef	May expose internal representatic
com.fenet.cdsJc.common.PermissionDef	May expose internal representatic
com.fenet.cdsJc.common.PermissionDef	May expose internal representatic
com.fenet.cdsJc.common.PermissionDef	May expose internal representatic
com.fenet.cdsJc.common.PermissionDef	May expose internal representatic
com.fenet.cdsJc.css.CSSApp	Exception is caught when Excepti
com.fenet.cdsJc.dao.BrioSessionFactory	Incorrect lazy initialization of static
com.fenet.cdsJc.dao.SessionPool	Inconsistent synchronization
com.fenet.cdsJc.dao.SessionPool	Incorrect lazy initialization of static
com.fenet.cdsJc.servlet.InstanceSessionPool	Unread field

切换到 Metrics 面板，Coding violations 中增加了 FindBugs 的统计信息：



## 2.3.8 运行测试代码及评估测试代码覆盖率

我们借助 JUnit 执行代码级的单元及集成测试，并由 EMMA 进行测试覆盖度的评估。

相关 target 定义如下。

```
<target name="build.tests" depends="build.srsc">
    description="Compile Test Java Files into Class Files">
    <mkdir dir="${test.classes}" />
    <javac destdir="${test.classes}" source="${jdk.version}"
        target="${jdk.version}" deprecation="false" optimize="false"
        debug="${javac.debug}" failonerror="true" memorymaximumsize="512m"
        fork="true">
        <classpath location="${build.classes}" />
        <classpath refid="runtime-lib" />
        <src refid="test.dir"></src>
    </javac>
    <copy todir="${test.classes}" preservelastmodified="true">
        <path refid="test.conf"></path>
    </copy>
</target>

<target name="emma.instrument">
    <taskdef resource="emma_ant.properties" classpathref="emma-lib"/>
    <delete dir="${test.instr.dir}"/>
    <delete dir="${emma.dir}"/>
    <mkdir dir="${test.instr.dir}"/>
    <mkdir dir="${emma.temp.dir}"/>
    <emma enabled="${emma.enabled}">
        <instr instrpath="${build.classes}" destdir="${test.instr.dir}"
            mode="copy"
            verbosity="verbose"
            metadatafile="${emma.temp.dir}/metadata.emma" merge="true"/>
    </emma>
</target>

<target name="run.test" depends="build.tests,emma.instrument">
    <mkdir dir="${junit.temp.dir}"/>
    <junit printsummary="off" fork="true" forkmode="perTest"
        outputtoformatters="true" showoutput="false" clonevm="true">
        <jvmarg value=
            "-Demma.coverage.out.file=${emma.temp.dir}/coverage.emma"/>
        <jvmarg value="-Demma.coverage.out.merge=true"/>
    </classpath>
```

```

        <pathelement location="${test.classes}"/>
        <pathelement location="${test.instr.dir}"/>
        <pathelement location="${build.classes}"/>
        <path refid="emma-lib"/>
        <path refid="runtime-lib"/>
    </classpath>
    <batchtest todir="${junit.temp.dir}">
        <formatter type="xml"/>
        <fileset dir="${test.classes}">
            <include name="com/longtop/bip/**/*Test.class"/>
        </fileset>
    </batchtest>
</junit>
<junitreport todir="${junit.dir}">
    <fileset dir="${junit.temp.dir}">
        <include name="TEST-*.xml"/>
    </fileset>
    <report format="noframes" todir="${junit.dir}"/>
</junitreport>
<delete dir="${junit.temp.dir}"/>
<emma enabled="${emma.enabled}">
    <report sourcepathref="src.dir" sort="+block,+name,+method,+class"
        metrics="method:70,block:80,line:80,class:100">
        <fileset dir="${emma.temp.dir}" includes="*.emma"/>
        <html outfile="${emma.dir}/index.html" depth="method"
            columns="name,class,method,block,line"/>
        <xml outfile="${emma.dir}/emma-report.xml" depth="method"/>
    </report>
</emma>
<delete dir="${emma.temp.dir}"/>
</target>

```

这里包括了三个 **target**: **build.tests** 编译测试代码; **emma.instrument** 处理字节码文件, 插入生成覆盖度报告必须的数据信息, 同时生成元数据文件 **metadata.emma**; **run.test** 使用 JUnit 执行测试代码, 同时生成覆盖度的评估报告。

为了生成覆盖度报告, **fork** 必须设为 **true**。

另外, **emma.enabled** 属性设置为 **false** 时将不会生成覆盖度报告。**classpath** 的前三个元素必须要注意他们之间的顺序, 不能颠倒。首先是测试类的 **class** 文件路径, 其次是经过 EMMA 处理的 **class** 文件路径, 最后是编译的源代码的文件路径。



CruiseControl 存在一 EMMA Coverage Tab 页，点击之后展示结果如下图所示。

EMMA Coverage Report				
OVERALL COVERAGE SUMMARY				
package	class, %	method, %	block, %	line, %
all classes	2% (5/282)	1% (16/2826)	0% (357/76135)	0% (55.8/18890)
OVERALL STATS SUMMARY				
total packages:	60			
total executable files:	266			
total classes:	282			
total methods:	2826			
total executable lines:	18890			
COVERAGE BREAKDOWN BY PACKAGE				
package	class, %	method, %	block, %	line, %
<a href="#">com.longtop.bip.auth</a>	0% (0/3)	0% (0/20)	0% (0/120)	0% (0/40)
No. source file	class, %	method, %	block, %	line, %
1 <a href="#">AuthServiceFactory.java</a>	0% (0/1)	0% (0/8)	0% (0/28)	0% (0/11)
2 <a href="#">BipAuthenticationFailureHandler.java</a>	0% (0/1)	0% (0/6)	0% (0/49)	0% (0/15)
3 <a href="#">BipAuthenticationSuccessHandler.java</a>	0% (0/1)	0% (0/6)	0% (0/43)	0% (0/14)
<a href="#">com.longtop.bip.auth.exception</a>	0% (0/1)	0% (0/1)	0% (0/4)	0% (0/2)
<a href="#">com.longtop.bip.auth.service.impl</a>	0% (0/17)	0% (0/136)	0% (0/4599)	0% (0/1045)
<a href="#">com.longtop.bip.auth.util</a>	0% (0/5)	0% (0/15)	0% (0/206)	0% (0/60)
<a href="#">com.longtop.bip.auth.vo</a>	0% (0/8)	0% (0/107)	0% (0/398)	0% (0/166)
<a href="#">com.longtop.bip.auth.web</a>	0% (0/33)	0% (0/207)	0% (0/11688)	0% (0/2912)
<a href="#">com.longtop.bip.base.common</a>	0% (0/1)	0% (0/2)	0% (0/52)	0% (0/25)
<a href="#">com.longtop.bip.base.common.hib</a>	0% (0/2)	0% (0/22)	0% (0/203)	0% (0/59)

### 2.3.9 使用 Ivy 进行 jar 包管理

Ivy 是 Apache 基金会推出的非常简洁和灵活的依赖管理工具。

如果在项目中采用了 Ivy，ANT 构建脚本会有些变化，在编译代码之前我们必须先解决项目或者模块依赖的 jar 包。

项目或者模块的 ivy.xml 文件配置如下，这个文件描述了当前项目或者模块依赖的所有 jar 包：

```
<ivy-module version="1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd"
>
    <info organisation="cdev" module="com.longtop.bi.bimp.common"/>
    <configurations>
```

```

    <include file="../../com.longtop.bi/default-ivy-configurations.xml"/>
</configurations>

<dependencies>
    <dependency org="apache-commons" name="commons-logging" rev="1.1.1"
transitive="false"/>
    <dependency org="ehcache" name="ehcache" rev="1.6.2"
transitive="false"/>
    <dependency org="intelliweb" name="intelliweb" rev="2.1"
transitive="false"/>
    <dependency org="intelliweb" name="intelligence-license" rev="2.1"
transitive="false"/>
    <dependency org="springframework" name="org.springframework.aop"
rev="3.0.5.RELEASE" transitive="false"/>
    <dependency org="springframework" name="org.springframework.asm"
rev="3.0.5.RELEASE" transitive="false"/>
    <dependency org="springframework" name="org.springframework.beans"
rev="3.0.5.RELEASE" transitive="false"/>
    <dependency org="springframework"
name="org.springframework.context" rev="3.0.5.RELEASE"
transitive="false"/>
    <dependency org="springframework"
name="org.springframework.context.support" rev="3.0.5.RELEASE"
transitive="false"/>
    <dependency org="springframework" name="org.springframework.core"
rev="3.0.5.RELEASE" transitive="false"/>
    <dependency org="springframework"
name="org.springframework.expression" rev="3.0.5.RELEASE"
transitive="false"/>
    <dependency org="springframework" name="org.springframework.jdbc"
rev="3.0.5.RELEASE" transitive="false"/>
    <dependency org="springframework" name="org.springframework.orm"
rev="3.0.5.RELEASE" transitive="false"/>
    <dependency org="springframework"
name="org.springframework.transaction" rev="3.0.5.RELEASE"
transitive="false"/>
</dependencies>
</ivy-module>

```

相关 target 定义如下：

```

.....
<path id="ivy-lib">
    <fileset dir="${ci.dir}/ivy/lib" includes="*.jar"/>

```

```

</path>
<target name="init">
    <taskdef name="ivy-configure"
classname="org.apache.ivy.ant.IvyConfigure" classpathref="ivy-lib"/>
    <taskdef name="ivy-cleancache"
classname="org.apache.ivy.ant.IvyCleanCache" classpathref="ivy-lib"/>
    <taskdef name="ivy-retrieve"
classname="org.apache.ivy.ant.IvyRetrieve" classpathref="ivy-lib"/>
    <property name="ivy.pattern"
value="\${build.lib}/[artifact]-[revision].[ext]"/>
    <ivy-configure file="\${basemodule.root}/ivysettings-ci.xml"/>
    <ivy-cleancache/>
</target>
<target name="build.module" depends="clean,init">
    <antcall target="template.module">
        <param name="module.name" value="bimp.util"/>
    </antcall>
    <antcall target="template.module">
        <param name="module.name" value="bimp.infrastructure"/>
    </antcall>
    .....
</target>
<target name="template.module">
    <property name="module.root"
value="\${basedir}/../com.longtop.bi.\${module.name}"/>
    <antcall target="template.module.ivy"/>
    <antcall target="template.module.build.srcs"/>
    <antcall target="template.module.jar"/>
    <antcall target="template.module.build.tests"/>
    <antcall target="template.module.run.tests"/>
</target>
<target name="template.module.ivy">
    <ivy-retrieve pattern="\${ivy.pattern}" file="\${module.root}/ivy.xml"/>
</target>
.....

```

这个是模块化之后的 BIMP 构建文件, BIMP 被分拆成为了几个功能上相互独立的模块, 因此构建文件也采用了函数式调用的方式分别构建每一个模块, 以 `template.`开头的是模板 `target`, 供其他 `target` 进行调用。



init target 首先定义了下文会用到的 ivy task 以及 ivy 配置文件，属性 ivy.pattern 定义了 jar 包存放的路径和格式。build.module 将模块的名称传递给模板 target template.module，template.module 分别调用各个模板 target 完成模块的 jar 包依赖、编译、测试和打包等任务。其中 template.module.ivy 会根据 ivy.xml 文件的定义从仓库中将依赖的 jar 按 ivy.pattern 定义的格式存放到目标路径，解决 jar 包依赖的问题。

## 2.4 在 CC 中集成 SVN

CruiseControl 与 SCM 的集成包括几个层面：

- CC 从 SCM 更新代码
- CC 从 SCM 获取版本号作为构建版本
- CC 操作 SCM，例如打标签、查询更新列表

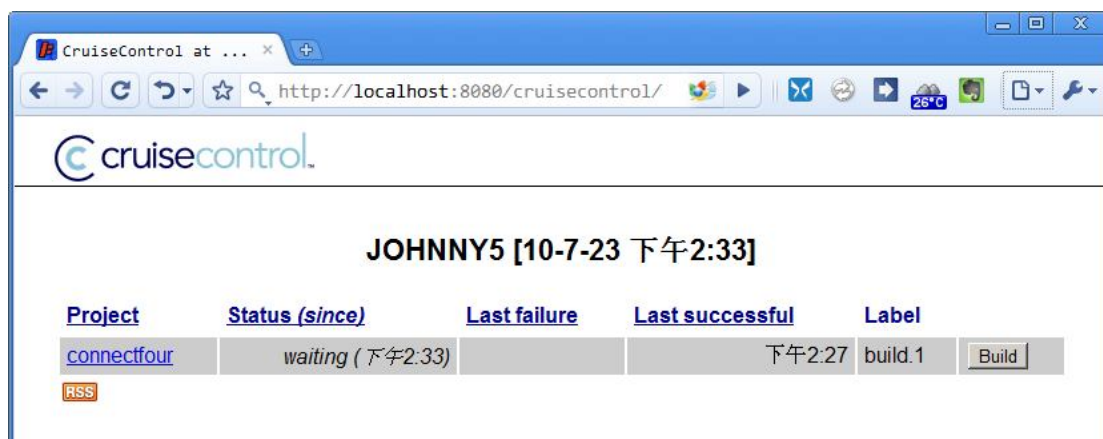
对于多人协作的团队开发来说，都会维护 SCM，作为 CC 的代码更新服务器，集成 SVN 做如下配置：

```
<project>
  .
  .
  <bootstrappers>
    <svnbootstrapper localWorkingCopy="projects/${project.name}"/>
  </bootstrappers>

  <modificationset quietperiod="1" ignoreFiles="">
    <svn LocalWorkingCopy="projects/${project.name}"/>
  </modificationset>
  .
  .
```

```
</project>
```

CC 默认使用类似 build.1 的形式来标识每一次构建，随着不断的构建，序号递增。



除了使用递增的序号进行标识外，我们还可以使用 SCM 的版本号作为本次构建的 Label，每一次构建就与源代码的版本做了关联，方便版本的管理。

使用 SVN 版本库的版本号作为构建 Label，做如下配置：

```
<project name="cds-gddb" buildafterfailed="false">
  <svnlabelincrementer workingcoppypath="projects/${project.name}"/>
  .
  .
</project>
```

CC 除了可以使用 SVN 的版本号作为 Label 外，还可以使用<cvslabelincrementer>、<p4changelistlabelincrementer>等作为 Label 的来源。CC 针对不同 SCM 工具提供了不同程度的支持。

详细的配置说明请参考：<http://cruisecontrol.sourceforge.net/main/configxml.html>。

## 2.5 增加 CC 访问控制

CruiseControl 服务器默认情况下任何人只要知道访问地址，都可以自由访问，查看报告，下载构建结果，这在项目组内部能够极大地提高工作的效率，方便交流，但是在这也带来了一些安全上的问题，比方说项目的源代码我们并不想让项目之外的人下载，至少不是任何人都可以下载。

CC 内置了 Jetty 作为 web 服务器，我们这里以项目构建结果页面的访问保护来说明 Jetty

的访问控制策略。

首先修改应用程序部署描述符 `web.xml`，加入访问控制的内容。

找到文件 {CruiseControl 安装目录}\webapps\cruisecontrol\WEB-INF\web.xml，增加以下内容：

```
<web-app>
.....
  <security-constraint>
    <web-resource-collection>
      <url-pattern>/artifacts/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>admin</role-name>
      <role-name>user</role-name>
    </auth-constraint>
    <user-data-constraint>
      <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Artifacts</realm-name>
  </login-config>
</web-app>
```

- `<web-resource-collection>` 标识需要限制访问的资源子集，包括两个子元素 `<url-pattern>` 和 `<http-method>`。
- `<url-pattern>` 匹配需要限制访问的 URL，这里表示访问 `artifacts` 路径下的文件需要进行认证。
- `<http-method>` 表示需要进行访问控制的 HTTP 方法，不配置的话表示限制所有的 HTTP 方法。
- `<auth-constraint>` 表示允许访问 `<web-resource-collection>` 元素标识的资源子集的用户的角色。
- `<login-config>` 配置认证信息。
- `<auth-method>` 表示认证的方式，这里采用 BASIC 基本认证。
- `<realm-name>` 这里表示用于进行认证的 realm 的名称。

配置 Jetty realm 提供认证。

找到文件{CruiseControl 安装目录}\jetty.xml，修改其中的 UserRealms 部分。注意修改红色标识的“Artifacts”为上一步应用中配置的 realm 的名称，/etc/realm.properties 为用户和角色配置文件，可以采用默认的，也可以修改为自己的文件。

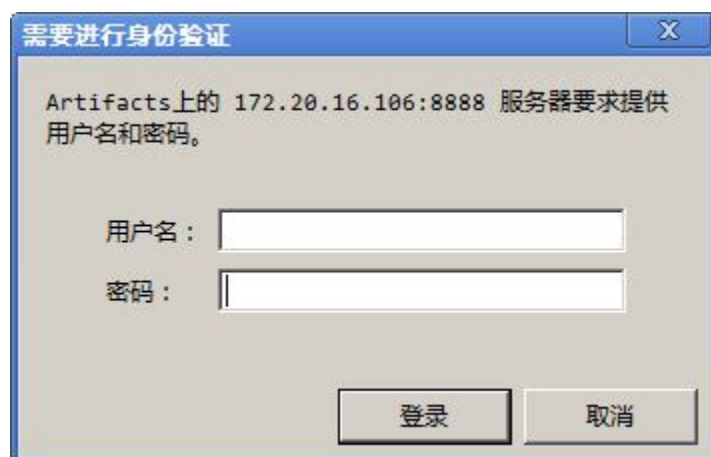
```
<Set name="UserRealms">
  <Array type="org.mortbay.jetty.security.UserRealm">
    <Item>
      <New class="org.mortbay.jetty.security.HashUserRealm">
        <Set name="name">Artifacts</Set>
        <Set name="config"><SystemProperty name="jetty.home"
default="."/>/etc/realm.properties</Set>
        <Set name="refreshInterval">0</Set>
      </New>
    </Item>
  </Array>
</Set>
```

然后配置用户和角色文件，找到{CruiseControl 安装目录}\etc\realm.properties，以#开头注释掉文件原来的内容，然后增加你的用户和角色信息

```
jetty: MD5:164c88b302622e17050af52c89945d44,user
user;password,user
.....
```

注意每行代表一个文件，格式为“用户名:密码,角色名”。其中密码可以是明文，也可以是加密。需要注意只有角色与应用中<auth-constraint>指定的角色相同的用户才能够访问受保护的资源。

配置完成之后，重新启动 CruiseControl，再次点击“Build Artifacts”链接，会弹出提示框要求输入用户名和密码：



我们知道 CruiseControl 自带了三个应用和一份文档，都位于 `webapps` 目录下，这里我们只对 `cruisecontrol` 设置了访问保护，而另一个应用 `dashboard` 还是可以自由访问构建结果。可以参照以上步骤为 `dashboard` 应用设置访问控制，或者直接删除 `dashboard` 应用（如果没有使用的话）。



## 3 使用 Hudson 实施 CI

### 3.1 Hudson 概述

最新的版本是 1.386，下载地址：<http://hudson-ci.org/>

hudson 只有一个 war 包，可以直接在命令行下执行：

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

但是需要指定 JAVA\_HOME 和 HUDSON\_HOME 环境变量，为了运行方便，建立一个批处理文件 hudson.bat，输入以下内容：

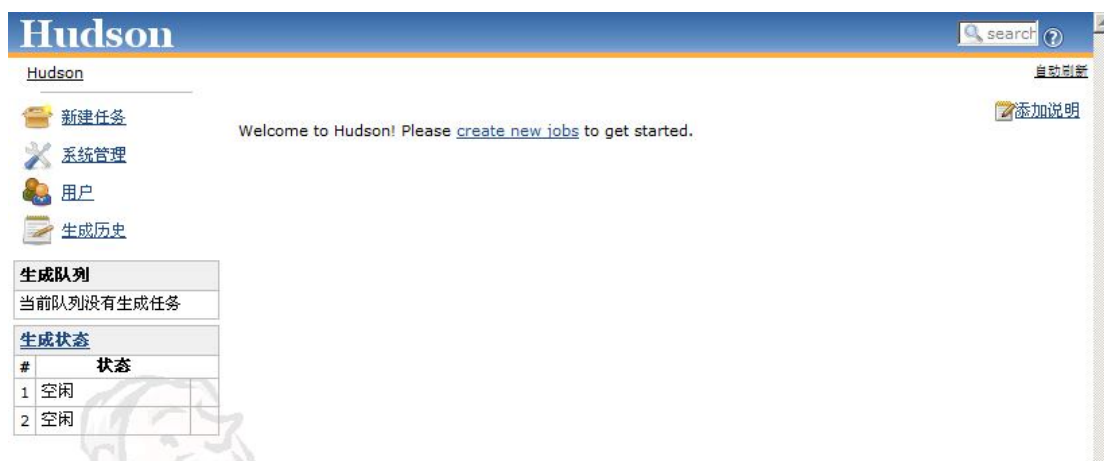
```
@echo off
set JAVA_HOME=C:\java
set ANT_HOME=D:\ci\hudson\apache-ant-1.7.0\
set HUDSON_HOME=D:\ci\hudson
cd /d D:\ci
%JAVA_HOME%\bin\java.exe -jar D:\ci\hudson.war
```

以上环境变量的路径请根据实际情况进行调整。

启动一个命令行终端，输入 hudson.bat 启动 hudson，会在 HUDSON\_HOME 指定的目录下初始化一些文件，这些文件包括 web 应用程序、插件、以及用来存放项目源代码和构建结果的目录：

名称	大小	类型
jobs		文件夹
plugins		文件夹
userContent		文件夹
war		文件夹
hudson.model.UpdateCenter.xml	1 KB	XML 文档
nodeMonitors.xml	1 KB	XML 文档
secret.key	1 KB	KEY 文件

打开 web 主页 <http://localhost:8080>，可以看到还没有配置任何项目。



## 3.2 配置一个新项目

执行 `hudson.bat` 启动 Hudson，通过浏览器访问应用主页：



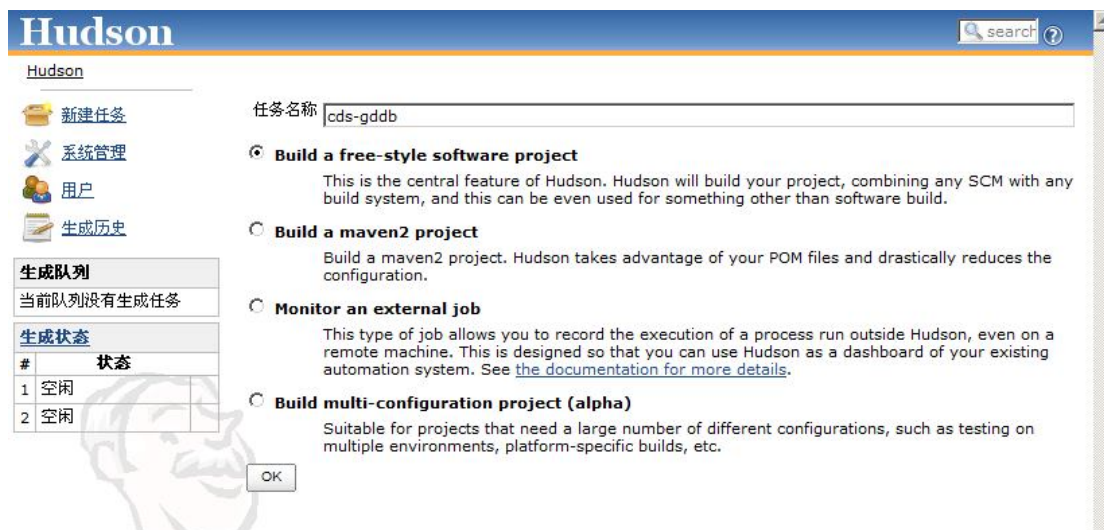
点击“create new jobs”新建一个项目，输入项目名称，选择新建一个 free-style 项目：

free-style:这是 hudson 最主要也是最基本的功能，包括构建，访问 SCM，代码检查，测试等更更多的功能；

maven2:如果项目中采用了 maven2，则可以选择此选项，Hudson 可以读取项目的 POM 文件，从而极大的简化配置工作。

external: 这个是将 Hudson 当做仪表盘，监控外部的构建过程甚至远程机器上的构建。可以与已经存在的自动构建系统配合使用。

multi-configuration: 适合于大量不同配置文件的项目，例如不同的测试环境或者不同的目标平台。



**Hudson**

任务名称: cds-gddb

☒ **Build a free-style software project**  
This is the central feature of Hudson. Hudson will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

☐ **Build a maven2 project**  
Build a maven2 project. Hudson takes advantage of your POM files and drastically reduces the configuration.

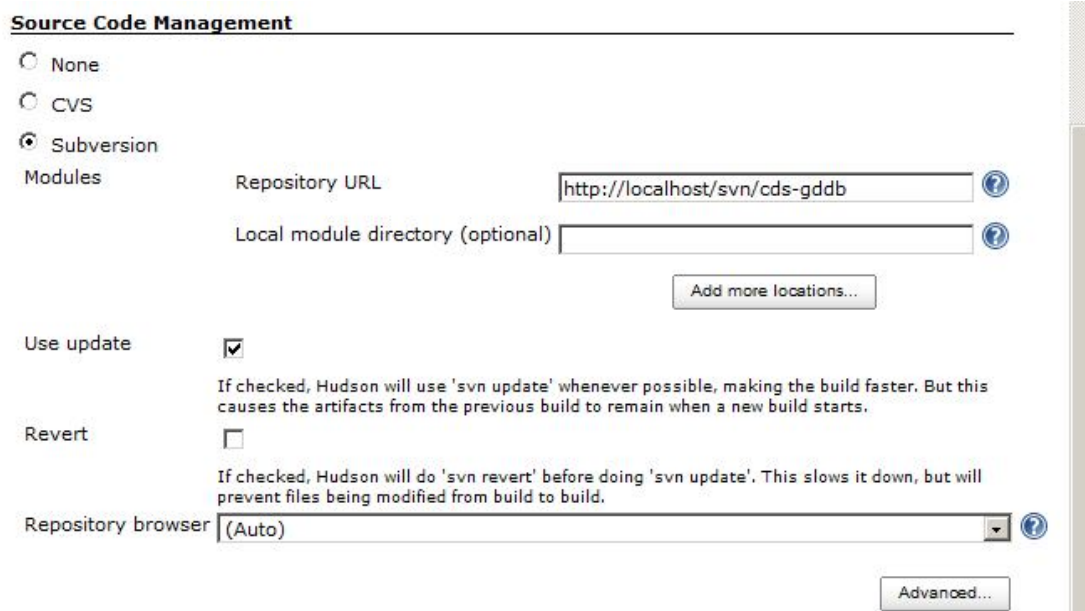
☐ **Monitor an external job**  
This type of job allows you to record the execution of a process run outside Hudson, even on a remote machine. This is designed so that you can use Hudson as a dashboard of your existing automation system. See [the documentation for more details](#).

☐ **Build multi-configuration project (alpha)**  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

OK

#	状态
1	空闲
2	空闲

在项目的配置页面，输入项目的各项信息，这里一一做个解释：



**Source Code Management**

☐ None

☐ CVS

☒ **Subversion**

Modules: Repository URL:  ?

Local module directory (optional):  ?

Add more locations...

Use update: ☒

If checked, Hudson will use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.

Revert: ☐

If checked, Hudson will do 'svn revert' before doing 'svn update'. This slows it down, but will prevent files being modified from build to build.

Repository browser:  ?

Advanced...

首先设置源代码库地址，我们的演示项目存放在一个 SVN 代码库中，因此选择 Subversion，并输入 URL，最后保存设置的时候会要求输入用户名密码。

设置触发策略，Schedule 表达式类似于克隆表达式，用于指定项目触发的时间信息。这里设置为每逢整点或者 30 分触发一次。

Build 的部分主要设置构建的信息，例如使用 Ant 进行构建，然后指定 Ant 脚本的位置（相对于 workspace 目录）和 targets 等信息。

**Build Triggers**

☐ Build after other projects are built

☒ Build periodically

Schedule

☐ Poll SCM

**Build**

**Invoke Ant**

Targets

Build File

Properties

Java Options

最后一部分是设置构建完成之后的动作，例如发布结果文件和报告等。这里发布了 JavaDoc 和代码包。

**Post-build Actions**

☒ Publish Javadoc

Javadoc directory

Directory relative to the root of the workspace, such as 'myproject/build/javadoc'

☐ Retain Javadoc for each successful build

☒ Archive the artifacts

Files to archive

☐ Aggregate downstream test results


☐ Publish JUnit test result report

☐ Build other projects

☐ Record fingerprints of files to track usage

☐ E-mail Notification

设置完成之后点击“Save”，项目就配置好了。

进入项目页面，点击图标手动触发一次构建。一旦启动，hudson 会自动从 svn 检出项目源代码，项目源代码位于 %HUDSON\_HOME%\jobs\workspace\ProjectName 目录下。



在左侧状态栏可以看到构建的进度信息，构建完成之后，如果成功会看到一个太阳图标，我们可以看到构建过程共消耗了 3min42sec。



点击项目名称，进入构建信息页，这里列出了上次构建的详细信息：

- 工作区显示的项目的源代码目录，当然也包括了上次构建生成的全部文件。
- “上次成功 Artifacts” 列出了发布的构建结果文件，包括一个应用包和源代码包。
- “最近变更集” 包括两次构建之间更新的文件列表。

### 3.3 插件管理

我们知道 Hudson 提供了强大的插件支持，我们可以根据项目需要非常方便的选择安装插件。我们的 Ant 脚本中包括了 pmd 和 findbugs 静态代码走查的 target，而且也生成了报告，但是我们没有将其发布到最终的结果中，我们可以借助相关插件来实现报告展示的功能。

通过地址访问插件管理页面 <http://localhost:8080/pluginManager/>，这里分别列出了可以更新、可以安装、和已经安装的插件信息，甚至在“高级”面板中我们还可以上传自己的插件。

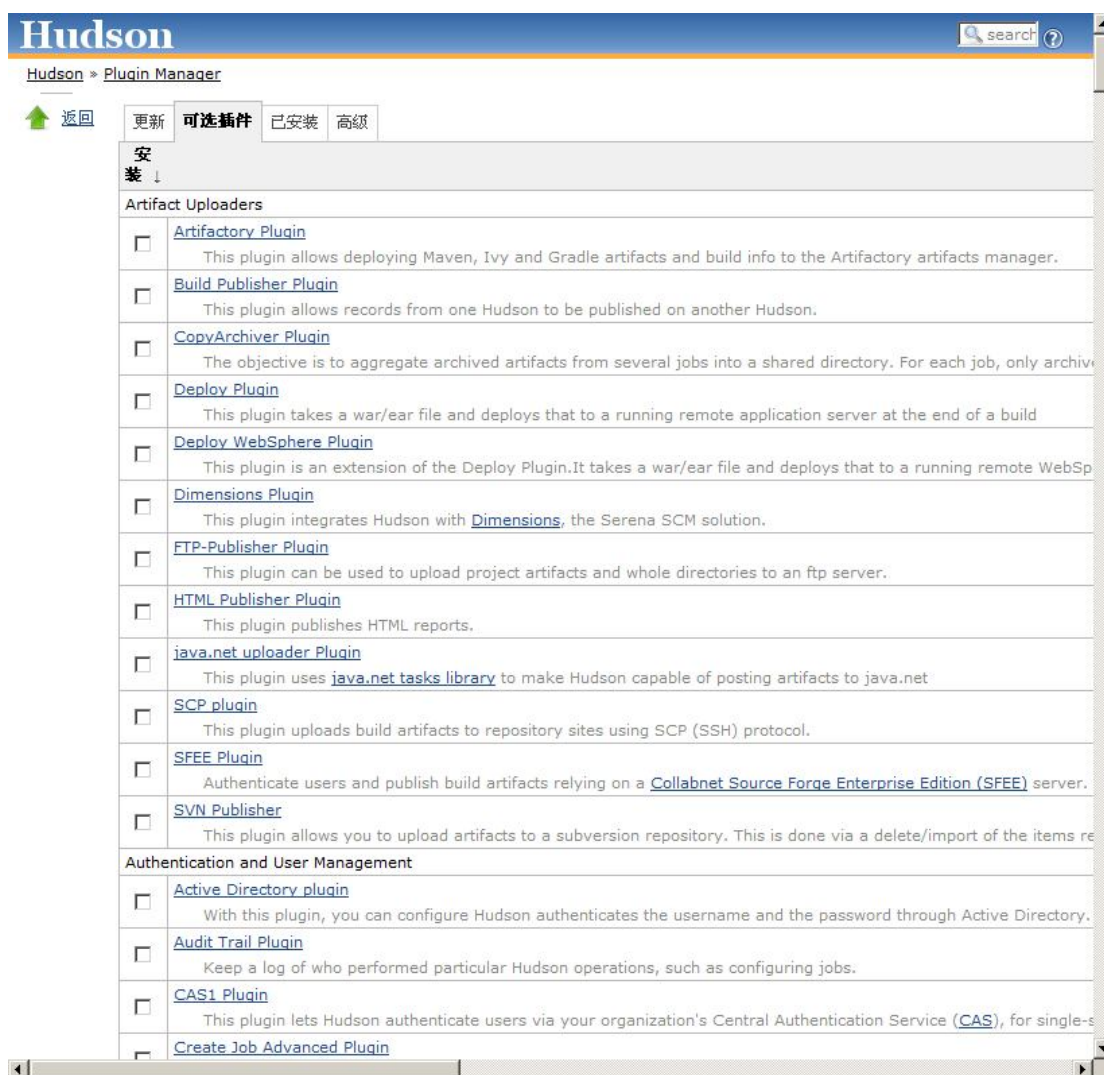
名称	版本	已安装
CVS Plugin This bundled plugin integrates Hudson with CVS version control system.	1.2	1.1
Maven 2 Project Plugin Hudson's Maven 2 project type <b>Warning: This plugin is built for Hudson 1.367 or newer. It may or may not work in your Hudson.</b>	1.367	1.363
SSH Slaves plugin This plugin allows you to manage slaves running on \*nix machines over SSH.	0.12	0.10

首先在“已安装”面板中可以看到系统已经安装了如下的插件：





切换到“可选插件”面板，这里按类别列出了大量可以安装的插件：

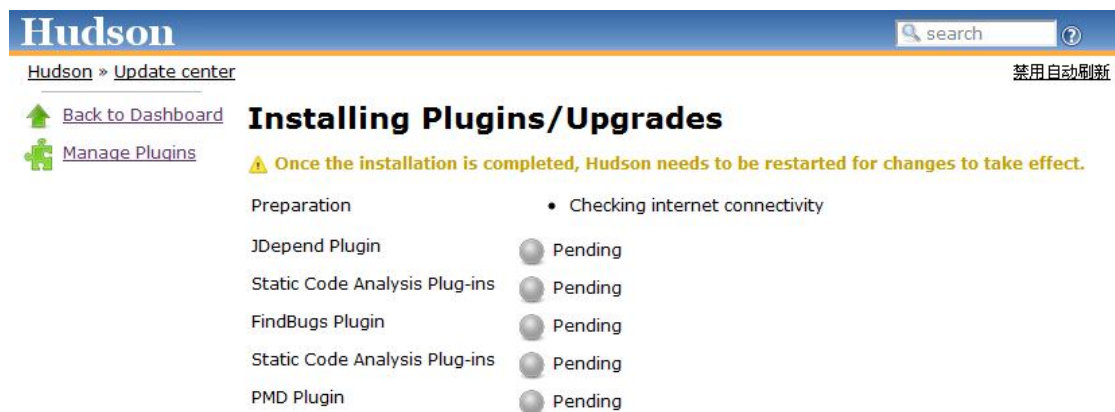


我们选择以下插件安装：

- PMD Plugin
- FindBugs Plugin
- JDepend Plugin

点击“安装”之后，等待安装完成，实际上除了选择的三个插件之外，实际上还需要安

装一个依赖的插件“Static Code Analysis Plug-ins”，Hudson 已经自动帮我们设置好了。



Hudson » Update center

Back to Dashboard Manage Plugins

## Installing Plugins/Upgrades

⚠ Once the installation is completed, Hudson needs to be restarted for changes to take effect.

Preparation	
Checking internet connectivity	
JDepend Plugin	Pending
Static Code Analysis Plug-ins	Pending
FindBugs Plugin	Pending
Static Code Analysis Plug-ins	Pending
PMD Plugin	Pending

等待安装完成，重启 Hudson，在项目信息页面，点击左侧“配置”，进入项目配置页面：



Hudson » cds-gddb

返回 状态 变更集 工作区 立即生成 删除Project 设置 Javadoc

## Project cds-gddb

添加说明

Javadoc 工作区 上次成功Artifacts

- cds-src.zip
- cds.war

最近变更集

### 永久连接

- Last build(#1), 7 min 21 sec前
- Last stable build(#1), 7 min 21 sec前
- Last successful build(#1), 7 min 21 sec前

Build History (趋势图)

#	Time	Status
#1	2010-7-26 16:30:17	成功

全部 失败

在“Post-build Actions”部分，这里已经增加了新的插件配置项，选择要发布的报告选项，并配置对应的路径，配置完成之后保存。



**Post-build Actions**

☒ Publish FindBugs analysis results

FindBugs results

Fileset includes setting that specifies the generated raw FindBugs XML report files, such as `**/findbugs.xml` or `**/findbugsXml.xml`. Basedir of the fileset is [the workspace root](#). If no value is set, then the default `**/findbugsXml.xml` or `**/findbugs.xml` are used for maven or ant builds, respectively. Be sure not to include any non-report files into this pattern.

[Advanced...](#)

☒ Publish PMD analysis results

PMD results

Fileset includes setting that specifies the generated raw PMD XML report files, such as `**/pmd.xml`. Basedir of the fileset is [the workspace root](#). If no value is set, then the default `**/pmd.xml` is used. Be sure not to include any non-report files into this pattern.

[Advanced...](#)

☒ Publish Javadoc

Javadoc directory

Directory relative to the root of the workspace, such as 'myproject/build/javadoc'

☐ Retain Javadoc for each successful build

☒ Archive the artifacts

Files to archive

[Advanced...](#)

☐ Aggregate downstream test results

☐ Publish JUnit test result report

☐ Build other projects

☐ Record fingerprints of files to track usage

☒ Report JDepend

Pre-generated JDepend File

Provide a path to a JDepend file created during the build. Use a preceding "/" to specify an absolute path, leave off the "/" to specify a path within the workspace. Leave blank to have the plugin generate its own file.

☐ E-mail Notification

[Save](#)

再次启动构建：

**Hudson**

Hudson

新建任务 系统管理 用户 生成历史

生成队列

当前队列没有生成任务

生成状态

#	状态
1	空闲
2	空闲

All +

S	W	任务 ↓	上次成功	上次失败	上次持续时间
		cds-gddb	12 min (#2)	N/A	3 min 23 sec

图标: S M L

图例: 全部 失败 最后一次

在构建结果页面的右侧第一项以图形显示了 FindBugs 和 PMD 发现问题数量的变化趋势，在左侧还显示了三个新的菜单，点击之后分别进入 FindBugs、PMD 和 JDepend 的结果页面：

# Hudson

[Hudson » cds-gddb](#)

[返回](#)

[状态](#)

[变更集](#)

[工作区](#)

[立即生成](#)

[删除Project](#)

[设置](#)

[FindBugs Warnings](#)

[PMD Warnings](#)

[Javadoc](#)

[Build \(趋势图\)](#)

History

#4 2010-7-26 17:36:25

#3 2010-7-26 17:18:12

#2 2010-7-26 17:05:41

#1 2010-7-26 16:30:17

[全部](#) [失败](#)

## Project cds-gddb

FindBugs Trend

count

800

700

600

500

400

300

200

100

0

3

4

添加说明

PMD Trend

count

700

600

500

400

300

200

100

0

3

4

Enlarge Configure


Javadoc

工作区

上次成功Artifacts

cds-src.zip

cds.war



40

