基于 Hudson 的持续集成研究和应用①

刘巧玲 范冰冰 黄兴平 (华南师范大学 计算机学院 广东 广州 510631)

摘 要: 持续集成 Continuous Integration(CI)是现代软件工程发展的一个重要里程碑。分析了现代软件开发集成存在的问题，以解决集成难题和提升项目可见性为目标，首次提出一套基于 Hudson 的持续集成设计与实施方案，并给出在 J2EE 项目中的实现指导，此方案在实际工程开发中取得了良好效果。

关键词: 持续集成; CI; Hudson; 自动化构建; 自动化测试

Research and Application of Continuous Integration Based on Hudson

LIU Qiao-Ling, FAN Bing-Bing, HUANG Xing-Ping

(College of Computer Science, South China Normal University, Guangzhou 510631, China)

Abstract: Continuous Integration (CI) is an important milestone of modern software engineering's development history. This paper analyzes existing problems of integration in modern software development, and aiming at solving integration problem & promoting project predictability, puts forward a CI design & implement solution for the first time, which gives the guide of implement in J2EE projects. This solution has got good effects in practical software development.

Keywords: continuous integration; CI; Hudson; automatic build; automatic test

1 引言

在现代软件开发中，不同的功能模块一般由不同的成员负责，同一功能模块的各层代码也可以由不同的程序员编写，在这种情况下，代码的集成和项目的管理尤为关键。但现实中往往出现这样情况：在同一个软件项目中，由不同程序员开发的单个小模块可以单独工作，但把它们集成为一个大的系统则可能失败。集成失败往往是把集成放在开发周期后期，甚至是在项目快结束前单列的一个“总装阶段”。

为了降低集成失败的风险，人们开始采用“早集成、常集成”的持续集成策略，从最初的分阶段集成，到后来的日构建，发展到目前的持续集成(Continuous Integration，CI)。CI 每次集成通过自动化构建来验证，一次构建包括编译、审查、测试、打包和部署等，它是将源代码集成在一起，并验证软件可以作为一个一致的单元运行的过程，尽快地检测出集成错误[1]。CI 以较小增量组装来发现和解决问题，根本上解决了项目开发中的集成难题。CI 能提高项目在进度、质量、代码风格、团队开发生产力、队员状态等软件工程要素的可见性，有助于管理者制定合理有效的计划和策略，从而使软件项目中的开发进度控制、成本控制、质量保证措施与风险管理等问题得到改善或解决[2]。

目前国内大部分软件开发团体对于代码集成只用代码版本管理工具，真正有助于项目的 CI 系统，必须包含代码版本控制、编译、测试、质量检查、代码风格检查、打包与部署，而这一过程配置非常复杂而繁琐，远超出普通开发人员的技术能力。建设一个完善的 CI 系统困难在于：一、如何由版本管理系统的分布式提交触发集成构建系统的自动构建；二、如何实现自动化测试，这是极其复杂而困难的任务；三、如何去分析自动化构建的结果，以得到代码风格统一性、

① 基金项目:广州科技支撑计划项目(2009Z2-D261)
收稿时间:2010-03-30;收到修改稿时间:2010-04-28

代码圈度、代码耦合度、代码重复率、代码覆盖率、分支覆盖率以及单元测试、集成测试的结果，这是困难的过程。目前无论公开出版刊物还是搜索引擎返回的结果，都没有上述三个关键技术的具体信息。

基于多个项目的开发经验和持续集成实践研究，本文提出一套较为完善的基于 Hudson 的配置简便、易于使用的持续集成设计与实施方案，希望能给软件开发团队实践持续集成提供借鉴价值，从而解决项目开发中的集成难题和提升项目可见性。

2　持续集成方案的设计

2.1 持续集成方案

持续集成场景起始于开发人员向代码库提交源代码；代码被提交后，版本控制库检测到变更，触发集成构建系统进行构建，集成构建系统先从版本控制库中取得最新的代码副本，然后执行构建脚本；最后集成构建系统会通过 email、SMS 等方式向指定的成员提供构建结果的反馈信息[3]。图 1 展示了上述所提到的 CI 系统基本要素。

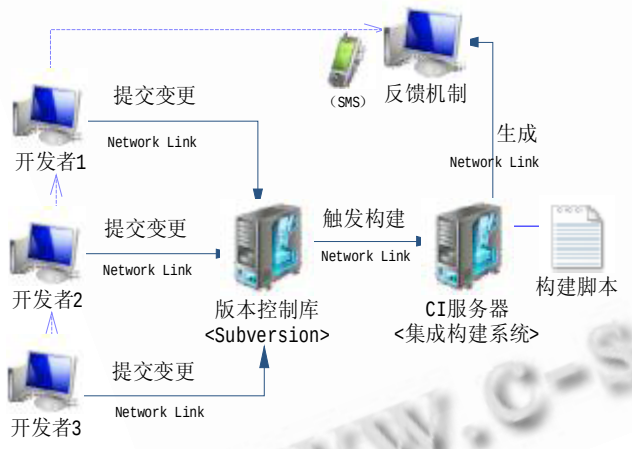

图 1　CI 系统的基本要素

基于 CI 基本理论和 CI 系统的基本要素，结合 J2EE 项目特点，提出了以下的 CI 设计方案，如图 2 所示。

集成构建系统中的 CI 服务器选用 Hudson[4]，它是一种基于 Java 开发的开源 CI 服务器。相比其他开源构建服务器，Hudson 主要有两个优点：一、容易安装和配置，它提供了直观灵活的基于 Web 的用户界面进行配置管理；二、具有强大的插件框架，这些插件可以显示测试结果和代码审查结果等反馈信息。集成构建系统中的构建工具选用 ANT，编译工具采用 Javac 工具，测试工具使用 Junit，代码风格检查工具使用 checkstyle，测试用例覆盖率检查工具选用 cobertura，J2EE 项目的目标代码部署到应用服务器 Jboss，版本控制工具使用 Subversion(简称 SVN)。

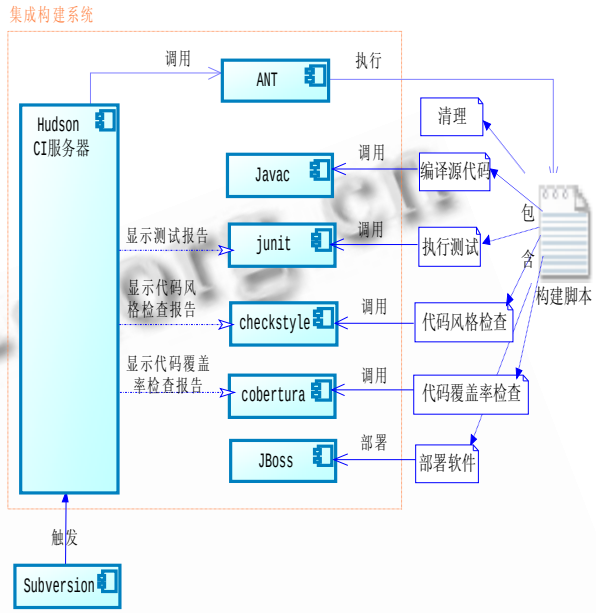

图 2　CI 设计方案

J2EE 项目的源代码修改后，提交到版本控制库 SVN，SVN 检测到变更，就触发 CI 服务器 Hudson 进行构建工作，Hudson 会先从 SVN 中取得最新的代码副本，Hudson 调用构建工具 Ant，Ant 会执行代码副本中的构建脚本 build.xml。build.xml 主要包括 6 部分配置代码——清理、编译源代码、执行测试、代码风格检查、测试用例覆盖率检查、部署软件，当执行到“编译源代码”时，Ant 会调用 Javac 对源代码进行编译；当执行到“执行测试”时，Ant 调用 Junit 工具执行测试用例代码；当执行到“代码风格检查”时，Ant 调用 checkstyle 工具进行源代码风格检查，并生成代码风格检查报告文件；当执行到“测试用例覆盖率检查”时，Ant 调用 cobertura 工具进行测试用例覆盖率检查，并生成测试用例覆盖率检查报告文件；当执行到“部署软件”时，Ant 把 J2EE 项目的目标代码部署到 Jboss。若成功部署，项目相关人员能访问集成构建系统上的目标软件，从而获得软件的功能和性能效果、项目开发进度、软件质量等相关信息。

2.2 持续集成方案的关键技术

(1) 版本管理系统的提交触发 CI 系统的自动构建。

开发者从 SVN 提交修改后，由 SVN 的 Hook 脚

本向 Husdon 发出项目版本变动通知，但由于 SVN 与 Hudson 是完全不同的工具，因此，如何由 Hook 触发 Hudson 来检出项目新版本是一个非常复杂的问题。本方案通过在Hook脚本中设置必要的环境参量，结合Ant提供的 HTTP 组件包，根据 SVN 改变的具体修订项目向 Hudson 发出项目修改通知，然后由 Hudson 向 SVN 请求检出最新版本，最终实现了系统的自动构建。

(2) 自动化测试的实现。

自动化测试包含单元测试自动化与集成测试自动化，由于单元测试仅仅是函数内部功能逻辑的正确性测试，没有与外部资源或组件的交互，因此只要有简单的测试框架即可完成，如 JUnit。集成测试包括了服务器端组件、文件系统、DBMS、服务器端响应时间、客户端脚本、网络连通性、流量负载等极其复杂的要素与结构，目前没有任何测试框架能够完成所有任务，本方案通过 JBoss Micro Container 来完成服务器端组件测试。生产 DBMS 到测试 DBMS 的自动切换，则是由测试用例驱动配置文件自动切换完成的。Web 客户端组件测试难点是 JavaScript，尤其是使用 Ajax 的项目，通过 selenium 结合 cubictest 与 dojotoolkit 组合实现了对 JavaScript 与 Ajax 的集成测试用例的实现，并同时完成网络连通性、流量负载测试。

(3) 自动化构建结果的分析和呈现。

由于以上功能的实现是由不同组件完成的，这些组件对运行环境要求不同，如 PMD 不能做为插件运行，而只能以独立的 java 程序运行。每个组件的配置包括输入数据过滤、处理过程配置与输出格式定义，以便为后续的报告生成提供格式化的信息。每个组件的配置都具有自己的表达模式，这也导致了配置了复杂性。经过长时间的研究，本方案给出了一套能够满足软件工程对质量要求的信息配置策略。

3 持续集成方案的实施

基于上文提到持续集成的设计方案，下文给出了基于 Hudson 的包含代码版本控制、代码风格检查、编译、测试、质量检查、打包与部署等功能的实施方案。

3.1 构建脚本 build.xml

build.xml 主要包括 6 部分配置代码——清理、编译源代码、执行测试、代码风格检查、测试用例覆盖率检查、部署软件，以下介绍脚本中的主要代码。

代码清单 1 展示了利用 javac 任务进行 java 代码编译，这个任务将编译特定目录下的文件，并将生成的.class 文件移到另一个目录下。

```
<!-- 编译 -->
<target name="compile" depends="init">
  <mkdir dir="${classes.dir}"/>
  <javac srcdir="${src.dir}" destdir="${classes.dir}"
    debug="yes" ><!--编译目标代码-->
   <classpath refid="app.classpath" /><!--设置编译依赖-->
   </javac>
</target>
```

代码清单 1 利用 Ant 编译

代码清单 2 是在 Ant 中执行 junit 任务，执行一组 Junit 测试用例。

```
<!--自动单元测试-->
<target name="junit" depends="app.ear" description="run all
unit tests">
  <junit fork="yes" forkmode="once" haltonfailure="false">
 ------
  <batchtest fork="yes" todir="${junitReport.dir}"><!--目标
   目录-->
  <fileset dir="${classes.dir}">
    <include name="junit/**/*Test.class"/><!--测试目标-->
  </fileset>
 </batchtest>
 </junit>
</target>
```

代码清单 2 利用 Junit 和 Ant 进行测试

代码清单 3 展示了 checkstyle 任务的执行，对源代码风格进行检查，以提高代码的可读性。

```
<!--检查代码风格-->
<target name="stylecheck" depends="copyconfigurefile">
   <checkstyle failonviolation="false" maxErrors="1000"
   config="${checkstyle.dir}/sun_checks.xml"><!--设置检
     查标准-->
    <fileset dir="${src.dir}" includes="**/*.java" />
      <formatter type="xml"
      toFile="${checkstyle.dir}/checkstyle_report.xml"/>
      <!--目标输出-->
    </checkstyle>
</target>
```

代码清单 3 利用 checkstyle 和 Ant 进行代码风格检查

代码清单 4 展示了 cobertura-instrument 任务的执行，对代码覆盖率进行检查。

```
<!--代码覆盖率检查-->
<target name="instrument" depends="stylecheck">
  <cobertura-instrument>
    <ignore regex="org.apache.log4j.*" />
    <fileset dir="${classes.dir}">
      <include name="**/*.class" /><!--检查目标-->
    </fileset>
  </cobertura-instrument>
</target>
```

代码清单 4　测试用例覆盖率检查

3.2 集成构建系统配置

(1) 创建一个 Job

进入 Hudson 服务首页后，点击 New Job，源代码项 Source Code Management 要填写 CiProject 源代码在版本控制库中的具体存储位置，编译项 Build 要选 Invoke ant。

(2) Junit 配置

进入 Hudson 服务首页后，点击项目 CiProject 进入 Configure 页面，Post-build Actions 项选择 Publish JUnit test result report，Test report XMLs 填上 junit 测试报告文件的路径，这样 Hudson 就能显示 junit 测试结果。

(3) Checkstyle 配置

首先要启用 Hudson 的 Checkstyle Plug-in 插件，然后进入 Post-build Actions 项所在页面，勾选 Publish Checkstyle analysis results，Checkstyle results 填上代码风格检查报告文件的路径，这样 Hudson 就能显示代码风格检查结果。

(4) Cobertura 配置

首先要启用 Hudson 的 Cobertura Plug-in 插件，然后进入 Post-build Actions 项所在页面，勾选 Publish Cobertura Coverage Report，Cobertura xml report pattern 填上测试用例覆盖率检查报告文件的路径，这样 Hudson 就能显示测试用例覆盖率检查结果。

3.3 自动构建结果

Hudson 中的 Build History 会记录当前构建状态和历史构建结果，蓝色灯表示构建成功，红色灯表示构建失败，灰色闪烁灯表示正在构建中。根据上述的配置，Hudson 会显示 junit 测试报告、checkstyle 代码风格检查报告、以及 cobertura 测试用例覆盖率检查报告，如图 3 所示。

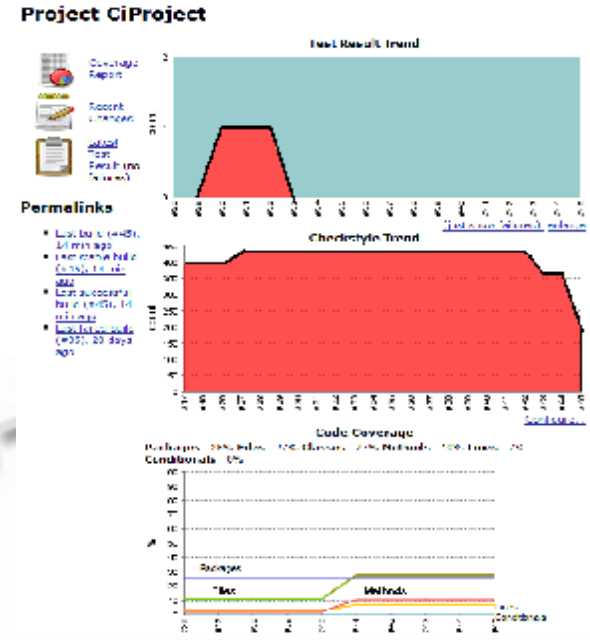

图 3　CI 报告

4 结束语

本文提出了基于 Hudson 的持续集成设计与实施方案，此方案除了能完全应用于 J2EE 项目开发中的集成实践，对.NET 项目和其他 B/S 项目也有较好的适用性。在项目开发中应用 CI，将会解决项目开发中的集成难题和提高项目可见性。

本方案的不足之处是没有加入持续数据库集成，通常情况下，项目开发中的数据库结构变动不大，对持续数据库集成的需求不迫切，对于数据库结构变化较大的项目建议加入持续数据库集成。

参考文献

1 Fowler M. Continuous Integration. [2009-9-7].http://martinfowler.com/articles/continuous Integration.html.

2 Shore J, Warden S. The Art of Agile Development.O'Reilly Media, Inc., 2008.

3 Duvall PM, Matyas S, Glover A. 持续集成--软件质量改进和风险降低之道.王海鹏,贾立群,译.北京:机械工业出版社, 2008.

4 Hudson M. [2009-11-20].http:// wiki. hudson-ci. org/display/HUDSON/Meet+Hudson.