

OpenDoc Series'

持续集成实践 之 CruiseControl

V1.0

作者：张辰雪

文档说明

参与人员:

作者	联络
张辰雪	

(at) 为 email @ 符号

发布记录

版本	日期	作者	说明
1.0	2005.05.1	张辰雪	原创
1.0	2005.06.1	夏昕	文档格式编排

OpenDoc 版权说明

本文档版权归原作者所有。

在免费、且无任何附加条件的前提下，可在网络媒体中自由传播。

如需部分或者全文引用，请事先征求作者意见。

如果本文对您有些许帮助，表达谢意的最好方式，是将您发现的问题和文档改进意见及时反馈给作者。当然，倘若有时间有能力，能为技术群体无偿贡献自己的所学为最好的回馈。

Open Doc Series 目前包括以下几份文档:

- Spring 开发指南
- Hibernate 开发指南
- ibatis2 开发指南
- Webwork2 开发指南
- 持续集成实践之 CruiseControl

以上文档可从 <http://www.redsaga.com> 获取最新更新信息

持续集成实践之 CruiseControl 篇

目录

目录.....	3
前言.....	5
读者.....	5
约定.....	5
持续集成的概念.....	5
剖析 CRUISECONTROL	7
CC 框架 8	
<i>Build Loop</i>	9
CC 插件 (Plugin)	10
CC 的配置文件 12	
<bootstrappers>	13
<modificationset>	14
<schedule>	14
<log >.....	17
<publishers >.....	17
<dateformat >.....	17
<plugin >.....	17
CRUISECONTROL 应用举例.....	17
基础知识: 18	
准备工作: 18	
安装和准备项目持续集成的环境 18	
准备 workspace.....	18
下载 CC 并创建 <i>cruisecontrol.jar</i>	19
创建 <i>cruisecontrol.war</i>	21
例子 HELLO WORLD! 22	
将 <i>helloworld</i> 导入 CVS.....	22
准备配置文件.....	24
启动 CC.....	26
结束语	30
附录一 HELLO WORLD!与 VSS.....	30
将 HELLOWORLD 导入 VSS 30	
准备配置文件 31	

附录二 OUT OF MEMORY ERROR 的解决方案.....	33
参考资料.....	34

前言

持续集成 (Continuous Integration) 这个术语源自 XP (极限编程) 的一个最佳实践, 随着 XP 社区在近几年的壮大, XP 的很多实践得到了广泛的推广, 持续集成就是其中之一, 但是持续集成并非 XP 的专利, 持续集成完全可以应用在采取非 XP 方法 (例如 RUP) 的项目里面。持续集成也不是一个新的概念, 在这个术语出现之前, 日创建 (daily build) 提供同样的含义, 他们的主要区别就在于实施的频率上, 随着 XP 社区的大师级人物 Martin Fowler 的一篇《[Continuous Integration](#)》正式为其正名, 持续集成这个术语就越来越多地出现在原来日创建出现的位置。同时, Martin Fowler 所在的公司 ThoughtWorks 开放了其持续集成的工具 CruiseControl 的源代码, 持续集成对于大部分开发人员来说就不再只是停留在口头上的漂亮的术语, 任何人在掌握了持续集成的基础理论后, 都可以使用 CruiseControl 来体会持续集成在项目开发中的巨大威力。

本文将简单地介绍持续集成的一些基本概念, 持续集成到底是做什么的? 持续集成的关键点在什么地方? 怎么结合工具进行实践? 最后一步步地演示如何使用 CruiseControl 这个目前最热门的持续集成工具。

读者

项目主管, 开发人员, 测试人员, 要求熟悉 Apache Ant, CVS 或者 Visual SourceSafe。

约定

- 本文的例子以 Windows 2000 作为操作系统平台。
- 本文以 Java 项目为例, 对于 .net 的持续集成请参考 CruiseControl.NET 的官方文档。
- 本文不是 CruiseControl 官方文档的中文版, 本文注重讲解 CruiseControl 的应用, 关于 CruiseControl 的配置文件的各参数的详细信息请参考官方文档。
- 本文中的创建均指 build。
- 本文的源码控制系统指版本控制系统, 源码库就是资料库 (Repository), 主要是因为本文重点针对的是源码。

持续集成的概念

在没有应用持续集成之前, 传统的开发模式是项目一开始就划分模块, 然后等所有的代码都开发完成之后再集成到一起进行测试, 随着软件技术的发展, 各种软件方法百花齐放, 软件规模也在扩大, 软件需求越来越复杂, 软件已经不能简单地通过划分模块的方式来开发, 需要项目内部互相合作, 划分模块这种传统的模式的弊端也越来越明显, 由于很多 bug 在项目的早期就存在, 到最后集成的时候才发现问题, 开发者需要在集成阶段花费大量的时间来寻

找 bug 的根源，加上软件的复杂性，问题的根源很难定位，甚至出现不得不调整底层架构的情况，在这个阶段的除虫会议（bug meetings）特别多，会议的内容基本上都是讨论 bug 是怎么产生的，最后往往发展成为不同模块的负责人互相推诿责任。

持续集成最大的优点是可以避免这种传统模式在集成阶段的除虫会议。持续集成主张项目的开发人员频繁的将他们对源码的修改提交(check in)到一个单一的源码库，并验证这些改变是否对项目带来了破坏，持续集成包括以下几大要点：

- ◆ 访问单一源码库，将所有的源代码保存在单一的地点（源码控制系统），让所有人都能从这里获取最新的源代码（以及以前的版本）。
- ◆ 支持自动化创建脚本，使创建过程完全自动化，让任何人都可以只输入一条命令就完成系统的创建。
- ◆ 测试完全自动化，要求开发人员提供自测试的代码，让任何人都可以只输入一条命令就运行一套完整的系统测试。
- ◆ 提供主创建，让任何人都可以只输入一条命令就可以开始主创建。
- ◆ 提倡开发人员频繁的提交（check in）修改过的代码。

持续集成的关键是完全的自动化，读取源代码、编译、连接、测试，整个创建过程都应该自动完成。对于一次成功的创建，要求在这个自动化过程中的每一步都不能出错，而最重要的一步是测试，只有最后通过测试的创建才是成功的创建。

在持续集成里面创建不再只是传统的编译和连接那么简单，创建还应该包括自测试，自测试的代码是开发人员提交源码的时候同时提交的，是针对源码的单元测试（源自 XP 的实践），将所有的这些自测试代码整合到一起形成测试集，在所有的最新的源码通过编译和连接之后还必须通过这个测试集的测试才算是成功的创建。这种测试的主要目的是为了验证创建的正确性，McConnell 称之为“冒烟测试”，在持续集成里面，这叫做集成验收测试 Build Verify Test，简称 BVT。BVT 测试是质量的基础，QA 小组不会感受到 BVT 的存在，他们只针对成功的创建进行测试（如功能测试）。

BVT 测试应该尽可能的详尽，详尽的测试才能发现更多的问题，而由此得到的反馈结果也更有参考意义，测试应该全部执行完毕，这样得到的反馈结果才是完整的，而不是遇到错误就放弃测试过程。

持续集成和日创建相比还有以下特点：

- ◆ 持续集成强调了集成频率，和日创建相比，持续集成显得更加频繁，目前推荐的最佳实践是每一个小时就集成一次。
- ◆ 持续集成强调及时反馈，日创建的目的是得到一个可以使用的稳定的发布版本，而持续集成强调的是集成失败之后向开发人员提供快速的反馈，当然成功创建的结果也是得到稳定的版本。
- ◆ 日创建并没有强调开发人员提交（check in）源码的频率，而持续集成鼓励并支持开发人员尽快的提交对源码的修改并得到尽快的反馈。

从上面列出的持续集成和日创建相比的特点来看，很明显，“频率”和“反馈”这两个词出现的特别多，持续集成有一个与直觉相悖的基本要点，那就是“经常性的集成比偶尔集成要好”。Martin Fowler 认为对于持续集成来说，集成越频繁，效果越好，如果你的集成不是经常进行的（少于每天一次），那么集成就是一件痛苦的事情，如果集成偶尔才进行一次（一周甚至一个月），等到集成阶段发现 bug，然后找原因解决 bug，会耗费你大量的时间与精力，而且这种方式有点象传统的集成模式，这违背了持续集成的初衷。

根据 Martin Fowler 的观点，项目 bug 的增加和时间并不是线性增长的关系，而是和时间的

平方成正比，两次集成间隔的时间越长，bug 增加的数量越超过你的预期，解决 bug 付出的工作量也越大，而你越觉得付出的工作量越大，你就越想推迟到以后去集成，企图到最后一次性解决问题，结果 bug 产生的就更多，导致下一次集成的工作量更大，你越感觉到集成的痛苦，就越将集成的时间推后，最后形成恶性循环。

因此如果集成的结果是让你感到痛苦，也许就说明你应该更频繁地进行集成。频繁的集成和及时的反馈鞭策着项目小组积极的面对问题，而不是将问题推到最后来解决，如果方法正确，更频繁的集成应该能减少你的痛苦，让你节约大量时间。

因为持续集成最终是通过测试来验证创建，所以你会发现对于持续集成的频率的要求跟 Kent Beck 提出的测试驱动的开发方法里面测试第一的理念完全一致。

需要注意的是从项目的一开始就引入持续集成可以尽早的发现 bug，但是并不代表持续集成可以帮你抓到所有的 bug。持续集成的排错能力取决于测试技术，众所周知，无法证明已经经过测试的代码就已经找到了所有的错误。

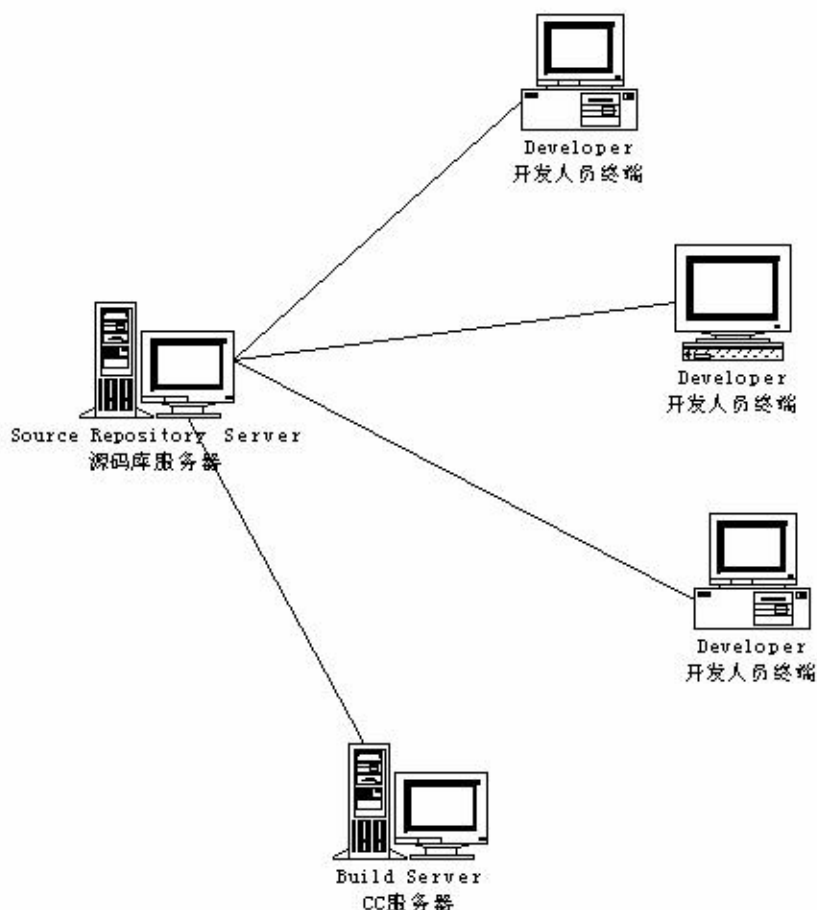
前面列举了持续集成这么多优点，但是创建一个持续集成的环境技术上是比较复杂的，也需要一定的时间，关键在于持续集成可以“及时”抓到足够多的 bug，从根本上消除传统模式的弊端，这就已经值回它的开销了。

对于持续集成的概念简单介绍到这里，下面开始持续集成的实践之旅。目前支持持续集成的工具已经越来越多，主流的包括 AntHill（商业化工具），CruiseControl，Apache Dump。本文将以 CruiseControl 为例来体会持续集成实践。

剖析 CruiseControl

CruiseControl 是一种持续集成过程的框架，包括了邮件通知，ant 和各种源码控制工具的插件。并提供了 web 接口，用于查看当前和以前的创建的结果。下面将用 CC 来代表 CruiseControl。

首先让我们看看下面的 CC 部署图，这个图描述了持续集成的硬件环境，图中包括了一台独立的源码库服务器，以及开发人员的终端（同时也是源码库服务器的客户端），我们注意到 CC 在一台独立的服务器上运行，这是推荐的一种结构，出自对性能和不影响原有的开发环境的考虑。当然你可以将 CC 放在源码库的同一台服务器上甚至某个开发人员的终端上，从这个图可以看出可以很容易的将 CC 引入到现有的开发模式中，只需要增加一台独立的服务器就可以了。



CC 框架

在学习使用 CC 之前，我们有必要研究一下 CC 的框架，这对我们了解 CC 的设计原理和 CC 对持续集成的支持程度很有帮助。通过前面对持续集成的一些概念的讲解，我们已经知道持续集成的几个要素，下面我们从工具的角度来看 CC 对这几个要素的支持。

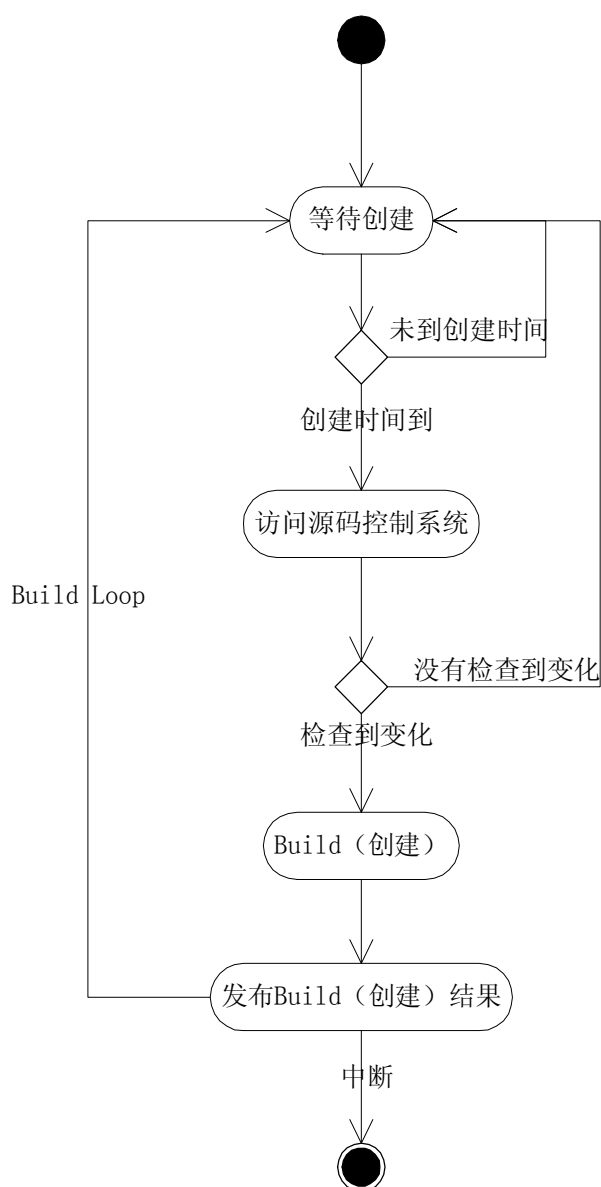
持续集成的要点	CC 的支持
单一代码源	CC 通过相应的插件支持对 CVS,VSS(Visual SourceSafe),ClearCase 等各类版本控制软件的源码库的访问。
自动化创建脚本	CC 通过 Build Loop 支持自动化创建，并通过 CC 插件支持 ant, maven 等创建工具。CC 对创建的支持实际上是通过一个代理(proxy)来完成的，你可以选择 Ant,Maven 这些流行的创建工具
自测试的代码	间接通过 ant 等自动化创建工具来支持对自测试的调用，但是这取决于项目自身是否提供自测试代码，CC 只是通过 ant 等工具激活对测试代码的调用。
主创建	间接通过 ant 等自动化创建工具支持，要求相应的 ant 或者 maven 脚本提供了主创建任务。
检入代码	属于版本控制的范畴，要求开发人员自己检入代码。

Build Loop

前面在讨论持续集成的时候讲到其最重要的特征之一是自动化，而 CC 的 Build Loop 就是为支持自动化而设计的，Build Loop 也是 CC 的核心。

Build Loop 从字面上理解就是循环创建的意思，CC 提供了一个 daemon 进程，该进程自动根据配置的时间间隔（也可以指定某个具体时间）读取 CC 配置文件并进行循环创建（build cycle），每次 CC 都会重新加载配置文件（修改了配置文件不用重新启动 CC）。

Build Loop 过程中所做的工作如下：访问源码控制系统，查看是否有代码被修改，如果有，获取源码的新版本，并根据配置对源码进行一次 Build，创建一个日志文件，最后向开发人员通知 build 的结果，活动图如下：



因为 Build Loop 是根据配置文件的内容来进行的，根据上面 BuildLoop 所做的工作，我们大概可以猜出配置信息主要应该包括：定时创建的时间和源码库的访问信息（定义检查源码变

化情况), 创建任务信息 (如指定 Ant 文件), 记录日志 (创建结果), 通知 (通知的内容可以定制)。现在对配置文件有个初步印象, 在下面学习配置文件的配置项的时候就会轻松很多。

补充说明, CC 的 Build Loop 刚开始只支持单个项目, 现在 CC 的 Build Loop 已经支持多项目(multiproject)。

CC 插件 (Plugin)

CC 设计思想是 one-size-fits-all, 也就是 CC 是由一个很精小(但是很强大)的核心(Build Loop)以及一些外部插件组成, 这给使用者提供了很大的扩展空间, 使用者可以根据需要扩展 CC 的功能 (提供新的插件), 而且 CC 是开源项目, 你还可以查看源码并修改 CC 提供的插件。CC 提供了六种不同类型的插件: Bootstrapper, SouceControl, Builder, LabelIncrementer, Publisher 以及 Listener, CC 的配置是围绕这些插件展开的, 下面对这些插件进行一个简单的介绍:

- **Bootstrapper:** 在 CC 进行创建之前运行, 是创建前的准备工作
- **SourceControl:** 访问源码源码控制系统 (如 CVS, VSS, ClearCase 等等), 查看源码自上一次 Build 之后是否被修改过, 并据此决定是否需要进行下一次 Build。
- **Builder:** 对项目进行创建, 熟悉 ANT 的使用者应该很清楚创建的含义, 这里简单提一下, 一次典型的创建包括了对项目源码的编译, 测试, 打包
- **LabelIncrementer:** 用于对源码打标签, 自动增加标签的编号
- **Publisher:** 用于发布创建的结果, 可以通过 email 的方式通知开发人员。
- **Listener:** 用于处理一些项目有关的事件 (CC 2.2 之后提供的新的功能)。

下面说说 CC 插件相关的配置, 在早期的 CC, 插件是需要注册的, 在<project>元素下面, 添加<plugin name="" classname=""/>就可以了, 其中 name 对应到你在 config.xml 使用到的插件功能的名字, 而 classname 则是实际的插件的 Java 类名, 有点类似 ANT 的<taskdef>。

最新的 CC 的新的特征是保持配置文件的精小, 所以 CC 自带的插件会自动注册, 不再需要显示地在配置文件里面添加注册信息了, 如果你有第三方插件要使用, 可以按照前面所讲到的方法进行注册。

由于版本的向下兼容性, 如果你很早以前就使用 CC, 一直保留了插件注册的习惯, 那么现在还可以继续象原来那样使用。

下面给出一个插件注册的例子, 里面配置的细节信息都省略了, 只给出 XML 文件中相应的元素的层次结构:

```
<cruisecontrol>
  <project name="myproject">
    <bootstrappers>
      <currentbuildstatusbootstrapper/>
    </bootstrappers>
    <modificationset>
```

```
        <cv/>
    </modificationset>
    <schedule>
        <ant/>
        <pause/>
    </schedule>
    <log>
        <merge/>
    </log>
    <publishers>
        <currentbuildstatuspublisher/>
        <email/>
    </publishers>
    <plugin name="cvs"

    classname="net.sourceforge.cruisecontrol.sourcecontrols.CVS" />
    <plugin name="currentbuildstatusbootstrapper"

    classname="net.sourceforge.cruisecontrol.bootstrappers.CurrentBuildStatusBootstrapper"/>
    <plugin name="ant"

    classname="net.sourceforge.cruisecontrol.builders.AntBuilder"/>
    <plugin name="pause"

    classname="net.sourceforge.cruisecontrol.PauseBuilder"/>
    <plugin name="email"

    classname="net.sourceforge.cruisecontrol.publishers.HTMLEmailPublisher"/>
    <plugin name="currentbuildstatuspublisher"

    classname="net.sourceforge.cruisecontrol.publishers.CurrentBuildStatusPublisher"/>
    <plugin name="labelincrementer"

    classname="net.sourceforge.cruisecontrol.labelincrementers.DefaultLabelIncrementer"/>
    </project>
</cruisecontrol>
```

在这个配置文件里面，我们可以看到每使用到某个插件的功能，就会添加相应的注册信息，例如：

```
<cruisecontrol>
```

```
<project>
  <modificationset>
    <cvsv/>
  </modificationset/>
</project/>
</cruisecontrol/>
```

就会对应到

```
<cruisecontrol>
  <project>
    <plugin name="cvs"
      classname="net.sourceforge.cruisecontrol.sourcecontrols.CVS" />
  </project/>
</cruisecontrol/>
```

至于该配置文件的细节，后面会有专门的章节来讲解，我们这里举这个例子主要是为了说明插件使用和注册之间的关系。记住，新版 CC 对于 CC 自带的插件已经不需要手工配置注册了。

CC 的配置文件

CC 的配置文件用于 build loop，默认文件名为 config.xml（如果使用其他的文件名，启动 CC 的时候需要手工指定该文件名，类似 ant 对 build.xml 文件的处理方式），该文件的 Builder 插件部分会引用相应的 build 文件，如 ant，maven 的 build 文件。

关于 CC 配置文件的各配置项参数的详细资料，请参考官方文档：

<http://cruisecontrol.sourceforge.net/main/configxml.html>

这里只讨论配置文件中常用的各元素之间的关系。

主配置文件 config.xml 的根元素是<cruisecontrol>，该元素很简单，没有什么需要配置的属性。<cruisecontrol>下支持三种元素，如下：

```
<cruisecontrol>
  <system/>
  <plugin/>
  <project/>
</cruisecontrol>
```

本文只讨论<project>元素，其他两个元素在常规的场所可以不用，目前 CC 支持多个项目（multiproject），因此可以有多个并行的<project>元素。在<project>元素下面都是跟 CC 插件有关的配置项，结构如下：

```
<cruisecontrol>
  <project>
    <plugin/>
    <dateformat/>
    <labelincrementer/>
```

```
<listeners/>
<bootstrappers/>
<modificationset/>
<schedule/>
<log/>
<publishers/>
</project>
</cruisecontrol>
```

下面对其中几个常用的元素的意义和用法进行详细的讲解

<bootstrappers>

<bootstrappers>的子元素就是 Bootstrapper 插件的配置信息, <bootstrappers>在创建之前运行, 这个前面我们已经讲过了, <bootstrappers>下面的每一个子元素都是相互独立的, 因此可以同时配置有多个 bootstrapper。

CC 提供的 bootstrapper 包括了两个种类, 一种用于向其他插件提供项目当前创建的状态, 还有一种是从某个源码控制系统更新本地文件, 其中最常用到的 bootstrapper 是 <currentbuildstatusbootstrapper>和<cvsbootstrappers>(本文使用 CVS 源码控制系统作为例子, 如果读者使用 Visual SourceSafe 或者 ClearCase, 那么这里应该分别使用<vssbootstrappers >或<clearcasebootstrappers>)。

前者<currentbuildstatusbootstrapper>指定了状态文件的位置, 主要是用来访问项目当前创建的状态, CC 的<currentbuildstatuspublisher>会将创建的状态写入这个文件, CC 的 web 用户接口要向用户提供最近一次创建的信息, 其 web 组件会访问该文件来获得当前项目进行创建的状态。顺便讲一下 <currentbuildstatusftpbootstrapper>, 其作用跟 <currentbuildstatusbootstrapper>完全一样, 不同的是状态文件保存在一个 FTP 上。

后者<cvsbootstrapper>的作用有点难理解, 因为我们每次项目的创建都应该基于最新的代码, 因此在创建之前就要获得最新的项目文件, 如果你使用的是 ant, 这个工作是由 ant 的 buildfile 来完成的, 如果这个 buildfile 本身在创建开始之前发生了变化, 我们是不是应该先更新这个 buildfile, 然后才通过 buildfile 来对项目进行创建呢? <cvsbootstrapper>就是从源码控制系统更新 buildfile 文件而设计的(还有一种替代方法是使用 wrapper buildfile, 这样就不用使用 <cvsbootstrapper>了, wrapper buildfile 也是推荐的方法, <modificationset>部分会进行详细的讨论)。

<bootstrappers>应用举例:

```
<bootstrappers>
  <currentbuildstatusbootstrapper file="currentbuildstatus.txt" />
  <cvsbootstrapper
cvsroot=":pserver:anonymous@localhost:2401/sandbox"
  file="helloworld/build.xml" />
</bootstrappers>
```

关于<cvsbootstrapper>和其他 bootstrapper 插件(如<vssbootstrapper>)的配置参数请参考官方参考手册。

<modificationset>

<modificationset>包括了 SourceControl 插件的配置信息，用于检查各个源码控制系统中是否发生变化，<schedule>会用到这里的配置信息，如果检测到变化，会触发创建过程。

<modificationset>的属性 quietperiod(单位为秒)定义了一个时间值。如果 CC 检查到了变化，会自检查到变化的源码控制系统的最后一次 check in 的时间开始等待，等待的时间由 quietperiod 决定，等待结束之后才触发创建 (build) 过程，主要是防止有人在 check in 的过程当中就触发创建过程 (可能 check in 只做了一半，这个时候触发创建显然是不正确的)。

下面给一个 cvs 的例子：

```
<modificationset quietperiod="30">
  <cvs cvsroot=":pserver:anonymous@localhost:2401/sandbox"
    localworkingcopy="helloworld"/>
</modificationset>
```

下面是一个 vss 的例子 (记得要在<bootstrappers>里面配置相应的<vssbootstrapper>)，熟悉 ant 的读者可以看出，其实很多参数跟 ant 里面对应的任务是一致的：

```
<modificationset quietperiod="30">
  <vssget serverPath="${vss.serverPath}" sssdir="${vss.sssdir}"
    login="${vss.username},${vss.password}"
    vsspath="${vss.vsspath}/${ot.src.dir}"/>
</modificationset>
```

还有对其他的源码控制系统如 pvcs,svn,clearcase 的访问的配置都差不多，请查看官方的参考手册。

<schedule>

<schedule>指定了创建的时间间隔，新的版本已经支持定义某个具体时间触发创建，这个新的特征是为了满足 Nightly build 的需要。

<schedule>定时驱动<modificationset>，如果检测到变化，就执行所指定的 builder 的任务。

例子：

```
<schedule interval="180" intervaltype="relative">
  <ant buildfile="projectname/build.xml" target="cleanbuild"
    multiple="5"/>
  <ant buildfile="projectname/build.xml" target="masterbuild"
    multiple="1"/>
</schedule>
```

目前 CC 支持的 builder 插件包括了 apache 的 ant 和 maven，本文以 ant 为例讲解 builder 插件的用法。

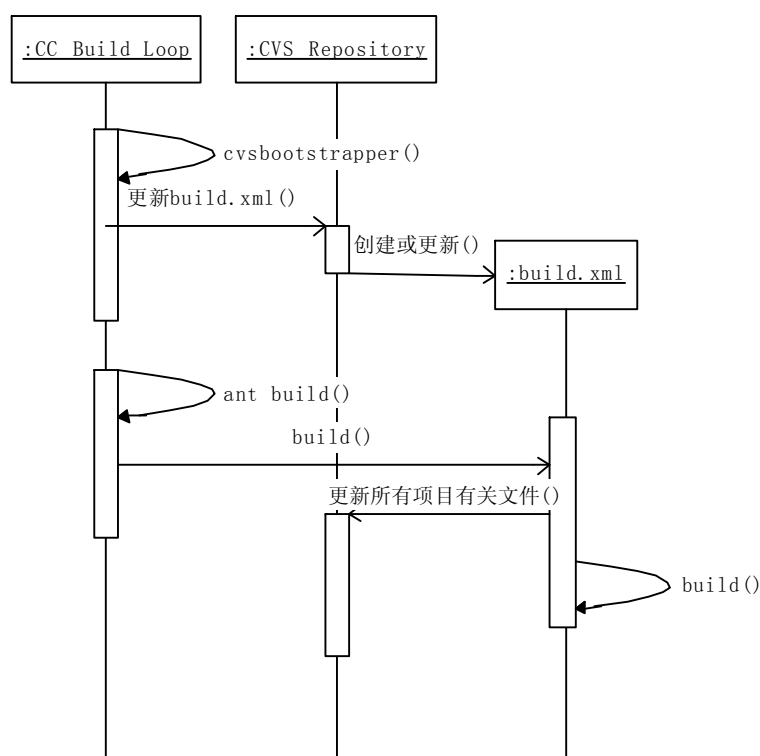
当 CC 启动 ant build 时，ant 是作为独立的 java 进程被调用的，CC 提供了两种调用 ant 的方式，第一种是使用 antscript 方式，第二种是使用和 CC 一起发布的 ant。其中第一种是推荐的方式，我们也将采用第一种方式，本文后面凡是使用到<ant>的地方，均指通过 antscript

方式来调用 ant。

Antscript 方式要求提供两个重要的参数，一个是 antscript，一个是 antWorkingDir，antscript 指定了一个独立的 ant 可运行的脚本文件，在 windows 下是 %ANT_HOME%\bin\ant.bat，antWorkingDir 指定了 ant 要运行的 build.xml 所在的目录，例如：

```
<ant antscript="C:\usr\local\ant\bin\ant.bat" antworkingdir="C:\home\ccuser\ccworkspace"
buildfile="build.xml" target="cleanbuild" multiple="5" />
```

关于 ant 的 build.xml 文件有两个访问的模式，第一个是前面我们讨论<bootstrappers>的时候讲到了的<cvsbootstrapper>插件，<cvsbootstrapper>在 build loop 运行前得到 build.xml 文件，并保存在指定的位置，而我们的 ant builder 插件会在 build loop 过程中去访问这个文件，其过程如下图：



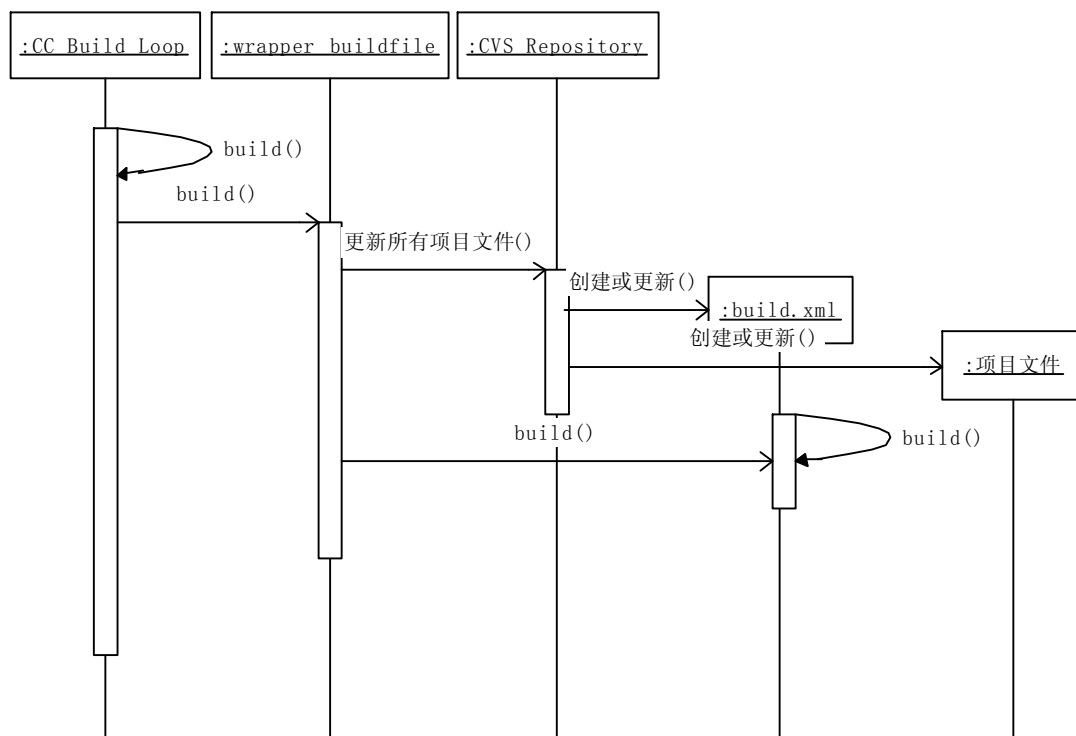
这种访问模式的特点对于现有的项目的 build.xml 要进行修改，增加新的 target 对源码控制系统的访问，同时增加一个和 CC 创建有关的 target，CC 自带的例子采取的就是这种方式，配置文件结构如下：

```
<project>
  <target name="build" depends="compile"
    description="Project original build"/>
  <target name="checkout" description="Update package from CVS"/>
  <target name="masterbuild" depends="checkout"
    description="Cruise control master build"/>
</project>
```

我们可以看出这种访问模式的缺点是对项目的 build.xml 有侵入性，除了项目原有的创建任务之外，还要为 CC 添加新的创建任务。而且这种模式在创建逻辑上是很难接受的，首先

build.xml 保存在源码控制系统里面，需要 CC 通过<cvstrapper>更新 build.xml，然后在创建阶段又通过这个 build.xml 的 checkout 更新整个项目的文件，同时也更新了自己，对于刚接触 CC 的新手来说很难理解。

第二个模式是 wrapper buildfile 文件模式，我们不在<bootstrappers>下使用<cvstrapper>，而是在文件系统的某个位置提供一个 wrapper buildfile 文件，这其实也是个 ant 的 build.xml，让 CC 直接去访问这个文件，然后通过这个文件再访问源码控制系统，获得新的源码和相应的 build.xml，然后将创建目标转发到新的 build.xml，如下图：



wrapper buildfile 模式的特点是需要增加一个独立的 build.xml，实际上是将第一种模式里面对原有项目的 build.xml 增加的部分独立出来放到一个新的 build.xml 里面，并将创建转发到项目的 build.xml，配置文件结构如下：

```

<project>
  <target name="checkout" description="Update package from CVS"/>
  <target name="masterbuild" depends="checkout">
    <ant antfile="${helloworld.dir}/build.xml" target="build"/>
  </target>
</project>
  
```

这种模式的优点是保持了原有项目的 build.xml 的完整性，对于原有的项目不需要做修改，而是通过添加新的文件的方式来完成。

在 [CC 应用举例](#) 的部分我们会体会到一个完整的 wrapper buildfile 例子。

<log >

<log >指定项目日志保存的地点，主要是合并项目创建过程 junit 测试结果的报表文件(xml)。
<log>的用法很简单，通常是指定 CC 的合并日志的目录就可以了，例如：

```
<log dir="logs/projectname">  
  <merge dir="checkout/projectname/test-results"/>  
</log>
```

<publishers >

<publishers >的子元素包括了 Publisher 插件的配置信息

在 build loop 结束之后将运行 Publishers，无论 build 是成功还是失败，都会运行 publisher，发布 build 的结果。

<publishers>下面最常用的 publisher 插件是 <currentbuildstatuspublisher>， <email> 和 <artifactspublisher>

<currentbuildstatuspublisher>的主要作用是用来显示 build loop 的结果，包括了编译信息，JUnit 测试结果信息，以及谁做了变更的信息。该插件和<currentstatusbootstrapper>插件访问的文件是同一个文件。

而<email>主要是用来通知使用者。最常用的用法是根据不同的结果发送到不同的邮件列表，如每次 build，无论成功失败都发送给某个邮件列表，还有失败的时候才发送的邮件列表。

<artifactspublisher>用于对创建过程中产生的人工制品进行发布

<dateformat >

<dateformat >指定日期格式，如果配置了这个部分，会修改默认日期格式。

<plugin >

<plugin >注册插件的信息，在 [CC 插件](#)的部分已经讲过。

CruiseControl 应用举例

前面讲了这么多和 CC 有关的基础知识，现在我们将这些知识整合起来，以一个完整的例子作为实践，从实践中体会 CC 的用法。

基础知识:

本例将使用 CC 自带的 helloworld 例子,使用 CVS 作为源码控制系统,ANT 作为 build 工具, Tomcat 作为 Web 容器,因此读者需要有这方面的基础知识。附录提供 VSS 作为源码控制系统的例子。

准备工作:

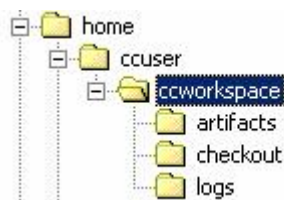
1. 知识准备: 阅读本文前部分的同时还应该去官方网站 cruisecontrol.sourceforge.net 将所有文档阅读一遍,其中 CruiseControl wiki 上的资料非常全面。
2. 安装好相关的软件,如 JDK, TOMCAT, CVS, 配置好环境变量如 JAVA_HOME, TOMCAT_HOME, CATALINA_HOME, CVS_HOME, CVSROOT, PATH 等等。
3. 构建良好的目录结构对于项目的开发和管理都非常重要,我个人喜欢在 windows 下模仿 LINUX 的目录结构来组织自己的目录,例如: c:\home\ccuser 作为我的%HOME%,而 c:\usr\local 作为我的 CVS, ANT, JDK 之类的安装目录,本文也将基于这样的目录组织原则,这样做还有一个好处,对于长期使用 Linux 而不熟悉 Windows 的用户在根据本文实践的时候不会有太大的困难。

安装和准备项目持续集成的环境

准备 workspace

创建一个目录作为我们的 workspace 目录,假设为 c:\usr\ccuser\ccworkspace,下面将用 CC_WORKSPACE 作为对该目录的引用,将来在%CC_WORKSPACE%目录下我们将创建 CC 的配置文件 config.xml 以及 ant 的 wrapper buildfile 文件 build.xml。

首先在这个目录下创建三个目录,如下图:

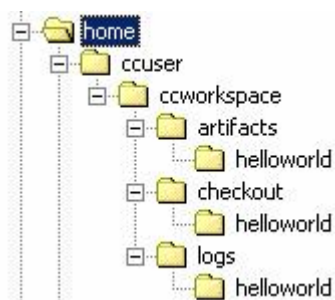


对于这三个目录的解释如下

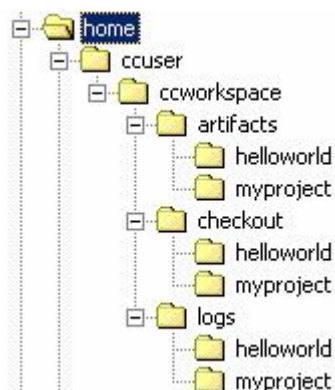
目录名	描述
%CC_WORKSPACE%\checkout	作为 CC 从 CVS 或者其他源码控制系统检出 helloworld 项目的目录
%CC_WORKSPACE%\logs	CC 的 build loop 过程中产生日志所在的目录
%CC_WORKSPACE%\artifacts	CC 在 build loop 过程中产生的人工制品所在的目录

注意,对于我们的每一个项目,都要在这三个目录下创建相应的子目录,子目录的目录名和

项目名是一致的，以 helloworld 为例，如下图：



如果你有多个项目，例如 helloworld 项目和另外一个项目 myproject，目录结构如下图：



下载 CC 并创建 `cruisecontrol.jar`

到官方站点下载一份最新的 CC 压缩包，我将使用版本 2.2 作为例子，同时讲讲和版本 2.1.6 在用法上的区别，将其解压到 `C:\var\local\cruisecontrol-2.2` 下，下面我们将用 `CC_HOME` 来引用这个目录，记得将 `%CC_HOME%\main\bin` 添加到系统的 `%PATH%` 环境变量中去。

`CC_HOME` 目录下有四个子目录 `contrib`, `docs`, `main`, `reporting`，如下



如果你使用的是 2.2 以前的版本，那么只有三个子目录 `docs`, `main`, `reporting`，如下



在 `%CC_HOME%/main/docs/helloworld` 下有一个例子，我们以后将直接用这个例子来作试

验，如下：



如果你使用的是 2.2 版本，那么在`%CC_HOME%\main\dist`目录下有一个 `cruisecontrol.jar` 文件。

如果你使用的是 2.1.6 或者以前的版本，就需要自己手工创建这个文件，创建方法如下：
运行 `%CC_HOME%\main\build.bat`，该命令会编译，测试，并创建 `%CC_HOME%\main\dist\cruisecontrol.jar`。

现在运行 `%CC_HOME%\main\bin\cruisecontrol.bat` 或者 `%CC_HOME%\main\dist\java -jar cruisecontrol.jar`，你会得到如下信息

```
C:\WINNT\system32\cmd.exe
C:\usr\local\cruisecontrol-2.2\main\dist>java -jar cruisecontrol.jar
[cc]一月-11 16:00:28 Main          - CruiseControl Version 2.2
[cc]一月-11 16:00:28 Main          - Config file not found: config.xml
[cc]一月-11 16:00:28 Main          - Usage:
[cc]一月-11 16:00:28 Main          -
[cc]一月-11 16:00:28 Main          - Starts a continuous integration loop
[cc]一月-11 16:00:28 Main          -
[cc]一月-11 16:00:28 Main          - java CruiseControl [options]
[cc]一月-11 16:00:28 Main          - where options (all optional) are:
[cc]一月-11 16:00:28 Main          -
[cc]一月-11 16:00:28 Main          - -port [number]           where number is the
port of the Controller web site; defaults to 8000
[cc]一月-11 16:00:28 Main          - -rmiport [number]       where number is the
RMI port of the Controller; defaults to 1099
[cc]一月-11 16:00:28 Main          - -xslpath directory     where directory is l
ocation of jmx xsl files; defaults to files in package
[cc]一月-11 16:00:28 Main          - -configfile file       where file is the co
nfiguration file; defaults to config.xml in the current directory
[cc]一月-11 16:00:28 Main          - -debug                 to set the internal
logging level to DEBUG
[cc]一月-11 16:00:28 Main          -
[cc]一月-11 16:00:28 Main          - Please keep in mind that the JMX server wil
l only be started if you specify -port and/or -rmiport
```

同时当前目录下会新增一个名为 `cruisecontrol.log` 的文件，该文件的内容和命令控制台得到的输出信息一样，恭喜你，第一步成功了。

补充：

在命令行运行 `cruisecontrol.bat` 的时候 CC 会在当前目录下自动查找是否存在 `config.xml` 文

件。你也可以直接在命令行通过 `-configfile` 和 `-projectname` 分别定 CC 所要访问的配置文件的
位置和所运行的项目的名字。

创建 `cruisecontrol.war`

如果你用的是 CC2.2 版本，那么在 `%CC_HOME%\reporting\jsp\dist` 目录下已经有一个 `cruisecontrol.war` 文件，你可以直接将这个文件 COPY 到你的 `%TOMCAT_HOME%\webapps` 目录下，不过这种方式通常都会出错，前面我们讲过 CC 的 WEB 组件要访问我们的项目 build 的状态文件，而下载的 CC2.2 里面自带的 `cruisecontrol.war` 并不知道你的状态文件的位置，（当然你可以打开 `cruisecontrol.war` 文件，直接修改 WEB-INF 目录里面的 `web.xml` 文件里的配置）。

下面是重新创建 `cruisecontrol.war` 的步骤：

假使我们用 Tomcat 4.1.30 作为 JSP 服务器，该版本的 Tomcat 支持 JSP1.2 规范，因此需要修改 `%CC_HOME%\reporting\jsp\buildresults.jsp` 文件

将

```
<%@ taglib uri="/WEB-INF/cruisecontrol-jsp11.tld" prefix="cruisecontrol"%>
```

修改为

```
<%@ taglib uri="/WEB-INF/cruisecontrol-jsp12.tld" prefix="cruisecontrol"%>
```

在创建 `cruisecontrol.war` 之前，我们需要传递三个参数给创建程序，分别用来指定 CC 产生的日志的目录（`user.log.dir`），创建过程中产生的状态文件的名称（`user.build.status.file`），还有创建过程中输出的人工制品所在的目录（`cruise.build.artifacts.dir`）。以我们前面创建的 `%CC_WORKSPACE%` 的目录结构为例，其对应的值如下：

参数名	参数值	说明
<code>user.log.dir</code>	<code>%CC_WORKSPACES%\logs</code>	
<code>user.build.status.file</code>	<code>status.txt</code>	该文件名应该和后面我们要创建的 CC 的 <code>config.xml</code> 文件中的 <code><currentbuildstatusbootstrapper></code> 中需要指定的状态文件的值一致
<code>cruise.build.artifacts.dir</code>	<code>%CC_WORKSPACE%\artifacts</code>	

然后运行以下命令：

```
%CC_HOME%\reporting\jsp\build -Duser.log.dir=%CC_WORKSPACES%\logs  
-Duser.build.status.file=status.txt -D cruise.build.artifacts.dir=%CC_WORKSPACE%\artifacts
```

还有一种方法，就是在该目录下创建一个新文件 `override.properties`，文件内容如下：

```
user.log.dir=%CC_WORKSPACE%\logs  
user.build.status.file=status.txt  
cruise.build.artifacts.dir=%CC_WORKSPACE%\artifacts
```

然后直接运行不带参数的 `build` 命令，如下：

```
%CC_HOME%\reporting\jsp\build
```

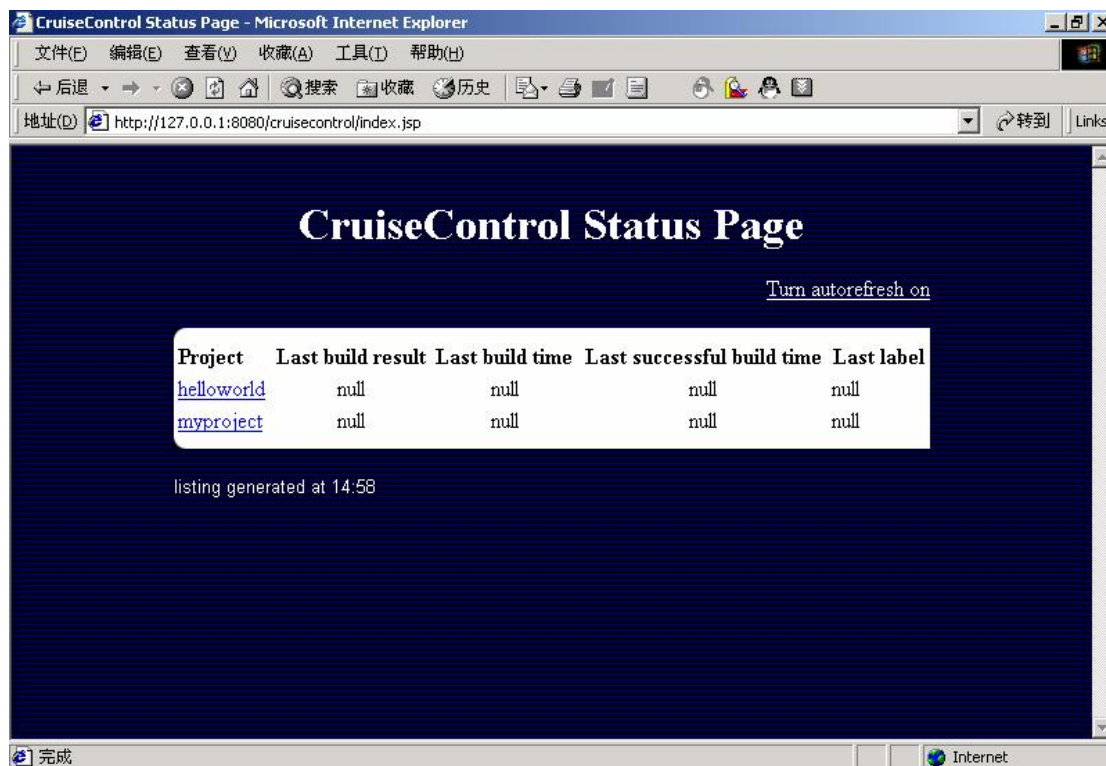
注意：

如果在执行 `%CC_HOME%\reporting\jsp\build` 命令的时候既不提供这三个参数

值，又不通过 `override.properties` 文件来提供这三个参数值，那么在创建的过程中会要求你输入这三个参数相应的值。

成功创建之后，将在 `dist` 目录下找到 `cruisecontrol.war` 文件，将其 COPY 到 `%TOMCAT_HOME%\webapps` 目录下，然后启动 tomcat 服务，CC 的 web 用户接口部分的准备工作就算大功告成。

最后我们可以在浏览器输入 <http://127.0.0.1:8080/cruisecontrol/index.jsp> 来验证一下结果（假设 tomcat 采用默认的设置，并没有更改端口号 8080 等等）



我们可以看到有两个项目，一个是 helloworld，另一个是 myproject，这和[准备 workspace](#) 创建的目录完全一致。

例子 Hello World!

将 helloworld 导入 CVS

我们以 cruisecontrol 自带的 helloworld 为例，该例子存在于 `%CC_HOME%\main\docs\helloWorld` 目录下，建议将其复制到你的 `%CC_WORKSPACE%\helloworld` 目录下。

然后进入 `%CC_WORKSPACE%\helloworld` 目录，只保留该目录下的 `cvs-build.xml` 文件和 `src` 子目录下的所有文件，因为这个例子是以 CVS 作为源码控制系统，为避免混淆将其他的文件删除。

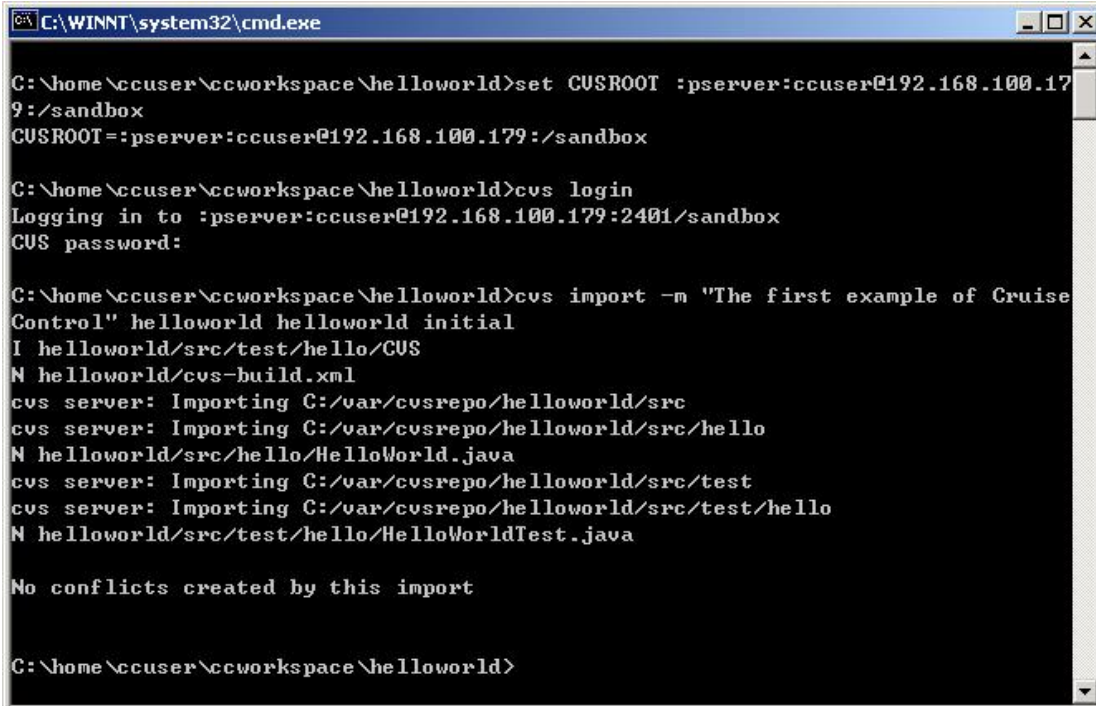
打开 cvs-build.xml 文件，我们注意其中的两个 target， all 是不依赖 CVS 而独立进行创建的 target，而 masterbuild 则要先访问 CVS 做 checkout 然后再做创建。因为我们将使用 wrapper buildfile 模式，在以后将会使用到 all 这个 target。

```
<target name="all" depends="init,clean,compile,test,jar" description="Build application"/>
<target name="masterbuild" depends="checkout,compile,test,jar"
description="Cruise control master build"/>
```

还有一点需要注意的是 CC 自带的 helloworld 例子 里面的 cvs-build.xml 使用 jikes 作为编译器，如果你没有 jikes，那么需要将相应的部分注释掉，如下

```
<!--property name="build.compiler" value="jikes"/-->
<!--property name="build.compiler.emacs" value="true"/-->
```

接下来准备将修改过的 helloworld 导入到你的 CVS，假设 CVS 服务器 ip 为 192.168.100.179，源码库为/sandbox，用户名为 ccuser，密码为 user4test，使用 pserver 协议来访问 repository（注意 pserver 是不安全的协议，本例只是示范作用，建议使用安全的访问协议），执行以下命令：

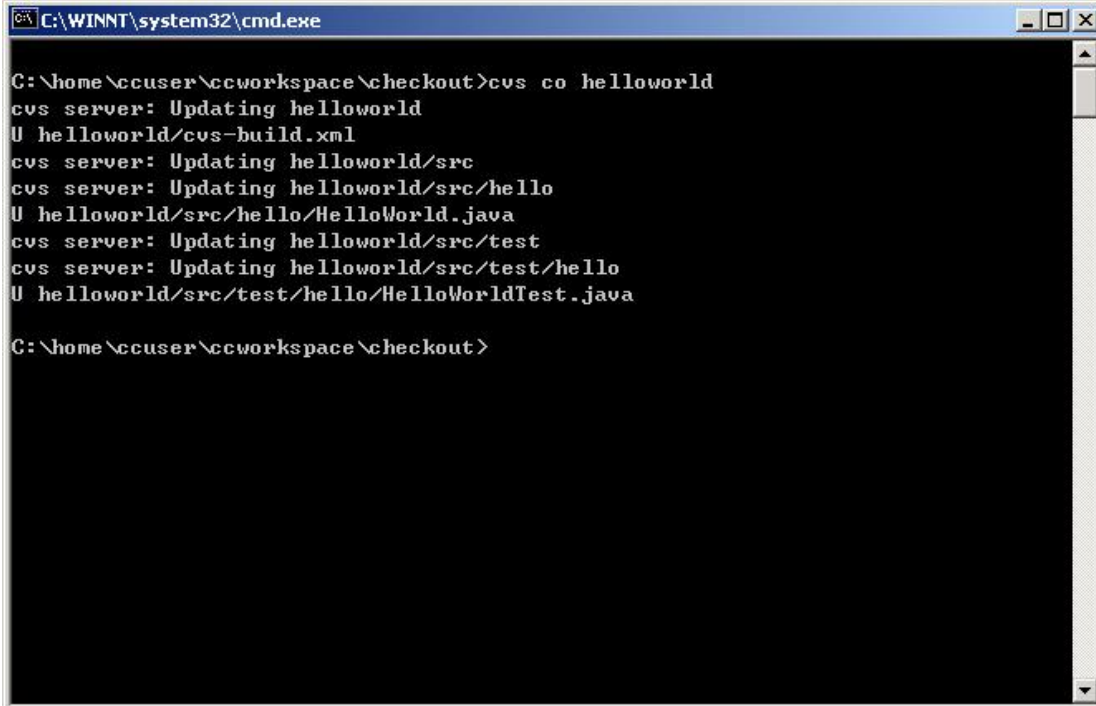


```
C:\WINNT\system32\cmd.exe
C:\home\ccuser\ccworkspace\helloworld>set CUSRROOT :pserver:ccuser@192.168.100.179:/sandbox
CUSRROOT=:pserver:ccuser@192.168.100.179:/sandbox
C:\home\ccuser\ccworkspace\helloworld>cvs login
Logging in to :pserver:ccuser@192.168.100.179:2401/sandbox
Cvs password:
C:\home\ccuser\ccworkspace\helloworld>cvs import -m "The first example of Cruise Control" helloworld helloworld initial
I helloworld/src/test/hello/CUS
N helloworld/cvs-build.xml
cvs server: Importing C:/var/cvsrepo/helloworld/src
cvs server: Importing C:/var/cvsrepo/helloworld/src/hello
N helloworld/src/hello/HelloWorld.java
cvs server: Importing C:/var/cvsrepo/helloworld/src/test
cvs server: Importing C:/var/cvsrepo/helloworld/src/test/hello
N helloworld/src/test/hello/HelloWorldTest.java

No conflicts created by this import

C:\home\ccuser\ccworkspace\helloworld>
```

确定将 helloworld 导入 CVS 之后，将%CC_WORKSPACE%\helloworld 目录删除，切换到 %CC_WORKSPACE%\checkout 目录下，并检出该项目，命令如下：



```
C:\WINNT\system32\cmd.exe

C:\home\ccuser\ccworkspace\checkout>cvs co helloworld
cvs server: Updating helloworld
U helloworld/cvs-build.xml
cvs server: Updating helloworld/src
cvs server: Updating helloworld/src/hello
U helloworld/src/hello/HelloWorld.java
cvs server: Updating helloworld/src/test
cvs server: Updating helloworld/src/test/hello
U helloworld/src/test/hello/HelloWorldTest.java

C:\home\ccuser\ccworkspace\checkout>
```

准备配置文件

现在在%CC_WORKSPACE%目录下面添加 CC 的配置文件 cvs-config.xml, 内容如下:

```
<cruisecontrol>
  <project name="helloworld">
    <bootstrappers>
      <currentbuildstatusbootstrapper
file="logs/helloworld/status.txt" />
    </bootstrappers>
    <modificationset quietperiod="30" >
      <cvs localworkingcopy="checkout/helloworld"/>
    </modificationset>
    <schedule interval="60" >
      <ant antscript="C:/usr/local/ant/bin/ant.bat"
        antworkingdir="C:/home/ccuser/ccworkspace"
        buildfile="cvs-build-wrapper.xml" target="cleanbuild"
multiple="5" />
      <ant antscript="C:/usr/local/ant/bin/ant.bat"
        antworkingdir="C:/home/ccuser/ccworkspace"
        buildfile="cvs-build-wrapper.xml" target="masterbuild"
multiple="1" />
    </schedule>
    <log dir="logs/helloworld">
      <merge dir="checkout/helloworld/test-results"/>
    </log>
  </project>
</cruisecontrol>
```

```
</log>
<publishers>
  <currentbuildstatuspublisher
file="logs/helloworld/status.txt" />
  <artifactspublisher dir="checkout/helloworld/dist"
    dest="artifacts/helloworld"/>
    <!--email mailhost="REPLACE WITH MAILHOST"
returnaddress="REPLACE WITH RETURN EMAIL ADDRESS" defaultsuffix=""
buildresultsurl="http://localhost:8080/cruisecontrol/buildresults">
  <always address="REPLACE WITH BUILD MASTER EMAIL ADDRESS" />
  <failure address="REPLACE WITH FAILURE EMAIL ADDRESS" />
  </email-->
  </publishers>
</project>
</cruisecontrol>
```

关于这个文件的说明如下：

1. 该配置文件里面的相对路径都是相对 `cvconfig.xml` 文件所在目录，也就是 `%CC_WORKSPACE%`。
2. `<currentbuildstatusbootstrapper>` 和 `<currentbuildstatuspublisher>` 指定的文件为 `logs/helloworld/status.txt`，这和我们[创建 cruisecontrol.war](#)的时候指定的 `user.log.dir` 和 `user.build.status.file` 的值是一致的。
3. `<cvcs>`元素的属性 `localworkingcopy` 值为 `checkout/helloworld`，因为我在此之前已经将 `helloworld` 项目 `check out` 到该目录下，因此 `CC` 的 `cvcs` 插件会自动在该目录下寻找相关的 `CVSROOT` 的信息。
4. `<log>`合并日志的源目录为 `checkout/helloworld/test-results`，该目录是 `helloworld` 项目自身创建过程中产生 `junit` 测试结果报表的目录，`<log>`合并日志后的目录为 `logs/helloworld`，该值和我们[创建 cruisecontrol.war](#)的时候指定的 `user.log.dir` 的值是一致的。
5. 为了说明`<artifactspublisher>`的用法，我们把创建过程中产生的 `jar` 文件作为人工制品，`helloworld`项目自身创建产生的 `jar` 文件保存在 `checkout/helloworld/dist` 目录下，每次 `build loop` 过程中`<artifactspublisher>`会自动在 `artifacts/helloworld` 目录下创建一个子目录，并将人工制品复制到这个目录下，而 `artifacts/helloworld` 和我们[创建 cruisecontrol.war](#)的时候指定的 `cruise.build.artifacts.dir` 的值是一致的。
6. 如果读者需要邮件通知的功能，需要取消`<email>`元素的注释，并将各属性值用实际的值代替。

还需要在`%CC_WORKSPACE%`目录下添加一个 `wrapper buildfile` 文件 `cvcs-build-wrapper.xml`，该文件会调用 `checkout/helloworld` 目录下的 `cvcs-build.xml` 文件，内容如下：

```
<?xml version="1.0"?>
<project name="helloworld" default="masterbuild" basedir=".">
  <property name="checkout.dir" value="checkout"/>
  <property name="helloworld.dir"
value="{checkout.dir}/helloworld"/>
  <property name="cvcsroot"
```

```
value=":pserver:ccuser@192.168.100.179:/sandbox"/>
  <target name="checkout" description="Update package from CVS">
    < cvs package="helloworld" cvsroot="${cvsroot}"
dest="${checkout.dir}"/>
  </target>
  <target name="masterbuild" depends="checkout">
    < ant inheritall="false" antfile="cvs-build.xml"
dir="${helloworld.dir}"
      target="all"/>
  </target>
  <target name="cleanbuild" depends="checkout">
    < ant inheritall="false" antfile="cvs-build.xml"
dir="${helloworld.dir}"
      target="clean"/>
  </target>
</project>
```

注意，在<ant>任务里面我们指定了 `dir`，这个值将作为被调用的 `cvs-build.xml` 的 `basedir`，而且我们将 `inheritall` 设为 `false`，这样被调用的 `cvs-build.xml` 将具备独立性，我们在这个文件中的配置信息将不会影响到将要调用的 `cvs-build.xml`。

启动 CC

一切准备就绪，下面在 `%CC_WORKSPACE%` 目录下运行 `cruisecontrol` 命令就可以启动我们的 CruiseControl 了，这次通过命令行参数 `-configfile` 来指定 `cvs-config.xml` 文件。

```
%CC_WORKSPACE%>cruisecontrol -configfile cvs-config.xml
```

命令行窗口如下（因为创建过程会显示很多信息，这里只提供部分显示信息）：

```

C:\home\ccuser\ccworkspace>cruisecontrol -configfile cvs-config.xml
"D:\j2sdk1.4.1_06\bin\java" -cp "D:\j2sdk1.4.1_06\lib\tools.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\dist\cruisecontrol.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\log4j.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\jdom.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\ant\ant.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\ant\ant-launcher.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\xerces.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\xalan.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\jakarta-oro-2.0.3.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\mail.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\junit.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\activation.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\commons-net-1.1.0.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\starteam-sdk.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\mx4j.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\mx4j-tools.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\mx4j-remote.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\smack.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\comm.jar;C:\usr\local\cruisecontrol-2.2\main\bin\..\lib\x10.jar;." -Djavax.management.builder.initial=mx4j.server.MX4JMBBeanServerBuilder CruiseControl -configfile cvs-config.xml
[cc]二月-22 16:26:12 Main - CruiseControl Version 2.2
[cc]二月-22 16:26:12 trolController- projectName = [helloworld]
[cc]二月-22 16:26:13 Project - Project helloworld: reading settings from config file [C:\home\ccuser\ccworkspace\cvs-config.xml]
[cc]二月-22 16:26:13 BuildQueue - BuildQueue started
[cc]二月-22 16:26:13 Project - Project helloworld starting
[cc]二月-22 16:26:13 Project - Project helloworld: idle
[cc]二月-22 16:26:13 Project - Project helloworld started
[cc]二月-22 16:26:13 Project - Project helloworld: next build in 1 minutes
[cc]二月-22 16:26:13 Project - Project helloworld: waiting for next time to build

```

创建结束之后，我们会在命令行窗口得到如下信息：

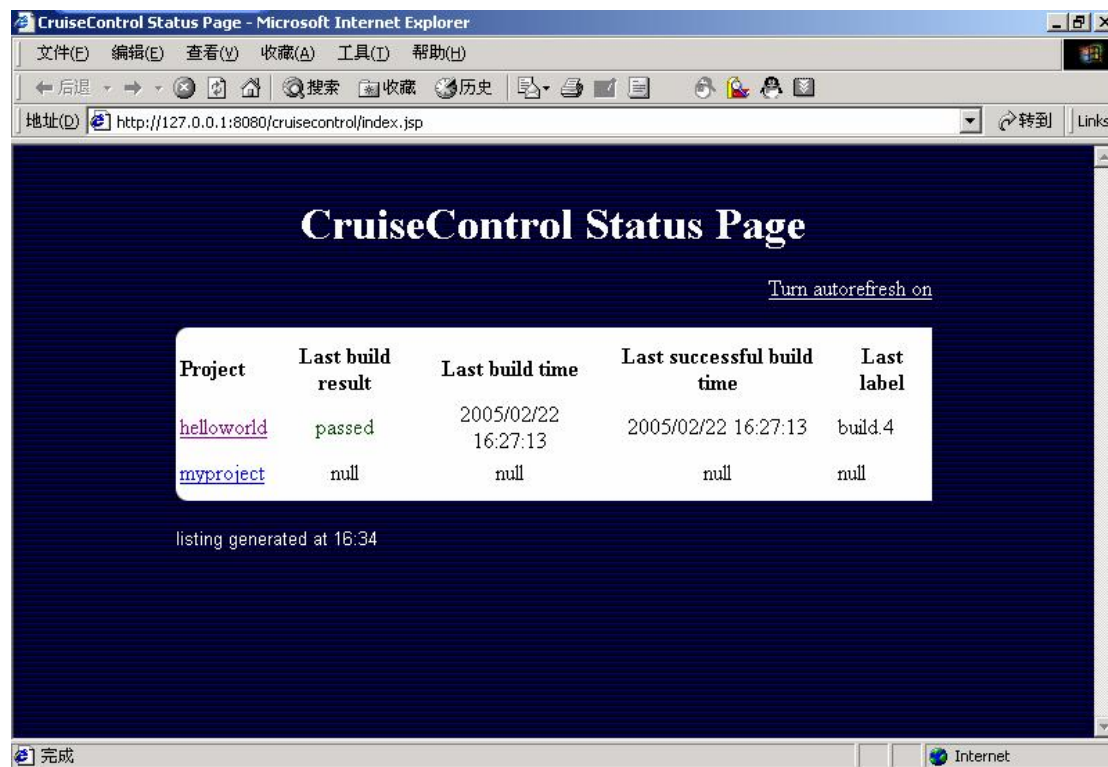
```

[javac] Compiling 2 source files to C:\home\ccuser\ccworkspace\checkout\helloworld\classes
test:
[mkdir] Created dir: C:\home\ccuser\ccworkspace\checkout\helloworld\test-results
[junit] Testsuite: test.hello.HelloWorldTest
[junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 1.432 sec
[junit] Testcase: testSayHello took 0.02 sec
jar:
[jar] Building jar: C:\home\ccuser\ccworkspace\checkout\helloworld\dist\helloworld.jar
all:
BUILD SUCCESSFUL
Total time: 13 seconds
[cc]二月-22 16:27:33 Project - Project helloworld: merging accumulated logging files
[cc]二月-22 16:27:33 Project - Project helloworld: build successful
[cc]二月-22 16:27:33 Project - Project helloworld: publishing build results
[cc]二月-22 16:27:34 Project - Project helloworld: idle
[cc]二月-22 16:27:34 Project - Project helloworld: next build in 1 minutes
[cc]二月-22 16:27:34 Project - Project helloworld: waiting for next time to build

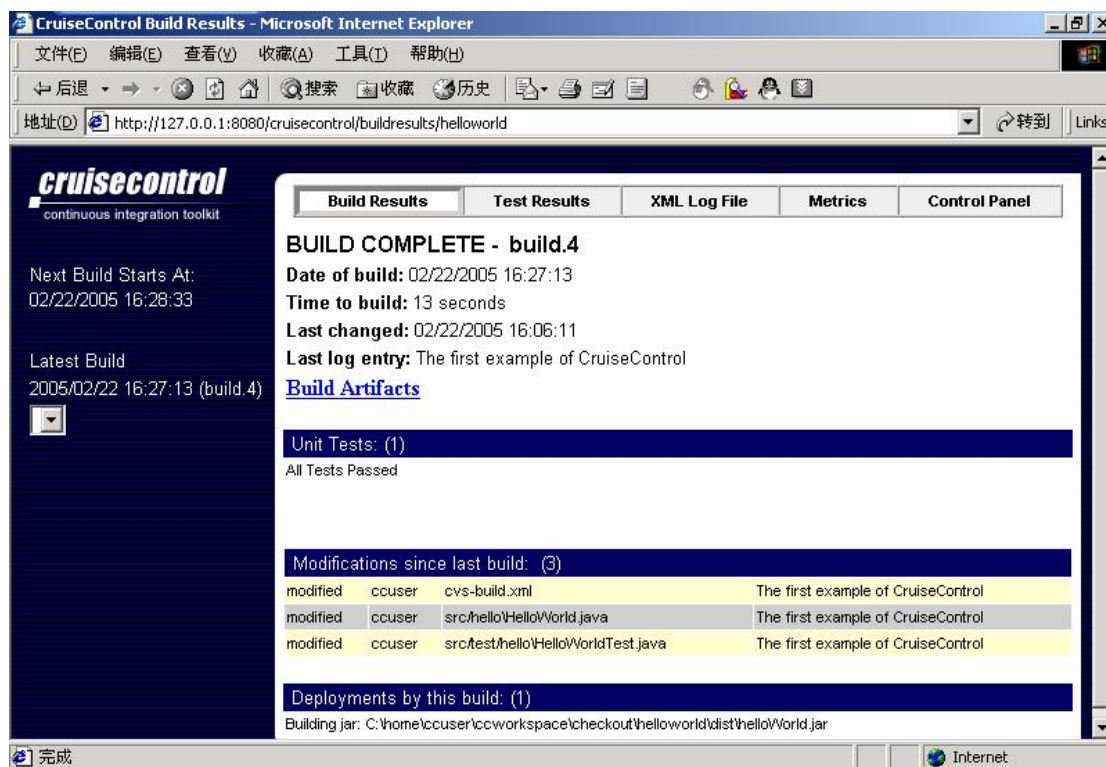
```

如果读者配置了邮件通知功能，那么在 `cvs-config.xml` 文件里面 `<email>` 元素下配置的用户会收到邮件，邮件内容是创建的结果的 `http` 超链接，点击该链接就可以查看创建的结果，如果没有配置邮件通知功能，可以通过下面的步骤查看结果：

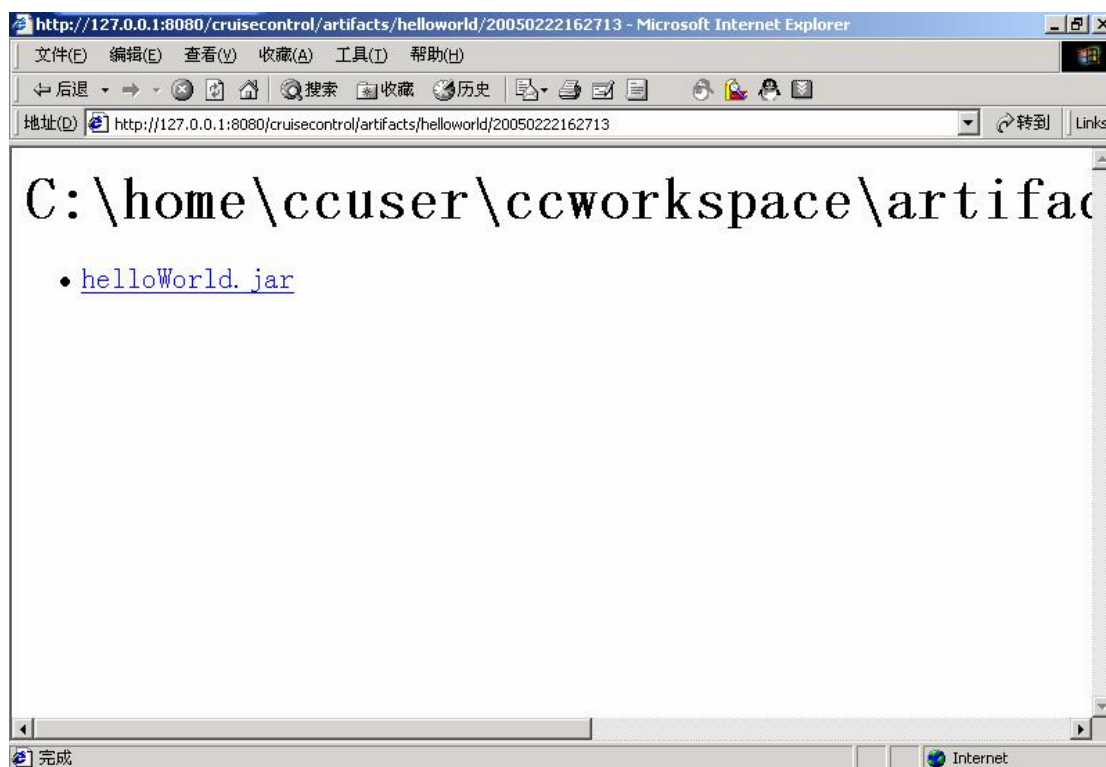
首先打开链接 <http://127.0.0.1:8080/cruisecontrol/index.jsp>，我们发现 `helloworld` 对应的属性值改变了，例如 Last build result 不再是刚开始的 `null` 值了，而是 `passed`，如下图：



点击项目列表中的 `helloworld`，如下图，其中右边的白色框里面列出了最近一次创建的摘要信息，例如创建的日期（**Date of build**），总共花了多少时间（**Time to build**），还有测试结果（**Unit Tests**）等等，左边 **Latest Build** 下面列出了历次创建的时间摘要：



继续点击 [Build Artifacts](#) 之后结果如下，我们看到创建过程中产生的 `helloWorld.jar` 文件作为人工制品被列出来了：



至此一个完整的使用 CC 进行持续集成的例子就结束了，适当的进行变化，例如采用不同的源码控制系统，导入自己的项目文件，定制项目的持续集成的时间表(<schedule>)，就可以

应用到实际的项目中去，[附录一](#)提供了采用 VSS（Visual SourceSafe）源码控制系统对 helloworld 这个例子进行持续集成需要做调整的一些步骤。

结束语

通过本文对持续集成的概念的讨论，并提供了结合 CruiseControl 这个先进的持续集成工具进行了实际演练的例子，读者对持续集成应该有了一个理性和感性的认识，只要在实际项目中多实践，就可以充分体会到持续集成的自动化和及时反馈给项目带来的好处，关于 CC 的更多的技巧需要读者在实践中摸索，本文只是起到抛砖引玉的作用。

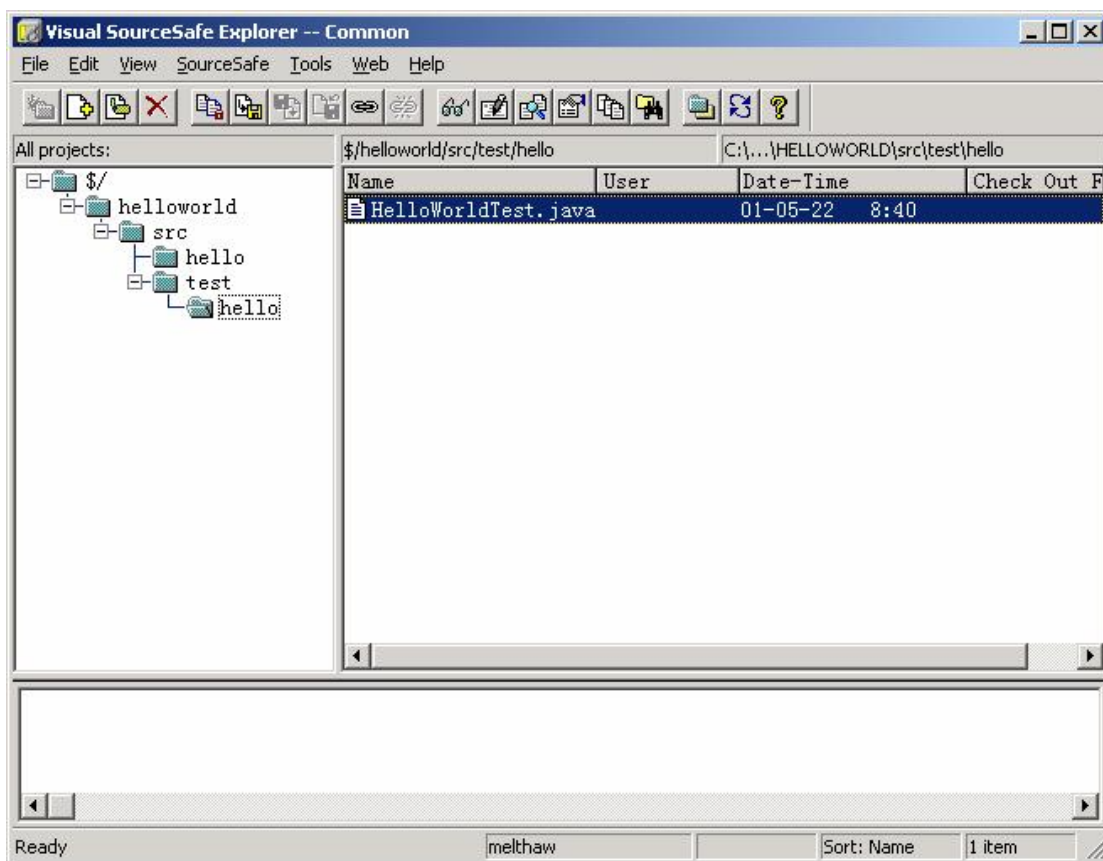
附录一 Hello World!与 VSS

下面是 Hello World!这个例子针对 VSS(Visual SourceSafe)源码控制系统所做的调整，除了下面的两个步骤其他和正文提供的例子完全一样。

将 helloworld 导入 VSS

方法同导入 CVS 的类似，只保留 vss-build.xml 文件和 src 子目录下的所有文件，将其他的文件删除。同样，如果你没有使用 jikes 编译器，修改 vss-build.xml 文件，将 jikes 有关的内容注释掉。

接下来准备将修改过的 helloworld 导入到你的 VSS，假设 VSS 服务器已经配置好，ip 为 192.168.100.179，而 VSS 服务器的 srcsafe.ini 所在的共享目录为 \\192.168.100.179\VSSREPO，用户名和密码分别为 vssuser 和 user4test，导入后的结果如下图：



准备配置文件

首先我们要确定 VSS 使用的日期格式，我测试用的环境的日期格式是 yy-MM-dd，时间格式是 HH:mm，如上图 VSS Explorer 里面 HelloWorldTest.java 的日期和时间分别是 01-05-22 和 8:40。

我们需要确定 VSS 客户端也就是 ss.exe 文件所在的目录，需要注意的是，用户的 VSS 客户端安装的路径中不能有空格，例如安装在“C:\Program Files\DevelopTools\Microsoft Visual Studio\”这样的目录下就会有问题。

假设 VSS 客户端安装在 C:\usr\local\Microsoft\VisualStudio\Common\VSS 下面，在 %CC_WORKSPACE% 下面添加 CC 的配置文件 vss-config.xml，内容如下（同样，如果需要邮件通知功能就要替换 <email> 里面的值）：

```
<cruisecontrol>
  <project name="helloworld">
    <bootstrappers>
      <currentbuildstatusbootstrapper
file="logs/helloworld/status.txt" />
    </bootstrappers>
    <modificationset quietperiod="30" >
      <vss
ssdir="C:/usr/local/Microsoft/VisualStudio/Common/VSS/win32"
      login="vssuser,user4test" vsspath="/helloworld"
      serverPath="//192.168.100.179/vssrepo"
```

```

dateformat="yy-MM-dd"
    timeformat="HH:mm" />
</modificationset>
<schedule interval="60" >
    <ant antscript="C:/usr/local/ant/bin/ant.bat"
        antworkingdir="C:/home/ccuser/ccworkspace"
        buildfile="vss-build-wrapper.xml" target="cleanbuild"
multiple="5" />
    <ant antscript="C:/usr/local/ant/bin/ant.bat"
        antworkingdir="C:/home/ccuser/ccworkspace"
        buildfile="vss-build-wrapper.xml" target="masterbuild"
multiple="1" />
</schedule>
<log dir="logs/helloworld">
    <merge dir="checkout/helloworld/test-results"/>
</log>
<publishers>
    <currentbuildstatuspublisher
file="logs/helloworld/status.txt" />
    <artifactspublisher dest="artifacts/helloworld"
        dir="checkout/helloworld/dist"/>
        <!--email mailhost="REPLACE WITH MAILHOST"
returnaddress="REPLACE WITH RETURN EMAIL ADDRESS" defaultsuffix=""
buildresultsurl="http://localhost:8080/cruisecontrol/buildresults">
        <always address="REPLACE WITH BUILD MASTER EMAIL ADDRESS" />
        <failure address="REPLACE WITH FAILURE EMAIL ADDRESS" />
        </email-->
    </publishers>
</project>
</cruisecontrol>

```

在%CC_WORKSPACE%目录下添加一个 wrapper buildfile 文件 vss-build-wrapper.xml, 除了访问 VSS 的 vssget 和正文的例子里面访问 CVS 的 checkout 不同以外, 其他都一样。

```

<?xml version="1.0"?>
<project name="helloworld" default="masterbuild" basedir=".">
    <property name="vsspath" value="/helloworld"/>
    <property name="serverPath" value="//192.168.100.179/vssrepo"/>
    <property name="ssdir"
        value="C:/usr/local/Microsoft/VisualStudio/Common/VSS/win32"/>
    <property name="username" value="vssuser"/>
    <property name="password" value="user4test"/>
    <property name="localpath" value="checkout/helloworld"/>
    <target name="vssget">
        <vssget vsspath="${vsspath}" login="${username},${password}"

```

```
        localpath="${localpath}" ssdir="${ssdir}"
        serverpath="${serverPath}" recursive="true"/>
    </target>
    <target name="cleanbuild">
        <ant dir="${localpath}" antfile="vss-build.xml"
target="cleanbuild"
        inheritall="false"/>
    </target>
    <target name="masterbuild" depends="vssget">
        <ant dir="${localpath}" antfile="vss-build.xml"
target="masterbuild"
        inheritall="false"/>
    </target>
</project>
```

最后运行以下命令来启动 CC 就可以了

```
%CC_WORKSPACE%>cruisecontrol -configfile vss-config.xml
```

附录二 Out Of Memory Error 的解决方案

在实际项目中用 CC 来做持续集成很容易遇到 Out Of Memory 错误，引起这个错误的原因很多，一个最常见的例子是 CC 的<log>插件合并日志文件的时候，如果要合并的日志太大，CC 会报告一个 Out Of Memory 错误，还有 ANT 的一些任务，如<javac>，<jar>，会占用很多的内存，可以通过给 CC 的配置文件中的<ant>元素添加子元素<jvmarg>来增加分配给 ANT 的内存

```
<ant>
    <jvmarg arg="-server" />
    <jvmarg arg="-Xms64m" />
    <jvmarg arg="-Xmx256m" />
</ant>
```

参考资料

CruiseControl 官方网站: <http://cruisecontrol.sourceforge.net>

CruiseControl wiki (学习 CruiseControl 的最佳站点):

<http://confluence.public.thoughtworks.org/display/CC/Home>

Martin Fowler 的 Continuous Integration:

<http://martinfowler.com/articles/continuousIntegration.html>

Gigix 翻译的持续集成: <http://dev.csdn.net/develop/article/12/12286.shtm>

Ant 官方网站: <http://ant.apache.org>

CVS 官方网站: <https://www.cvshome.org/>

软件质量之路 (二) 日创建:

http://www-900.ibm.com/developerWorks/cn/linux/software_engineering/l-frmwk/index2.shtml