

持续集成环境在项目管理中的应用

Nevaeheerf 2009 年 08 月 03 日星期一

目录

一、前言	3
二、依赖工具集合	4
三、代码管理.....	4
1、工具选择	4
2、版本目录结构	5
3、项目内目录结构.....	8
(1)Document.....	9
(2)Setup:	9
(3)Src:.....	10
(4)Test:.....	10
(5)releasebuild.bat:	10
4、版本库钩子	11
5、公共引用库的管理.....	12
1)公共引用库的设置	12
2)公共引用库的更新	14
四、测试驱动开发模式的引入.....	16
五、Wix 打包脚本的编写	22

六、MSBuild 脚本的编写	22
七、持续集成环境的搭建	22
1、持续集成介绍	22
2、持续集成搭建工具介绍	23
3、持续集成脚本的编写	33
1)触发持续集成的方式	33
2)标签的定义	33
3)从版本库签出代码	34
4)执行集成步骤	35
(1)编译解决方案.....	36
(2)自动化测试.....	36
(3)代码分析	37
(4)修改程序集的版本号	39
(5)部署程序集.....	50
(6)MSI 安装包	52

一、前言

目前网络上介绍敏捷开发的资料有很多，相关的书籍也有一些，我强烈推荐[《敏捷软件开发—原则、模式与实践》](#)这本书。

在敏捷开发模式中，持续集成是一个很重要的实践，我在最初接触这个概念的时候的认知是片面的，认为它只是借助一个工具把开发过程中的各个环节都串联起来，实现自动化而已，但是当我尝试着在自己负责的开发团队内部推行持续集成的时候，发现要处理的问题有很多，涉及到很多个方面，比如代码目录组织，版本控制中的权限分配，钩子脚本的编写，测试驱动模式的引入，MSBuild 脚本的编写，Wix 打包脚本的编写等，需要掌握多方面的知识，多亏网络上很多同行的文章帮助，目前来说，我的问题都解决了，而且自己确实觉得这个环境的搭建为我们整个团队带来了很大的好处，从团队人员相互之间的协作效率的提高，到代码质量的提高，进而到产品的稳定，产品维护成本的下降。都有很大的提升。我想肯定有很多兄弟也会碰到我曾经遇到过的问题，所以我决定把自己搭建持续集成的过程记录下来，如果能对大家的工作有所帮助，我将很高兴。

我会把和项目管理或者持续集成相关的部分尽量写详细，其他部分不会展开，比如SubVersion 的如何安装为服务，配置使用，展开的话就太多了，有兴趣的同学，可以去网络上找到相关的文章，我会尽量都给出参考文档的相关连接。

我首先会介绍持续集成依赖的几个技术点，包括代码目录结构，版本库的设置，测试驱动的引入，MSBuild 脚本的编写，Wix 脚本的编写，然后才介绍持续集成的搭建，并以几个实际在我的开发过程中在应用的例子来给大家说明持续集成脚本的编写。

感谢我曾经的部门经理，涛哥，从你那里我学到了很多很多，不光是技术，还有考

虑问题的方式，还有对待工作的态度：**静下心来，做好自己手头的事儿，
沉浸其中，享受整个过程！**

二、依赖工具集合

- 1、版本控制工具：[Subversion](#)
- 2、版本控制前台客户端：[TortoiseSVN](#)
- 3、代码校验工具：[FxCopSetup](#)
- 4、测试驱动工具：[NUnit](#)
- 5、测试驱动 Visual Studio 下面插件：[TestDriven.NET](#)
- 6、打包工具：[Wix](#)
- 7、持续集成环境搭建：[CruiseControl.NET](#)
- 8、开源 Wiki：[ScrewTurn Wiki](#)

三、代码管理

1、工具选择

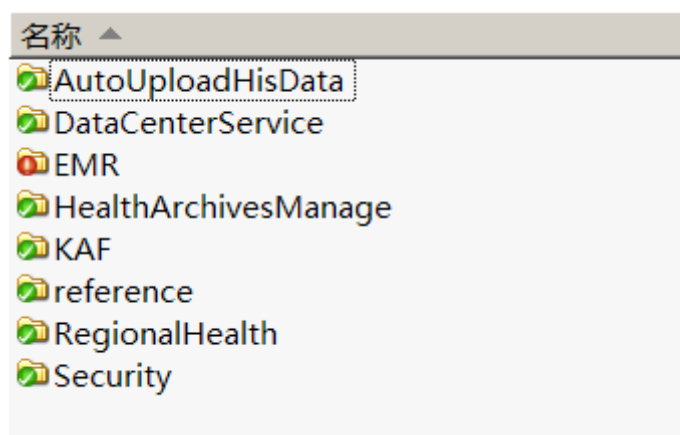
我选择的版本控制工具是 Subversion，以前使用的是 VSS(Visual SourceSafe)，VSS 是基于“锁定--编辑--解锁”模式的，这个模式有一个弊端，就是当其他人在编辑相关单元文件的时候，此单元文件处于锁定状态，其他人如果想编辑这个单元文件的话，只能处于等待状态。后来在持续集成中我使用的 Subversion 替代 VSS，Subversion 是基于“修改--冲突--合并”的一个模式，也就是说多个人可以同时签出一个单元文件，编辑然后提交，如果多个人都修改了同一文件的某一行的话，就会发生冲突，手工

解决冲突。我自己的体会是，这种情况极少发生，更多的时候是：a 同学编辑了 File1 文件的上半部分，b 同学在 File1 文件的最后又添加了一些代码，这样是会造成冲突的，而且本身这种多人编辑同一个文件的情况就很少。b 同学不用因为 a 同学在编辑 File1 文件而只能等待，他也能同时编辑 File1 文件，直观的好处就是这样能提高开发速度。而且 Subversion 提供的配置文件非常容易掌握。可以快速配置出对于整个开发团队成员负责模块的读写权限配置，而且可以有组的概念[groups]，这就直接对应了团队中以小组为划分的结构。而且有很多 Subversion 的扩展工具可以选择，我选择的是 TortoiseSVN，我们都称它为“小乌龟”，这是一个和 Windows 操作系统 Shell 绑定的工具，它把所有对 Subversion 的操作都包装成了可视化操作，而且由于是和操作系统绑定的，任何一个目录都可以成为存放代码副本的地方，个人觉得非常好用，而且还有很多为 Subversion 编写的钩子脚本，可以让我们在版本控制这一块做的更好，后面有一个例子来说明我是如何使用钩子脚本的。

Subversion 的配置脚本的编写如果展开的话，会占用很多篇幅，所以在此不做过多描述了，有兴趣的同学可以去看一下 SubVersion 的 [PDF](#) 教程，全中文的，写的非常详细。

2、版本目录结构

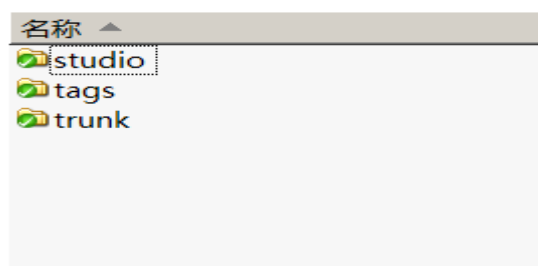
先给大家看一下我组织的版本结构，如图一：



<图一>

图一中就是我搭建的多个版本库中的其中一个的根目录，在搭建版本库的时候，尽量不要往里面放置太多的项目，这样会造成整个版本库的庞大笨重，如果在根目录更新会非常缓慢。建议可以按照业务逻辑划分的方式组织多个版本库。

我选取其中的一个项目 KAF 为例子来进一步说明我的版本分配结构：



<图二>

如图二所示，这是我负责的一个项目 KAF(KWMK Application Framework)目录的结构，KAF 是我们公司所有产品的底层框架，所有的项目的代码结构都是这样组织的：一个主版本(trunk)，一个分支版本(branch)studio，一个标签目录 tags。组织好 trunk 版本的内容后提交到版本库里面，然后使用 SubVersion 的命令行工具创建分支版本，命令如下：

`svn copy svn://192.168.100.3/KWMK/KAF/trunk svn://192.168.100.180/KWMK/KAF/studio -m "创建KAF项目的分支"`

通过上面的命令行就能创建KAF项目的分支版本了，开始的时候主版本和分支版本是完全一样的。

Trunk 版本:作为当前项目的发布副本,并不作为日常的开发副本,编译生成程序集时,选择的是 Release 编译。

Branch 版本:作为日常的开发副本,开发人员的日常工作都在分支版本里面进行,编译生成程序集时,选择的是 Debug 编译。

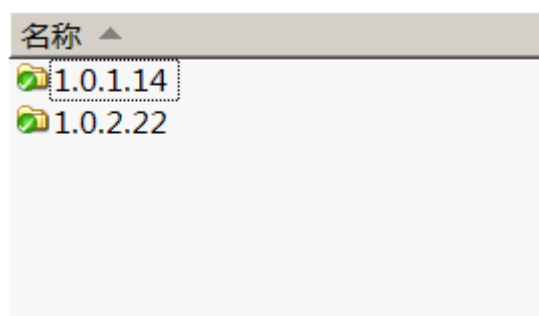
Tags 目录:存储主版本的标签,主版本的每次成功发布,都会在 Tags 目录下面以主版本的发布版本号为标签,保存一份当前主版本的代码。

下面我来说明这么划分的意义:当完成一个功能模块,或者是解决了一组 Bug 的时候,就通过 SubVersion 的合并操作,把分支版本和主版本之间的差异部分合并到主版本里面去,从这个意义上说,主版本就是基线版本,这么做的好处在于:日常开发过程当中,很可能开发人员的提交会非常频繁,每一次的提交都会造成版本库的对应版本号码提升,但是仅仅这个版本库的版本号码对于发布来说是没有意义的,对于主版本来说,很可能是经历了重大的修改,或者是加入了很多新的功能的时候,要不就是作为具有里程碑意义的功能增加,或者 Bug 修改的时候,我们才会发布一个新版本,如果我们对版本不做主版本和分支版本的区分,只有一个主版本的时候,那么每一次提交,哪怕是修改了一个很小的错误,比如说对某一个类的重构修改,很可能开发人员在填写提交日志的时候都不知道如何描述这个改动,但是分支版本的持续集成也会触发一次,发布一个版本,这样就太频繁了,从产品发布角度来说,也是没有任何意义的。所以我们要区分主版本和分支版本,前面说过了,主版本就是基线版本。主版本被配置成不支持自动的持续集成,而是由管理人员手工触发一次持续集成。这样的话,主版本的主要任务就是为了发布。

分支版本的每一次代码提交,都会自动的触发持续集成,在这个持续集成的环节,不包含只有正式发布的时候才会做的一些操作,比如说修改程序集的版本号,或者打包等操作,只是会做编译,测试,代码检验等操作。

在把分支版本的差异合并到主版本之后,由开发人员主观判断是否需要发布一个新的版本,如果需要的话,那么借助持续集成工具,手工触发一次持续集成,这个时候会完成持续集成所有的步骤,这里额外提一下,如果按照这个流程做下来的话,在主版本的持续集成里面,编译,测试等环节基本上不会出现问题了,因为你已经在分支版本里面都解决完了,基本上也就是跑完修改版本号和部署两个环节了,所以不用担心主版本和分支版本都做一遍持续集成的情况下,程序员会有重复解决问题的情况发生。

主版本的持续集成过程中,有一个环节就是打标签,就是把你发布的版本号对应的版本代码做一个标记,保存在 Tags 目录里面,如图三所示。

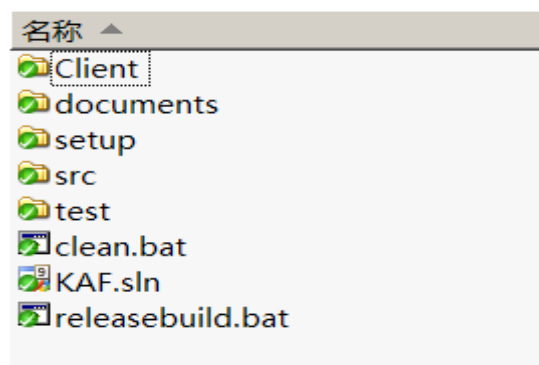


<图三>

这么做的好处显而易见,每一个发布的版本对应的代码都能在 Tags 里面找到,对于快速定位错误,非常方便。还有一个更大的好处,随时都能用这里的任何一个版本号,生成对应的发布产品,因为有的时候多个客户需要的是一个产品的不同版本,当然这一点对于 KAF 这个项目是意义不大的,因为它是产品的底层依赖库,但是对于管理客户端产品意义非常大。

3、项目内目录结构

我再介绍一下项目目录内部的结构,如图四所示。

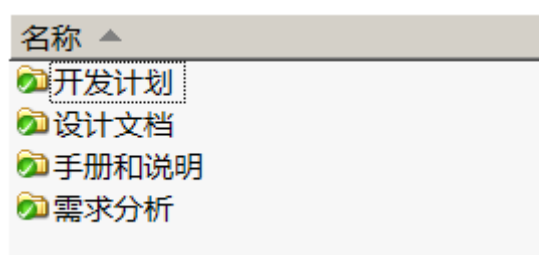


<图四>

目录里面各个文件夹得含义如下：

(1)Document

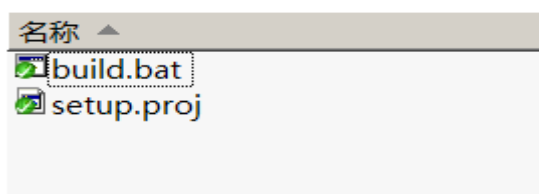
当前项目所涉及到的所有文档，如图五所示。



<图五>

(2)Setup:

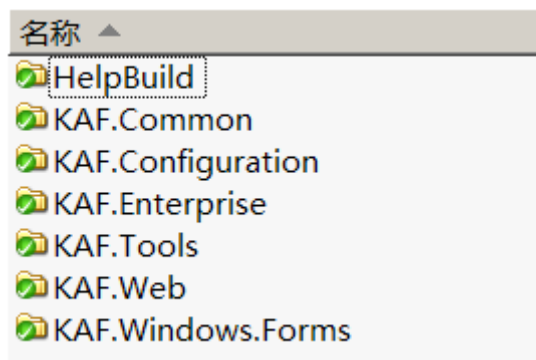
当前项目的安装、部署脚本，如图六所示。



<图六>

(3)Src:

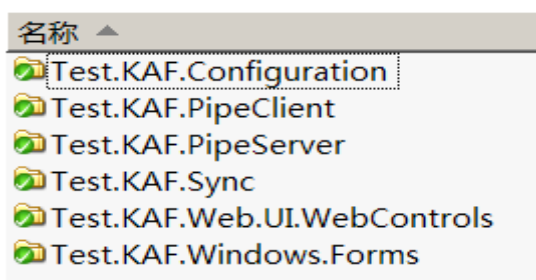
当前项目的源代码，如图七所示。



<图七>

(4)Test:

当前项目的测试案例，如图八所示。



<图八>

(5)releasebuild.bat:

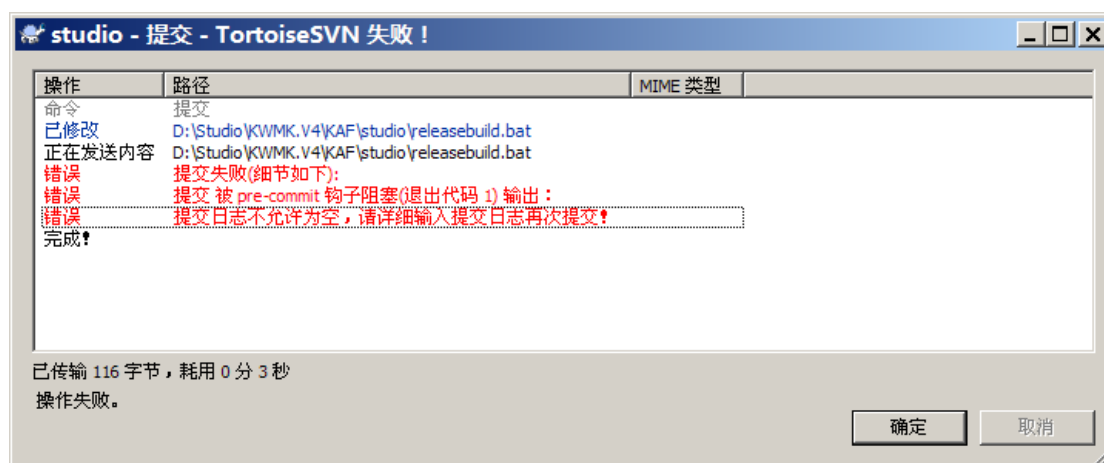
任何人把当前项目签出到一个目录后，都能执行这个批处理，尝试对项目进行 Release

版本的编译，而不用打开笨重的 Visual Studio2008

个人认为良好的目录结构组织，也是有利于整个项目的管理的。

4、版本库钩子

我要求所有的团队成员在提交代码的时候必须要对本次提交做一个文字的描述，比如是修改的 Bug 追踪系统中的第几号 Bug，或者添加的某个功能等，但是不是所有的人都能做到，后来在阅读 SubVersion 的文档的时候发现它支持钩子脚本，就是说在提交的环节可以执行预定义的一个操作，这个操作是一个脚本，我就在网上找到了如下的脚本，进行了一些修改，当提交人员没有写提交日志就提交的时候，SubVersion 会给出一个错误，如图九所示。



<图九>

这个钩子脚本内容如下：

```
@echo off

set SVN_BINDIR="C:\Program Files\Subversion\bin"

setlocal

set REPOS=%1

set TXN=%2

rem check that logmessage contains at least 10 characters

%SVN_BINDIR%\svnlook log "%REPOS%" -t "%TXN%" | findstr "." > nul
```

```
if %errorlevel% gtr 0 goto err
```

```
exit 0
```

```
:err
```

```
echo 提交日志不允许为空，请详细输入提交日志再次提交！ 1>&2
```

```
exit 1
```

把上面的文本保存为一个批处理文件，叫做 checklog-commit.bat 放在 SubVersion 版本库目录下的 hooks 目录里面就可以了。

这样的话，大家在提交的时候，就必须写上提交日志信息，提交日志的功能对于程序员来说有类似于程序注释的功能，帮助程序员想起每次的提交自己都干了什么，当然还有一点很重要的一点是帮助其他人员了解本次提交做的操作，虽然我的团队中有些兄弟写的提交日志，除了他自己以外谁也看不懂，但是总比没有强吧 😊。

详细准确的提交日志描述，还有一个很重要的目的，就是在后边介绍的持续集成中，可以在产品发布的时候，把发布版本中的从上次版本发布到本次版本发布之间的提交日志用程序抽取出来形成一个文档，伴随产品一起发布，可以给实施人员或者最终用户一个更新日志，说明这个版本都做了什么改进，修改了那些 Bug，添加了哪些功能，比如说 QQ 或者其他很多成熟的软件发布，在安装后，都能找到这么一个类似的文档。我们也可以做到这种效果。

5、公共引用库的管理

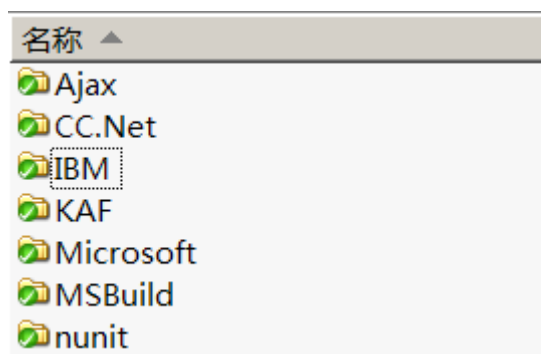
1)公共引用库的设置

在我没有管理这个团队以前，所有的人的代码中的对组件的依赖都是使用的相对路径，就是说开发人员把要依赖的程序集放到自己开发机器上的一个路径下面，然后就在程序中添加引用。这样做的坏处显而易见，每个人负责的代码只能在自己的机器上面编译的过去，其

他人在自己的机器上签出别人的代码来就肯定编译不过去。还有一个问题就是对于团队内部人员之间开发的程序集，如果存在依赖关系的话，比如 KAF，那么更新的时候比较麻烦。

比如说 a 同学的代码中引用了 b 同学的一个组件 c，那么当 b 同学负责的这个组件更新的时候，就要主动告诉 a 同学去某某地方取这个包含 c 组件的已经更新的程序集，复制粘贴满天飞，那么这个问题如何解决哪？

解决方法就是创建一个公共的引用目录，大家可以在图一中看到在版本库的根目录下存在一个叫做 reference 的目录，这个目录就是公共的引用目录，这里面放置的是所有的依赖程序集，如图十所示。



<图十>

这个目录里面按照公司的名称或者功能，分了很多类别，每一个目录里面放置的就是被其他项目依赖的组件，每个开发人员的 SubVersion 账号，都有读取这个根目录的权限，并且被要求一定要把这个目录签出来，然后在各自负责的项目中都使用相对路径引用 reference 目录下的相关依赖程序集，所以各个项目的*.csproj 文件中对于引用的程序集的描述，类似如下格式：

```
<HintPath>..\..\..\reference\libs\KAF\KAF.Common.dll</HintPath>
```

```
<HintPath>..\..\..\reference\libs\KAF\KAF.Web.dll</HintPath>
```

```
<HintPath>..\..\..\reference\libs\KAF\KAF.Windows.Forms.dll</HintPath>
```

这样就能保证任何一个项目成员把其他人的项目签下来的时候，只要保证他的副本根目

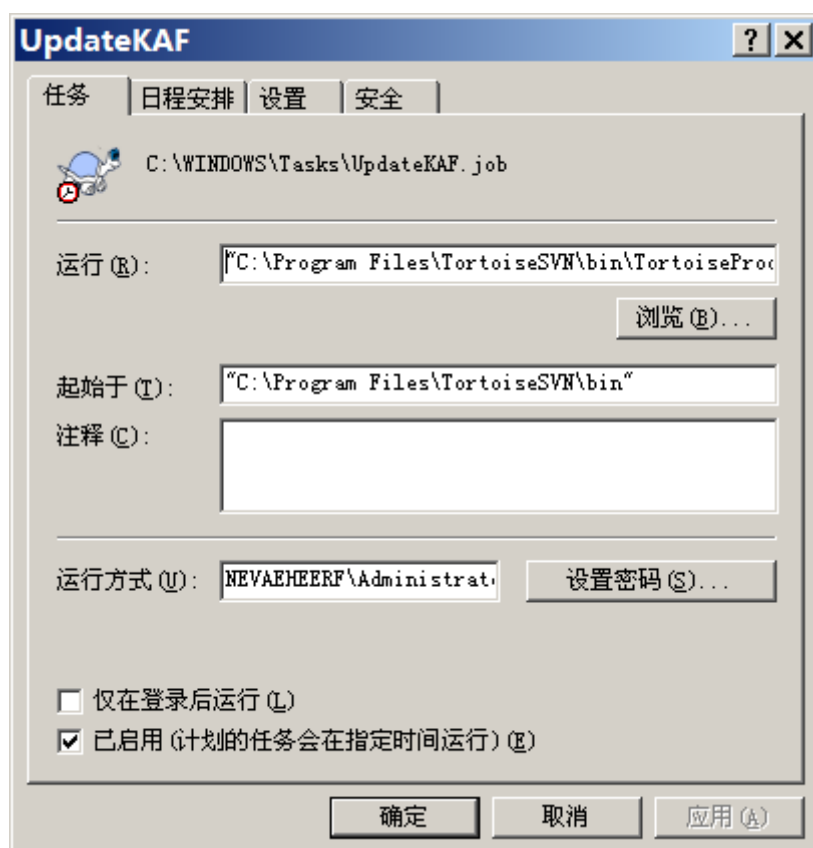
录中存在 reference 这个文件夹 ,那么执行编译的时候就不会报出引用程序集找不到的错误。

2)公共引用库的更新

公共引用库在没有使用持续集成进行管理的时候, 它的更新还是比较麻烦的, 以 KAF 为例, 前边我说了, KAF 项目是框架库, 团队中的每个人的项目都要引用这个项目, KAF 项目代码在版本库的根目录下面, 如图一中所示, 它在公共引用库中的位置如图十中所示, 我每次发布 KAF 更新的时候都要手工的把更新的 KAF 程序集文件复制到.\reference\libs\KAF 这个目录下面, 提交到版本库中, 然后告诉其他团队人员手工更新自己机器上面的公共引用库的相关目录, 这样很麻烦, 使用持续集成后就解决了这个问题。

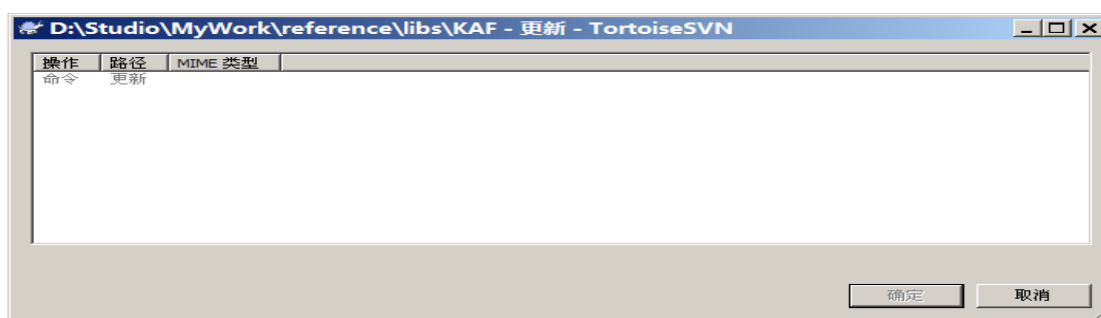
在对 KAF 的 Trunk 版本进行持续集成过程的最后, 加入一步操作, 即当各个操作都成功后, 把生成的程序集 (Release) 复制到公共引用库下面的对应目录, 即.\reference\libs\KAF 下面, 然后提交这几个程序集到版本库, 然后在每个开发人员的机器上的控制面板中的计划任务中, 创建一个计划任务, 如图十一所示, 在运行框中输入如下命令行:

```
"C:\Program Files\TortoiseSVN\bin\TortoiseProc.exe" /command:update /path:  
" D:\Studio\reference\libs\KAF\" /notempfile /closeonend:1
```



图十一

使用 TortoiseSVN(小乌龟)提供的命令行工具,定时更新每台机器上面的工作副本中的公共引用库目录.\reference\libs\KAF,更新时间间隔可以灵活设置,我的 KAF 因为目前比较稳定了,更新不是很频繁了,所以我设置的是 4 个小时更新一次,运行时如图十二所示。



图十二

至于持续集成中的相关设置,我会在后继章节介绍。

四、测试驱动开发模式的引入

测试驱动 TDD (Test-Driven Development) 也是敏捷开发模式提倡的开发方法 , TDD 不是测试的方法 , 而是开发的方法 , 它是轻量级的软件开发过程 , 非常适合开发小组或者小型的团队进行实践。我个人的体会 , 测试驱动的开发模式首先转变的就是开发人员的考虑问题的角度 , 尽量的从自己编写的类的调用者的角度去考虑问题 , 测试驱动提出的口号就是 “总是从一个失败的测试案例开始” , 就是说 , 当你想设计一个类的时候 , 你首先写下这个类的测试案例 , 因为这个类还没有实现 , 你的测试肯定过不去 , 所以 “总是从一个失败的测试案例开始”。

测试案例肯定体现的是调用者的想法 , 就是你想怎么使用这个类 , 然后把这个想法反馈到这个类的实现过程中去 , 我们总是在提 “要把业务逻辑和业务展现相分离 , 增加类的可复用性 , 要把你的 UI 类榨干 , UI 类只是用来和用户交互的 , 不应该包含业务逻辑代码” 等等 , 但是现实情况是 , 这直接和开发人员的能力有关 , 能力好的开发人员写出来的类的封装性就好一些 , 复用性也随之强一些 , 能力差的开发人员往往会直接点开一个按钮就开始往里面写数据库连接代码了 , 生猛无比。现在情况好了一些 , 测试驱动的引入 , 可以说是给开发人员提供了一个设计的拐杖 , 帮你把你的产品类封装的相对更好一些 , 帮你将业务逻辑和业务展现的剥离进行的相对更彻底一些。

TDD 模式可以提升你的设计质量 , 强迫自己把速度降下来 , 多思考 , 是否还有重构的必要 , 还有一个优点是每一份测试案例都是对于它所测试类的非常好的使用说明 , 对于程序员来说 , 给他一个类的使用例子 , 要远比给他一份描述这个类如何使用的文档来的更让他高兴。

TDD 的原则 , 如下 :

- 1、让不可测试的代码最小化
- 2、杜绝在产品代码中出现测试代码
- 3、一个测试对应一个条件
- 4、保持测试的独立性
- 5、小步累进
- 6、快速构建

我做测试驱动开发使用的工具是 [NUnit](#) ,这是一个从 JUnit 移植而来的工具 ,它提供命令行和工作窗口两种工作模式 ,持续集成中就用到了它的命令行工作模式。在使用 VS2008 开发的过程中 ,强烈推荐使用 [TestDriven.NET](#) ,非常得方便 ,下面我用一个我写的例子给大家简单介绍一下如何写测试案例 :

我有一个管理电子病历中元素的类叫做 ElementManager.cs 对于这个类的单元测试类叫做 ElementManagerTest.cs :

```
using System.Collections.Generic;

using System.IO;

//要添加Nunit提供的框架程序集的引用

using NUnit.Framework;

using KAF.Data.SqlClient;

namespace EMR.Common.Test

{

    /// <summary>

    /// 标示这是一个测试类
```

```
/// </summary>
```

```
[TestFixture]
```

```
public class ElementManagerTest
```

```
{
```

```
    private static SQLDB _sqlServerDB;
```

```
/// <summary>
```

```
/// 初始化测试类所需资源
```

```
/// </summary>
```

```
[TestFixtureSetUp]
```

```
public void MyInitialize()
```

```
{
```

```
    _sqlServerDB = new SQLDB("Data Source=192.168.100.22;Initial
```

```
    Catalog=CaseRoom;Persist Security Info=True;User
```

```
    ID=sa;Password=abc123", "System.Data.OracleClient");
```

```
}
```

```
/// <summary>
```

```
/// 释放测试类所占用资源
```

```
/// </summary>
```

```
[TestFixtureTearDown]
```

```
public void MyFinalize()

{

    _sqlServerDB.Close();

}

/// <summary>

/// 测试ElementManager类提供的GetElementByCode方法

/// 如果element.Name不等于"起病诱因",则测试失败。

/// 如果element.Name等于"起病诱因",则测试成功。

/// </summary>

[Test]

public void GetElementByCodeTest()

{

    Element element =

ElementManager.Instance.GetElementByCode("661c1dad-119a-4e84-87e3-c5501

14a85c5");

    Assert.AreEqual("起病诱因", element.Name);

}

/// <summary>

/// 测试ElementManager类提供的GetElementDataByCode方法。

/// 如果normailElementData.Data等于"左右相等",则测试成功。
```

```
/// 如果normailElementData.Data不等于"左右相等",则测试失败。

/// </summary>

[Test]

public void GetElementDataByCodeTest()

{

    NormailElementData normailElementData =

ElementManager.Instance.GetElementDataByCode("3924b61d-9f82-40b2-9f89-b8c74c

5784bf");

    Assert.AreEqual("左右相等", normailElementData.Data);

}

.....

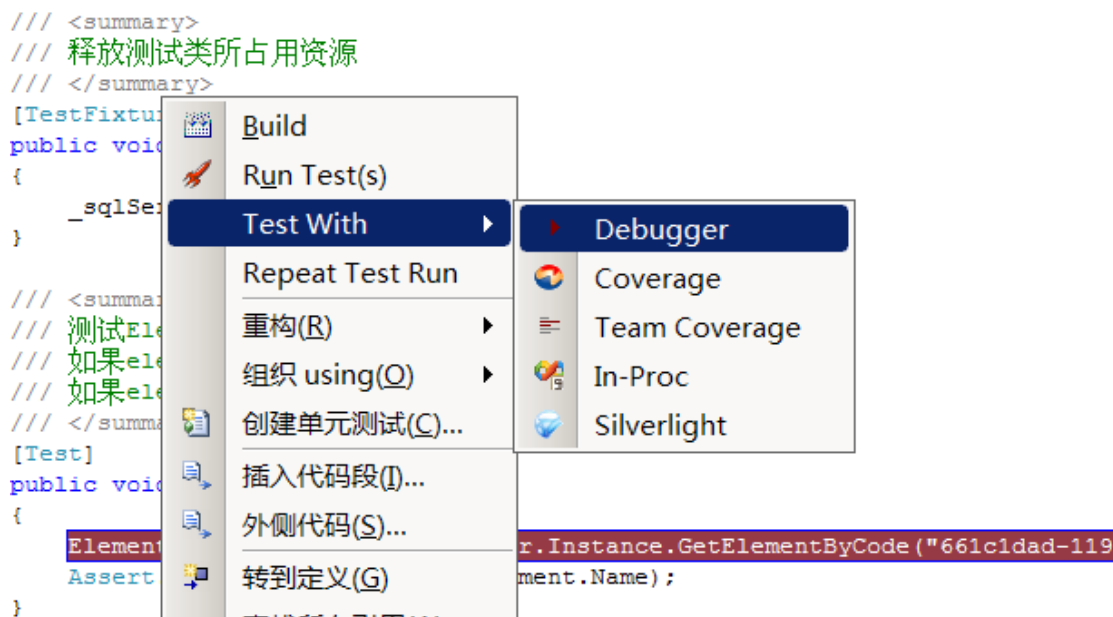
}

}
```

在上面的测试案例中，可以看到，测试类使用 Assert（断言）来判断一个变量是否等于期望的值，通过这种方式来判断测试是否成功。

Assert 类提供了很多方法，主要的方法就是 AreEqual，针对不同数据类型，提供了 23 中重载。

在编写一个测试方法的同时，就可以进行调试，不必等到完全写完了才能测试，这个时候可以使用前边推荐的 [TestDriven.NET](#) 这个 VS 插件，如图十三所示，这个工具简直是太好



<图十三>

用了，我喜欢的不得了，虽然 VS2008 也提供了代码测试的功能 (VSTS)，但是我一直没有使用，觉得太笨重，而且也是因为习惯了这个 TestDriven.NET，在图十三中大家能看到，直接在一个测试方法里面打上断点后，就能调用“Test With”的子菜单里面的“Debugger”运行调试了，再也不用在产品代码里面写测试语句了，再也不用为了测试一个方法，或者验证一个想法而去创建一个可执行程序了。

测试驱动的规则和测试类型等其他方面，还有 NUnit 的使用要是讲的话，还有很多，我就不展开讲了，对于 NUnit 的使用，大家可以参考一下 NUnit 的文档。

在本章，我主要是根据自己的实践经验，讲了自己对测试驱动的体会，个人感觉测试驱动是一个非常好的工具，或者说是一种非常好的思考问题的方式，能帮助我们提高代码水平和产品质量。通过测试驱动，我的团队中使用测试驱动的每个人，都加深了对“面向对象”编程的理解。

五、Wix 打包脚本的编写

尚未编写

六、MSBuild 脚本的编写

尚未编写

七、持续集成环境的搭建

好了，具备了上面几章介绍的技术点，就可以搭建我们的持续集成流程了，本章我将用我们团队的几个项目作为例子，介绍持续集成环境的搭建。

1、持续集成介绍

“持续集成”概念首先是在敏捷开发模式中提出的概念，原来叫做“日编译”(DailyBuild),简单来说就是频繁持续的把开发团队中的各个成员的工作进行集成，并且把集成结果反馈给团队中的各个成员。持续集成的目的不是减少 Build 失败的次数，而是尽早发现问题，在最短的时间内解决问题，减少风险和浪费。

持续集成一般都顺序包含下面的几个步骤：

- (1) 更新代码(SubVersion)
- (2) 编译项目(MSBuild)
- (3) 自动化测试(NUnit)

- (4) 代码分析(FxCop)
- (5) 代码覆盖率检查(NCover)
- (6) 发布(Wix 生成的 MSI)
- (7) 反馈集成结果

当然，还能做的步骤更多，比如数据库结构的更新，版本号的更新等，我会在后边进行说明。

2、持续集成搭建工具介绍

搭建持续集成环境的工具不止一个，我选择了 CruiseControl.NET，这是一个开源项目，你可以对源码进行相应的修改。

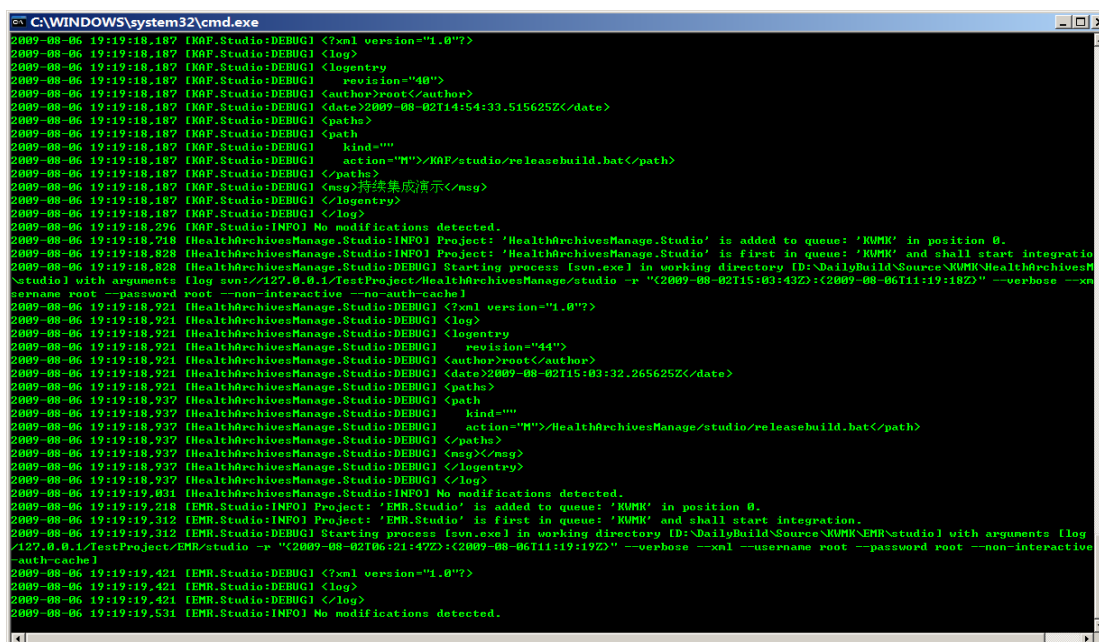
CruiseControl.NET 有两个安装包，一个是服务器端的安装包 CruiseControl.NET，一个是客户端的安装包 CCTray，简称为 CC。服务端的安装完成后，会在 IIS 服务器下面生成一个网站，叫做“ccnet”，在 CruiseControl.NET 的安装目录下的“server”目录下有一个名为“ccnet.config”的文件，这个文件就是持续集成中各个项目的配置文件，是以 XML 的格式存储的，在这个文件中配置每个项目的集成步骤，如图十四所示，就是配置好的站点截图。

Dashboard							
Farm Report Download CCTray Servers local							
Server	Project Name	Last Build Status	Last Build Time	Last Build Label	CCNet Status	Activity	Admin
local	DoctorWorkStation.studio	Success	2009-06-17 16:41:15	1.0.1.1167	Running	Sleeping	Force Stop
local	HealthArchivesManage.studio	Success	2009-07-10 09:13:02	1.0.19.787	Running	Sleeping	Force Stop
local	KAF.Studio	Success	2009-08-04 12:13:21	1.0.56.933	Running	Sleeping	Force Stop
local	KAF.Trunk	Success	2009-07-07 15:32:53	1.0.40.645	Running	Sleeping	Force Stop
local	RegionalHealth.Studio	Success	2009-07-31 14:53:29	1.0.282.929	Running	Sleeping	Force Stop
local	RegionalHealth.Trunk	Success	2009-04-09 13:53:45	1.0.0.0	Running	Sleeping	Force Stop
local	Security.Studio	Success	2009-07-16 10:56:58	1.0.11.854	Running	Sleeping	Force Stop

<图十四>

[CruiseControl.NET](#) 还会在安装机器上面生成一个服务,大家可以在机器安装的服务列表里面找到一个名为“CruiseControl.NET Server”的服务,就是它了,这个服务就负责对脚本中定义的各个项目进行持续集成操作,然后把集成结果反馈到 IIS 的站点里面。这个服务在安装完成后,是默认停止的,您要手工启动它。

在进行集成操作的时候,[CruiseControl.NET](#) 还会动态生成日志,这个日志对于在配置持续集成的项目时,是非常有用的,这个日志存在于 [CruiseControl.NET](#) 的安装目录下的 server 目录下,名为 ccnet.log,大家可以在控制台,通过如下的命令行查看日志的动态更新: **tail -f ccnet.log**, 这个日志对于调试持续集成的配置是非常有用的,日志截图如下:



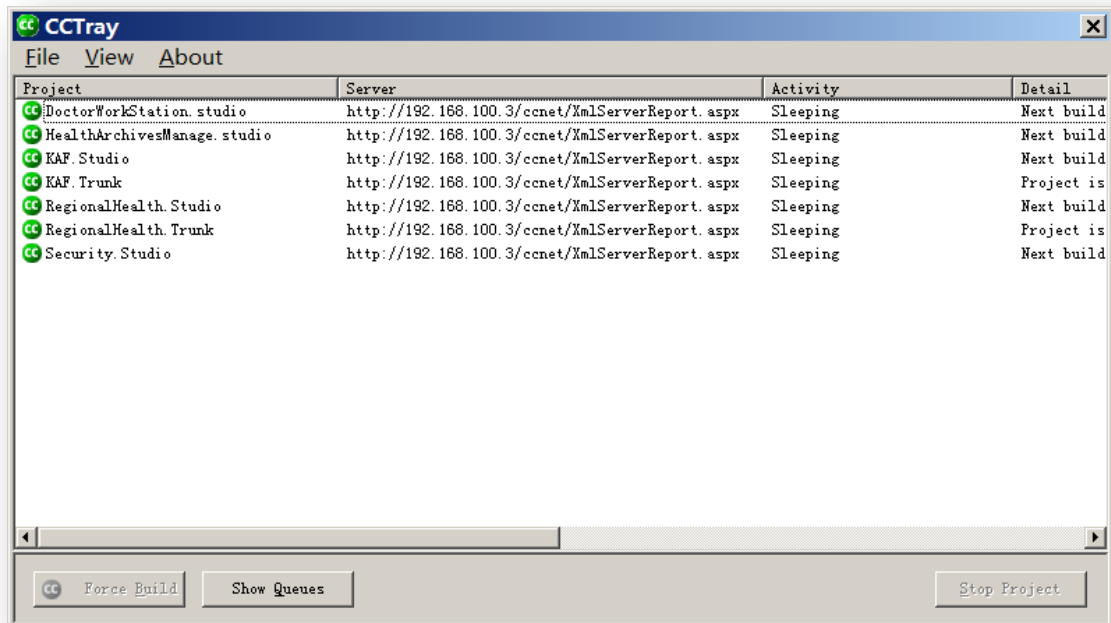
```

C:\WINDOWS\system32\cmd.exe
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG] <?xml version="1.0"?>
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG] <log>
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG] <logentry
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG]   revision="40">
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG]   <author>root</author>
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG]   <date>2009-08-02T14:54:33.515625Z</date>
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG]   <paths>
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG]     <path
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG]       kind=""
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG]       action="M"/KAF/studio/releasebuild.bat</path>
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG]   </paths>
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG] </logentry>
2009-08-06 19:19:18.187 [KAF.Studio:DEBUG] </log>
2009-08-06 19:19:18.296 [KAF.Studio:INFO] No modifications detected.
2009-08-06 19:19:18.718 [HealthArchivesManage.Studio:INFO] Project: 'HealthArchivesManage.Studio' is added to queue: 'KUMK' in position 0.
2009-08-06 19:19:18.828 [HealthArchivesManage.Studio:INFO] Project: 'HealthArchivesManage.Studio' is first in queue: 'KUMK' and shall start integration.
2009-08-06 19:19:18.828 [HealthArchivesManage.Studio:DEBUG] Starting process [svn.exe] in working directory [D:\DailyBuild\Source\KUMK\HealthArchivesManage.Studio] with arguments [log svn:/127.0.0.1/TestProject/HealthArchivesManage/studio -r "(2009-08-02T15:03:43Z):(2009-08-06T11:19:18Z)" --verbose --xml --username root --password root --non-interactive]
2009-08-06 19:19:18.921 [HealthArchivesManage.Studio:DEBUG] <?xml version="1.0"?>
2009-08-06 19:19:18.921 [HealthArchivesManage.Studio:DEBUG] <log>
2009-08-06 19:19:18.921 [HealthArchivesManage.Studio:DEBUG] <logentry
2009-08-06 19:19:18.921 [HealthArchivesManage.Studio:DEBUG]   revision="44">
2009-08-06 19:19:18.921 [HealthArchivesManage.Studio:DEBUG]   <author>root</author>
2009-08-06 19:19:18.921 [HealthArchivesManage.Studio:DEBUG]   <date>2009-08-02T15:03:32.265625Z</date>
2009-08-06 19:19:18.921 [HealthArchivesManage.Studio:DEBUG]   <paths>
2009-08-06 19:19:18.937 [HealthArchivesManage.Studio:DEBUG]     <path
2009-08-06 19:19:18.937 [HealthArchivesManage.Studio:DEBUG]       kind=""
2009-08-06 19:19:18.937 [HealthArchivesManage.Studio:DEBUG]       action="M"/HealthArchivesManage/studio/releasebuild.bat</path>
2009-08-06 19:19:18.937 [HealthArchivesManage.Studio:DEBUG]   </paths>
2009-08-06 19:19:18.937 [HealthArchivesManage.Studio:DEBUG] </logentry>
2009-08-06 19:19:18.937 [HealthArchivesManage.Studio:DEBUG] </log>
2009-08-06 19:19:19.031 [HealthArchivesManage.Studio:INFO] No modifications detected.
2009-08-06 19:19:19.218 [EMR.Studio:INFO] Project: 'EMR.Studio' is added to queue: 'KUMK' in position 0.
2009-08-06 19:19:19.312 [EMR.Studio:INFO] Project: 'EMR.Studio' is first in queue: 'KUMK' and shall start integration.
2009-08-06 19:19:19.312 [EMR.Studio:DEBUG] Starting process [svn.exe] in working directory [D:\DailyBuild\Source\KUMK\EMR\studio] with arguments [log svn:/127.0.0.1/TestProject/EMR/studio -r "(2009-08-02T16:21:47Z):(2009-08-06T11:19:19Z)" --verbose --xml --username root --password root --non-interactive]
2009-08-06 19:19:19.421 [EMR.Studio:DEBUG] <?xml version="1.0"?>
2009-08-06 19:19:19.421 [EMR.Studio:DEBUG] <log>
2009-08-06 19:19:19.421 [EMR.Studio:DEBUG] </log>
2009-08-06 19:19:19.531 [EMR.Studio:INFO] No modifications detected.
  
```

然后在每个团队成员的机器上面安装 CCTray, 然后运行 CCTray, 会在系统托盘的右下角有一个绿色的图标, 如图十五所示, 双击图标, 弹出配置界面, 如图十六所示, 按照如下步骤操作, 导入项目:

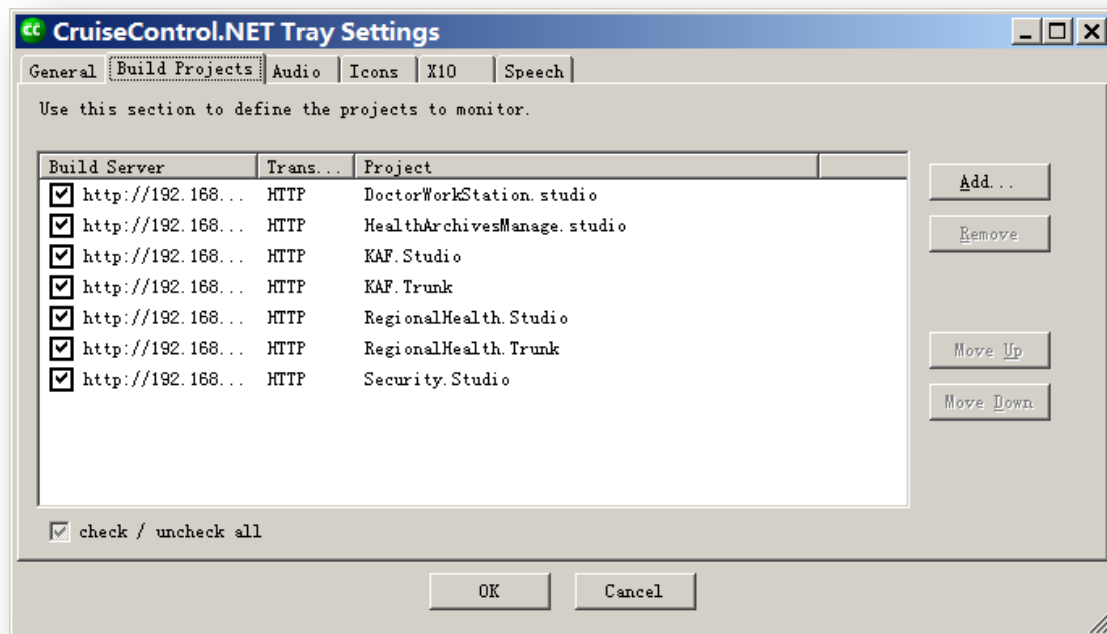


<图十五>



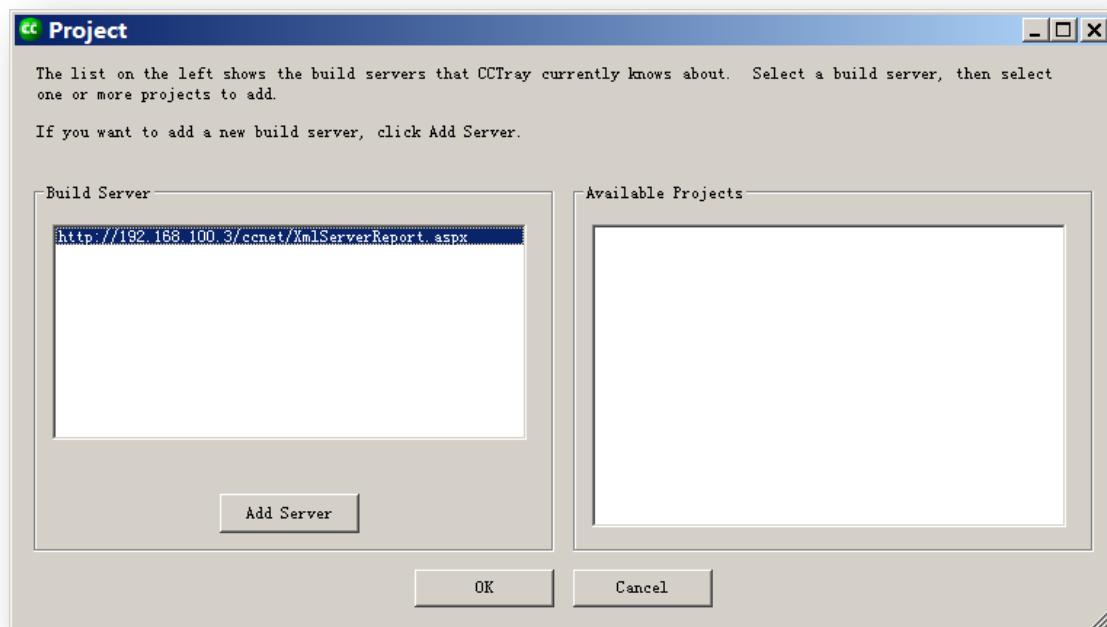
<图十六>

1)点击“File”菜单,选择“Settings...”项,弹出如图十七所示界面,选择“Build Projects”页。



<图十七>

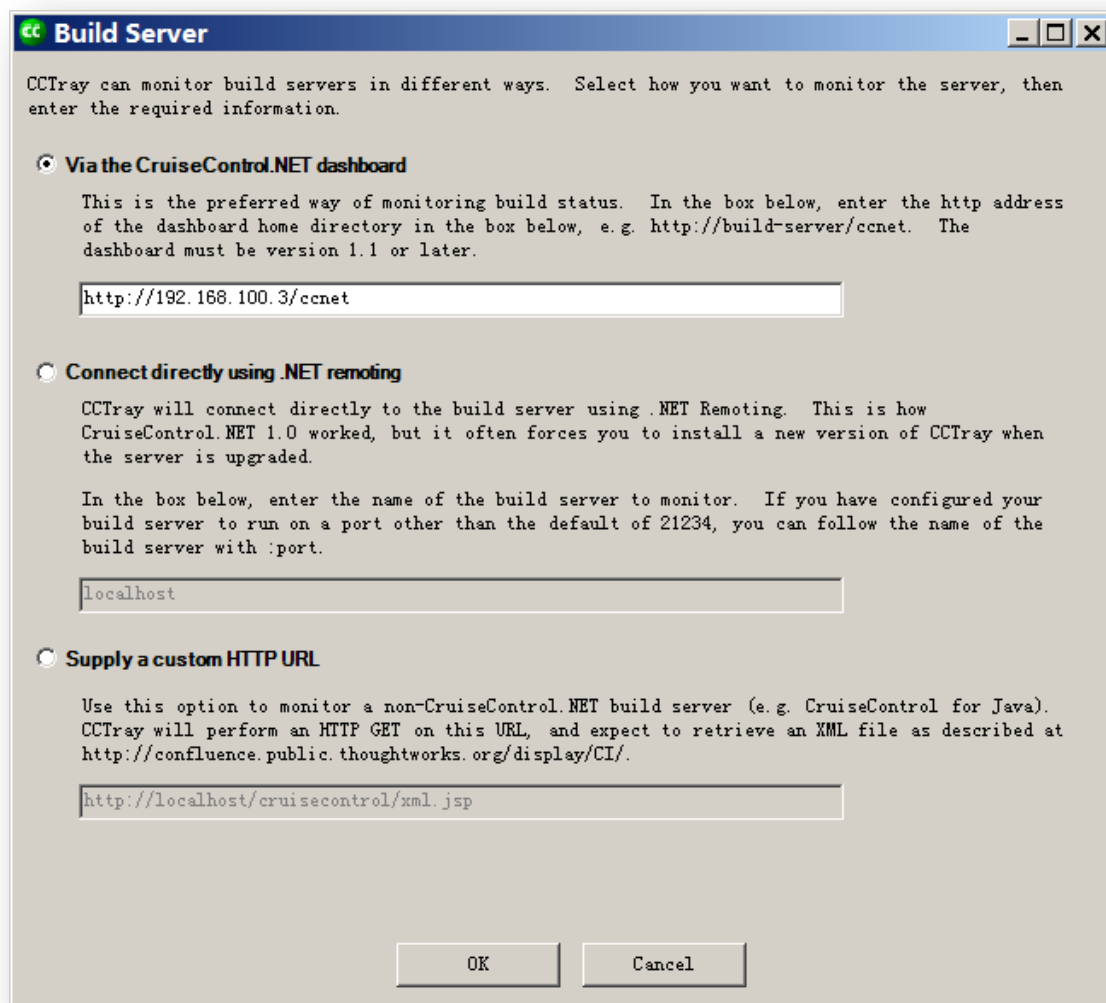
2)点击 “Add” 按钮，弹出如图十八所示界面。



<图十八>

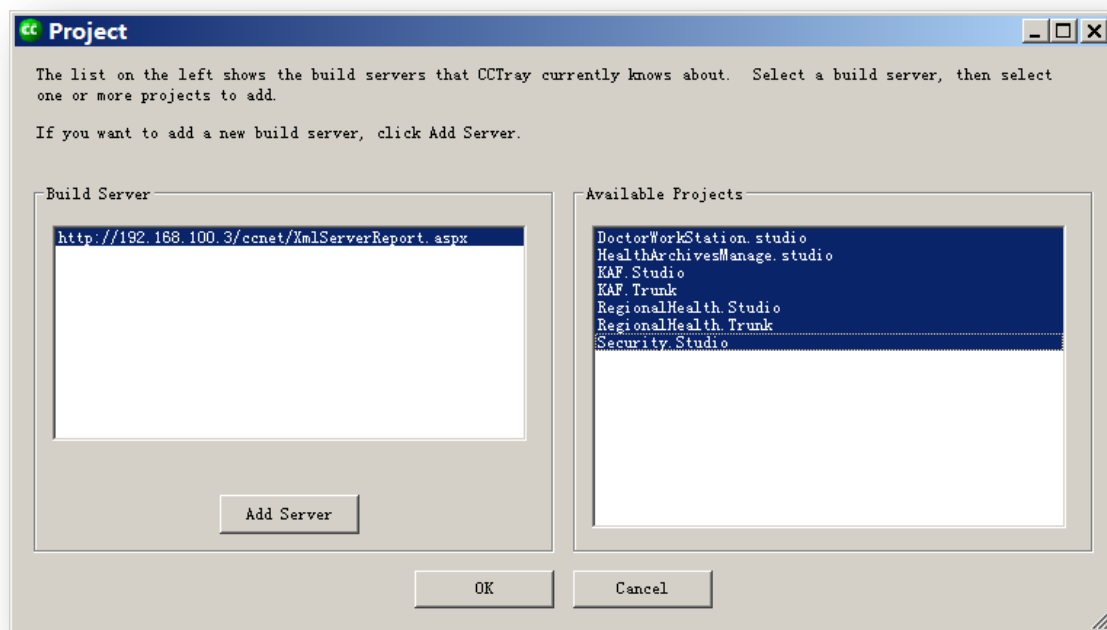
3)点击 “Add Server” 按钮，弹出如图十九所示界面。选择第一个选项 “Via the

CruiseControl.NET dashboard”，然后在下面的文本框里面输入 CruiseControl.NET 的网站所在地址，比如我的 CruiseControl.NET 服务器安装在了 192.168.100.3 这台机器上面，那么我就在文本框里面输入 <http://192.168.100.3/ccnet>，然后点击“OK”就可以了。



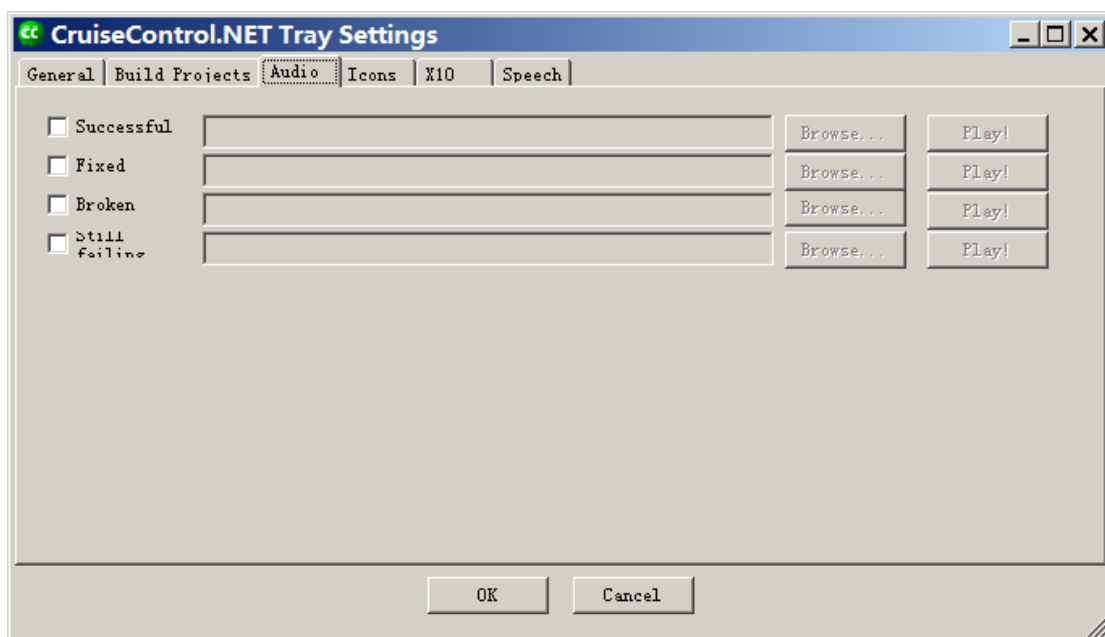
<图十九>

4)这个时候如图二十所示，显示出了配置好的所有项目，只要把和当前开发人员相关的项目项选中，然后点击“OK”就可以了，这里要说一下，对于团队负责人员来说，应该把所有的项目都选上，为了你观察每个项目的集成情况，当失败的时候，你可以提醒失败项目的负责人员及时的进行解决问题。



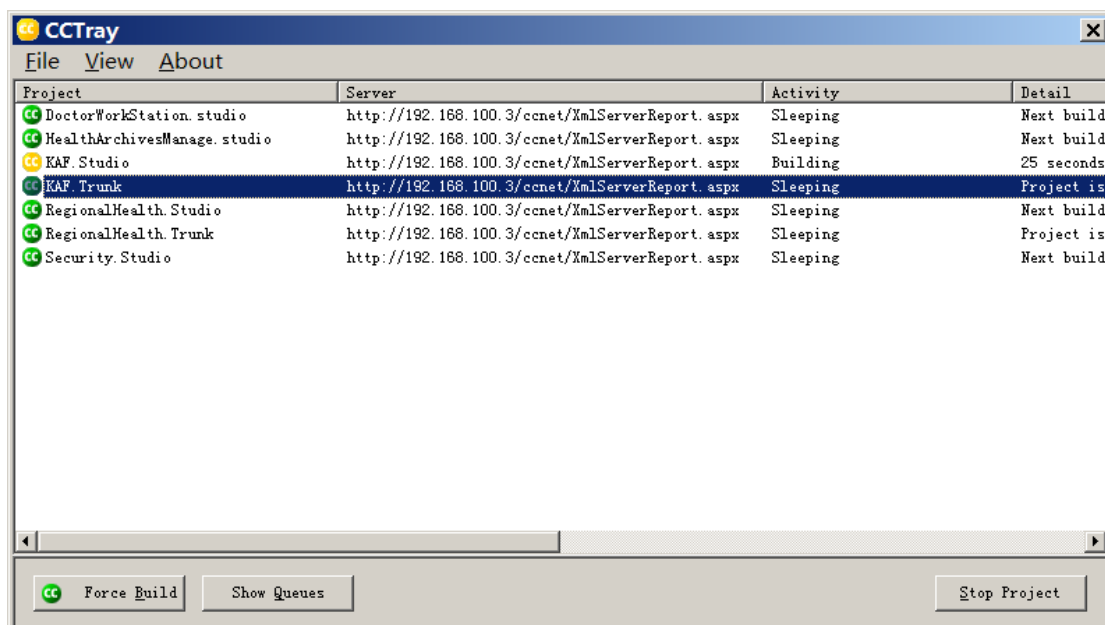
<图二十>

5)到此就设置好了持续集成的客户端了，还有一个设置这里要提一下，在第一步里面的图十七所示的界面上，有一个“Audio”页的设置，如图二十一所示，这里可以设置项目各个状态的对应的声音，成功的时候播放什么声音，失败的时候播放什么声音，你可以用一台机器啥也不干，连上一个大音箱，就是用来播放项目集成中的各个状态，这也就是集成的反馈了，成功的时候放一些轻松的，喜悦的音乐，失败的时候可以播放救火车的声音，不知道这样会不会让开发人员更快的解决问题？😁

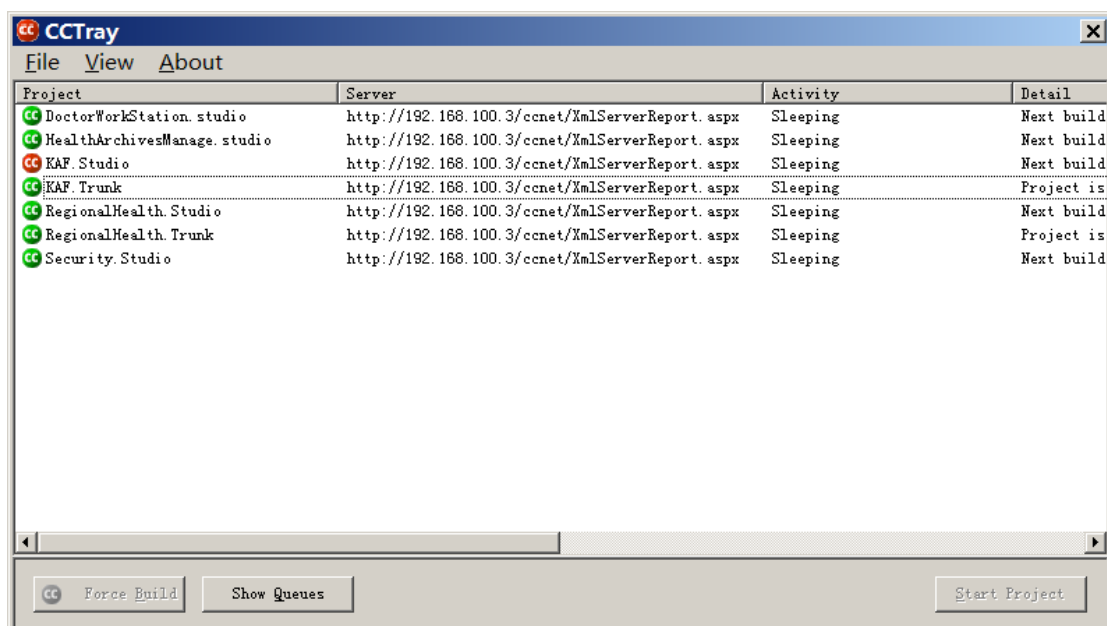


<图二十一>

6)在某个项目的持续集成触发的时候，那个项目在 CCTray 中的对应项会变成黄色，如图二十二所示，同时托盘里面的绿色小图标也会变成黄色，如果集成成功了，图标会又变成绿色，如果失败，图标就变成了红色了如图二十三所示，这个时候，谁负责这个项目，就要马上去解决，要不然一阵我就去问他，为啥还不解决这个错误，你要造反吗。



<图二十二>



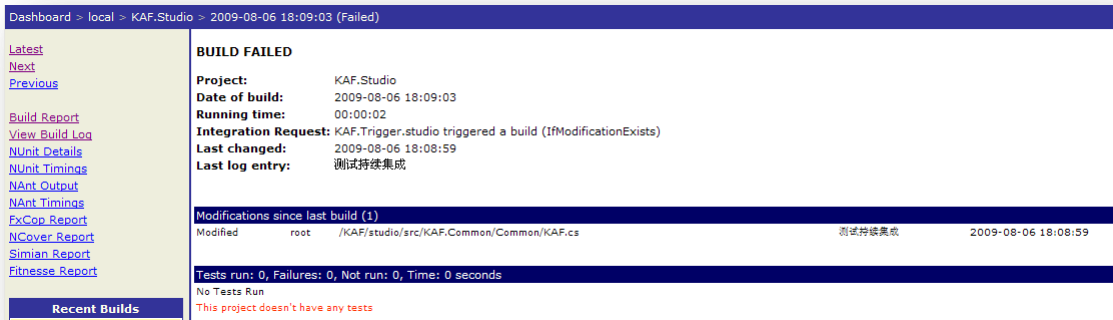
<图二十三>

7)当集成失败的时候，可以用鼠标双击图二十三里面所示的这个失败的项目，则会用默认的浏览器打开当前这个项目的集成报告，如图二十四所示。



<图二十四>

点击“here”链接，进入当前项目的集成报告页面，弹出如图二十五所示。



<图二十五>

在此页面中显示如下信息：

项目名称	值	注释
BUILD FAILED		项目是否成功
Project	KAF.Studio	项目名称
Date of build	2009-08-06 18:09:03	编译时间
Running time	00:00:02	持续时间
Integration Request	KAF.Trigger.studio triggered a build (IfModificationExists)	标明持续集成的触发形式 ,当前项目是当提交的代码有修改 (IfModificationExists)的时候会触发。
Last changed	2009-08-06 18:08:59	
Last log entry	测试持续集成	SubVersion 的本次提交日志

在 “Modifications since last build(1)” 区域，会列出本次提交代码被修改的文件列表。

文件状态	提交	文件路径	提交	提交时间

	人 员		日 志	
Modified	cwk	/KAF/studio/src/KAF.Common/Common/KAF.cs	测试持续集成	2009-08-06 18:08:59

点击界面左边功能项里面的“View Build log”项目，可以看到当前项目（KAF.Studio）的编译报告，如图二十六所示。

```

</msbuild>生成启动时间 2009-8-6 18:09:05。
节点 0 上的项目“D:\DailyBuild\Source\KWMK.V4\KAF\studio\KAF.sln”(默认目标)。
正在生成解决方案配置“Release|Any CPU”。
项目“D:\DailyBuild\Source\KWMK.V4\KAF\studio\KAF.sln”(1)正在节点 0 (默认目标) 上生成“D:\DailyBuild\Source\KWMK.V4\KAF\stud
Processing 0 EDMX files.
Finished processing 0 EDMX files.
CoreResGen:
没有相对于自身源文件过期的资源。将跳过资源生成。
Common\KAF.cs(48,1): error CS0116: 命名空间并不直接包含诸如字段或方法之类的成员
已完成生成项目“D:\DailyBuild\Source\KWMK.V4\KAF\studio\src\KAF.Common\KAF.Common.csproj”(默认目标) -- 失败。
项目“D:\DailyBuild\Source\KWMK.V4\KAF\studio\KAF.sln”(1)正在节点 0 (默认目标) 上生成“D:\DailyBuild\Source\KWMK.V4\KAF\stud
Processing 0 EDMX files.
Finished processing 0 EDMX files.
CoreCompile:
正在跳过目标“CoreCompile”，因为所有输出文件相对于其输入文件来说都是最新的。
CopyFilesToOutputDirectory:
KAF.Configuration -> D:\DailyBuild\Source\KWMK.V4\KAF\studio\src\KAF.Configuration\bin\Release\KAF.Configuration.dll
已完成生成项目“D:\DailyBuild\Source\KWMK.V4\KAF\studio\src\KAF.Configuration\KAF.Configuration.csproj”(默认目标)。
已完成生成项目“D:\DailyBuild\Source\KWMK.V4\KAF\studio\KAF.sln”(默认目标) -- 失败。

生成失败。

“D:\DailyBuild\Source\KWMK.V4\KAF\studio\KAF.sln”(默认目标)(1) ->
“D:\DailyBuild\Source\KWMK.V4\KAF\studio\src\KAF.Common\KAF.Common.csproj”(默认目标)(2) ->
(CoreCompile 目标) ->
Common\KAF.cs(48,1): error CS0116: 命名空间并不直接包含诸如字段或方法之类的成员

0 个警告
1 个错误
    
```

<图二十六>

图二十六，是截屏的最后部分，大家可以看到，这是 MSBuild 对项目进行编译后给出的报告。代码负责人员根据这个编译报告，解决出现的错误，然后再次提交，进行持续集成，这个过程一直重复，直到项目集成成功。

3、持续集成脚本的编写

使用支持 XML 编辑的工具，打开 CruiseControl.NET 的安装目录下面的 server 目录下面的 ccnet.config 文件。这个文件就是持续集成项目的配置文件，我用 KAF 项目的持续集成配置来给大家简单介绍一下如何配置 ccnet.config。

1)触发持续集成的方式

在一个项目中，通过 `<triggers>` 节来定义触发方式：

KAF.Studio 的触发方式：

```
<triggers>

    <!--自动检测版本库中的KAF.Studio项目，当有代码提交，并且有改动发生时，触
发集成-->

    <intervalTrigger name="KAF.Trigger.studio" seconds="10"
buildCondition="IfModificationExists"/>

</triggers>
```

KAF.Trunk 的触发方式：

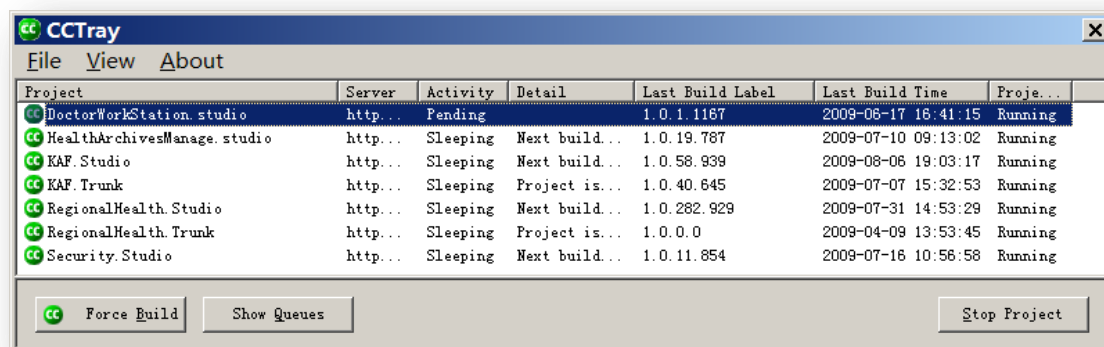
```
<!--手工触发-->

<triggers/>
```

2)标签的定义

如图二十七所示，是客户端的 CCTray 的显示界面，其中第五列(Last Build Label)，就是显示的标签，这个标签可以是一个日期，或者是一个版本号，它是支持自定义的，如果 CruiseControl.NET 提供的预定义标签形式不能满足你的要求，你可以自己编写程序集扩展

[CruiseControl.NET](#) 的标签。



<图二十七>

KAF.Studio 的标签就是用的 [CruiseControl.NET](#) 预定义的：

<!--以时间作为标签-->

<labeller type="dateLabeller" />

KAF.Trunk的标签是一个发布的版本号,类似于1.0.1.1167这样的格式,其中最后一位(1167)是取得SubVersion中的版本号,你可以通过实现ThoughtWorks.CruiseControl.Core.dll程序集中的ILabeller的方式,来实现自己的标签,然后把包含这个类的程序集命名为“ccnet.*.plugin.dll”,*可以替换成一个具体的名称,然后把这个程序集放到和ccnet.config同一个目录下就可以了。

3)从版本库签出代码

这一步,就是从版本库中把更新的代码签出到指定的目录下面。KAF.Studio 项目和 KAF.Trunk 项目这一步基本都是一样的,只是 KAF.Trunk 项目的代码控制节中多加入了如下两行配置,用来在主版本持续集成成功的情况下,为当前对应的版本代码打上一个标签,存放在 tags 目录下面,这么做的道理在前面的章节已经说过了:

<tagOnSuccess>true</tagOnSuccess>

```
<tagBaseUrl>svn://192.168.100.3/KWMKTestProject/KAF/tags</tagBaseUrl>
```

源码控制节配置如下

```
<!--源码控制节，版本库类型为SubVersion-->
```

```
<sourcecontrol type="svn">
```

```
<!--自动获取源码-->
```

```
<autoGetSource>true</autoGetSource>
```

```
<tagOnSuccess>true</tagOnSuccess>
```

```
<tagBaseUrl>svn://192.168.100.3/KWMKTestProject/KAF/tags</tagBaseUrl>
```

```
<!--当前项目对应的版本库路径-->
```

```
<trunkUrl>svn://192.168.100.3/KWMK/KAF/studio</trunkUrl>
```

```
<!--工作目录-->
```

```
<workingDirectory>D:\DailyBuild\Source\KWMK\KAF\studio</workingDirectory>
```

```
<!--用户名-->
```

```
<username>root</username>
```

```
<!--密码-->
```

```
<password>aaaaaaa</password>
```

```
</sourcecontrol>
```

4)执行集成步骤

所有的集成的步骤都要放在<tasks>节中，KAF.Studio 项目的集成包括了如下步骤。

(1)编译解决方案

<!--使用MSBuild进行编译-->

<msbuild>

<!--MSBuild.exe所处的位置-->

<executable>C:\WINDOWS\Microsoft.NET\Framework\v3.5\MSBuild.exe</executab

le>

<!--工作目录-->

<workingDirectory>D:\DailyBuild\Source\KWMK\KAF\studio</workingDirectory>

<!--解决方案的名称-->

<projectFile>KAF.sln</projectFile>

<!--MSBuild编译项目时，传入的参数-->

<buildArgs>/property:Configuration=Release</buildArgs>

<!--超时时长设置-->

<timeout>60</timeout>

<!--依赖程序集的位置-->

<logger>D:\DailyBuild\Source\KWMK\reference\libs\MSBuild\ThoughtWorks.Cruise

Control.MsBuild.dll</logger>

</msbuild>

(2)自动化测试

<!--使用NUnit进行自动化测试-->

<nunit>

```
<!--NUnit提供的控制台工具的位置-->

<path>D:\DailyBuild\Source\KWMK\reference\tools\nunit\nunit-console.exe</path>

>

<!--加入已经写好的包含测试案例的程序集-->

<assemblies>

<assembly>D:\DailyBuild\Source\KWMK\KAF\studio\test\KAF.Common.Test\bin\Release\Test.KAF.Common.dll</assembly>

<assembly>D:\DailyBuild\Source\KWMK\KAF\studio\test\KAF.Enterprise.Test\bin\Release\Test.KAF.Enterprise.dll</assembly>

<assembly>D:\DailyBuild\Source\KWMK\KAF\studio\test\KAF.Windows.Forms.Test\bin\Release\Test.KAF.Windows.Forms.dll</assembly>

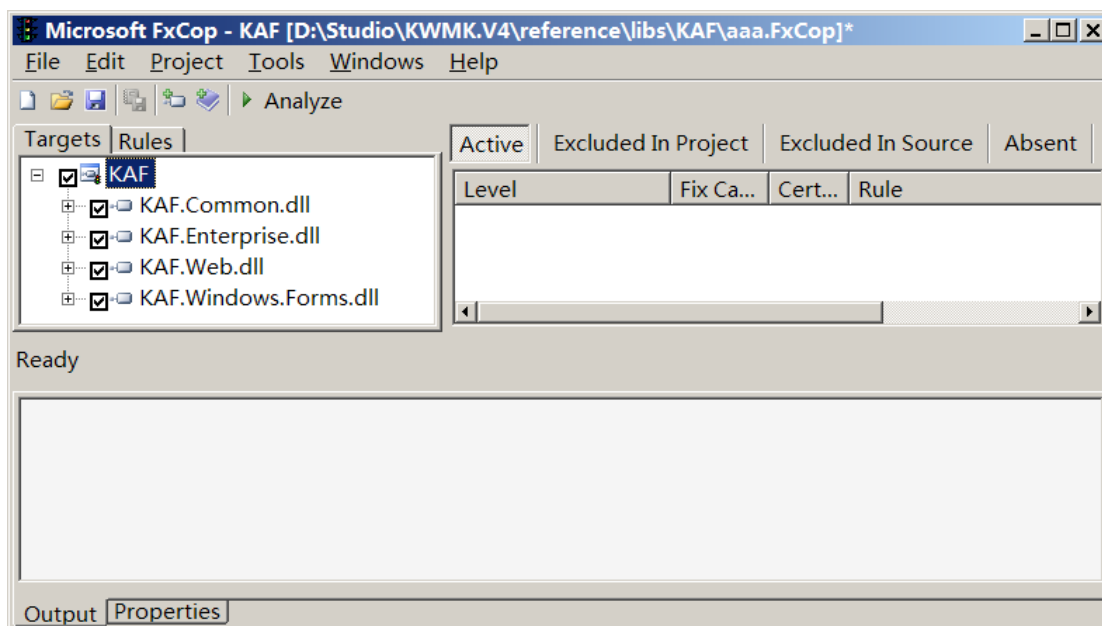
</assemblies>

</nunit>
```

(3)代码分析

代码分析，需要第三方工具的支持，我选择的是微软的FxCop，首先要生成待分析文件的模板文件。

首先打开Microsoft FxCop 1.36，如图二十八所示，在Targets面板的根节点中，点击右键选择“Add Targets”项目，添加要分析的程序集，然后保存为文件KAF.FxCop



<图二十八>

然后，在ccnet.config文件中添加相关配置，MSBuild和NUnit等步骤都是在CruiseControl.NET里面预定义好的，所以能直接使用<msbuild>或者<nunit>等标签，对于没有预定义的可执行程序，就都是用<exec>这个标签就可以了。

<exec>

<!--FxCop控制台的位置-->

<executable> C:\Program Files\Microsoft FxCop 1.36\FxCopCmd.exe</executable>

<!--工作目录，要分析的程序集的所在目录-->

<baseDirectory> D:\DailyBuild\Source\KWMK\reference\libs\KAF</baseDirectory>

<!--FxCop运行的参数-->

<buildArgs> /project:D:\DailyBuild\Source\KWMK\reference\libs\KAF\KAF.Common.Da
ilyBuild.FxCop

/out:D:\DailyBuild\Log\KWMK\KAF\studio\fxCop-results.xml</buildArgs>

<!--超时时长设置-->

<buildTimeoutSeconds> 120</buildTimeoutSeconds>

</exec >

<!--合并输出文件-->

<merge>

<!--把代码分析报告合并到KAF.Studio项目的所有报告中去-->

<files> <file> D:\DailyBuild\Log\KWMK\KAF\studio\fxCop-results.xml</file>

</files>

</merge>

(4)修改程序集的版本号

这一步骤，只有KAF.Trunk项目才有，因为KAF.Trunk是发布版本。我们先来看一下配置内容。

<msbuild>

 <executable> C:\WINDOWS\Microsoft.NET\Framework\v3.5\MSBuild.exe</execut

able>

 <workingDirectory> D:\DailyBuild\Source\KWMK\KAF\trunk\setup</workingDirec

tory>

 <projectFile> setup.proj</projectFile>

<buildArgs> /property:ReferencePath=..\..\..\reference;OutputPath=..\..\output;Solutio

nDirectory=..\</buildArgs>

 <timeout> 60</timeout>

 <logger> D:\DailyBuild\Source\KWMK\reference\libs\MSBuild\ThoughtWorks.Crui

seControl.MsBuild.dll</logger>

```
</msbuild>
```

大家可以看到这次MSBuild执行的不是KAF.sln文件了，而是位于项目目录中的setup文件夹下面的setup.proj，我们来看一下setup.proj的内容：

```
<?xml version="1.0" encoding="utf-8"?>

<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003"
DefaultTargets="CreateSetup">

  <PropertyGroup>

    <ProductVersion>$(CCNetLabel)</ProductVersion>

    <UpdateFilesCount>0</UpdateFilesCount>

  <DefineConstants>ProductVersion=$(ProductVersion)</DefineConstants>

</PropertyGroup>

  <Target Name="CreateSetup">

    <CallTarget Targets="UpdateVersion" />

    <CallTarget Targets="Build"/>

    <CallTarget Targets="Deploy"/>

  </Target>

  <Target Name="UpdateVersion">

    <Message Text="更新组件的版本号码" />

    <Message Text="传入的版本号码为 $(ProductVersion)" />

    <UpdateVersion SolutionDirectory="$(SolutionDirectory)"
Version="$(ProductVersion)">

      <Output TaskParameter="UpdateFilesCount"
```



```
PropertyName="UpdateFilesCount" />

</UpdateVersion>

<Message Text="更新的文件数量为 $(UpdateFilesCount)" />

</Target>

<Target Name="Build">

  <Message Text="编译解决方案"/>

  <MSBuild Projects="..\KAF.sln" Properties="OutputPath=$(OutputPath);
Configuration=Release" />

</Target>

<Target Name="Deploy">

  <Message Text="复制文件到引用目录"/>

  <Copy SourceFiles="..\output\KAF.Common.dll"
DestinationFolder="$(ReferencePath)\libs\KAF"/>

  <Copy SourceFiles="..\output\KAF.Enterprise.dll"
DestinationFolder="$(ReferencePath)\libs\KAF"/>

  <Copy SourceFiles="..\output\KAF.Web.dll"
DestinationFolder="$(ReferencePath)\libs\KAF"/>

  <Copy SourceFiles="..\output\KAF.Windows.Forms.dll"
DestinationFolder="$(ReferencePath)\libs\KAF"/>

  <Message Text="删除输出目录"/>

  <RemoveDir Directories="..\output" />

</Target>
```

```
<UsingTask
```

```
AssemblyFile= "..\..\reference\libs\MSBuild\MyTask\MSBuild.MyTask.dll"
```

```
TaskName= "MSBuild.MyTask.UpdateVersion" />
```

```
</Project>
```

简要说明一下：`<PropertyGroup>`是属性组的概念，其中包含的属性，是可以在整个脚本中使用的变量，如果要把一个外部变量赋给一个程序组中的变量的话，可以把外部变量名放在`$()`中，比如这句话：`<ProductVersion>$(CCNetLabel)</ProductVersion>`的含义就是把外部变量CCNetLabel的值赋给ProductVersion，DefaultTargets是默认的要执行的任务，在这个脚本中，默认的任务是“CreateSetup”，其中包含三个子任务，分别是：

```
<CallTarget Targets="UpdateVersion" />
```

```
<CallTarget Targets="Build"/>
```

```
<CallTarget Targets="Deploy"/>
```

“UpdateVersion”是用来更新版本号的自定义任务，“Build”任务用来编译项目，“Deploy”任务用来部署。

```
<Target Name="UpdateVersion">
```

```
<Message Text="更新组件的版本号码" />
```

```
<Message Text="传入的版本号码为 $(ProductVersion)" />
```

```
<UpdateVersion SolutionDirectory="$(SolutionDirectory)"
```

```
Version="$(ProductVersion)">
```

```
<Output TaskParameter="UpdateFilesCount" PropertyName="UpdateFilesCount"
```

```
/>
```

```
</UpdateVersion>
```

```
<Message Text="更新的文件数量为 $(UpdateFilesCount)" />
```

```
</Target>
```

上面的配置就是UpdateVersion任务的具体内容,其中<Message Text="更新组件的版本号码" /> 是用来向控制台输出消息的。

```
<UpdateVersion SolutionDirectory="$(SolutionDirectory)"
```

```
Version="$(ProductVersion)">
```

```
<Output TaskParameter="UpdateFilesCount" PropertyName="UpdateFilesCount" />
```

```
</UpdateVersion>
```

用来调用更新版本号的自定义任务 “UpdateVersion”, SolutionDirectory是传入的解决方案的所在目录, Version是要更新的版本号码, <Output

TaskParameter="UpdateFilesCount" 用来返回输出参数UpdateFilesCount, 即更新文件的数量, 这个输出参数主要是为了调试这个任务类的时候使用, 没有其他意义。然后在脚本中添加如下语句, 来引入包含自定义任务的程序集。

```
<UsingTask AssemblyFile="..\..\reference\libs\MSBuild\MyTask\MSBuild.MyTask.dll"
```

```
TaskName="MSBuild.MyTask.UpdateVersion" />
```

前面讲过了, 我的每一个项目的代码组织结构都如图四所示, 知道了解决方案文件*.sln所在的位置, 就知道了程序集的位置在\src\目录下面, 我会递归遍历src目录, 找到所有的

AssemblyInfo.cs 文件, 然后使用正则表达式把文件中的

[assembly: AssemblyVersion("1.0.0.0")]替换成传入的版本号Version。

下面我们来看一下这个自定义任务的源码:

```
using Microsoft.Build.Utilities;
```

```
using Microsoft.Build.Framework;
```

```
using System;

using System.Text.RegularExpressions;

using System.IO;

namespace MSBuild.MyTask
{
    /// <summary>

    /// 编写人员:曹文奎 caowenkui@126.com

    /// 编写日期 : 2008年12月26日 星期五

    /// 此类描述:此任务更新指定项目的AssemblyInfo信息

    /// </summary>

    public class UpdateVersion : Task
    {

        private delegate bool ProcessFileDelegate(string aFilePath);

        private string _solutionDirectory = string.Empty;

        private string _version = string.Empty;

        private int _filesCount = 0;

        private const string _assemblyInfo = "AssemblyInfo.cs";

        /// <summary>

        /// 指定解决方案所在的文件夹路径

        /// </summary>
```

[Required]

public string SolutionDirectory

```
{  
  
    get { return _solutionDirectory; }  
  
    set { _solutionDirectory = value; }  
  
}
```

/// <summary>

/// 要写入的版本号码

/// </summary>

[Required]

public string Version

```
{  
  
    get { return _version; }  
  
    set { _version = value; }  
  
}
```

/// <summary>

/// 返回更新文件的数量

/// </summary>

[Output]

public int UpdateFilesCount

```
{

    get { return _filesCount; }

    set { _filesCount = value; }

}

public override bool Execute()

{

    SearchDirectory(_solutionDirectory, UpdateAssemblyInfoFile);

    return true;

}

/// <summary>

/// 根据传入的文件夹路径，遍历文件夹

/// </summary>

/// <param name="aTargetDirectory">目标路径</param>

/// <param name="aProcessFile">处理文件的相关方法</param>

private void SearchDirectory(string aTargetDirectory, ProcessFileDelegate

aProcessFile)

{

    string[] fileEntries = Directory.GetFiles(aTargetDirectory, _assemblyInfo,

SearchOption.AllDirectories);

    _filesCount = 0;
```

```
foreach (string item in fileEntries)

{

    if (aProcessFile != null)

    {

        aProcessFile(item);

        _filesCount++;

    }

}

}

/// <summary>

/// 修改指定的AssemblyInfo.cs文件

/// </summary>

/// <param name="aFullFileName">文件的绝对路径，路径 + 文件名</param>

/// <returns>修改是否成功</returns>

private bool UpdateAssemblyInfoFile(string aFullFileName)

{

    try

    {

        string regularExpressions = "\\[assembly:

AssemblyVersion\\(\\\".*\\\"\\)\";

StreamReader streamReader = new StreamReader(aFullFileName,
```

```
System.Text.Encoding.UTF8);

    String allText = streamReader.ReadToEnd();

    streamReader.Close();

    Regex regex = new Regex(regularExpressions,
RegexOptions.Multiline);

    string formatVersion = String.Format("[assembly:
AssemblyVersion(\"{0}\")]\" , _version);

    allText = regex.Replace(allText, formatVersion);

    StreamWriter streamWriter = new StreamWriter(aFullFileName, false,
System.Text.Encoding.UTF8);

    streamWriter.Write(allText);

    streamWriter.Close();

    return true;

}

catch (Exception ex)

{

    throw new Exception(ex.Message);

}

}

private string NewPublishVersion(string aTrunkVesion)

{
```



```
string revision = MyTaskHelper.GetRevision(_solutionDirectory);

string[] versionArray = aTrunkVesion.Split(new char[1] { '.' },
StringSplitOptions.RemoveEmptyEntries);

string str = string.Empty;

for (int i = 0; i < 4; i++)

{

    str = str + versionArray[i] + ".";

    if (i == 2)

    {

        str = str + revision;

        break;

    }

}

return str;

}

}
```

开发人员在调试的时候，可以执行 setup 目录下面的 build.bat 批处理来调试这个 MSBuild 脚本，build.bat 的内容如下：

```
cls

%windir%\microsoft.net\framework\v3.5\msbuild                setup.proj

/property:ReferencePath=..\..\reference;OutputPath=..\..\output;SolutionDirectory
```

=..\CCNetLabel=1.0.0.123

pause

批处理中 CCNetLab 的值为瞎编的一个版本号码，就是为了便于调试使用，在真实环境中，也就是在 CruiseControl.NET 中，预定义了一些变量，可以给用户调用，CCNetLab 就是一个，返回当前项目的标签值，其他预定义的变量，大家可以参考 CruiseControl.NET 的帮助文档，批处理运行的时候如图二十九所示。

```

C:\WINDOWS\system32\cmd.exe - build.bat

D:\Studio\KUMK.U4\KAF\trunk\setup>C:\WINDOWS\microsoft.net\framework\v3.5\msbuild setup.proj /property:ReferencePath=..\reference;OutputPath=..\output;SolutionDirectory=..\CCNetLabel=1.0.0.123
Microsoft (R) 生成引擎版本 3.5.30729.1
[Microsoft .NET Framework 版本 2.0.50727.3082]
版权所有 (C) Microsoft Corporation 2007。保留所有权利。

生成启动时间 2009-08-07 14:06:09。
节点 0 上的项目 "D:\Studio\KUMK.U4\KAF\trunk\setup\setup.proj" (默认目标)。
更新组件的版本号码
传入的版本号码为 1.0.0.123
更新的文件数量为 12
Build:
  编译解决方案
  项目 "D:\Studio\KUMK.U4\KAF\trunk\setup\setup.proj" (1) 正在节点 0 (默认目标) 上生成 "D:\Studio\KUMK.U4\KAF\trunk\KAF.sln" (2)。
    正在生成解决方案配置 "Release|Any CPU"。
  项目 "D:\Studio\KUMK.U4\KAF\trunk\KAF.sln" (2) 正在节点 0 (默认目标) 上生成 "D:\Studio\KUMK.U4\KAF\trunk\src\KAF.Common\KAF.Common.csproj" (3)。
    Processing 0 EDMX files.
    Finished processing 0 EDMX files.
  PrepareForBuild:
    正在创建目录 "..\..\output\"。
    正在创建目录 "obj\Release\"。
  CoreResGen:
    正在将资源文件 "10\KAF.IO.Ports.KUGSMSender.resx" 处理到 "obj\Release\KAF.IO.Ports.KUGSMSender.resources" 中。
  CopyFilesMarkedCopyLocal:
    正在将文件从 "..\..\..\reference\libs\IBM\BM.Data.DB2.dll" 复制到 "..\..\output\IBM.Data.DB2.dll"。
    正在将文件从 "..\..\..\reference\libs\IBM\BM.Data.DB2.Entity.dll" 复制到 "..\..\output\IBM.Data.DB2.Entity.dll"。
    正在将文件从 "..\..\..\reference\libs\IBM\BM.Data.U2.Ucinet.dll" 复制到 "..\..\output\IBM.Data.U2.Ucinet.dll"。
  CopyFilesToOutputDirectory:
    正在将文件从 "obj\Release\KAF.Common.dll" 复制到 "..\..\output\KAF.Common.dll"。
    KAF.Common -> D:\Studio\KUMK.U4\KAF\trunk\output\KAF.Common.dll
    正在将文件从 "obj\Release\KAF.Common.pdb" 复制到 "..\..\output\KAF.Common.pdb"。
    已完成生成项目 "D:\Studio\KUMK.U4\KAF\trunk\src\KAF.Common\KAF.Common.csproj" (默认目标)。
  项目 "D:\Studio\KUMK.U4\KAF\trunk\KAF.sln" (2) 正在节点 0 (默认目标) 上生成 "D:\Studio\KUMK.U4\KAF\trunk\src\KAF.Configuration\KAF.Configuration.csproj" (4)。
    Processing 0 EDMX files.
    Finished processing 0 EDMX files.
  PrepareForBuild:
    正在创建目录 "obj\Release\"。
  CopyFilesToOutputDirectory:
    正在将文件从 "obj\Release\KAF.Configuration.dll" 复制到 "..\..\output\KAF.Configuration.dll"。
    KAF.Configuration -> D:\Studio\KUMK.U4\KAF\trunk\output\KAF.Configuration.dll
    正在将文件从 "obj\Release\KAF.Configuration.pdb" 复制到 "..\..\output\KAF.Configuration.pdb"。
    已完成生成项目 "D:\Studio\KUMK.U4\KAF\trunk\src\KAF.Configuration\KAF.Configuration.csproj" (默认目标)。
  
```

<图二十九>

执行完<Target Name="UpdateVersion">后，则解决方案中包含的所有程序集的版本号码都更新完毕，然后执行<Target Name="Build">项目，生成发布程序集。

(5)部署程序集

生成最后的结果程序集后，我们要发布它，对于 KAF 这个项目来说，就是要把它复制到公共引用程序库中的.\reference\libs\KAF 下面，然后把它提交到版本库中。在 KAF 项目的 setup.proj 脚本中的<Target Name="Deploy">用来完成向目标目录的复制操作。

```

<Target Name="Deploy">

  <Message Text="复制文件到引用目录"/>

  <Copy SourceFiles="..\output\KAF.Common.dll"

DestinationFolder="$(ReferencePath)\libs\KAF"/>

  <Copy SourceFiles="..\output\KAF.Enterprise.dll"

DestinationFolder="$(ReferencePath)\libs\KAF"/>

  <Copy SourceFiles="..\output\KAF.Web.dll"

DestinationFolder="$(ReferencePath)\libs\KAF"/>

  <Copy SourceFiles="..\output\KAF.Windows.Forms.dll"

DestinationFolder="$(ReferencePath)\libs\KAF"/>

  <Message Text="删除输出目录"/>

  <RemoveDir Directories="..\output" />

</Target>

```

复制完成后，在持续集成 KAF.Trunk 项目的配置任务中最后一个任务就是提交到版本库中去。

```

<exec>

  <executable>C:\Program Files\Subversion\bin\svn.exe</executable>

  <baseDirectory>D:\DailyBuild\Source\KWMK\reference\libs\KAF</baseDirectory>

>

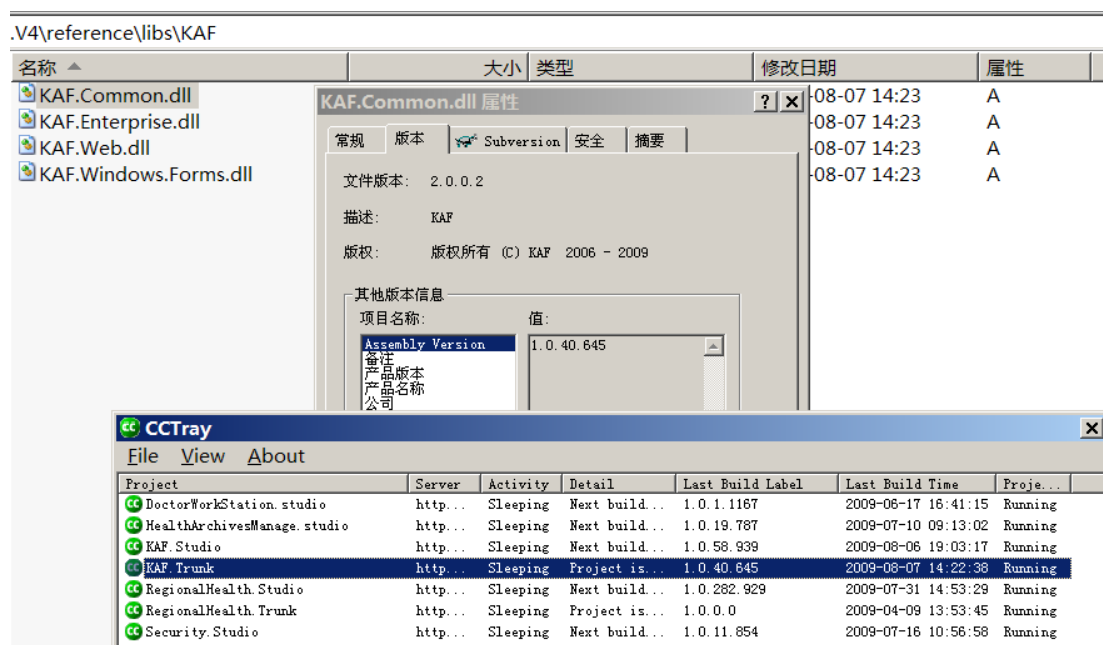
  <buildArgs>commit -m "ccnet commit" --username root --password

aaaaaaaa</buildArgs>

</exec >

```

然后每个开发人员的机器上面都添加了一个计划任务，它会定时的调用 TortoiseSVN 的命令把最新版本的 KAF 项目更新下来。大家可以在图三十中看到 KAF 程序集的版本号码和 CCTray 中的标签的对应关系。



<图三十>

(6)MSI 安装包

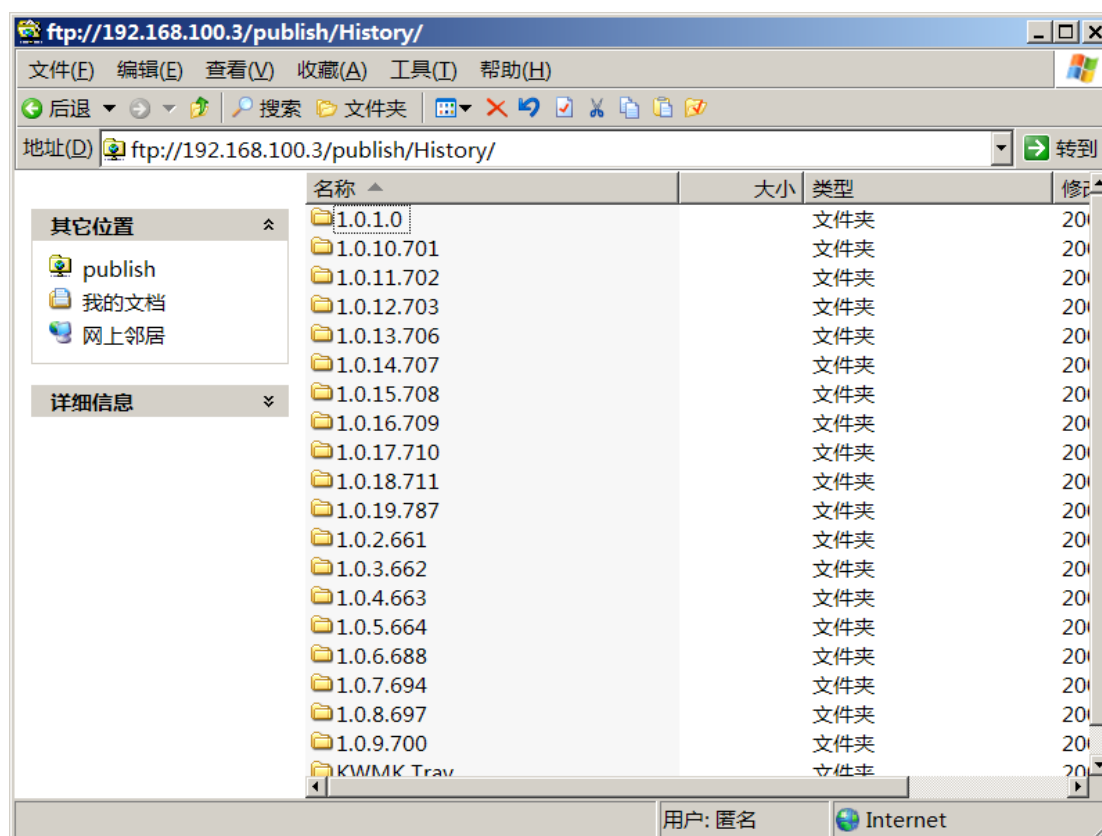
对于客户端程序来说，它们的发布环节，不仅仅是把程序集复制到指定目录这么简单，一般都是提供一个安装包。对于这种需求，我采用的是 Wix 脚本技术，下面我们来看一个例子。

我负责的项目中有一个项目叫做 [HealthArchivesManage](#)(健康档案管理)这是一个桌面程序，用来采集调查表的数据，然后通过 WCF 上传到中心服务器，它最后发布的环节，[CruiseControl.NET](#)使用 MSBuild 执行 [HealthArchivesManage](#) 这个项目中的 setup 目录下的 setup.wixproj 文件，setup.wixproj 是个安装类型项目，包含了一个使用 Wix 技术编写的安装脚本，生成 MSI 格式的可执行程序。

每次 [HealthArchivesManage.Trunk](#) 的持续集成项目执行最后发布环节的时候，都会把生成的 MSI 文件复制到我架设的一个 FTP 上面的两个文件夹中，一个叫做 History，一个叫做 Latest，其中 History 按照版本号存放每一次持续集成生成的 MSI 文件，这样就和前边讲的每个项目的 tags 中的标签对应上了，每一个发布的 MSI 对应的源码都能在 tags 目录下面找到。当多个用户需要一个项目的不同版本的时候，这个功能会非常有用。

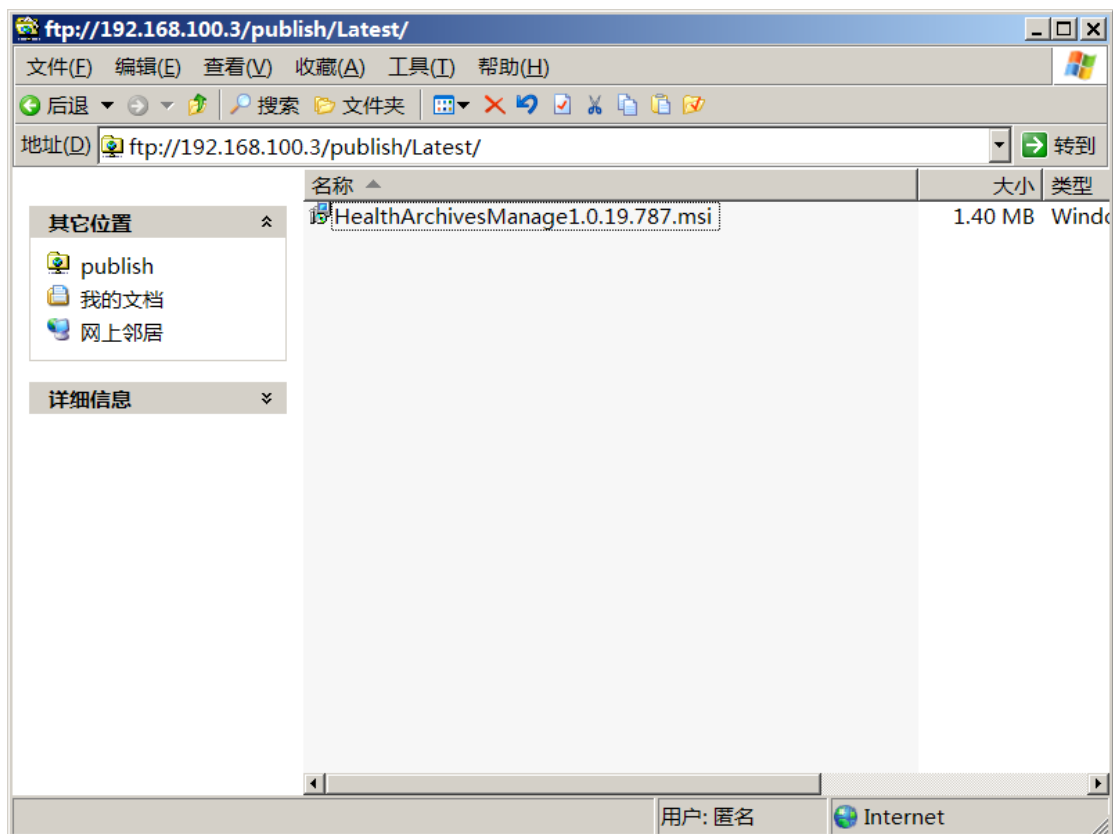
在 Latest 目录中存放的是最新生成的 MSI 文件。

如图三十一所示，是 History 目录。



<图三十一>

Latest 目录，如图三十二所示。



<图三十二>

Setup.wxs 文件的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>

<?include config.wxi ?>

<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi"

xmlns:netfx="http://schemas.microsoft.com/wix/NetFxExtension">

  <Product Name="$(var.ProductShortName) $(var.ProductVersion)"

Id="$(var.ProductCode)" UpgradeCode="$(var.UpgradeCode)" Language="2052"

Codepage="936" Version="$(var.ProductVersion)"

Manufacturer="$(var.Manufacturer)">

    <Package Manufacturer="$(var.Manufacturer)" InstallerVersion="200"

Languages="2052" SummaryCodepage="936" Compressed="yes" />
```

```

<Media Id="1" Cabinet="HealthArchivesManage.cab" EmbedCab="yes"/>

<PropertyRef Id="NETFRAMEWORK20" />

<Condition Message="本程序需要.NET Framework 2.0 框架的支持，请先安装框架然
后再运行本安装程序。">Installed OR NETFRAMEWORK20</Condition>

<Directory Id="TARGETDIR" Name="SourceDir">

  <Component Id="updateRegistry"

    Guid='{13EF4211-D022-4edd-B019-949BE022F616}'>

      <RegistryKey Root="HKLM" Key="SOFTWARE\KWMK" Action="create" >

        <RegistryKey Key="Tray" Action="create" >

          <RegistryKey Key="UpdateItem" Action="create" >

            <RegistryKey Key="$(var.InstallName)"

              Action="createAndRemoveOnUninstall" >

                <RegistryValue Name="Version" Value="$(var.ProductVersion)"

                  Type="string" />

                <RegistryValue Name="UpdateDateTime" Value="2008-01-21

                  19:14:00" Type="string" />

                <RegistryValue Name="ManifestUri" Value="$(var.UpdateURL)"

                  Type="string" />

                <!--设置默认自动更新时间间隔为5分钟-->

                <RegistryValue Name="UpdateInterval" Value="300000"

                  Type="integer" />

                <RegistryValue Name="ApplicationId" Value="$(var.InstallName)"

```

```

    Type="string" />

        <RegistryValue Name="Description"

Value="$ (var.ProductShortName)" Type="string" />

        </RegistryKey>

        </RegistryKey>

        </RegistryKey>

        </RegistryKey>

    </Component>

    <Directory Id="ProgramFilesFolder" Name="PFiles">

        <Directory Id="KWMK" Name="KWMK">

            <Directory Id="INSTALLDIR" Name="HealthArchivesManage">

                <Component Id="CompriseFile"

Guid="E60864BF-49C4-4a0d-B6CC-3B384ADD557F">

                    <File Id="HealthArchivesManage.exe"

Name="HealthArchivesManage.exe" Source="..\output\HealthArchivesManage.exe" />

                    <File Id="HealthArchivesManage.exe.config"

Name="HealthArchivesManage.exe.config"

Source="..\output\HealthArchivesManage.exe.config" />

                    <File Id="config.xml" Name="config.xml" Source="..\output\config.xml"

/>

                </Component>

                <Directory Id="image" Name="image">

```



```
<Component Id="image"

Guid="117469DB-9A4C-439f-9E9A-B707F09A5176">

    <File Id="bg1.gif" Name="bg1.gif" Source="..\output\image\bg1.gif"

/>

    <File Id="bg.jpg" Name="bg.jpg" Source="..\output\image\bg.jpg"

/>

    <File Id="pic_a_2.png" Name="pic_a_2.png"

Source="..\output\image\pic_a_2.png" />

    <File Id="pic_a_3.png" Name="pic_a_3.png"

Source="..\output\image\pic_a_3.png" />

    <File Id="pic_b_1.png" Name="pic_b_1.png"

Source="..\output\image\pic_b_1.png" />

    <File Id="pic_b_2.png" Name="pic_b_2.png"

Source="..\output\image\pic_b_2.png" />

    <File Id="pic_b_3.png" Name="pic_b_3.png"

Source="..\output\image\pic_b_3.png" />

    <File Id="pic_c_1.png" Name="pic_c_1.png"

Source="..\output\image\pic_c_1.png" />

    <File Id="pic_c_2.png" Name="pic_c_2.png"

Source="..\output\image\pic_c_2.png" />

    <File Id="pic_c_3.png" Name="pic_c_3.png"

Source="..\output\image\pic_c_3.png" />
```

```
<File Id="pic_d_1.png" Name="pic_d_1.png"
Source= "..\output\image\pic_d_1.png" />

<File Id="pic_d_2.png" Name="pic_d_2.png"
Source= "..\output\image\pic_d_2.png" />

<File Id="pic_d_3.png" Name="pic_d_3.png"
Source= "..\output\image\pic_d_3.png" />

<File Id="pic_e_1.png" Name="pic_e_1.png"
Source= "..\output\image\pic_e_1.png" />

<File Id="pic_e_2.png" Name="pic_e_2.png"
Source= "..\output\image\pic_e_2.png" />

<File Id="pic_e_3.png" Name="pic_e_3.png"
Source= "..\output\image\pic_e_3.png" />

<File Id="pic_f_1.png" Name="pic_f_1.png"
Source= "..\output\image\pic_f_1.png" />

<File Id="pic_f_2.png" Name="pic_f_2.png"
Source= "..\output\image\pic_f_2.png" />

<File Id="pic_f_3.png" Name="pic_f_3.png"
Source= "..\output\image\pic_f_3.png" />

<File Id="pic_g_1.png" Name="pic_g_1.png"
Source= "..\output\image\pic_g_1.png" />

<File Id="pic_g_2.png" Name="pic_g_2.png"
Source= "..\output\image\pic_g_2.png" />
```

```
<File Id="pic_g_3.png" Name="pic_g_3.png"
Source="..\output\image\pic_g_3.png" />

<File Id="pic_h_1.png" Name="pic_h_1.png"
Source="..\output\image\pic_h_1.png" />

<File Id="pic_h_2.png" Name="pic_h_2.png"
Source="..\output\image\pic_h_2.png" />

<File Id="pic_d_4.png" Name="pic_d_4.png"
Source="..\output\image\pic_d_4.png" />

<File Id="pic_e_4.png" Name="pic_e_4.png"
Source="..\output\image\pic_e_4.png" />

<File Id="pic_f_4.png" Name="pic_f_4.png"
Source="..\output\image\pic_f_4.png" />

</Component>

</Directory>

</Directory>

</Directory>

</Directory>

<Directory Id="ProgramMenuFolder" Name="ProgramMenuFolder">

  <Directory Id="ProgramMenuFolderKWMK" Name="KWMK">

    <Component Id="ProgramMenuKWMK"
Guid="51F97418-ABF1-43B7-BB4C-2E77437FC7F1">

      <RegistryKey Root="HKCU"
```

```
Key="Software\KWMK\HealthArchivesManage\Program">

    <RegistryValue Value="HealthArchivesManage" Type="string"

KeyPath="yes" />

    </RegistryKey>

    <Shortcut Id="shortcut.prog" WorkingDirectory="INSTALLDIR"

Directory="ProgramMenuFolderKWMK" Name="健康档案采集卡"

Target="[INSTALLDIR]\HealthArchivesManage.exe" Icon="HealthArchivesManage.exe"

IconIndex="0" />

    <RemoveFolder Id="RemoveShorcutFolder" On="uninstall" />

    </Component>

</Directory>

</Directory>

<Directory Id="DesktopFolder" Name="Desktop">

    <Component Id="DesktopKWMK"

Guid="aede1637-df5a-4c41-94b6-f077d03e5372">

    <RegistryKey Root="HKCU"

Key="Software\KWMK\HealthArchivesManage\desktop">

    <RegistryValue Value="HealthArchivesManage" Type="string"

KeyPath="yes" />

    </RegistryKey>

    <Shortcut Id="shortcut.desk" WorkingDirectory="INSTALLDIR"

Directory="DesktopFolder" Name="健康档案采集卡"
```

```
Target="[INSTALLDIR]\HealthArchivesManage.exe" Icon="HealthArchivesManage.exe"
```

```
IconIndex="0" />
```

```
</Component>
```

```
</Directory>
```

```
</Directory>
```

```
<Property Id="WIXUI_INSTALLDIR" Value="INSTALLDIR" />
```

```
<UIRef Id="WixUI_InstallDir" />
```

```
<Feature Id="main" Title="main" Level="1">
```

```
<ComponentRef Id="CompriseFile" />
```

```
<ComponentRef Id="image" />
```

```
<ComponentRef Id="updateRegistry" />
```

```
<ComponentRef Id="ProgramMenuKWMK" />
```

```
<ComponentRef Id="DesktopKWMK" />
```

```
</Feature>
```

```
<InstallExecuteSequence>
```

```
<RemoveExistingProducts After="InstallInitialize" />
```

```
</InstallExecuteSequence>
```

```
<Icon Id="HealthArchivesManage.exe"
```

```
SourceFile="..\output\HealthArchivesManage.exe" />
```

```
</Product>
```

```
</Wix>
```

下面我把 KAF 项目的完整的配置贴在下面，下面的配置是把 KAF.Trunk 和 KAF.Studio

两个项目整合到一起了，请大家注意：

```
<project name="KAF.Studio" queue="KWMK" queuePriority="1">

  <!--默认的工作目录-->

  <workingDirectory>D:\DailyBuild\Source\KWMK\KAF\studio</workingDirectory>

  <!--当前项目的集成过程中生成的结果文件的存放位置，为了发布到CruiseControl.NET的网站中做准备-->

  <artifactDirectory>D:\DailyBuild\Log\KWMK\KAF\studio</artifactDirectory>

  <!--配置当前项目在网站中的页面-->

  <webURL>http://192.168.100.3/ccnet/server/local/project/KAF.Studio/ViewProjectReport.aspx</webURL>

  <category>KWMK</category>

  <!--配置触发器类型-->

  <triggers>

    <intervalTrigger name="KAF.Trigger.studio" seconds="10"

buildCondition="IfModificationExists"/>

  </triggers>

  <!--配置标签类型-->

  <labeller type="CustomLabeller">

    <majorVersion>1</majorVersion>

    <minorVersion>0</minorVersion>

    <solutionFilePath>D:\DailyBuild\Source\KWMK\KAF\studio</solutionFilePath>

  </labeller>
```

```

<!--配置源码控制-->

<sourcecontrol type="svn">

    <autoGetSource>true</autoGetSource>

    <trunkUrl>svn://192.168.100.3/KWMK/KAF/studio</trunkUrl>

    <workingDirectory>D:\DailyBuild\Source\KWMK\KAF\studio</workingDirectory>

    <username>root</username>

    <password>www.hao123.com</password>

</sourcecontrol>

<!--任务集合-->

<tasks>

<!--编译项目-->

    <msbuild>

        <executable>C:\WINDOWS\Microsoft.NET\Framework\v3.5\MSBuild.exe</execut
able>

        <workingDirectory>D:\DailyBuild\Source\KWMK\KAF\studio</workingDirectory>

        <projectFile>KAF.sln</projectFile>

        <buildArgs>/property:Configuration=Release</buildArgs>

        <timeout>60</timeout>

        <logger>D:\DailyBuild\Source\KWMK\reference\libs\MSBuild\ThoughtWorks.Crui
seControl.MsBuild.dll</logger>

    </msbuild>

<!--自动化测试-->
  
```

```
<nunit>

<path>D:\DailyBuild\Source\KWMK\reference\tools\nunit\nunit-console.exe</path>

<assemblies>

<assembly>D:\DailyBuild\Source\KWMK\KAF\studio\test\Test.KAF.Common\bin\Release\Test.KAF.Common.dll</assembly>

</assemblies>

</nunit>

<!--更新版本号码，并复制到公共引用目录-->

<msbuild>

<executable>C:\WINDOWS\Microsoft.NET\Framework\v3.5\MSBuild.exe</executable>

<workingDirectory>D:\DailyBuild\Source\KWMK\KAF\studio\setup</workingDirectory>

<projectFile>setup.proj</projectFile>

<buildArgs>/property:ReferencePath=..\..\reference;OutputPath=..\..\output;SolutionDirectory=..\</buildArgs>

<timeout>60</timeout>

<logger>D:\DailyBuild\Source\KWMK\reference\libs\MSBuild\ThoughtWorks.CruiseControl.MsBuild.dll</logger>

</msbuild>

<!--把公共引用程序库下面的程序集提交到版本库中-->

<exec>

<executable>C:\Program Files\Subversion\bin\svn.exe</executable>
```



```
<baseDirectory>D:\DailyBuild\Source\KWMK\reference\libs\KAF</baseDirectory>
```

```
<buildArgs>commit -m "ccnet commit" - -username root - -password
```

```
aaaaaaa</buildArgs>
```

```
</exec >
```

```
<!--代码检查-->
```

```
<exec>
```

```
<executable>C:\Program Files\Microsoft FxCop
```

```
1.36\FxCopCmd.exe</executable>
```

```
<baseDirectory>D:\DailyBuild\Source\KWMK\reference\libs\KAF</baseDirectory>
```

```
<buildArgs>/project:D:\DailyBuild\Source\KWMK\reference\libs\KAF\KAF.Common.Da
```

```
ilyBuild.FxCop
```

```
/out:D:\DailyBuild\Log\KWMK\KAF\studio\fxCop-results.xml</buildArgs>
```

```
<buildTimeoutSeconds>120</buildTimeoutSeconds>
```

```
</exec >
```

```
<!--合并代码检查结果到集成结果文件存放位置，为了在网站中进行查看-->
```

```
<merge>
```

```
<files>
```

```
<file>D:\DailyBuild\Log\KWMK\KAF\studio\fxCop-results.xml</file>
```

```
</files>
```

```
</merge>
```

```
</tasks>
```

```
<publishers>
```

```
<xmllogger logDir="D:\DailyBuild\Log\KWMK\KAF\studio\buildlogs" />
```

```
</publishers>
```

```
</project>
```

至此，我就把持续集成中的基本环节都给大家讲了一下，我并没有详细的描述

[CruiseControl.NET](#)中各个技术点包含的内容，这些大家可以去查询它的帮助文件。我只是把我的实际应用的案例写出来希望给大家一个参考，希望能对大家的工作有所帮助。

个人感觉就是持续集成本身并不难配置，但是在各个环节涉及到的技术点比较多，如果能把这些工具掌握好，配置出一个适合自己团队的持续集成环境，会对整个团队的开发效率有很大的提高。

在搭建这个平台的过程中，我有一个体会就是对于一个程序员来说，能够快速的掌握一种技术的能力是非常重要的。

至于数据库的持续集成和MSBuild以及Wix脚本的编写，我以后会补上，要写的实在是太多了，所以这次就暂时不写了。