目录

[1. 持续集成是个什么鬼 2](#_Toc496707985)

[2. 有什么好处 2](#_Toc496707986)

[3. 需要什么 2](#_Toc496707987)

[3.1具有版本控制功能的代码库 2](#_Toc496707988)

[3.2构建工具 2](#_Toc496707989)

[3.3测试 3](#_Toc496707990)

[3.4使用持续集成，通常还涉及到另外两个概念： 3](#_Toc496707991)

[3.5 CI工具 4](#_Toc496707992)

[4. 怎样做 4](#_Toc496707993)

[4.1例子背景描述 4](#_Toc496707994)

[4.2集成代码 5](#_Toc496707995)

[4.3自动构建 5](#_Toc496707996)

[4.4自动部署 5](#_Toc496707997)

[4.5功能测试（集成测试） 6](#_Toc496707998)

[4.6交付 6](#_Toc496707999)

[4.7总结 6](#_Toc496708000)

# 持续集成是个什么鬼

持续集成（CI, Continuous Integration），是极限编程（XP, Extremely Programming）中的一种实践。它主张每天提交若干次代码，每次提交都经过自动化编译和测试（最低要求为单元测试）后，集成到代码库的主干上去。

# 有什么好处

通过持续集成，我们可以：

快速发现问题（逻辑错误、bug等）。并且，我们只需要在很小范围内的代码修改中去寻找问题所在。

避免了在项目后期才做集成所造成的混乱。当大家在项目后期才进行代码/功能集成的话，所产生的冲突必然增多。并且当你发现问题或bug的时候，需要检查代码的范围也会增大。这样会提高集成的难度，甚至难以集成。

当一个bug导致必须回滚代码时，所损失的代码量也大大地降低。

可以让产品快速迭代，同时还能保持质量。

不需要固定一个人去维护一段代码，功能的添加或者程序的修改都可以通过自动化测试去保证其正确性。缓解了猿们在修改不熟悉的代码时那种害怕牵一发而动全身的情况。

# 需要什么

实现持续集成，一般需要以下内容：

## 3.1具有版本控制功能的代码库

例如：SVN, Git。相信现在的项目没有不对代码进行版本管理的，所以这方面内容大家也应该非常熟悉。在这里不再详述。

## 3.2构建工具

在持续集成的过程中，需要对已存在的或者新提交的代码进行编译、打包等操作。这样，就需要构建工具帮助构建一个编译环境，并对代码进行编译、集成、打包等操作。而构建的方式越简单越好，最好是一句命令就可以启动构建。现在，各种语言都有自己的构建工具，例如Java中的Maven、Gradle、Ant，前端中的Grunt、Gulp等等，好好利用这些工具，就能帮你完成这部分工作。

## 3.3测试

测试是持续集成中重要的一环。代码提交前，需要在本地运行单元测试，通过测试后再提交代码。构建完成后，需要运行全部测试（单元测试，功能测试，端到端测试）以确保产品质量。如果有一个测试没有通过，那么这次提交的代码不能进入主干；或者这次构建的产物是一个失败的构建品，不能用于发布。另外，由于持续集成依赖于这些测试去保证产品质量，所以测试的覆盖率要尽可能高。测试覆盖率不够高（包含代码覆盖率和功能覆盖率），就无法充分反映代码的变动是否对系统带来影响。而低覆盖率的测试，压根就无法保证产品质量。当上线的时候才发现问题就太迟了。

## 3.4使用持续集成，通常还涉及到另外两个概念：

**3.4.1 持续交付**

持续交付（Continuous Delivery），指的是频繁地将软件的新版本，交付给质量团队或者用户，以供评审。如果评审通过，代码就进入生产阶段。

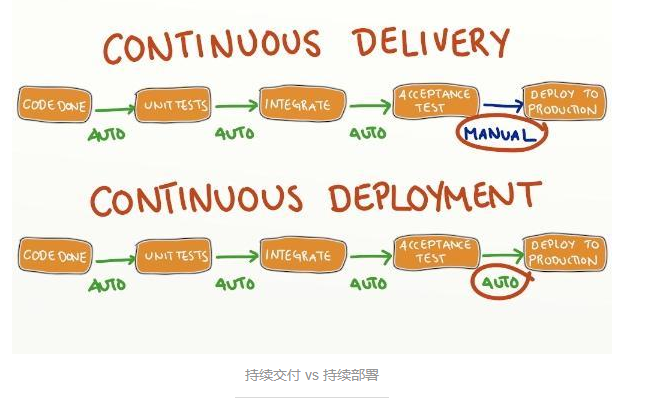
持续交付可以看作持续集成的下一步。它强调的是，不管怎么更新，软件是随时随地可以交付的。

**3.4.2 持续部署**

持续部署（Continuous Deployment），是持续交付的下一步，指的是代码通过评审以后，自动部署到生产环境。

持续部署的目标是，代码在任何时刻都是可部署的，可以进入生产阶段。

* + 1. **用一张图描述两者的关系**



## 3.5 CI工具

CI工具的作用是将整个CI过程管理起来并自动化，结果可视化。部分工具还结合了CD（持续交付）的功能。现在已经有很多CI工具去满足你不同的需求，例如Jenkins，专为Github开源项目提供的Travis，.Net用的CruiseControl.Net。他们各有特色，根据自己的需求选择适合自己的工具即可。

# 怎样做

好了，当集齐以上内容后，我们可以开始召唤神龙了。我们用一个例子来介绍一下一个典型流程是怎样的。

## 4.1例子背景描述

假设我们现在有一个产品P，以war包形式发布，由三个模块module A, module B, module C构建而成。三个模块的关系为：A和B为独立模块提供不同功能。C依赖A和B，然后构成产品P。我们使用了Git作为我们代码库的版本管理工具，用Java进行开发，maven作为我们的构建工具。在每个模块里，有我们基于JUnit写的单元测试代码。独立于三个模块外，有一块代码，也是基于JUnit写的，作为我们的功能测试代码（集成测试）。

## 4.2集成代码

当我们完成开发工作，需要提交代码到代码库前，我们至少需要在本地跑一次单元测试，在保证全部测试通过后，才可以将代码提交至我们的代码库Git上面去。例如，在我们上面描术的项目中，我对module A的代码进行了修改，那我最起码得在本地运行一次mvn test（执行Maven命令，test代表将会执行到maven default生命周期中从validate到test阶段）， 执行成功后，我才会将代码commit and push到远程Git库上去。要做到这样效果的话，就需要保证单元测试代码也同步完成。而在极限编程中（XP），人们比较倾向于测试驱动开发（TDD, Test-driven Development）的实践，通过提前写好自动化的单元测试，保证好每一步功能开发的质量。

## 4.3自动构建

通过CI工具，可以设置一个勾子，当代码提交后触发相应构建。例如，我们提交了module A的代码时，Jenkins会扫描到我们这次提交，勾子触发module A的构建。这个过程会做如下操作：

1.Jenkins调用Git插件，从Git库上下载最新代码；

2.Jenkins调用Maven插件，执行Maven命令（一般为mvn install，如果需要上传至远端Maven库，也可以执行mvn deploy）对该模块进行构建。经过编译、通过单元测试后，便可以打包并安装到本地Maven库，以供其它依赖所用。这次构建成功，意味module A在模块自身的单元测试范围内是正常的。

3.因为module A是包含在产品P里面，所以，也需要回归一产品功能测试。由于module C依赖A，并构建成产品。所以在CI工具里面，我们需要配置好在module A构建成功后，自动触发module C的构建，经过类似步骤1、2这样的构建后，最终会生成产品P的war包。而C的构建成功，只代表着通过了module C自身的单元测试，还不能对生成的war包进行功能测试。然后就要看我们下一步的工作－－自动部署了。

## 4.4自动部署

在功能测试之前，我们需要在CI工具里配置一项任务，用于将最新构建出来的产品包部署到测试工环境中去。这个任务由产品构建任务成功而被触发，而部署方式根据不同使用方式及不同的实际情况而多种多样。例如通过脚本将新构建的war包上传至指定位置，等待web容器自动扫描及部署。能或者产品有自己的安装脚本，我们在任务中配置好运行安装脚本，就可以自动将产品部署到指定的测试环境中去。

## 4.5功能测试（集成测试）

当部署成功后，真正的功能测试就可以开始了。一般情况下，我们可以独立出一块代码，基于JUnit编写好我们的功能测试代码（JUnit是作为测试的入口以及基本测试框架。如果你的需求比较复杂，那你完全可以将其它三方框架与JUnit集成使用）。功能测试过程和构建过程非常相似，均是依赖Git和Maven去完成：

1.Jenkins调用Git插件，从Git库上下载最新代码；

2.Jenkins调用Maven插件，执行Maven命令：mvn clean test。

区别在于功能测试阶段，Maven只执行到default生成周期的test阶段，不会执行后面的package和install。因为它只需要Maven帮忙运行测试代码即可，它本身没有什么可以构建的。

**P.S.**

如果还需要更复杂的端到端测试的话，可能就需要准备更复杂的部署脚本，或者预先准备好整套端到端测试环境，后面只需要部署好war包即可。但无论怎样，最终原来理还是相同。

## 4.6交付

当新提交代码后构建出来的产品包，通过了各种各样残酷的测试后，就说明这个包是稳定的，能达到基本交付条件的（前提是自动化测试的覆盖率足够高，当然，有一些极端的情况需要人工测试的另说）。那么，我们就可以将这个包放到指定目录作为交付品，供其它测试团队获取并进行进一步的测试，甚至供生产环境部署使用。

## 4.7总结

持续集成作为极限编程中的一个实践，现在已被很多公司使用。但是，使用持续集成，并不是说你得接受极限编程的全部东西。相反，它可以独立开来，与其它实践结合使用。

持续集成是敏捷开发中快速迭代的重要保证。

自动化测试是持续集成中重要一环，要真正用好持续集成，就要尽量提高自动化测试的覆盖率。