

持续集成 CI

Maven2, Hudson ...

CI 背景

□ 项目风险

- 没有可部署软件，很晚才发现bug，缺少项目可见性，低品质软件.....

□ 什么是持续集成？

- CI服务器循环检查（每天，每小时，每半小时.....）代码提交，启动一个过程（编译，测试，部署.....）来构建项目。

CI 价值

- 降低风险
 - 尽早的发现bug，静态代码分析
 - 自动化
 - 自动编译，自动测试，自动部署，自动审查.....
 - 增强项目可见性
 - maven site，测试报告..... 项目健康状态
 - 可部署的软件
 - 建立对产品的信心
 - 测试都通过了，可部署的软件一直存在！
-

CI 实践 – Martin Flower

- 维护一个SVN仓库和一个Maven仓库
 - 自动化构建
 - 构建自测试
 - 频繁提交
 - 集成服务器为每次提交执行构建
 - 保持快速构建
 - 模拟产品环境测试
 - 所有人都能很方便的拿到最新的可运行程序
 - 所有人都能看到项目的状态
 - 自动化部署
-

维护一个SVN仓库和一个Maven仓库

□ 将以下东西提交至Subversion

- 源代码，安装脚本，配置脚本，测试脚本，数据库Schema，Sql脚本.....
- svn commit的时候，写注释！
- 原子提交：一个任务一次提交

□ 不要将jar, war等内容提交到Subversion

- 使用Maven仓库，如Nexus

□ 最后，使用svn中的源码 + Nexus中的lib，就能构建你的项目

自动化构建

□ 须满足

- 一台virgin机器 ->
- svn checkout ->
- mvn install ->
- 得到可运行程序

□ 当然，你可能还需要

- 安装svn客户端，安装maven
- 配置一下maven profile

□ 所有机器使用同样的环境

- 同样的jdk, maven, svn, os版本

构建自测试

- 很多风险是通过测试来避免的
 - 测试是构建过程必要的一个阶段
 - mvn install默认会执行test阶段
 - 好好使用TestNG或者JUnit组织测试
 - TestNG支持测试分组
 - 结合TestNG该特性和Maven的生命周期，可以将单元测试，集成测试分层。
 - 当测试失败的时候，重构代码！
 - 如果你跳过测试，用不了多久，你会发现你再也没欲望写测试用例了，因为那个时候大片的测试都失败了。
-

频繁提交

- 高频度的提交能尽快的发现冲突
 - 越早发现冲突，越容易解决
 - 步骤是：
 - svn update
 - 解决冲突
 - 编码，通过本地的构建和测试
 - svn commit
 - 高频度的提交，能帮助你将任务分成小的独立的子任务，以对自己做更合理的安排
 - 提交的时候写注释！
-

集成服务器为每次提交执行构建

- 每台开发者的机器都不一样，在A机器上成功的构建，B机器上可能就失败了
 - 建立一台持续集成服务器，它为每次提交进行构建。
 - 只有通过了CI服务器的构建，才是合格的提交。
 - CI服务器需要给提交者提供反馈。
 - CruiseControl很有名，但Hudson更好
 - 一个经验实践：每隔半小时检查是否有新的提交。
 - 当CI服务器的构建失败了，应该尽快修复它。
-

保持快速构建

- 持续集成的关键点之一是要快速反馈
 - 一些提高构建速度的手段
 - 高性能CI服务器
 - 移除不必要的log输出
 - 多阶段构建，如
 - 一阶段：编译，单元测试 (5 min)
 - 二阶段：集成测试，打包 (15 min)
 - 如果提交代码之后2小时才收到反馈，工作上下文已经丢失大半
 - 尽量将CI服务器的构建控制在10分钟以内
 - 实在不行，分阶段，第一阶段在10分钟以内
-

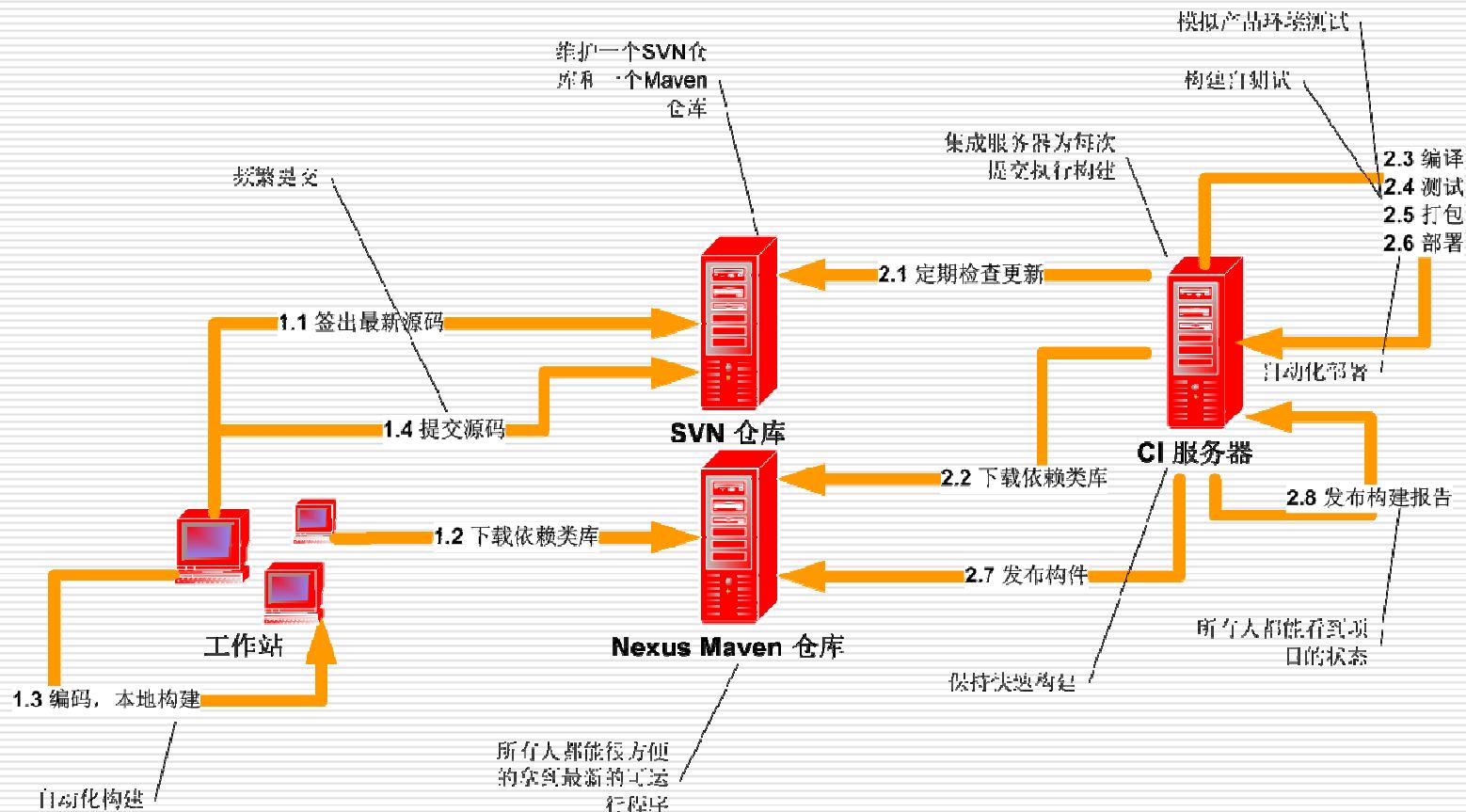
CI 实践

- 模拟产品环境测试
 - 测试环境与产品环境越相近，越多的风险就能提早发现。
 - 使用同样版本的数据库，OS，硬件，网络。总之尽量与产品环境一致。
 - 使用虚机，方便定制和克隆。
- 所有人都能很方便的拿到最新的可运行程序
 - 方便演示，测试，确认之前的想法
 - 每次成功构建之后，将构件发布到Nexus中，大家随时可以下载。
- 所有人都能看到项目的状态
 - 促进交流，大家都能看到全局
 - Hudson的页面能然给你看到项目状态
 - 构建稳定？测试都通过了？最近谁提交了代码？

自动部署

- 开发的时候我们会部署，测试的时候会部署，产品发布的时候要部署。
 - 于是重复的，清理数据库，复制粘贴文件，停止/启动服务器.....
 - 重复的劳动都应该自动化！
-

总体流程



Maven2在CI中的角色

□ 组织自动化构建

- 最简单的情况 : mvn clean

□ 帮助生成项目报告

- mvn site

□ 统一构建的用户接口

- mvn命令在所有项目中通用

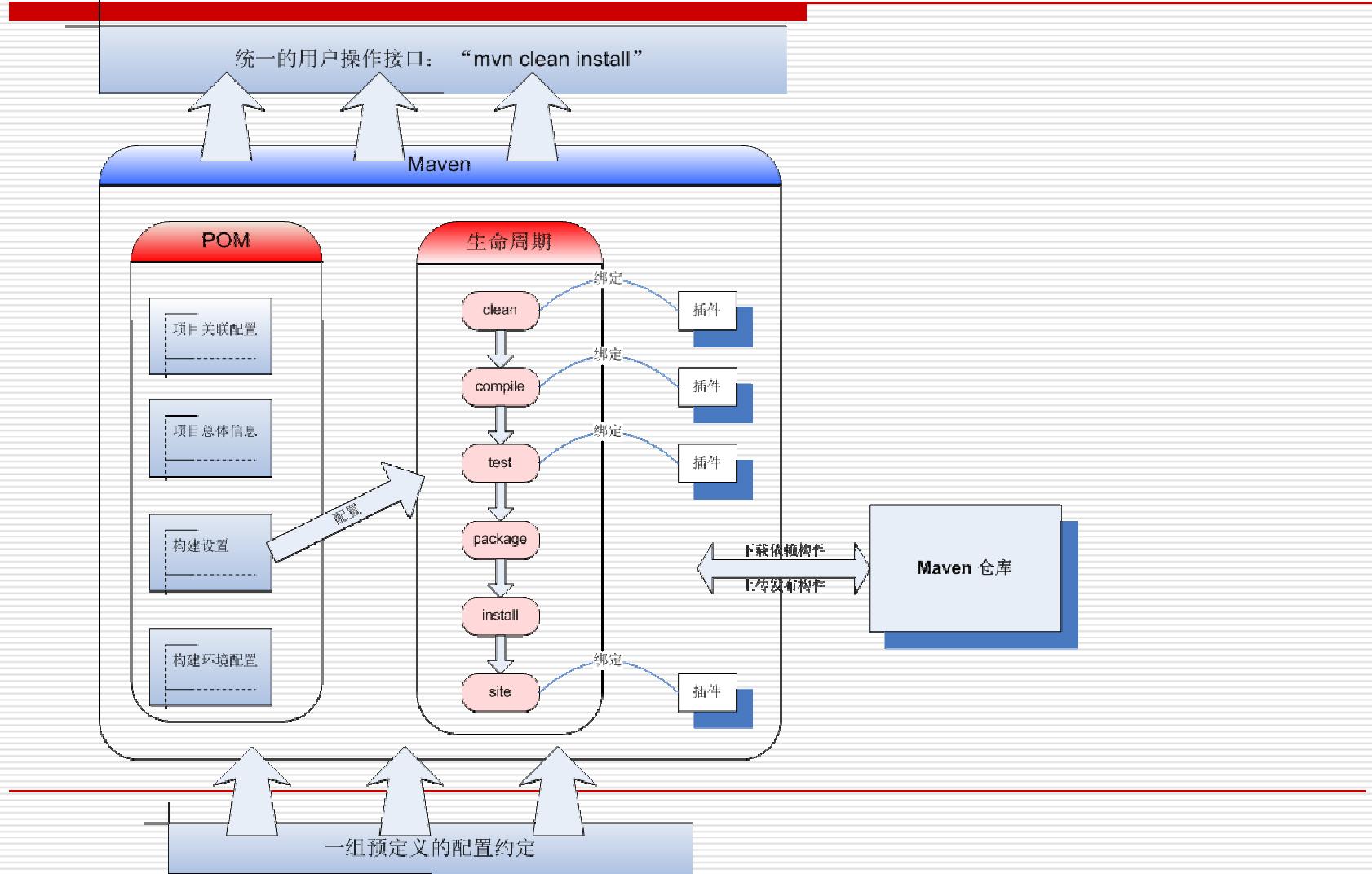
□ 最终构件的仓库管理

- 第三方依赖 , 本身的release或snapshot都存储于maven仓库中(nexus)
-

Maven2 核心介绍

- 一组通用的用户接口
 - 一组预定义的配置约定
 - 一个POM
 - 一套生命周期
 - 一堆插件
 - 一个构件仓库
-

Maven2 核心介绍



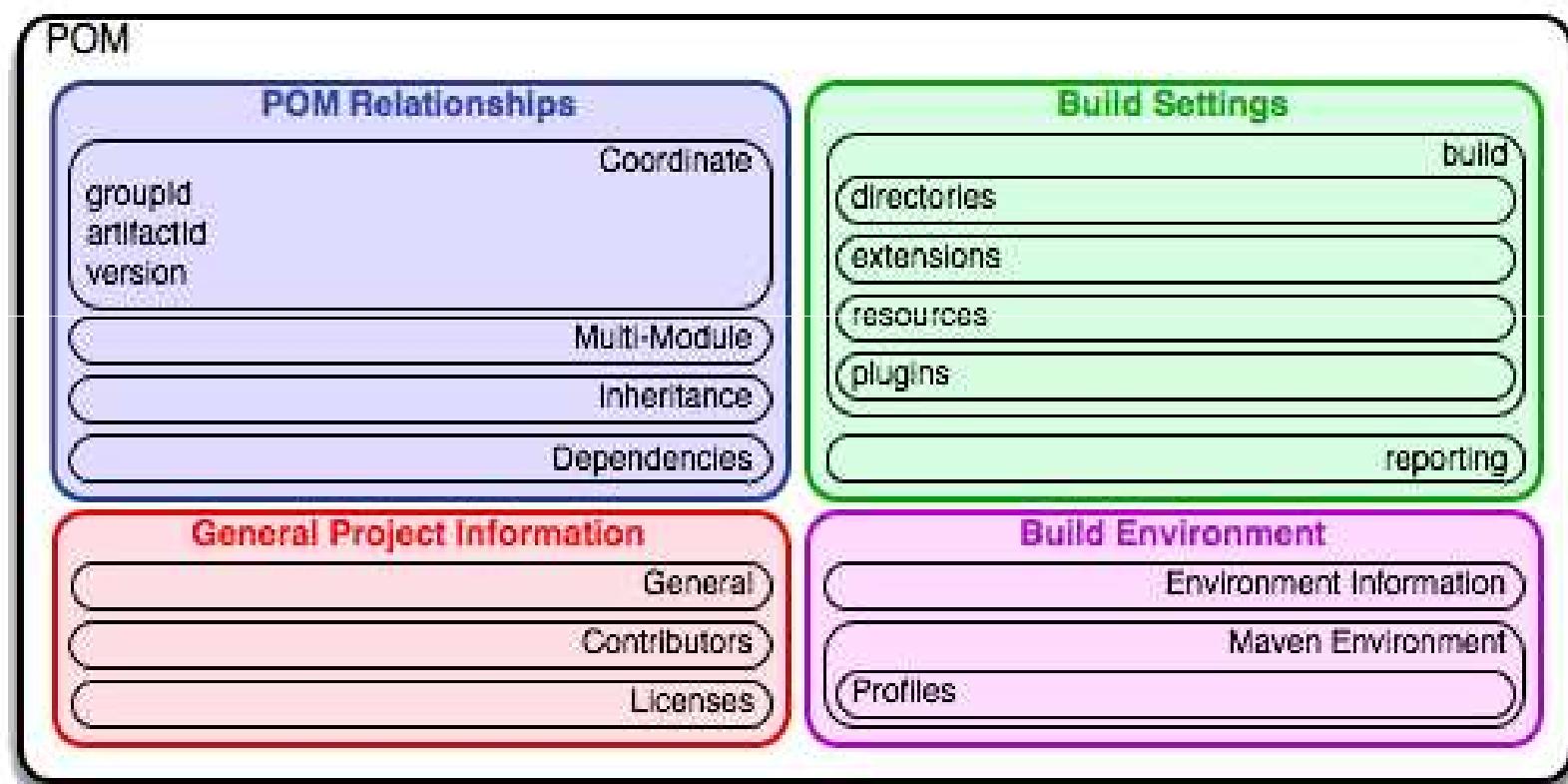
一组通用的用户接口

- Ant, make, shell可以完成Maven大部分的任务。
 - Maven的一大优点是，使用Maven的项目暴露出来的构建接口是一致的。
 - 因此，一个健康的Maven项目，签出源码，运行 mvn clean install 就构建好了。
-

一组预定义的配置约定

- Maven能自动帮你编译，测试，打包等等。
 - 前提是你遵守它的约定：
 - src/main/java
 - src/main/resources
 - src/test/java
 - src/test/resources
 - src/main/webapp
 - 输出位于target/目录
-

一个POM



一套生命周期

- 三套相互独立的生命周期
 - Clean Lifecycle 在进行真正的构建之前进行一些清理工作。
 - Default Lifecycle 构建的核心部分，编译，测试，打包，部署等等。
 - Site Lifecycle 生成项目报告，站点，发布站点。
- 每套生命周期中，执行后一个阶段会自动执行它前面的所有阶段
 - process-resources 复制并处理资源文件，至目标目录，准备打包。
 - compile 编译项目的源代码。
 - process-test-resources 复制并处理资源文件，至目标测试目录。
 - test-compile 编译测试源代码。
 - test 使用合适的单元测试框架运行测试。这些测试代码不会被打包或部署。
 - package 接受编译好的代码，打包成可发布的格式，如 JAR。
 - install 将包安装至本地仓库，以让其它项目依赖。
 - deploy 将最终的包复制到远程的仓库，以让其它开发人员与项目共享。
- 插件绑定至生命周期

一堆插件

- 生命周期什么都不做，因此Maven的安装文件很小。所有的事情都交给了插件来完成。
 - Maven的default生命周期中定义了一个compile阶段，这个定义本身什么都不会做，真正编译代码的是Compiler插件
- 主要插件列表：<http://maven.apache.org/plugins/>
- 插件可以被配置以完成一些自定义的行为
 - 配置compiler插件

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
        <source>1.5</source>
        <target>1.5</target>
    </configuration>
</plugin>
```

一个构件仓库

- 高度可配置的介于你的组织与公开Maven仓库之间的代理
 - 为你的组织提供了一个可部署你组织内部生成的构件的地方。
 - Nexus
-

CI 阶段

- 持续编译
 - 持续测试
 - 持续数据库集成
 - 持续审查
 - 持续部署
 - 持续反馈
-

持续编译

- mvn compile
 - 使用最新的源码，集成编译，能快速发现很多集成的问题。
 - 遵循Maven的约定：
 - src/main/java
 - src/main/resources
 - ...
-

持续测试

- 单元测试
 - 自动
 - 可重复
 - 不依赖环境
 - 自我验证
- 集成测试
 - 对环境有依赖
 - 执行较慢
- 测试分组 TestNG
- 分步执行
 - mvn test
 - mvn integration-test
- 坚持编写高质量的测试用例

持续数据库集成 CDBI

- 数据库脚本应该有所有开发者共同维护
 - 每次脚本修改，数据库重新创建并测试
 - 可以快速的获得一个干净的数据库
 - 使用maven-sql-plugin实现持续数据库集成(CDBI)
-

持续审查

- 帮助提高代码质量，组织编码规范，大大降低code review时间。
 - mvn site
 - 动态测试 vs 静态审查
 - 测试覆盖率，PMD，CheckStyle
 - 应该以怎样的频度执行审查？
 - 生成site很耗时
 - 审查出的问题不会造成构建失败
 - 每晚执行，生成报告
-

持续部署

- 随时随地的发布可工作的软件
 - 快速发现一些部署时才会出现的错误
 - mvn deploy不是持续部署
 - mvn deploy是将构件发布到Maven仓库中
 - 持续部署是将应用部署到运行环境中
 - Maven cargo插件可以：
 - 启动/停止J2EE容器
 - 部署应用到J2EE容器
 - <http://cargo.codehaus.org/Maven2+plugin>
-

持续反馈

□ 发送信息要考虑：

- 内容：只提供有用的信息
- 对象：只给需要的人发
- 时间：尽快发送反馈
- 方式：email, rss ...

□ CruiseControl和Hudson都提供很好的反馈配置机制。

□ 对反馈信息采取积极的态度。
