MAVEN

基于项目对象模型(POM)的项目管理和构建自动化工具

MAVEN 基础概念

- Lifecycle
- Phase
- Plugin / Goal

LIFECYCLE

- 生命周期，是对项目构建过程的抽象和统一
- Maven 拥有三套相对独立的生命周期

Lifecycle	含义
clean	清理项目
default	构建项目
site	发布项目站点

PHASE

每个生命周期都由多个"阶段"（Phase）组成

- clean 生命周期
  - pre-clean
  - clean
  - post-clean

PHASE

- default 生命周期
  - validate
  - initialize
  - ...
  - compile
  - ...
  - package
  - install
  - deploy

PHASE

- site 生命周期
  - pre-site
  - site
  - post-site
  - site-deploy

PHASE

每个“阶段"与它生命周期前后的"阶段"有依赖关系，执行一个"阶段"会自动执行在它之前的所有"阶段"

1. 执行 "mvn clean"，实际执行的阶段为: pre-clean -> clean
2. 执行 "mvn compile"，实际执行的阶段为: validate -> initialize -> generate-sources -> ... -> compile
3. 执行 "mvn clean install"，实际执行的阶段为: pre-clean -> clean -> validate -> initialize -> ... -> install

PLUGIN / GOAL

- "生命周期"和"阶段"都是 Maven 对构建过程的抽象，实际任务都是交由"插件"（Plugin）完成的
- 一个"插件"往往提供了多个功能，每一个功能就叫做"目标"（Goal）。正如"插件"的概念，使得每个"生命周期中阶段"的行为都可以很容易的修改和替换
- 我们可以在 pom 文件中引入第三方插件来代理一个具体的"阶段"

PLUGIN / GOAL

即使不引入任何第三方插件，Maven 也内置了一套"阶段"对应"插件"的关系

Lifecycle	Phase	>	Plugin	Goal
clean	clean		maven-clean-plugin	clean
default	compile		maven-compiler-plugin	compile
default	package		maven-jar-plugin	jar
default	deploy		maven-deploy-plugin	deploy
site	site		maven-site-plugin	site

MAVEN 命令行

Maven Usage

```
usage: mvn [options] [goal(s)] [phase(s)]
```

调用插件的一个具体目标

```
mvn maven-install-plugin:install
```

思考

在不引入第三方插件的情况下，为何这两条命令是等价的?

```
# phase
mvn clean

# plugin:goal
mvn maven-clean-plugin:clean
```

依赖管理

项目依赖的 jar 在 maven 中通过配置的方式自动引入，只需将依赖声明在 pom 文件中即可

```
<dependency>
  <groupid>com.zaxxer</groupid>
  <artifactid>HikariCP</artifactid>
  <version>2.3.2</version>
</dependency>
```

依赖传递

什么是依赖传递?

- 假设项目依赖 "log4j 2.0"，又依赖 "libA"，而 "libA" 依赖 "log4j 1.2"
- 这样 "libA" 依赖的 "log4j 1.2" 也会被传递给项目
- 最终导致项目同时依赖 "log4j 2.0" 和 "log4j 1.2"

如何限制依赖传递

Dependency Scope

```
<dependency>
  <groupid>junit</groupid>
  <artifactid>junit</artifactid>
  <version>4.6</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupid>javax.servlet</groupid>
  <artifactid>servlet-api</artifactid>
  <version>2.5</version>
  <scope>provided</scope>
</dependency>
```

如何限制依赖传递

Dependency Scope

Scope	传递	含义
compile	是	默认值，依赖在所有过程中有效
provided	X	此依赖已由容器提供（例如 servlet-api, jsp-api），仅在编译、测试阶段使用
runtime	是	此依赖仅在运行时使用，编译期不需要
test	X	此依赖仅在测试编译、测试运行阶段使用
system	X	类似 provided，但期望你指出依赖的明确位置，而不会去仓库查询

如何限制依赖传递

Exclusions

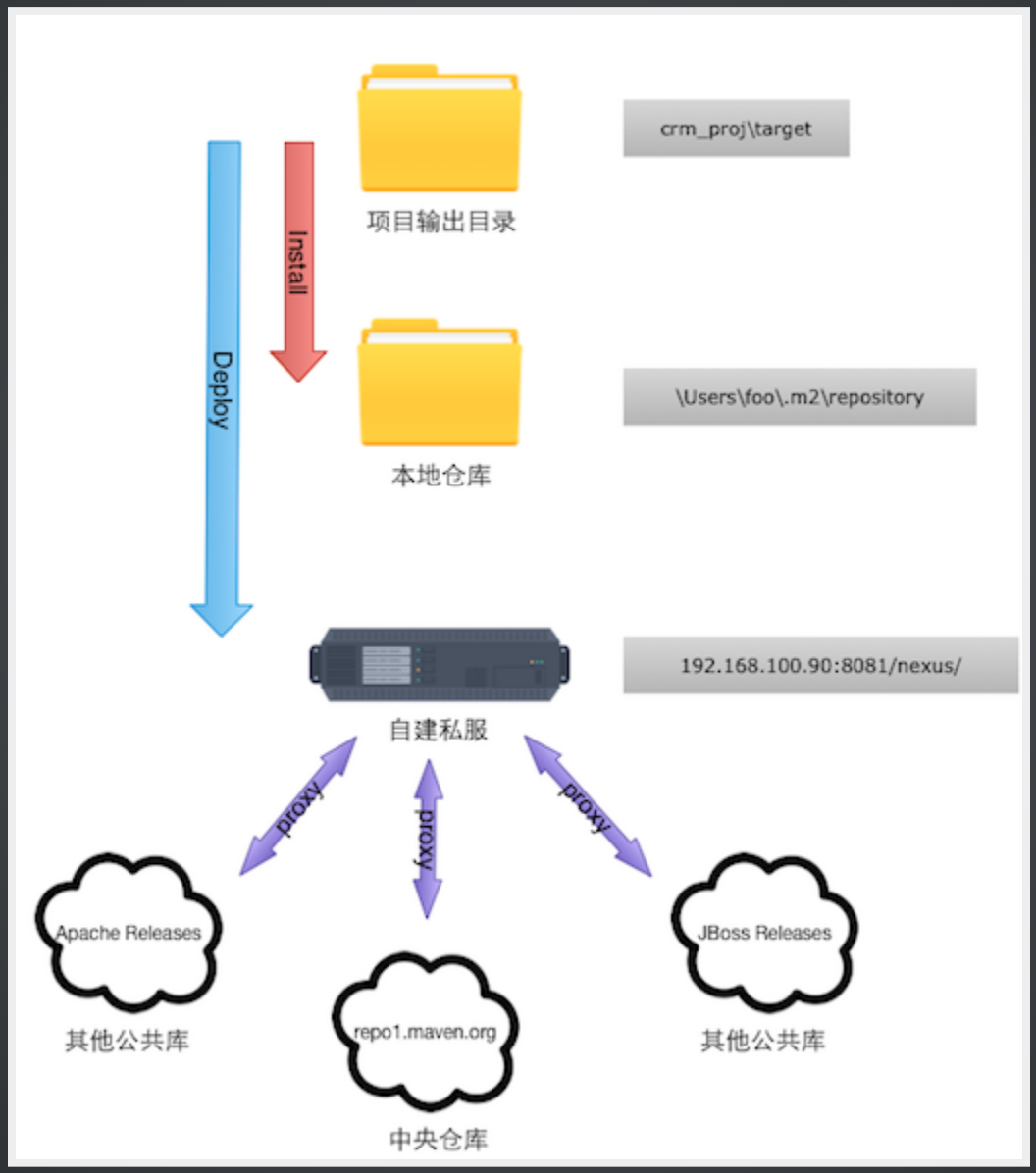
```
<dependency>
  <groupid>org.apache.poi</groupid>
  <artifactid>poi</artifactid>
  <version>3.1-FINAL</version>
  <exclusions>
    <exclusion>
      <groupid>log4j</groupid>
      <artifactid>log4j</artifactid>
    </exclusion>
  </exclusions>
</dependency>
```

MAVEN 仓库

Maven 仓库分两大类，其中"远程仓库"又分三种：

- 本地仓库
- 远程仓库
  - 自建仓库 (私服)
  - 中央仓库
  - 其他公共库

MAVEN 仓库


crm_proj\target
项目输出目录
Install
Deploy
\Users\foo\.m2\repository
本地仓库
192.168.100.90:8081/nexus/
自建私服
proxy
proxy
proxy
Apache Releases
其他公共库
repo1.maven.org
中央仓库
JBoss Releases
其他公共库

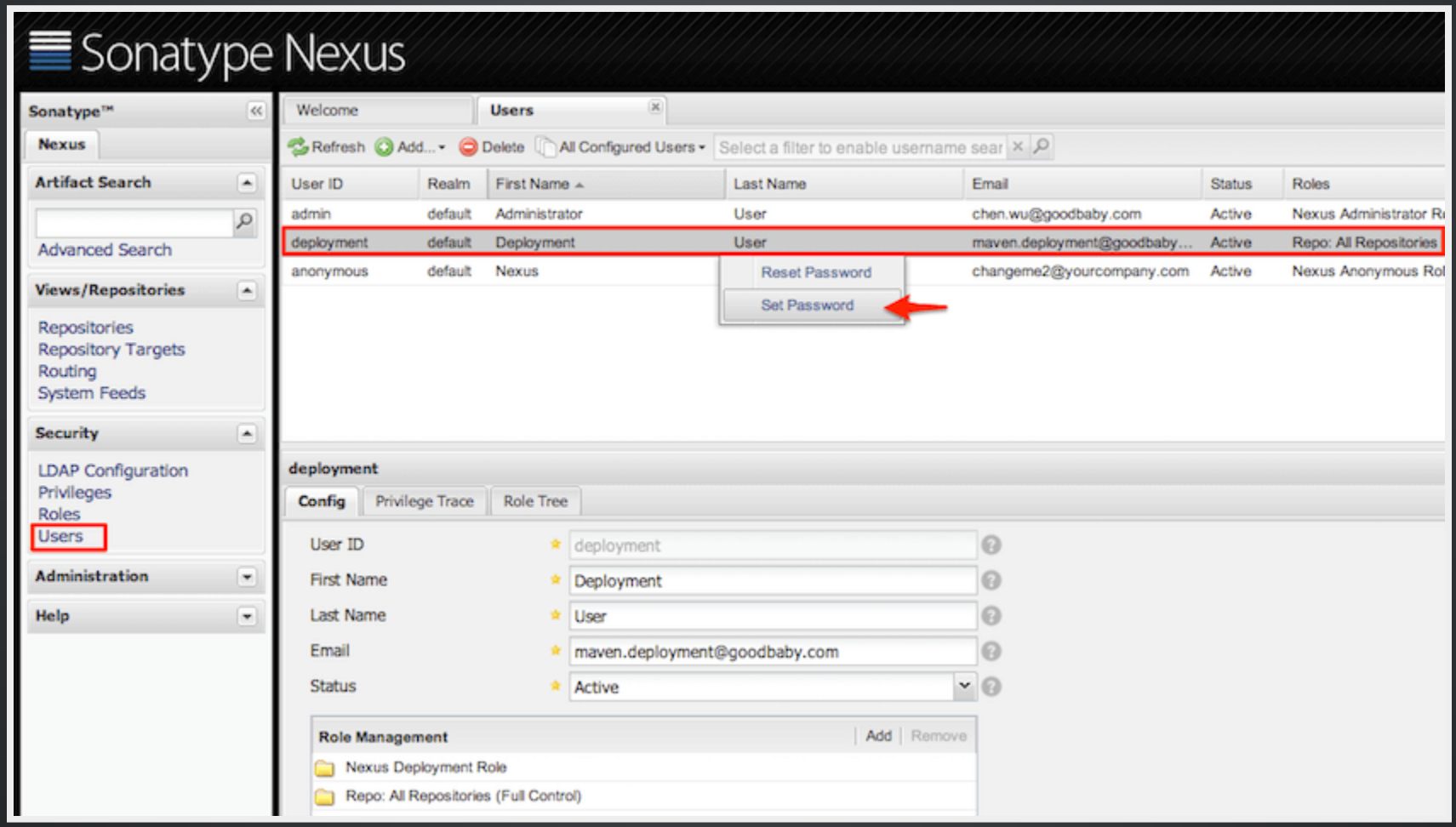
自建仓库(私服)

1. 在服务器安装 Sonatype Nexus，并启动
2. 在开发机修改 Maven 的 settings.xml 以添加 nexus 的地址

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http
  ...
  <mirrors>
    <mirror>
      <id>local.mirror.repository</id>
      <name>Local Nexus Repository</name>
      <url>http://192.168.100.90:8081/nexus/content/groups/public/</url>
      <mirrorof>*</mirrorof>
    </mirror>
  </mirrors>
  ...
</settings>
```

如何将我开发的 LIB 部署到自建仓库

- 首先要用管理员登录 Nexus，设置 deployment 用户的密码
- 否则 deploy 将提示错误: 401, ReasonPhrase:Unauthorized



如何将我开发的 LIB 部署到自建仓库

修改项目的 pom.xml，增加 distributionManagement 部分

```
<project>
  ...
  <distributionmanagement>
    <repository>
      <id>releases</id>
      <name>Internal Releases</name>
      <url>http://192.168.100.90:8081/nexus/content/repositories/releases
    </repository>
    <snapshotrepository>
      <id>snapshots</id>
      <name>Internal Releases</name>
      <url>http://192.168.100.90:8081/nexus/content/repositories/snapsho
    </snapshotrepository>
  </distributionmanagement>
  ...
</project>
```

如何将我开发的 LIB 部署到自建仓库

修改 Maven 配置文件 settings.xml，增加 servers 部分

```
<!--?xml version="1.0" encoding="UTF-8"?-->
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http
  ...
  <servers>
    <server>
      <id>releases</id>
      <username>deployment</username>
      <password>fake_passwd</password><!-- 前面设置的 deployment 用户的密码
    </server>
    <server>
      <id>snapshots</id>
      <username>deployment</username>
      <password>fake_passwd</password><!-- 前面设置的 deployment 用户的密码
    </server>
  </servers>
  ...
</settings>
```

如何将我开发的 LIB 部署到自建仓库

至此，执行 deploy 命令就将会把当前项目按照 pom 中的版本部署到自建仓库

```
mvn clean deploy
```

MAVEN 插件

IDE	可用插件
Intellij IDEA	已内置
Eclipse	m2e (m2eclipse)
MyEclipse	已内置

小技巧: 如何知道一个 CLASS 来自哪个 LIB

当碰到类找不到的错误，例如:

```
java.lang.NoClassDefFoundError: com.foo.FooServiceImpl
```

可通过 GrepCode 搜索 FooServiceImpl 来自哪个 Lib

小技巧: 如何在中央仓库查找 LIB

- http://mvnrepository.com/
- http://search.maven.org/#browse

THE END

讨论环节