



每个人都有自己的特长，选对了团队才会发挥出自己的超长价值 一起在编程的世界畅游吧

目录视图

摘要视图

RSS 订阅

【活动】Python创意编程活动开始啦!!! CSDN日报20170502 ——《程序学徒与导师》 深入浅出，带你学习 Unity

eclipse插件——maven

标签：eclipse eclipse插件 maven配置及搭建

2016-09-05 14:19

24866人阅读

评论

2

分类： java (43) maven (1)

版权声明：本文为博主原创文章，未经博主允许不得转载。欢迎点击屏幕左侧的QQ联系我，或者私信我

目录(?)

[+]

项目开发中遇到的问题

- 都是同样的代码，为什么在我的机器上可以编译执行，而在他的机器上就不行？
- 为什么在我的机器上可以正常打包，而配置管理员却打不出来？
- 项目组加入了新的人员，我要给他说明编译环境如何设置，但是让我挠头的是，有些细节我也记不清楚了
- 我的项目依赖一些jar包，我应该把他们放哪里？放源码库里？
- 这是我开发的第二个项目，还是需要上面的那些jar包，再把它们复制到我当前项目的svn库里吧
- 现在是第三次，再复制一次吧 —— 这样真的好吗？
- 我写了一个**数据库**相关的通用类，并且推荐给了其他项目组，现在已经有五个项目组在使用它了，今天我发现了一个bug，并修正了它，我会把jar包通过邮件发给其他项目组,这不是一个好的分发机制，太多的环节可能导致出现bug
- 项目进入**测试**阶段，每天都要向测试服务器部署一版。每次都手动部署，太麻烦了

什么是maven

Maven的概念

- maven翻译为“专家”，“内行”。Maven是基于项目对象模型(POM)，可以通过一小段描述信息来管理项目的构建，报告和文档的软件项目管理工具。
- Maven是跨平台的项目管理工具。主要服务于基于Java平台的项目构建，依赖管理和项目信息管理。
- Maven主要有两个功能：
 - 项目构建
 - 依赖管理

什么是构建

QQ加我技术群

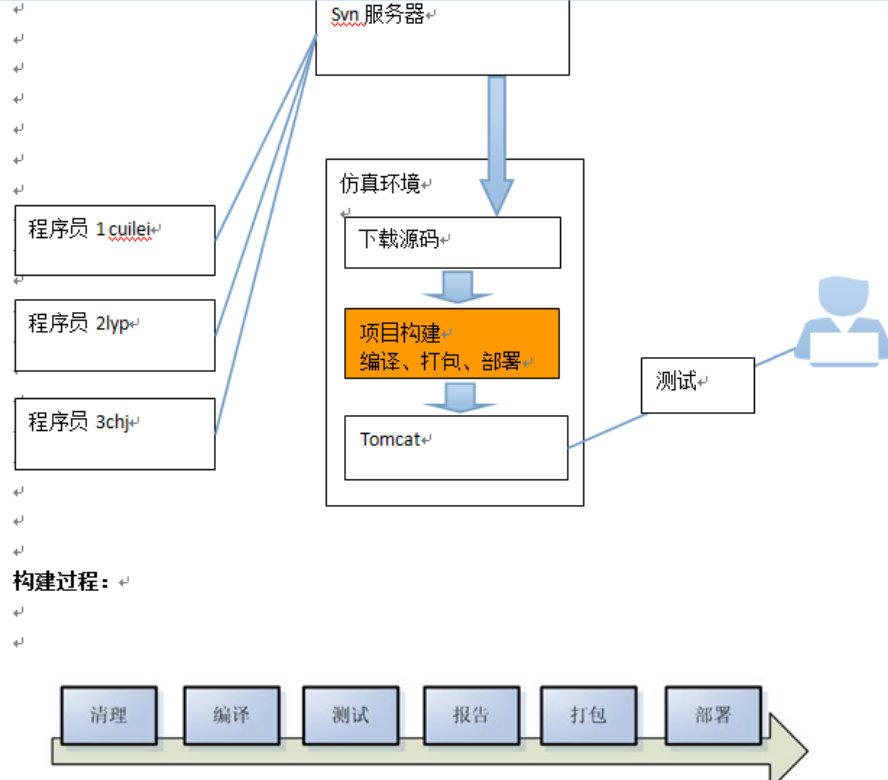
篇
5条

加群

么聊

内喜怒

篇
6126



为什么使用maven?

- **Eclipse**

手工操作较多，编译、测试、部署等工作都是独立的，很难一步完成
每个人的IDE配置都不同，很容易出现本地代码换个地方编译就出错

- **Ant**

没有一个约定的目录结构

必须明确让ant做什么，什么时候做，然后编译，打包

没有生命周期，必须定义目标及其实现的任务序列

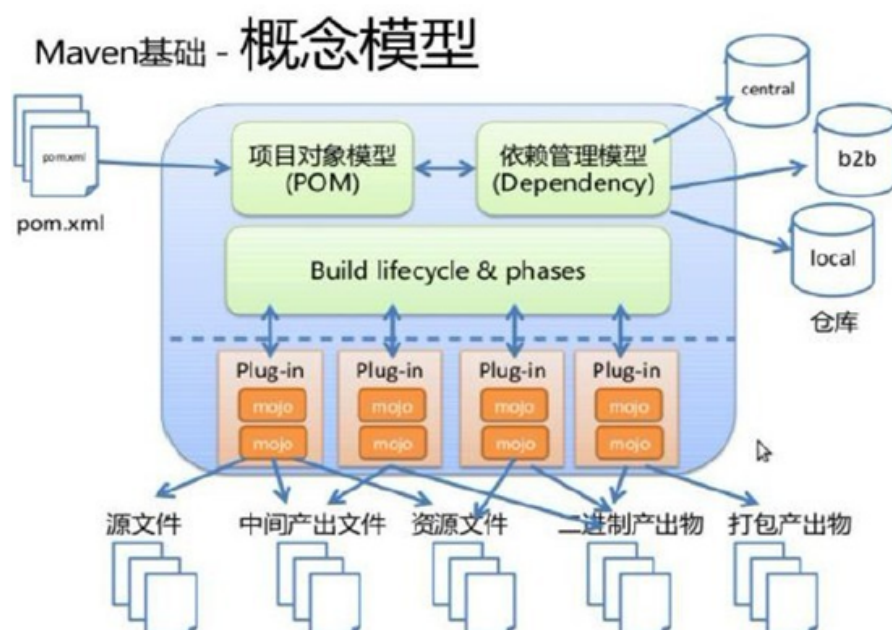
没有集成依赖管理

- **Maven**

拥有约定，知道你的代码在哪里，放到哪里去

拥有一个生命周期，例如执行 mvn install 就可以自动执行编译，测试，打包等构建过程只需要定义一个 pom.xml,然后把源码放到默认的目录，Maven帮你处理其他事情

Maven模型



展开

37021)

33796)

29913)

26928)

24863)



app开发报价单



(20)

(11)

(10)

(8)

(5)

(4)

(4)

(4)

(3)

(3)

一《程


评论

境搭

构浅

的中文

QQ加我技术群



数据处
后台的
Juid":

rows

rows

类是
要条件
以没

类是
要条件
可以不
生...

望楼

心声

1不等

=1不

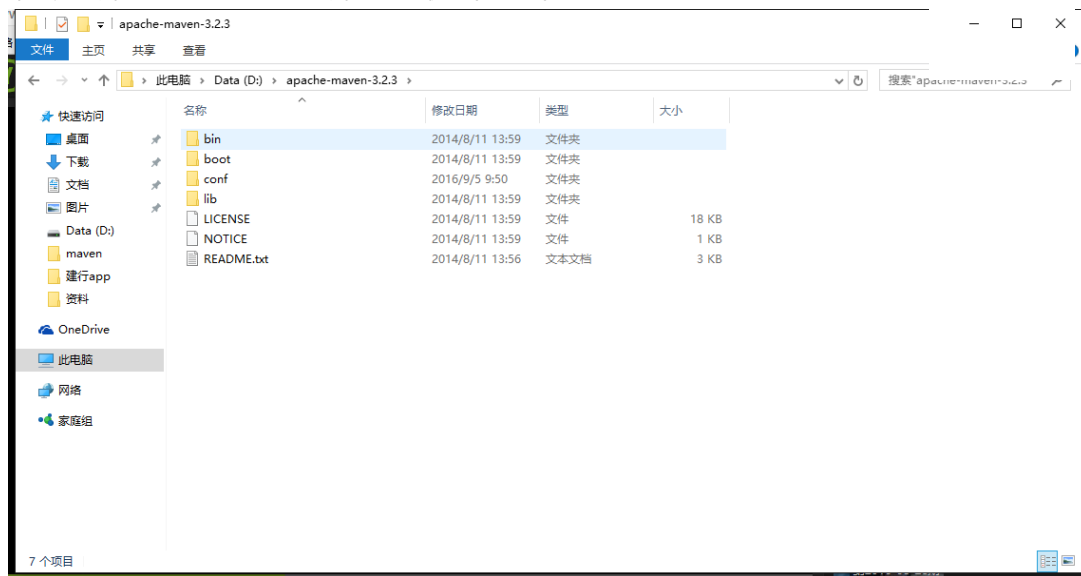
Maven的安装及配置

下载maven

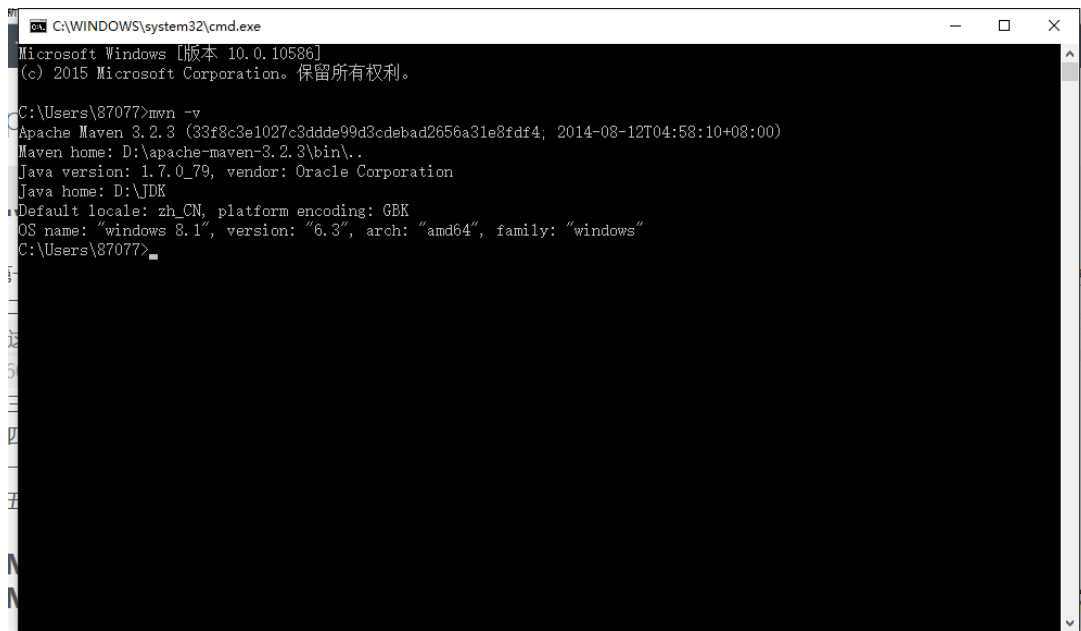
- 官方网站：<http://maven.apache.org>
- 本课程使用的maven的版本为3.0.5

Maven的安装

- 第一步：安装jdk，要求1.6或以上版本
- 第二步：解压maven的压缩包。解压目录最好不要有中文。



- 第三步：配置系统环境变量，MAVEN_HOME
- 第四步：设置环境变量Path，将%Maven_HOME%\bin加入Path中，一定要注意要用分号；与其他值隔开
- 第五步：5.验证安装是否成功，打开cmd窗口，敲入mvn -v 查看



Maven安装目录分析

- bin：含有mvn运行的脚本
- boot：含有plexus-classworlds类加载器框架。plexus-classworlds是一个类加载器框架，相对于默认
的java类加载器，它提供了更丰富的语法以方便配置，Maven使用该框架加载自己的类库。对于一般的Maven
用户来说，不必关心该文件。
- conf：含有settings.xml配置文件，此配置为maven的全局配置。
- lib：含有Maven运行时所需要的java类库
LICENSE.txt, NOTICE.txt, README.txt针对Maven版本，第三方软件等简要介绍



Maven的配置

• Maven的默认配置

- 在MAVEN_HOME下的配置文件：MAVEN_HOME/conf/settings.xml 是maven全局的配置。如果不对其修改的话，默认本地仓库的路径就是：当前用户路径C:\Users[UserName].m2

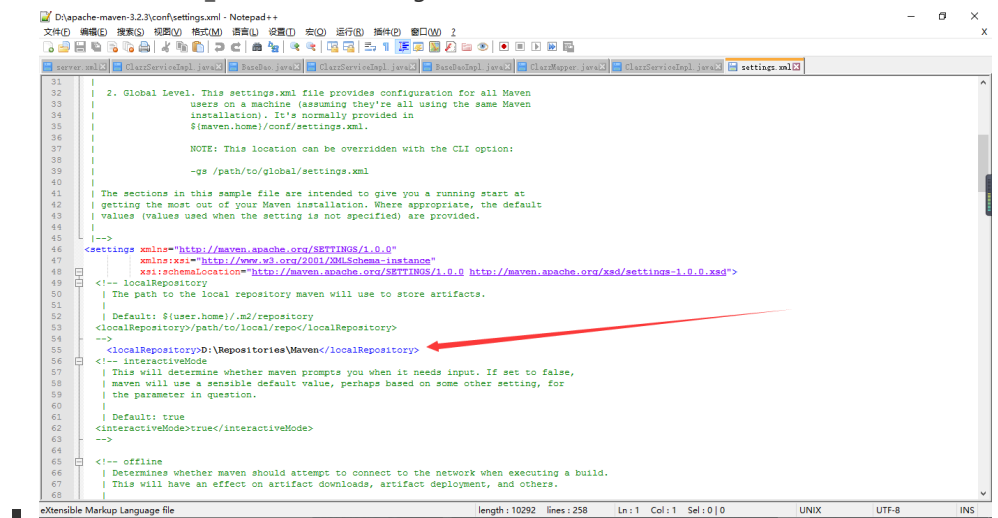


- localRepository：用户仓库，用于检索依赖包路径

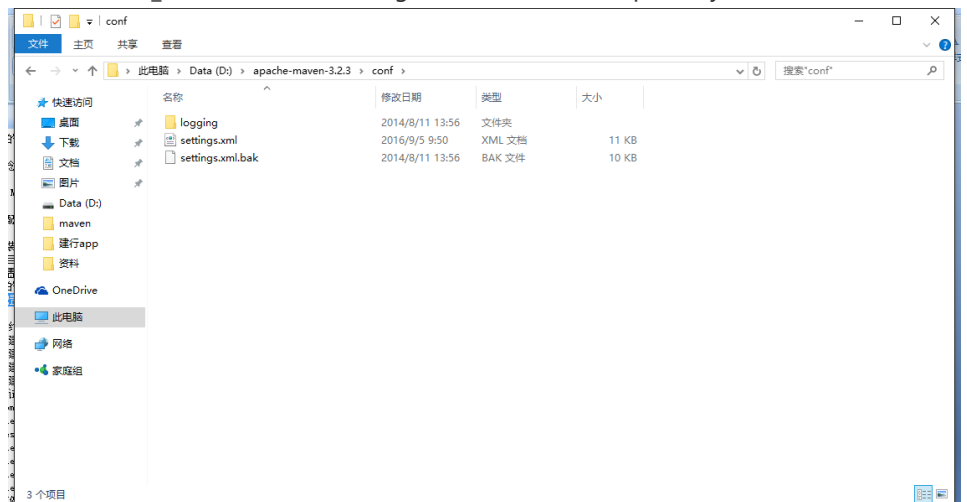
• 用户配置

◦ 配置方法：

- 1、在你想保存仓库的位置创建一个目录作为仓库目录。例如在d:盘创建一个MavenRepository目录
- 2、在其中创建一个名为“repository”的目录。
- 3、修改MAVEN_HOME\conf\settings.xml



- 4、从MAVEN_HOME\conf中把settings.xml复制到MavenRepository目录。得到如下结构：



■ ！注意：

用户级别的仓库在全球配置中一旦设置，全局配置将不再生效，转用用户所设置的仓库，否则使

QQ加我技术群





创建Maven工程

• 4.1 遵从Maven约定

src/main/**Java** —— 存放项目的.java文件

src/main/resources —— 存放项目资源文件，如**spring, hibernate**配置文件

src/test/java —— 存放所有测试.java文件，如JUnit测试类

src/test/resources —— 测试资源文件

target —— 项目输出位置

pom.xml——maven项目核心配置文件

Project

| -src

| | -main

| | | -java —— 存放项目的.java文件

| | | -resources —— 存放项目资源文件，如spring, hibernate配置文件

| | -test

| | -java —— 存放所有测试.java文件，如JUnit测试类

| | -resources —— 测试资源文件

| -target —— 目标文件输出位置例如.class、.jar、.war文件

| -pom.xml —— maven项目核心配置文件

• 4.2 第一步：创建maven工程

首先建立MavenHelloWorld项目同时建立Maven约定的目录结构

MavenHelloWorld

| -src

| | -main

| | | -java —— 存放项目的.java文件

| | | -resources —— （暂时省略）

| | -test

| | -java —— 存放所有测试.java文件，如JUnit测试类

| | -resources —— （暂时省略）

| -target —— （不手工创建）

| -pom.xml —— maven项目核心配置文件

• 4.3 第二步：创建HelloWorld.java

- 在src/main/java/cn/change/maven目录下新建文件Hello.java

```
1 package cn.change.maven;
2
3 public class HelloWorld {
4     public String sayHello(String name) {
5         return "Hello World : " + name + "!";
6     }
7 }
```

• 4.4 第三步：新建测试文件HelloTest.java

- 在/src/test/java/cn/change/maven目录下新建测试文件HelloTest.java

```
1 package cn.change.maven;
2
3 import org.junit.Test;
```

QQ加我技术群





QQ加我技术群



```
4 import static junit.framework.Assert.*;
5
6 public class HelloWorldTest{
7
8     @Test
9     public void testHello(){
10
11         HelloWorld hello = new HelloWorld();
12         String results = hello.sayHello("maven");
13         assertEquals("Hello World :maven!",results);
14
15     }
16
17 }
```

- 4.5 第四步：创建pom.xml

- 在项目MavenHelloWorld根目录建立pom.xml

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/20
2     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/m
3     <modelVersion>4.0.0</modelVersion>
4     <groupId>com.bshinfo.el</groupId>
5     <artifactId>ccb_el_manager</artifactId>
6     <version>0.0.1-SNAPSHOT</version>
7     <name>ccb_el_manager Maven Webapp</name>
8     <url>http://maven.apache.org</url>
9     <packaging>war</packaging>
10    <build>
11        <finalName>ccb_el_manager</finalName>
12    </build>
13
14    <dependencies>
15        <dependency>
16            <groupId>junit</groupId>
17            <artifactId>junit</artifactId>
18            <version>4.11</version>
19            <scope>test</scope>
20        </dependency>
21
22        <!-- spring jar -->
23        <dependency>
24            <groupId>org.springframework</groupId>
25            <artifactId>spring-core</artifactId>
26            <version>4.2.5.RELEASE</version>
27        </dependency>
28        <dependency>
29            <groupId>org.springframework</groupId>
30            <artifactId>spring-context</artifactId>
31            <version>4.2.5.RELEASE</version>
32        </dependency>
33        <dependency>
34            <groupId>org.springframework</groupId>
35            <artifactId>spring-context-support</artifactId>
36            <version>4.2.5.RELEASE</version>
37        </dependency>
38        <dependency>
39            <groupId>org.springframework</groupId>
40            <artifactId>spring-tx</artifactId>
41            <version>4.2.5.RELEASE</version>
42        </dependency>
43        <dependency>
44            <groupId>org.springframework</groupId>
45            <artifactId>spring-aop</artifactId>
46            <version>4.2.5.RELEASE</version>
47        </dependency>
48        <dependency>
49            <groupId>org.springframework</groupId>
```




QQ加我技术群



```
50         <artifactId>spring-beans</artifactId>
51         <version>4.2.5.RELEASE</version>
52     </dependency>
53     <dependency>
54         <groupId>org.springframework</groupId>
55         <artifactId>spring-jdbc</artifactId>
56         <version>4.2.5.RELEASE</version>
57     </dependency>
58     <dependency>
59         <groupId>org.springframework.security</groupId>
60         <artifactId>spring-security-core</artifactId>
61         <version>4.0.4.RELEASE</version>
62     </dependency>
63     <dependency>
64         <groupId>org.springframework.security</groupId>
65         <artifactId>spring-security-web</artifactId>
66         <version>4.0.4.RELEASE</version>
67     </dependency>
68     <dependency>
69         <groupId>org.springframework.security</groupId>
70         <artifactId>spring-security-config</artifactId>
71         <version>4.0.4.RELEASE</version>
72     </dependency>
73     <dependency>
74         <groupId>org.springframework.security</groupId>
75         <artifactId>spring-security-cas</artifactId>
76         <version>4.0.4.RELEASE</version>
77     </dependency>
78     <dependency>
79         <groupId>org.springframework</groupId>
80         <artifactId>spring-jms</artifactId>
81         <version>4.2.5.RELEASE</version>
82     </dependency>
83
84     <!-- spring mvc jar -->
85     <dependency>
86         <groupId>org.springframework</groupId>
87         <artifactId>spring-web</artifactId>
88         <version>4.2.5.RELEASE</version>
89     </dependency>
90     <dependency>
91         <groupId>org.springframework</groupId>
92         <artifactId>spring-webmvc</artifactId>
93         <version>4.2.5.RELEASE</version>
94     </dependency>
95
96     <!-- mybatis jar -->
97     <dependency>
98         <groupId>org.mybatis</groupId>
99         <artifactId>mybatis</artifactId>
100         <version>3.3.1</version>
101     </dependency>
102
103     <!-- mybatis-paginator jar -->
104     <dependency>
105         <groupId>com.github.miemiedev</groupId>
106         <artifactId>mybatis-paginator</artifactId>
107         <version>1.2.10</version>
108     </dependency>
109
110     <!-- mybatis-spring jar -->
111     <dependency>
112         <groupId>org.mybatis</groupId>
113         <artifactId>mybatis-spring</artifactId>
114         <version>1.2.4</version>
115     </dependency>
116
117     <!-- mysql-connector-java jar -->
118     <dependency>
```



QQ加我技术群



```
119         <groupId>mysql</groupId>
120         <artifactId>mysql-connector-java</artifactId>
121         <version>5.1.38</version>
122     </dependency>
123     <!-- 数据库连接池 jar -->
124     <dependency>
125         <groupId>commons-dbcp</groupId>
126         <artifactId>commons-dbcp</artifactId>
127         <version>1.4</version>
128     </dependency>
129     <dependency>
130         <groupId>commons-pool</groupId>
131         <artifactId>commons-pool</artifactId>
132         <version>1.6</version>
133     </dependency>
134
135     <!-- log jar -->
136     <dependency>
137         <groupId>log4j</groupId>
138         <artifactId>log4j</artifactId>
139         <version>1.2.17</version>
140     </dependency>
141     <dependency>
142         <groupId>org.slf4j</groupId>
143         <artifactId>slf4j-api</artifactId>
144         <version>1.7.19</version>
145     </dependency>
146     <dependency>
147         <groupId>org.slf4j</groupId>
148         <artifactId>slf4j-log4j12</artifactId>
149         <version>1.7.19</version>
150     </dependency>
151
152     <!-- jsp Template jar -->
153     <dependency>
154         <groupId>org.freemarker</groupId>
155         <artifactId>freemarker</artifactId>
156         <version>2.3.23</version>
157     </dependency>
158
159     <!-- apache servlet api jar -->
160     <dependency>
161         <groupId>org.apache.tomcat</groupId>
162         <artifactId>tomcat-servlet-api</artifactId>
163         <version>8.0.32</version>
164     </dependency>
165
166     <dependency>
167         <groupId>org.aspectj</groupId>
168         <artifactId>aspectjweaver</artifactId>
169         <version>1.8.9</version>
170     </dependency>
171
172     <dependency>
173         <groupId>org.aspectj</groupId>
174         <artifactId>aspectjrt</artifactId>
175         <version>1.8.9</version>
176     </dependency>
177
178     <dependency>
179         <groupId>org.aspectj</groupId>
180         <artifactId>aspectjtools</artifactId>
181         <version>1.8.9</version>
182     </dependency>
183
184     <!-- activemq -->
185     <dependency>
186         <groupId>org.apache.activemq</groupId>
187         <artifactId>activemq-core</artifactId>
```




QQ加我技术群



```
188         <version>5.7.0</version>
189     </dependency>
190     <dependency>
191         <groupId>org.apache.activemq</groupId>
192         <artifactId>activemq-pool</artifactId>
193         <version>5.13.2</version>
194     </dependency>
195
196     <!-- cas security -->
197     <dependency>
198         <groupId>org.jasig.cas.client</groupId>
199         <artifactId>cas-client-core</artifactId>
200         <version>3.4.1</version>
201     </dependency>
202     <!-- end of cas security -->
203     <dependency>
204         <groupId>net.sf.json-lib</groupId>
205         <artifactId>json-lib</artifactId>
206         <version>2.4</version>
207         <classifier>jdk15</classifier>
208     </dependency>
209     <dependency>
210         <groupId>commons-codec</groupId>
211         <artifactId>commons-codec</artifactId>
212         <version>1.10</version>
213     </dependency>
214     <dependency>
215         <groupId>org.codehaus.jackson</groupId>
216         <artifactId>jackson-mapper-asl</artifactId>
217         <version>1.5.0</version>
218     </dependency>
219     <dependency>
220         <groupId>commons-fileupload</groupId>
221         <artifactId>commons-fileupload</artifactId>
222         <version>1.2.2</version>
223     </dependency>
224     <dependency>
225         <groupId>org.apache.commons</groupId>
226         <artifactId>commons-io</artifactId>
227         <version>1.3.2</version>
228     </dependency>
229 </dependencies>
230 </project>
```

• 4.6 第五步：测试maven命令

- 打开cmd命令行，进入Hello项目根目录(包含有pom.xml文件的目录)

◦ 4.6.1 mvn compile命令

- 执行完毕后，生成了maven工程编译完成后字节码文件的生成目录target
- 执行 mvn compile命令
- 执行完毕后，生成了maven工程编译完成后字节码文件的生成目录target

◦ 4.6.2 mvn clean命令

- cmd 中继续录入mvn clean命令
- 执行完毕后，字节码生成目录被删除

◦ 4.6.3 mvn test命令

- cmd 中录入 mvn test命令
- 执行完毕后，字节码生成目录中生成了被测试类与测试类的编译字节码和测试执行过程日志与详细报告

◦ 4.6.4 mvn clean compile命令

- cmd 中录入 mvn clean compile命令



QQ加我技术群



- 组合指令，先执行clean，再执行compile，通常应用于上线前执行，清除测试类

◦ 4.6.5 mvn clean test命令

- cmd 中录入 mvn clean test命令
- 组合指令，先执行clean，再执行test，通常应用于测试环节

◦ 4.6.6 mvn clean package命令

- cmd 中录入 mvn clean package命令
- 组合指令，先执行clean，再执行package，将项目打包，通常应用于发布前
- 执行过程：
 - 清理———清空环境
 - 编译———编译源码
 - 测试———测试源码
 - 打包———将编译的非测试类打包

◦ 4.6.7 mvn clean install命令

- cmd 中录入 mvn clean install 查看仓库，当前项目被发布到仓库中
- 组合指令，先执行clean，再执行install，将项目打包，通常应用于发布前
- 执行过程：
 - 清理———清空环境
 - 编译———编译源码
 - 测试———测试源码
 - 打包———将编译的非测试类打包
 - 部署———将打好的包发布到资源仓库中
 - mvn archetype:create：创建 Maven 项目
 - mvn compile：编译源代码
 - mvn test-compile：编译测试代码
 - mvn test：运行应用程序中的单元测试
 - mvn site：生成项目相关信息的网站
 - mvn clean：清除目标目录中的生成结果
 - mvn package：依据项目生成 jar 文件
 - mvn install：在本地 Repository 中安装 jar
 - mvn eclipse:eclipse：生成 Eclipse 项目文件
 - mvn -Dmaven.test.skip=true：忽略测试文档编译

◦ 4.6.8 错误范例

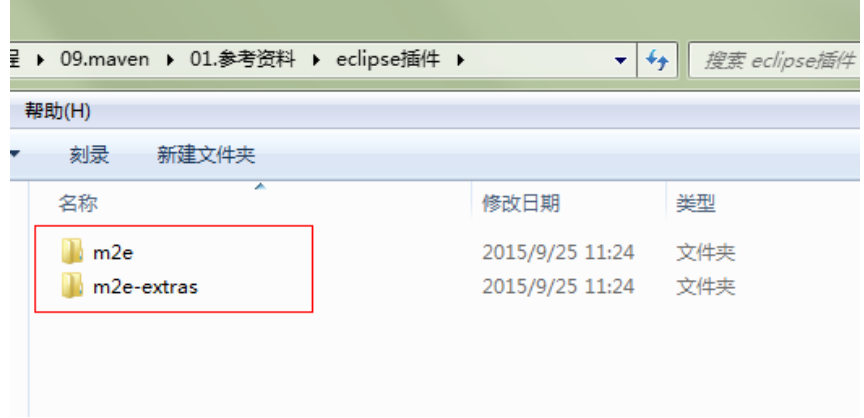
说明：MAVEN 命令输入错误，命令名称显示在[ERROR] Unknown lifecycle phase “compoile” 引号中的内容为输入错误的名称

[ERROR] Unknown lifecycle phase “compoile” . You must specify a valid lifecycle phase or a goal in the format : or :

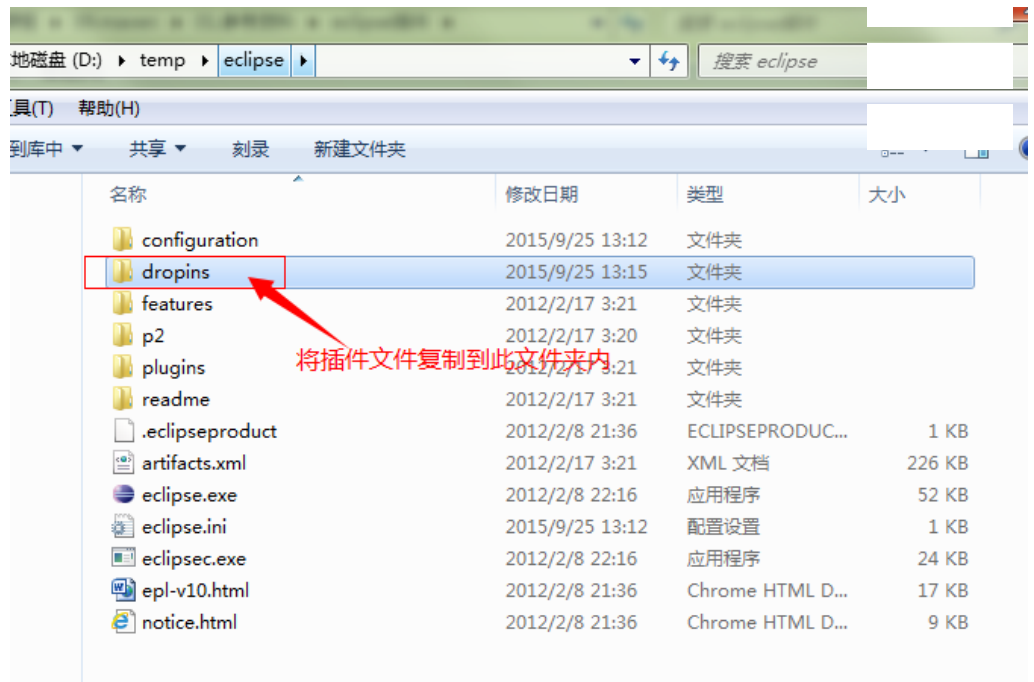
M2Eclipse

• 5.1 安装m2eclipse插件

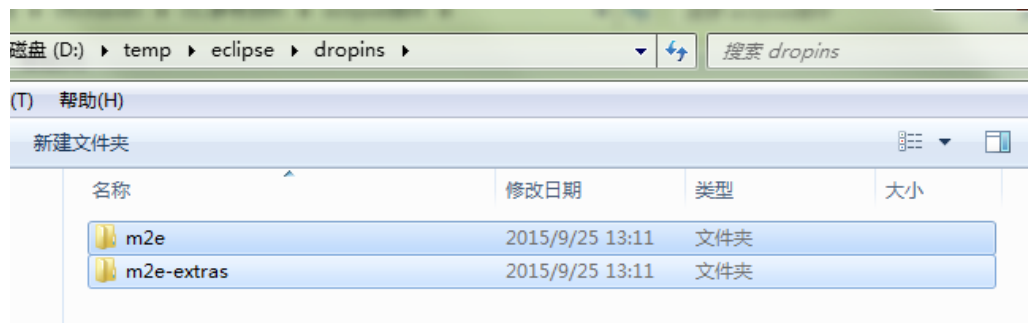
- Maven插件的位置，在参考资料\eclipse插件目录下：



-
- 只需要把这两个文件复制到Eclipse安装目录下的dropins目录下即可。



○

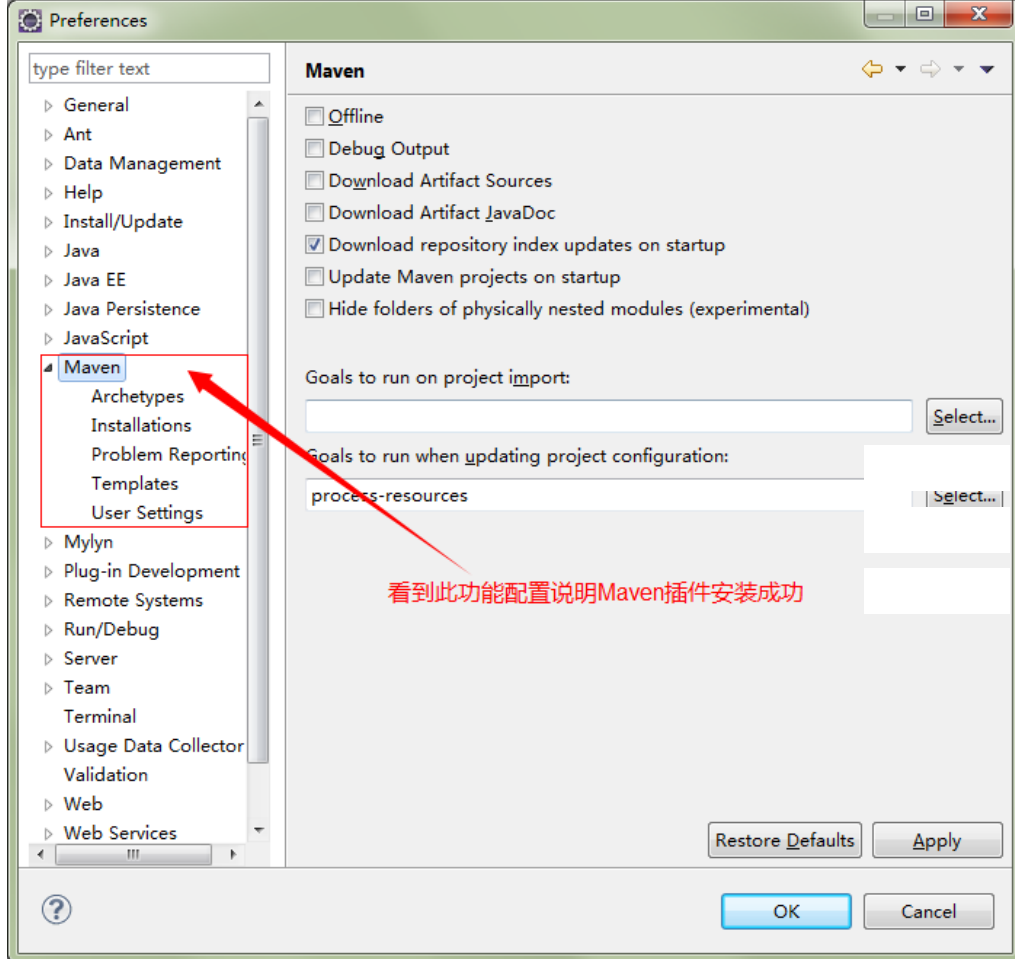


○

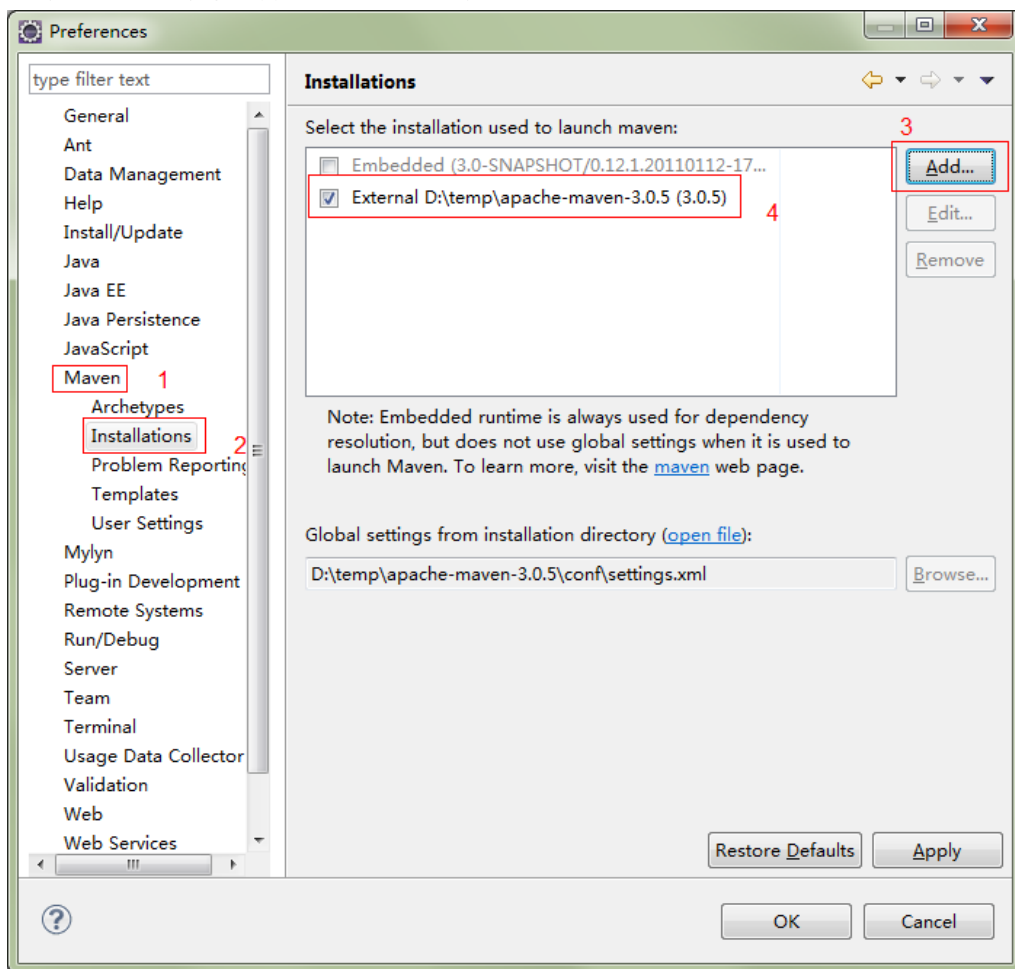
- 5.2 配置maven插件
 - 查看maven插件是否安装成功：启动Eclipse→window→preference→Maven

QQ加我技术群





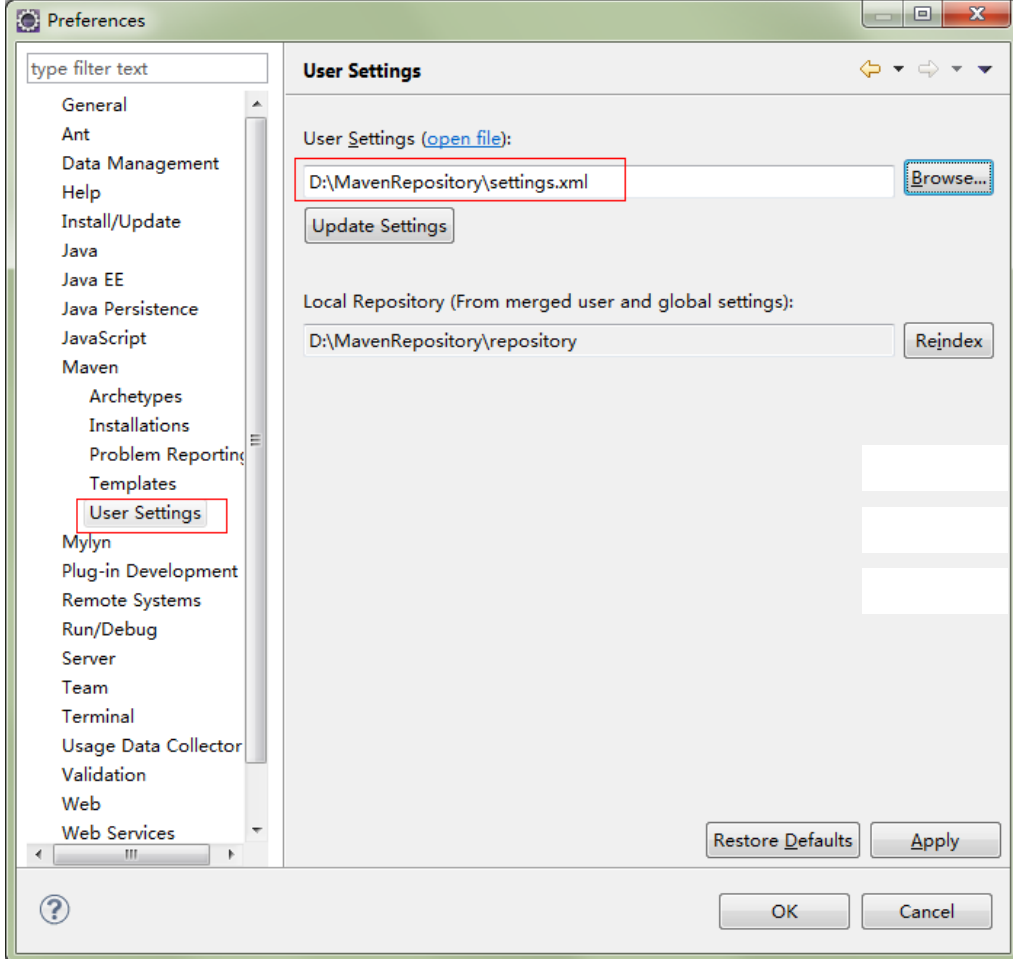
- 2、设置maven环境，也就是指定maven的安装目录



- 3、加载用户配置

QQ加我技术群



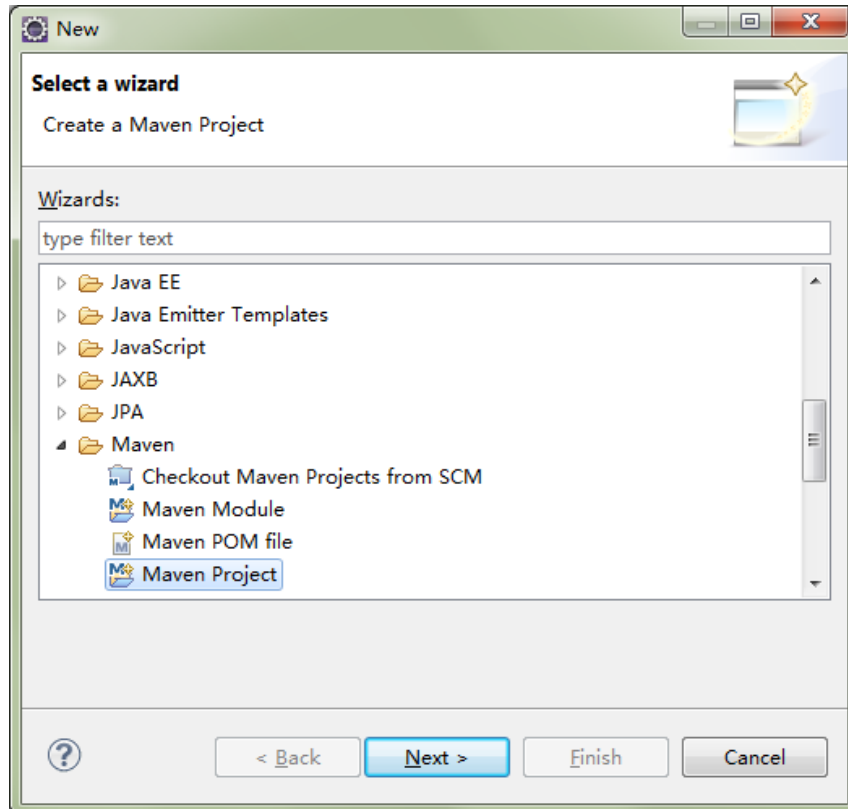


- 5.3 创建maven工程

- 5.3.1 第一个maven工程

- 第一步：选择new→maven→Maven Project

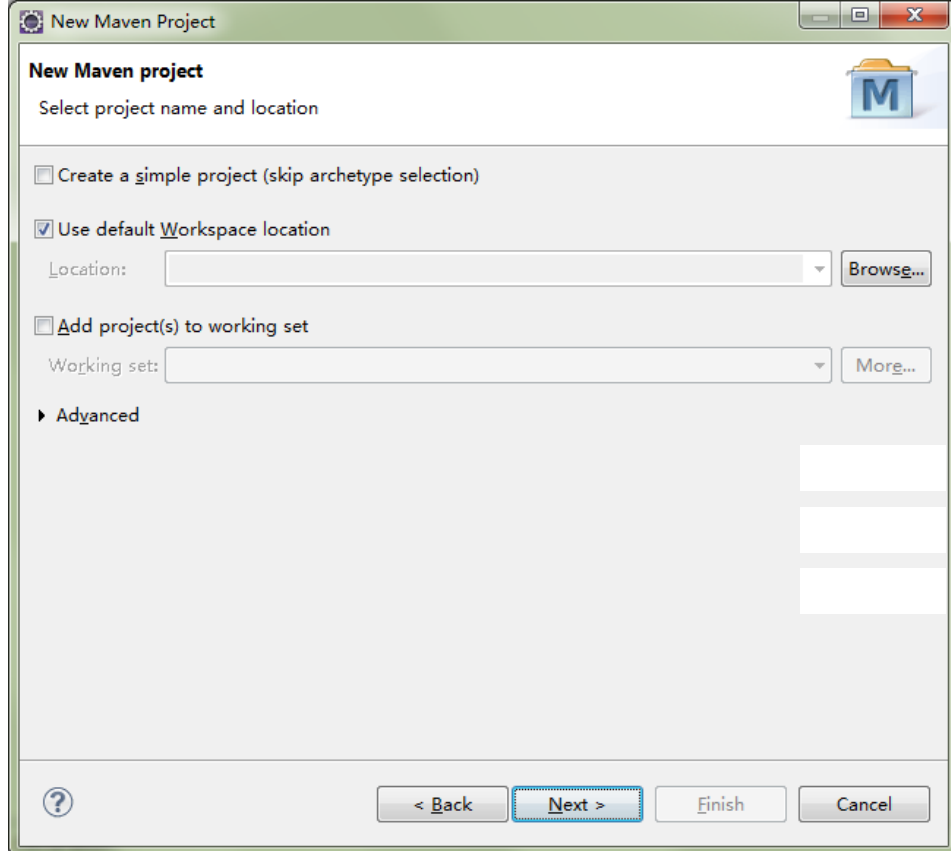
QQ加我技术群



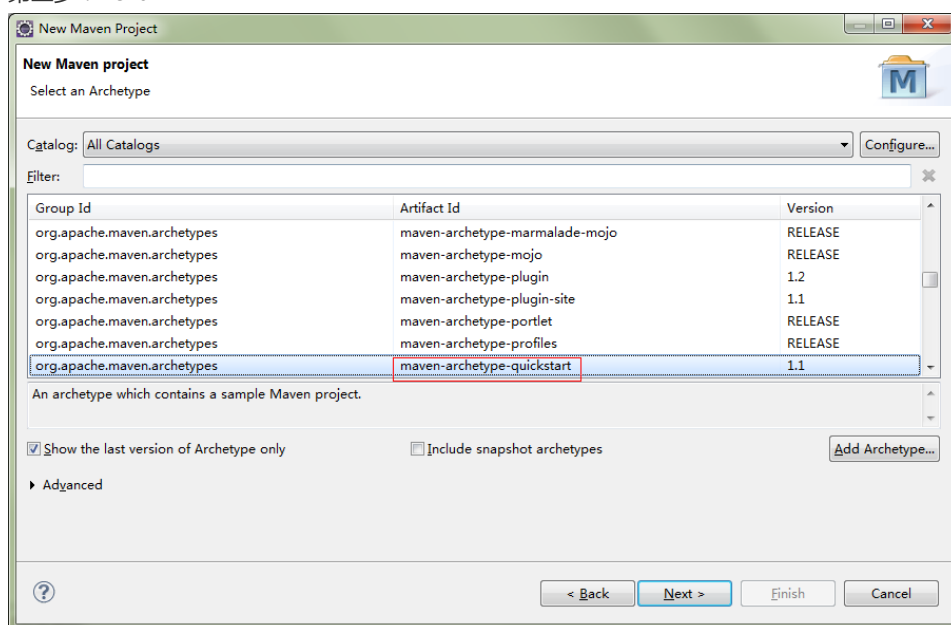
- 第二步：next



QQ加我技术群

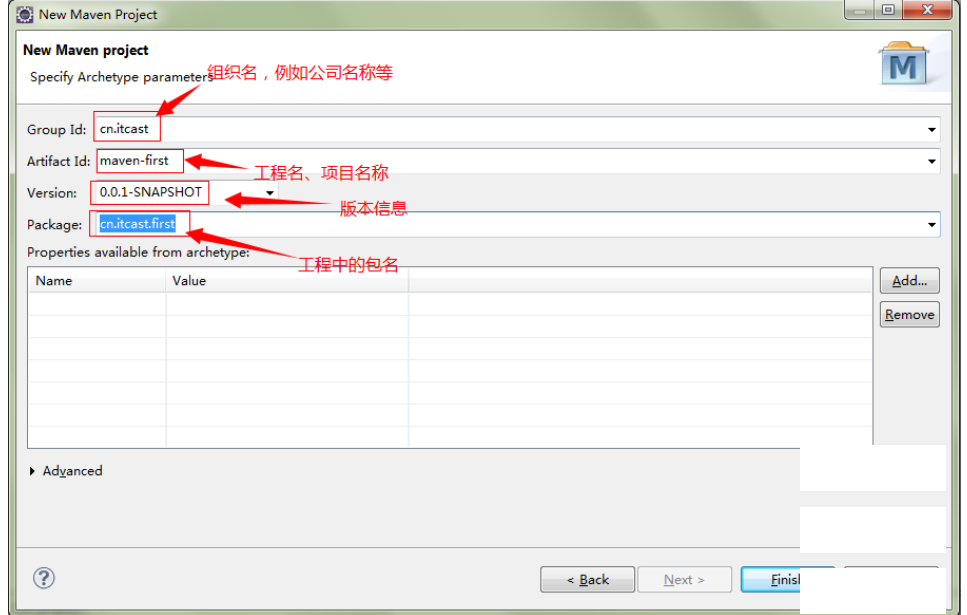


■ 第三步：next

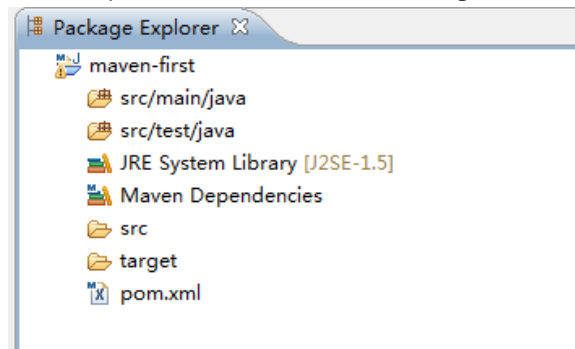


■ 选择maven的工程骨架，这里我们选择quickstart。点击next。

■ 第四步：next

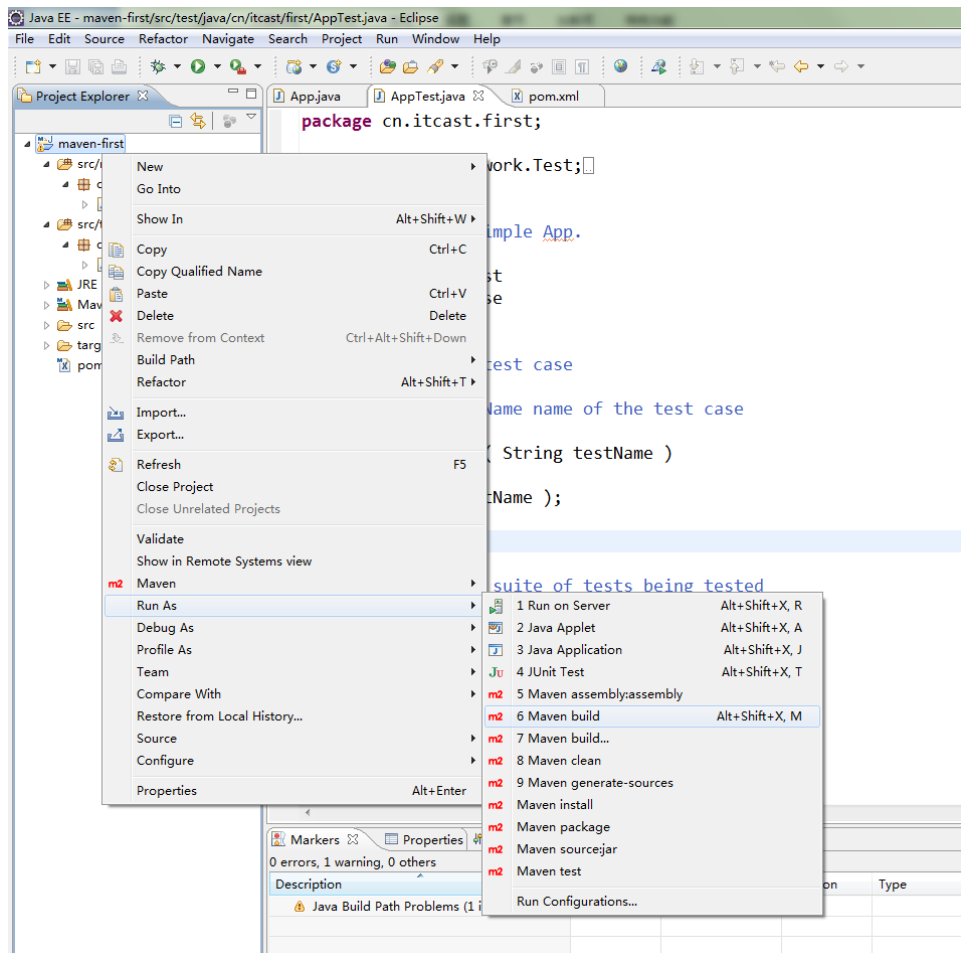


- 输入GroupId、ArtifactId、Version、Package信息点击finish完成。



5.3.2 执行maven命令

- 在Eclipse的maven插件中执行maven命令，需要在maven工程或者pom.xml文件上点击右键，选择Run as→maven build。



5.3.3 第二个maven工程

5.3.3.1 创建工程

QQ加我技术群



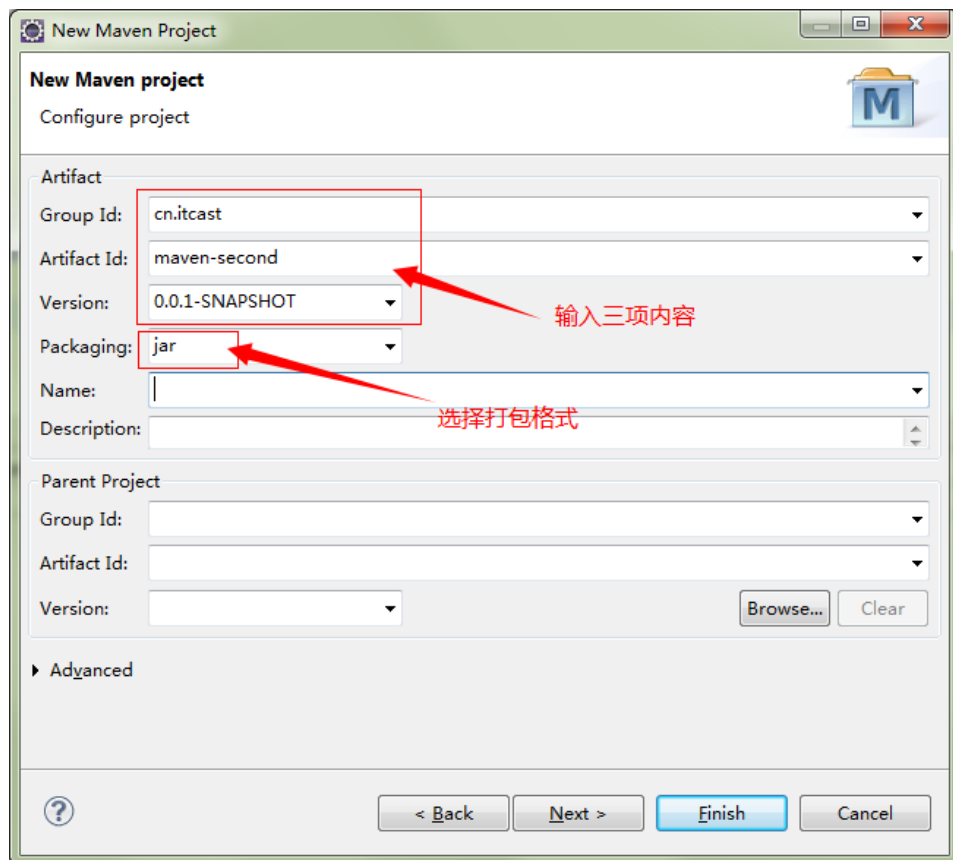
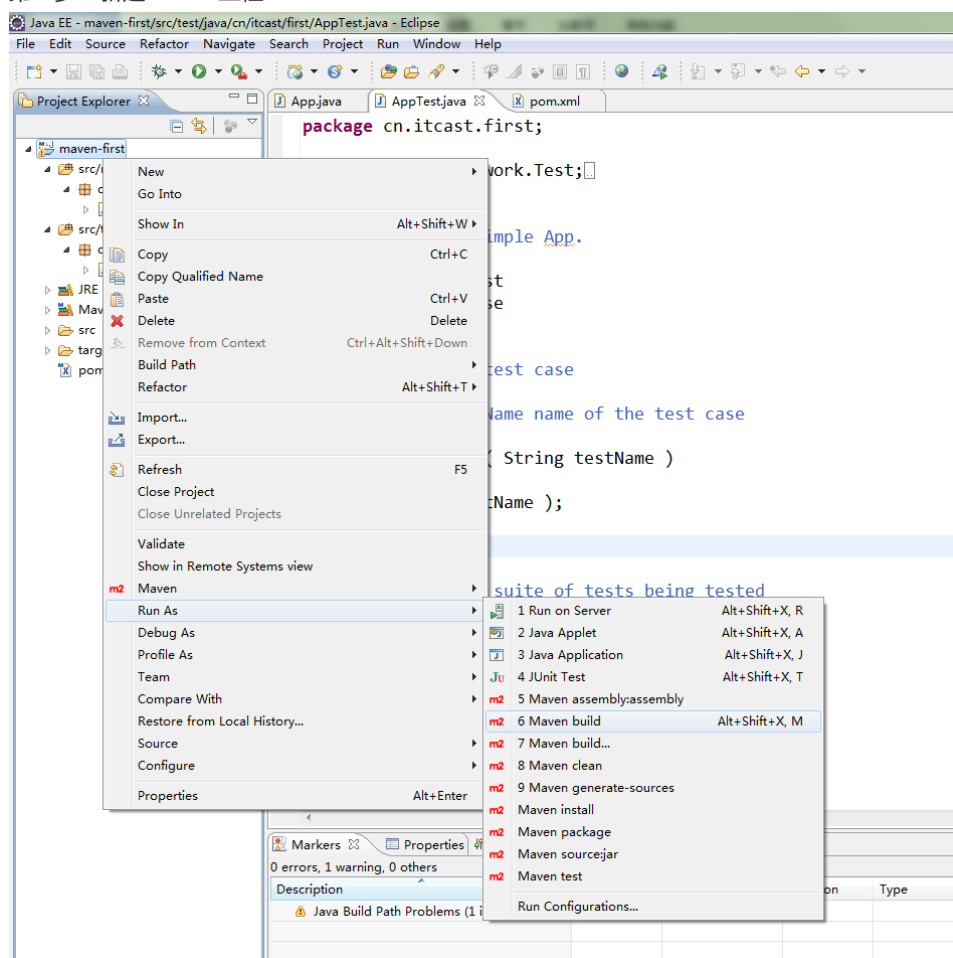


QQ加我技术群



- 第一个工程创建时选用的是maven插件提供的工程骨架，每次选择骨架时都需要联网下载，如果网络不通或者较慢的情况下会有很长时间的等待。使用很是不方便，所以创建工程时可以不选择骨架直接创建工程。

■ 第一步：新建maven工程



- 输入Groupid、artifactid、version三项内容，然后选择packaging也就是工程的打包格式，常用的打包格式为jar、war、pom三种。
- 第三步：点击finish完成。
- 5.3.3.2 修改pom文件



QQ加我技术群



- 需求：在second工程中调用first工程中的First类的方法，然后测试。
- 、

```
1 package cn.change.second;  
2  
3 import cn.change.first.First;  
4  
5 public class Second {  
6  
7     public String say(String name) {  
8         First first = new First();  
9         String result = first.say(name);  
10        return result + ":second";  
11    }  
12 }
```

– 5.3.3.4 创建SecondTest.java

```
1 package cn.change.sencond;  
2  
3 import org.junit.Assert;  
4 import org.junit.Test;  
5  
6 import cn.change.second.Second;  
7  
8 public class SecondTest {  
9  
10    @Test  
11    public void testSay() {  
12        Second second = new Second();  
13        String result = second.say("张三");  
14        Assert.assertEquals("hello 张三:first:second", result);  
15    }  
16 }
```

– 5.3.3.5 测试工程

- ![这里写图片描述](http://img.blog.csdn.net/20160905135716735)
- 如果maven-first工程没有安装则会出现以下错误:

[INFO] Scanning for projects...

[INFO]

[INFO] _____

[INFO] Building maven-second 0.0.1-SNAPSHOT

[INFO] _____

[WARNING] The POM for cn.change:maven-first:jar:0.0.1-SNAPSHOT is missing, no dependency information available

[INFO] _____

[INFO] BUILD FAILURE

[INFO] _____

[INFO] Total time: 0.218s

[INFO] Finished at: Fri Sep 25 15:06:00 CST 2015

[INFO] Final Memory: 4M/15M

[INFO] _____

[ERROR] Failed to execute goal on project maven-second: Could not resolve dependencies for project cn.change:maven-second:jar:0.0.1-SNAPSHOT: Could not find artifact cn.change:maven-first:jar:0.0.1-SNAPSHOT -> [Help 1]

[ERROR]

[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.

[ERROR] Re-run Maven using the -X switch to enable full debug logging.



QQ加我技术群



[ERROR]

[ERROR] For more information about the errors and possible solutions, please read the following articles:

[ERROR] [Help 1] <http://cwiki.apache.org/confluence/display/MAVEN/DependencyResolutionException>

提示找不到maven-first的jar包。当系统运行时是从本地仓库中找依赖的jar包的，所以必须先将maven-first安装才能正常运行，需要在maven-first工程上运行 mvn install命令安装到本地仓库。

- 5.3.4 第三个工程
 - 创建第三个工程此工程使用maven-second工程中的Second的方法，并测试。
 - 5.3.4.1 创建工程
 - ![这里写图片描述](http://img.blog.csdn.net/20160905135828749)
 - 5.3.4.2 修改pom文件

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
2     <modelVersion>4.0.0</modelVersion>
3     <groupId>cn.change</groupId>
4     <artifactId>maven-third</artifactId>
5     <version>0.0.1-SNAPSHOT</version>
6     <properties>
7         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
8     </properties>
9
10    <dependencies>
11        <dependency>
12            <groupId>junit</groupId>
13            <artifactId>junit</artifactId>
14            <version>4.9</version>
15            <scope>test</scope>
16        </dependency>
17        <!-- 依赖maven-first工程 -->
18        <dependency>
19            <groupId>cn.change</groupId>
20            <artifactId>maven-second</artifactId>
21            <version>0.0.1-SNAPSHOT</version>
22        </dependency>
23    </dependencies>
24 </project>
```

- 5.3.4.3 依赖关系
 - ![这里写图片描述](http://img.blog.csdn.net/20160905135916970)
- 5.3.4.4 创建Third.java

```
1 package cn.change.third;
2
3 import cn.change.second.Second;
4
5 public class Third {
6
7     public String say(String name) {
8         Second second = new Second();
9         String result = second.say(name);
10        return result + ":third";
11    }
12 }
```

- 5.3.4.5 创建ThirdTest.java

```
1 package cn.change.third;
2
3 import org.junit.Assert;
4 import org.junit.Test;
5
6 public class ThirdTest {
7     @Test
```



app开发报价单



```
8      public void testSay() {
9          Third third = new Third();
10         String result = third.say("张三");
11         Assert.assertEquals("hello 张三:first:second:third", result);
12     }
13 }
14
15
```

Maven核心概念

Maven的核心概念包括：坐标、依赖管理、仓库管理、生命周期、插件和目标、聚合继承。

- 6.1 坐标
 - 6.1.1 什么是坐标
 - groupId：定义当前Maven组织名称
 - artifactId：定义实际项目名称
 - version：定义当前项目的当前版本
 - packaging：定义该项目的打包方式，如果没有此项则默认为jar包。
 - 6.1.2 Maven坐标主要组成
 - Maven世界拥有大量构建，我们需要找一个用来唯一标识一个构建的统一规范
拥有了统一规范，就可以把查找工作交给机器
 - 6.1.3 Maven为什么使用坐标
 - 依赖声明主要包含如下元素：

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.10</version>
  <scope>test</scope>
</dependency>
```

- 6.2 依赖管理

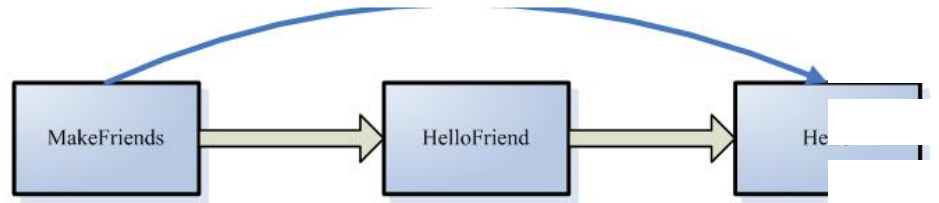
依赖范围 (Scope)	对于主代码 classpath有效	对于测试代码 classpath有效	被打包，对于 运行时 classpath有效	例子
compile	Y	Y	Y	log4j
test	-	Y	-	junit
provided	Y	Y	-	servlet-api
runtime	-	-	Y	JDBC Driver Implementation

- 其中依赖范围scope 用来控制依赖和编译，测试，运行的classpath的关系. 主要的是三种依赖关系如下：
 - 1.compile：默认编译依赖范围。对于编译，测试，运行三种classpath都有效
 - 2.test：测试依赖范围。只对于测试classpath有效
 - 3.provided：已提供依赖范围。对于编译，测试的classpath都有效，但对于运行无效。因为由容器已经提供，例如servlet-api



app开发报价单

- 4.runtime:运行时提供。例如:jdbc驱动
 - 5.import
 - 6.system
 - 如果Two中使用One，Three中使用Two则称Two是Three的直接依赖，而称One是Three的间接依赖。
- C->B B->A
- C直接依赖B
- C间接依赖A



◦

	compile	test	provided	runtime
compile	compile	-	-	runtime
test	test	-	-	test
provided	provided	-	provided	provided
runtime	runtime	-	-	runtime

◦

6.3 生命周期

-

Maven生命周期就是为了对所有的构建过程进行抽象和统一。包括项目清理，初始化，编译，打包，测试，部署等几乎所有构建步骤。
生命周期可以理解为构建工程的步骤。

在Maven中有三套相互独立的生命周期，请注意这里说的是“三套”，而且“相互独立”，这三套生命周期分别是：

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。再次强调一下它们是相互独立的，你可以仅仅调用clean来清理工作目录，仅仅调用site来生成站点。当然你也可以直接运行 mvn clean install site 运行所有这三套生命周期。

Maven三大生命周期

6.3.2.1 clean：清理项目

■

clean生命周期每套生命周期都由一组阶段(Phase)组成，我们平时在命令行输入的命令总会对应于一个特定的阶段。比如，运行mvn clean，这个的clean是Clean生命周期的一个阶段。有Clean生命周期，也有clean阶段。Clean生命周期一共包含了三个阶段：

pre-clean 执行一些需要在clean之前完成的工作 clean 移除所有上一次构建生成的文件 post-clean 执行一些需要在clean之后立刻完成的工作

mvn clean 中的clean就是上面的clean，在一个生命周期中，运行某个阶段的时候，它之前的所有阶段都会被运行，也就是说，mvn clean 等同于 mvn pre-clean clean，如果我们运行 mvn post-clean，那么pre-clean，clean 都会被运行。这是Maven很重要的一个规则，可以大大简化命令行的输入。

6.3.2.2 default：构建项目

-

QQ加我技术群





app开发报价单



Default生命周期Default生命周期是Maven生命周期中最重要 的一个，绝大部分工作都发生在这个生命周期中。这里，只解释一些比较重要和常用的阶段：

validate generate-sources process-sources generate-resources
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-classes generate-test-sources process-test-sources
generate-test-resources 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。 process-test-classes 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。 prepare-package package
接受编译好的代码，打包成可发布的格式，如 JAR 。 pre-integration-test integration-test
post-integration-test verify 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。

运行任何一个阶段的时候，它前面的所有阶段都会被运行，这也就是为什么我们运行mvn install 的时候，代码会被编译，测试，打包。此外，Maven的插件机制是完全依赖Maven的生命周期的，因此理解生命周期至关重要。

- 6.3.2.3 site: 生成项目站点

Site生命周期

- pre-site 执行一些需要在生成站点文档之前完成的工作
- site 生成项目的站点文档
- post-site 执行一些需要在生成站点文档之后完成的工作，并且为部署做准备
- site-deploy 将生成的站点文档部署到特定的服务器上

这里经常用到的是site阶段和site-deploy阶段，用以生成和发布Maven站点，这可是Maven相当强大的功能，Manager比较喜欢，文档及统计数据自动生成，很好看。

- 6.4 Maven插件
 - Maven的核心仅仅定义了抽象的生命周期，具体的任务都是交由插件完成的。每个插件都能实现一个功能，每个功能就是一个插件目标。Maven的生命周期与插件目标相互绑定，以完成某个具体的构建任务。
例如compile就是插件maven-compiler-plugin的一个插件目标

QQ加我技术群



顶

18

踩

9

上一篇 eclipse使用技巧

下一篇 面试框架部分总结

我的同类文章

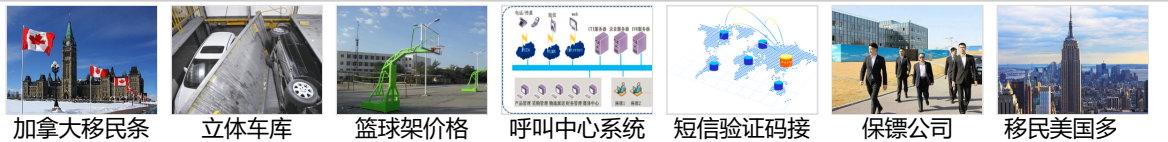
java (43)		maven (1)	
• 简单认识Nginx---负载均衡	2017-04-28 阅读 195	• Spring 集成Kafka(完整版)	2017-04-16 阅读 940
• Kafka集群配置---Windows版	2017-04-10 阅读 339	• 大话 Spring Session 共享	2017-04-01 阅读 1597
• 通过注解实现通用导出Excel	2017-03-22 阅读 331	• java 注解的使用	2017-03-02 阅读 643
• Javaweb表格加载---DataTa...	2017-02-27 阅读 678	• Spring项目集成ShiroFilter...	2017-02-13 阅读 763
• java反射	2017-02-13 阅读 815	• java swing 开发 -JTable	2017-01-19 阅读 1368
• Redis回顾	2017-01-18 阅读 1046		

更多文章



app开发报价单





参考知识库



MySQL知识库

22144 关注 | 1449 收录



Java 知识库

26202 关注 | 1457 收录



.NET知识库

3810 关注 | 836 收录



软件测试知识库

4582 关注 | 318 收录



Java SE知识库

25922 关注 | 479 收录



Java EE知识库

17945 关注 | 1324 收录

猜你在找

单元测试和JUnit4实战视频教程

Maven下载安装和配置

spring3.2入门到大神（备java基础、jsp、servlet、j

gradle学习笔记

大规模敏捷需求管理

webmagic采集CSDN的Java_WebDevelop页面

【直通华为HCNA/HCNP系列R篇3】路由器接口配置与管理

Android 开源项目分类汇总

JUnit免费课程

Android 应用开发GitHub 优秀的 Android 开源项目

QQ加我技术群



查看评论

暂无评论

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题

Hadoop

AWS

移动游戏

Java

Android

iOS

Swift

智能硬件

Docker

OpenStack

VPN

Spark

ERP

IE10

Eclipse

CRM

JavaScript

数据库

Ubuntu

NFC

WAP

jQuery

BI

HTML5

Spring

Apache

.NET

API

HTML

SDK

IIS

Fedora

XML

LBS

Unity

Splashtop

UML

components

Windows Mobile

Rails

QEMU

KDE

Cassandra

CloudStack

FTC

coremail

OPhone

CouchBase

云计算

iOS6

Rackspace

Web App

SpringSide

Maemo

Compuware

大数据

aptech

Perl

Tornado

Ruby

Hibernate

ThinkPHP

HBase

Pure

Solr

Angular

Cloud Foundry

Redis

Scala

Django

Bootstrap

广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |