# Eclipse

## Eclipse右键New菜单项的自定义设置

大家都知道一般在Eclipse中新建一个项目或者文件都是通过菜单项的File-New来创建，然而有些右键出来的选项可能从来都不会使用到，而有些可能会常用到但是右键中又没有，这个时候就可以自定义New中的右键选项，具体设置方式如下：

**Window --> Customize Perspective... --> Shortcuts --> 选择"New"**  进行相关设置即可(如下图)



## web项目run as 菜单中没有 run on server

eclipse中已安装好tomcat，某maven web项目原本也可以在tomcat中运行，突然发现右击该项目的 run as 菜单中，没有了 run on serven 选项，项目也无法在tomcat中跑了。

原因是该项目的Dynamic Web Module 被我不经意间删除了，如下图。MyEclipse中的Web项目导入到Eclipse后，也会出现这样的情况。主要把Dynamic Web Module勾上即可。



有时勾选Dynamic Web Module 时会出现这个错误提示：Dynamic web module 3.0 requires java 1.6 or newer



解决这个错误的办法是：在项目的pom.xml的<build></build>标签中加入：

1. **<plugins>**
2. **<plugin>**
3. **<groupId>**org.apache.maven.plugins**</groupId>**
4. **<artifactId>**maven-compiler-plugin**</artifactId>**
5. **<version>**2.3.2**</version>**
6. **<configuration>**
7. **<source>**1.6**</source>**
8. **<target>**1.6**</target>**
9. **</configuration>**
10. **</plugin>**
11. **</plugins>**

保存，项目构建完成后在项目文件夹上点右键，选择Maven->Update Project Configuration，问题解决。

## 为什么eclipse新建项目的默认编码为何是GBK,而不是UTF-8

我们用的操作系统是中文操作系统，windows操作系统的编码是GBK，所以eclipse默认的新项目的编码是GBK的，你在Window->Preferences->General->Workspace->Text file encoding 将其改为UFT-8即可，同样，如果你想修改html,css等文件的编码格式，你在Window->Preferences->General->Content Types中可以修改

## 如何删除Eclipse弃用的工作空间？

在 eclipse\configuration\.settings\org.eclipse.ui.ide.prefs 文件里找到相应的空间，删除即可。

## eclipse因引用本地的JDK而不是JRE。



## Eclipse中安装Tomcat

1、在本地安装并配置好tomcat

2、新建一个Web项目，右击该项目，选择Run as -> run on server.

3、在弹出的run on server框上选择要安装的tomcat版本







## 将MyEclipse的Web项目导入到Eclipse

将一个MyEclipse的Web项目导入到Eclipse后，项目并没有报错，奇怪的是，右击该项目，选择run as菜单，此时并没有出现run on server 选项。项目结构如下：



解决的办法如下：

1、为项目添加Dynamic Web Module



2、添加Dynamic Web Module后，run as菜单出现run on server了。执行run on server，在tomcat上运行该项目，结果切报错。



3、原来添加Dynamic Web Module后，项目结构中多了一个WebContent目录。Myeclipse中默认的Web目录是WebRoot，而Eclipse中默认的Web目录是Webcontent，因此报错。



4、在项目属性中修改项目的部署目录。添加Dynamic Web Module后项目部署目录如下：



删除WebContent目录，然后点击Add按钮添加Folder，让Folder指向WebRoot





5、在项目的目录结构中，删除WebContent目录，删除后如下：



6、重新执行run on server即可。

## maven项目导入Eclipse

maven web 项目导入Eclipse后，执行maven build 命令，编译运行报错，找不到配置文件。

原因是导入项目的部署路径没有设置好，右击项目选择属性，按下图设置好即可。



## Eclipse中Project Explorer和Package Explorer这两个有什么区别

Package Explorer不会显示修改某些文件后会自动更改的文件，如class文件，在web工程里web-inf下的classes文件夹在Package Explorer下会隐藏  
  
Project Explorer会显示所有文件

# tomcat

## tomcat的安装与配置

**以tomcat7.0为例，Tomcat 7.0 （请到官网下载）的免安装版的配置(假如将Tomcat 解压到C:\Program Files目录，目录结构为：C:\Program Files\apache-tomcat-7.0.39。**

**tomcat的安装目录最好在某个盘的根目录下，路径不能带中文、空格或符号，例如：D:\tomcat7。以免在运行过程中产生其他问题。**

**1、添加环境变量：在 我的电脑->属性->高级->环境变量**

**2、新建系统变量，变量名：CATALINA\_HOME 变量值：C:\Program Files\apache-tomcat-7.0.39 (Tomcat 解压到的目录)。**

**3、在系统变量 Path的最后面添加%CATALINA\_HOME%\lib;%CATALINA\_HOME%\lib\servlet-api.jar;%CATALINA\_HOME%\lib\jsp-api.jar 注意它们之间的分号，一定是英文的分号。**

**4、Tomcat 7.0的管理员的配置，进入C:\Program Files\apache-tomcat-7.0.11(Tomcat目录)下的conf目录，编辑tomcat-users.xml，找到最后的：**

**<!--**

**<role rolename="tomcat"/>**

**<role rolename="role1"/>**

**<user username="tomcat" password="tomcat" roles="tomcat"/>**

**<user username="both" password="tomcat" roles="tomcat,role1"/>**

**<user username="role1" password="tomcat" roles="role1"/>**

**-->**

**在上面这段后面 添加上：**

1. **<role** rolename="admin-gui"**/>**
2. **<role** rolename="manage-gui"**/>**
3. **<user** username="admin" password="admin" roles="admin-gui,manage-gui"**/>**

**到这应该知道我们添加的管理员的用户名和密码是多少了吧!**

**5、进入Tomcat 目录下的bin目录，双击startup.bat 启动Tomcat 在命令行窗口会出来英文提示。**

**6、浏览器输入：http://localhost:8080 可以看到Tomcat的欢迎页面就说明配置成功了，点击右上角上manager连接，输入上面配置的用户名和密码，就可以进入管理页面。**

****

# java 基础

## [使用命令wsimport构建WebService客户端](http://www.cnblogs.com/simle/archive/2011/11/03/2233417.html)

建立 a 文件夹，执行命令：

E:\阳光\医院自助终端\华侨医院>wsimport -s a -p com.sun309.frontend.remote.settle http://www.jd120.com/Settle/SettleService.asmx?wsdl

# maven

## Eclipse创建Maven项目部署war包到外部tomcat

Eclipse+Maven创建webapp项目<二>

1、开启eclipse，右键new——》other，如下图找到maven project



2、选择maven project，显示创建maven项目的窗口，此次不勾选Create a simple project，直接next，如图所示：



3、在Filter中搜索webapp，选择如图所示，并next：



4、输入maven项目的基本信息，如下图所示：



5、完成maven项目的创建，生成相应的webapp项目结构，如下所示，此处有部分结构是项目不需要的，我们需要去掉：



6、选择项目，右键选择Properties，进入属性页面，选择到Project Facets菜单下，如下图所示：



7、选择java版本为1.7，并去掉其他两项，如下图：



8、点击ok之后，再次回到项目结构，此时项目结构比较清晰，符合我们想要创建的maven项目：



9、创建没有显示的源代码文件夹，如下图所示：





10、最后得到完成的maven webapp项目结构如下：



11、右击项目，执行 Run s -> Maven install . 在项目的target目录下生成对应的war包maven.archetype.webapp.war 。

12、将生成的war包拷到tomcat的webapps目录下



启动tomcat，在浏览器输入地址：http://localhost:8080/maven.archetype.webapp/ 即可访问。



正常情况tomcat启动时会把war把解压，生成一个同名的文件夹在webapps目录下。如果有的tomcat不能解压war包导致war包不能正常运行，则把正常解压后的项目文件拷到这个tomcat下就OK了。

例如SunPlatform.war 在本机的tomcat上能正常运行，在服务器上就不能。则把本机tomcat启动后解压出来的SunPlatform目录复制到服务器，就可以了。

13、也可以不用外部tomcat，而在项目中使用tomcat-maven-plugin 运行项目。

在pom.xml中加入以下代码：

<build>

<finalName>maven.archetype.webapp</finalName>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.0</version>

</plugin>

</plugins>

</pluginManagement>

</build>

保存后右击项目，执行 Run As -> Run Configurations



即可在tomcat-maven插件中运行该项目。打开浏览器输入地址http://localhost:8080/maven.archetype.webapp/ 访问项目首页。

注意：此时在项目属性的Project Facets 中，并没有引入Dynamic Web Module，因此不能执行Run As -> Run on Server 来引入本地tomcat运行。

## Eclipse+Maven构建springmvc项目

一.背景介绍

　　对于初学者，用maven构建项目并不是一件容易的事，springmvc并不是MVC中的主流，但是本人觉得springmvc比struts要好用，真正做到了零配置。一次使用，就放不下了。

二.准备工作

　1.Eclipse 3.7

　　2.maven

　　3.Eclipse 需要安装maven插件。url:maven - <http://download.eclipse.org/technology/m2e/releases> 。安装maven-3.0.4。并且选择本地的maven，如下图：



三.构建工程

　　1.用maven插件构建项目框架

　　maven具有强大构建功能，使用maven可以构建多种不同类型的工程。这里我们构建maven-archetype-webapp类型的项目。在Eclipse->New中选择other,找到maven Project型。如下图：



在选完路径之后，我们选择构建类型，如下图：



接下来，填写工程的Group Id，Artifact Id，如下图：



这里的Group Id就是大项目的id，Arifact Id就是该项目的Id。就像一个大项目中有许多小项目组成一样。此时，我们的项目已经成型了，样子如下图：



接下来，我们要完善项目的目录，配置。

　　2.完善项目

　　首先，完善目录，增加重要的source Folder，这个不是简单的Floder,这些文件夹是会参与编译的。增加src/main/java，src/test/resources，src/test/java目录。让目录变成标准的maven结构。如下图：



接下来，改变一些配置：

让工程的JDK用本地的jdk;

让工程的字符集为UTF-8;

改变工程的目录顺序；

　　这些都完成之后，工程目录应该是如下的样子：



　　3.将工程变成web工程

　　此时，我们的工程还不是标准的web工程，可以在eclipse中增加web工程的特性，选择工程的Properties,选Project Facets,如下图：



这里，我们选择Dynamic Web Module,版本选择2.4，这个版本比较通用。如下图：



此时，我们看到目录中多了一个WebContent目录，由于使用maven构建，web目录是src/main/webapp,所以我们删除WebContent目录。接下来，要配置web项目的发布目录，就是Deployment Assembly，如图：



test目录不要发布，WebContent目录没有了，所以这三项都删掉。并且增加src/main/webapp目录，和Maven Dependenices,完成之后如下图：



于是，我们的工程就完全是一个web工程了。

　　4.赋予工程的springmvc特性

　　配置web.xml，使其具有springmvc特性，主要配置两处，一个是ContextLoaderListener，一个是DispatcherServlet。代码如下：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 <?xml version="1.0" encoding="UTF-8"?>

2 <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"

3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

4 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

5 http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

6

7 <listener>

8 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

9 </listener>

10

11 <servlet>

12 <servlet-name>exam</servlet-name>

13 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

14 </servlet>

15

16 <servlet-mapping>

17 <servlet-name>exam</servlet-name>

18 <url-pattern>/</url-pattern>

19 </servlet-mapping>

20

21 <welcome-file-list>

22 <welcome-file>index.jsp</welcome-file>

23 </welcome-file-list>

24 </web-app>

[复制代码](javascript:void(0);)

　　配置ContextLoaderListener表示，该工程要以spring的方式启动。启动时会默认在/WEB-INF目录下查找applicationContext.xml作为spring容器的配置文件，这里可以初始化一些bean，如DataSource。我们这里什么也不做。代码如下：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

1 <?xml version="1.0" encoding="UTF-8"?>

2 <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">

3 <beans>

4 </beans>

　　配置DispatcherServlet表示，该工程将采用springmvc的方式。启动时也会默认在/WEB-INF目录下查找XXX-servlet.xml作为配置文件，XXX就是DispatcherServlet的名字，该文件中将配置两项重要的mvc特性：

HandlerMapping,负责为DispatcherServlet这个前端控制器的请求查找Controller；

ViewResolver,负责为DispatcherServlet查找ModelAndView的视图解析器。

代码如下：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 <?xml version="1.0" encoding="UTF-8"?>

2 <!-- Bean头部 -->

3 <beans xmlns="http://www.springframework.org/schema/beans"

4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"

5 xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:context="http://www.springframework.org/schema/context"

6 xmlns:util="http://www.springframework.org/schema/util"

7 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

8 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd

9 http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd

10 http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.0.xsd">

11

12 <!-- 激活@Controller模式 -->

13 <mvc:annotation-driven />

14 <!-- 对包中的所有类进行扫描，以完成Bean创建和自动依赖注入的功能 需要更改 -->

15 <context:component-scan base-package="cc.monggo.web.controller" />

16

17 <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />

18

19 <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">

20 <property name="prefix">

21 <value>/WEB-INF/jsp/</value>

22 </property>

23 <property name="suffix">

24 <value>.jsp</value>

25 </property>

26 </bean>

27 </beans>

[复制代码](javascript:void(0);)

　　5.让maven自动配置jar包

　　在用maven生成框架时，就生成了pop.xml，这就是maven的配置文件。我们要引入spring-web,servlet等特性的包。代码如下：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

2 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

3 <modelVersion>4.0.0</modelVersion>

4 <groupId>exam</groupId>

5 <artifactId>exam\_3</artifactId>

6 <packaging>war</packaging>

7 <version>0.0.1-SNAPSHOT</version>

8 <name>exam\_3 Maven Webapp</name>

9 <url>http://maven.apache.org</url>

10 <dependencies>

11 <dependency>

12 <groupId>junit</groupId>

13 <artifactId>junit</artifactId>

14 <version>3.8.1</version>

15 <scope>test</scope>

16 </dependency>

17

18 <dependency>

19 <groupId>org.springframework</groupId>

20 <artifactId>spring-web</artifactId>

21 <version>3.0.5.RELEASE</version>

22 </dependency>

23

24 <dependency>

25 <groupId>org.springframework</groupId>

26 <artifactId>spring-webmvc</artifactId>

27 <version>3.0.5.RELEASE</version>

28 </dependency>

29

30 <dependency>

31 <groupId>org.apache.geronimo.specs</groupId>

32 <artifactId>geronimo-servlet\_2.5\_spec</artifactId>

33 <version>1.2</version>

34 </dependency>

35

36 </dependencies>

37 <build>

38 <finalName>exam\_3</finalName>

39 </build>

40 </project>

[复制代码](javascript:void(0);)

　　maven就是这么简单，一旦保存，maven就会自动下载pop.xml的jar包。此时可以看到目录中Maven Dependencies下生成了jar包。

　　更多的jar包可以在maven中心库下载：[http://mvnrepository.com](http://mvnrepository.com/)。

　　6.做个测试

　　说了一大堆，只有运行起来才有意思，下面写个简单的测试。先写Controller。编写两个类，LoginControler.java，LoginForm.java。代码如下：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 package cc.monggo.web.controller;

2

3

4 import javax.servlet.http.HttpServletRequest;

5 import javax.servlet.http.HttpServletResponse;

6

7 import org.springframework.stereotype.Controller;

8 import org.springframework.web.bind.annotation.RequestMapping;

9 import org.springframework.web.servlet.ModelAndView;

10

11 import cc.monggo.domain.LoginForm;

12

13 @Controller

14 public class LoginController {

15 @RequestMapping(value="login")

16 public ModelAndView login(HttpServletRequest request,HttpServletResponse response,LoginForm command ){

17 String username = command.getUsername();

18 ModelAndView mv = new ModelAndView("/index/index","command","LOGIN SUCCESS, " + username);

19 return mv;

20 }

21 }

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 package cc.monggo.domain;

2

3

4 public class LoginForm {

5 private String username;

6 private String password;

7 public String getUsername() {

8 return username;

9 }

10 public void setUsername(String username) {

11 this.username = username;

12 }

13 public String getPassword() {

14 return password;

15 }

16 public void setPassword(String password) {

17 this.password = password;

18 }

19 }

[复制代码](javascript:void(0);)

　　再增加一些jsp,首页的index.jsp,主要是做跳转，代码如下：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

1 <%

2 request.getRequestDispatcher("/WEB-INF/jsp/login/login.jsp").forward(request,response);

3 %>

　  还有两个jsp，做些简单的功能，一个表单login.jsp,一个表单提交的返回index.jsp，代码如下：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 <%@ page language="java" contentType="text/html; charset=utf-8"

2 pageEncoding="utf-8"%>

3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

4 <html>

5 <head>

6 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

7 <title>Insert title here</title>

8 </head>

9 <body>

10 <div>

11 <form action="login" methed="get">

12 <input type="text" name="username">

13 <input type="submit" value="SUBMIT">

14 </form>

15 </div>

16 </body>

17 </html>

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 <%@ page language="java" contentType="text/html; charset=utf-8"

2 pageEncoding="utf-8"%>

3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

4 <html>

5 <head>

6 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

7 <title>Insert title here</title>

8 </head>

9 <body>

10 ${command}

11 </body>

12 </html>

[复制代码](javascript:void(0);)

　　整个项目的目录结构如下：



　　在tomcat中运行，即可。不要使用Tomcat插件，可能有问题。就用普通的方式。运行效果如下图：



## Maven项目打包缺少XML文件

解决办法，在pom.xml中把xml文件包含进去。

<build>

<resources>

<resource>

<directory>src/main/java</directory>

<includes>

<include>com/sun/mapping/ChargeMapper.xml</include>

<include>com/sun/mapping/ConsumeMapper.xml</include>

<include>com/sun/mapping/RefundMapper.xml</include>

<include>com/sun/mapping/UserMapper.xml</include>

</includes>

<filtering>true</filtering>

</resource>

</resources>

</build>

## pom.xml更新下载jar包失败





1.问题描述

在通过pom.xml中的依赖坐标,从私服下载依赖包时,往往会遇到如下的类似问题:

The container 'Maven Dependencies' references non existing library '${groupid}/${artifactid}-${version}.jar'

e.g.

|  |  |
| --- | --- |
| 1 | The container 'Maven Dependencies' references non existing library 'E:\Maven\repo\org\apache\hadoop\hadoop-yarn-api\2.0.0-cdh4.5.0\hadoop-yarn-api-2.0.0-cdh4.5.0.jar' |

2.解决方案

上面问题往往是在下载依赖过程中网络出现问题导致的.

此时我们本机已经开始下载依赖代码,但是下载失败,本机仓库中会在${MAVEN\_repo}/${groupid}/${artifactid}/${version} 路径下面生成 \*.lastUpdated 的文件.

\*.lastUpdated实在可恶,下次再下载时由于它的存在Maven不会再去私服中下载

2.1 解决方法:(针对PC与Maven私服之间网络传输问题)

1. 打开本地仓库所在目录, 通过win文件夹的搜索功能,查找 \*.lastUpdated ,然后将找到的文件全部删除
2. 重新 Maven Update Project,
   1. 若在eclipse中操作时勾选上 Force Update of Snapshots / Releases
   2. 若在命令行操作时增加 -U 参数

一般情况下上面方法即可解决问题. 若问题依然存在,请确认下主机与Maven私服之间网络是否畅通.可以通过在浏览器中打开私服地址确认.

若本机与私服之前网络畅通,那这个问题可能是由于私服与中央仓库(或者其他第三方仓库)之间下载依赖时网络故障导致的.

2.2 解决方法:(针对私服与中央仓库或第三方仓库之间网络传输问题)

1. 使用管理员登陆私服页面(Sonatype Nexus)
2. 在仓库的浏览页面找到下载失败的包, 右键菜单 "Update Index", 右键菜单 "Expire Cache"

[](http://static.oschina.net/uploads/space/2013/1206/162311_NoW2_1189023.png)

1. 按照PC与私服之间网络问题的解决步骤处理一遍

# Hibernate

## Eclipse 使用Hibernate Tools插件生成POJO

1. 很多时候我们已经设计好了数据库，需要使用hibernate来做数据持久化，因此需要根据数据库中的表结构生成相应的POJO.
3. 本例使用hibernatetools来自动创建pojo。
5. 测试环境：eclipse3.7（indigo）+hibernate4.1+hibernatetools3.3+oracle10g
6. 生成POJO只需要导入数据库驱动包，不需要导入其他Hibernate包。
7. **1.下载hibernatetools插件**
9. hibernatetools可从官网下载，但是需要注意下载和自己使用的eclipse版本对应的版本包，然后安装到eclipse中取，link安装方式参见：<http://maimode.iteye.com/blog/1164524>
11. 下载地址:
12. 1、首先，要去Hibernate的网站下载这个工具包。一个大约14M的ZIP压缩文件。  
    <http://jaist.dl.sourceforge.net/sourceforge/jboss/HibernateTools-3.2.4.Beta1-R200810311334.zip>  
    2、解压缩下载好的文件，释放出features和plugins这两个文件夹和其中的内容。完成后将这两个文件夹复制到Eclipse的目录下（这两个文件夹在Eclipse中是默认存在的，直接复制就行了）。  
    3、到这步为止HibernateTools的安装就算结束了
13. **2.创建hibernate配置文件及console配置文件**
15. 新建java项目，建立包test,test.hibernate,test.model。
16. 选中项目，右键-》new-》other-》hibernate-》hibernate configuration file：
17. 
19. 然后 next，配置console文件：
20. 

23. **3.创建reverse文件**
25. 选中项目，右键-》new-》other-》hibernate-》hibernate reverse engineering file
26. 
28. 下一步，配置要生成POJO对应的表：
29. 
31. **4.反向工程创建pojo**
33. 在hibernate code generation configuration菜单中配置
34. 
36. 
38. 
40. 执行run后便生成了相应的pojo文件：
42. 

45. 一般自动生成的mapping文件（XXX.hbm.xml）都需要修改后才能使用。

## 日期类型

某字段数据库中数据类型为date时，Hibernate的实体类中，该字段可以是Date类型，也可以是String类型。String类型时，默认是yyyy-MM-dd HH:mm:ss格式。

# spring

## 通过ClassPathXmlApplicationContext读取配置文件

ApplicationContext context =

**new** ClassPathXmlApplicationContext(**new** String[]

{ "file:D:/eclipse-jee-juno/mavenspace/maven.springmvc/src/main/webapp/WEB-INF/applicationContext.xml" });

SpringPublisher publisher = (SpringPublisher) context

.getBean("springPublisher");

## Windows下ActiveMQ下载、安装部署

1、下载：http://activemq.apache.org/download.html 最新Windows版本

2、安装

(1) 首先配置JAVA环境变量

JAVA\_HOME=D:\Program Files\Java\jdk1.5.0

CLASSPAHT=.;D:\Program Files\Java\jdk1.5.0\lib\tools.jar;D:\Program Files\Java\jdk1.5.0\lib\td.jar;D:\Program Files\Java\jdk1.5.0\lib

PATH=D:\Program Files\Java\jdk1.5.0\bin;

(2)直接解压至任意目录(例如：D:\apache-activemq-5.3.0)

3、启动ActiveMQ服务器：直接运行\bin\win32\activemq.bat

当运行成功后

4、打开ActiveMQ消息管理后台系统 http://localhost:8161/admin/

## 消息队列：Spring JMS

### [SPRING与JMS结合的实例](http://www.cnblogs.com/phoebus0501/archive/2011/02/28/1966896.html)

|  |
| --- |
| package jms.activemq.myexample.spring;  import java.util.Date;  import javax.jms.JMSException;  import javax.jms.Message;  import javax.jms.Session;  import javax.jms.TextMessage;  import org.springframework.jms.core.MessageCreator;    public class MyMessageCreator implements MessageCreator {        /\*\*       \* 消息序号       \*/      private int msgNo;        public MyMessageCreator(int no) {          this.msgNo = no;      }        @Override      public Message createMessage(Session session) throws JMSException {          TextMessage textMsg = session.createTextMessage();          textMsg.setText(new Date() + "第" + this.msgNo + "条消息发出");            return textMsg;      }    } |
| package jms.activemq.myexample.spring;    import javax.jms.\*;    /\*\*   \* Text消息监听   \*   \* @author xiaochuanyu   \*   \*/  public class TextListener implements MessageListener {        /\*\*       \* Casts the message to a TextMessage and displays its text.       \*       \* @param message       \*            the incoming message       \*/      public void onMessage(Message message) {          TextMessage msg = null;            try {              if (message instanceof TextMessage) {                  msg = (TextMessage) message;                  System.out.println("Reading message: " + msg.getText());              } else {                  System.out.println("Message of wrong type: "                          + message.getClass().getName());              }          } catch (JMSException e) {              System.out.println("JMSException in onMessage(): " + e.toString());          } catch (Throwable t) {              System.out.println("Exception in onMessage():" + t.getMessage());          }      }  } |

|  |
| --- |
| package jms.activemq.myexample.spring;    import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    public class SpringJmsTestMain {        public static void main(String[] args) throws InterruptedException {            ApplicationContext context = new ClassPathXmlApplicationContext(                  new String[] { "SpringJms/SpringJms.xml" });            SpringPublisher publisher = (SpringPublisher) context                  .getBean("springPublisher");          publisher.start();      }  } |
| package jms.activemq.myexample.spring;    import javax.jms.Destination;    import org.springframework.jms.core.JmsTemplate;    public class SpringPublisher {      /\*\*       \* Jms模板       \*/      private JmsTemplate template;        /\*\*       \* Topic       \*/      private Destination topic;        public JmsTemplate getTemplate() {          return template;      }        public void setTemplate(JmsTemplate template) {          this.template = template;      }        public Destination getTopic() {          return topic;      }        public void setTopic(Destination topic) {          this.topic = topic;      }        /\*\*       \* Start       \*       \* @throws InterruptedException       \*/      public void start() throws InterruptedException {            int messageCount = 10;            while ((--messageCount) > 0) {              sendMessage(messageCount);              Thread.sleep(1000);          }      }        /\*\*       \* 消息发送       \*/      protected void sendMessage(int msgNO) {            this.template.send(this.topic, new MyMessageCreator(msgNO));      }  } |

[复制代码](javascript:void(0);)

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">  
  
  
  
 <!-- jms 连接工厂 -->  
 <bean id="jmsFactory" class="org.apache.activemq.ActiveMQConnectionFactory">  
 <property name="brokerURL">  
 <value>tcp://localhost:61616</value>  
 </property>  
 </bean>  
  
  
 <!-- jms 连接池 -->  
   
 <!--   
 <bean id="pooledJmsFactory" class="org.apache.activemq.pool.PooledConnectionFactory">  
 <property name="connectionFactory">  
 <ref local="jmsFactory" />  
 </property>  
 </bean>  
 -->  
  
 <!-- jms Topic -->  
 <bean id="myTopic" class="org.apache.activemq.command.ActiveMQTopic"  
 autowire="constructor">  
 <constructor-arg value="STOCKS.JAVA" />  
 </bean>  
  
  
 <!-- 消息监听器 -->  
 <bean id="myTextListener" class="jms.activemq.myexample.spring.TextListener">  
 </bean>  
  
  
 <!-- jms Consumer -->  
 <bean id="javaConsumer"  
 class="org.springframework.jms.listener.DefaultMessageListenerContainer">  
 <property name="connectionFactory" ref="jmsFactory" />  
 <property name="destination" ref="myTopic" />  
 <property name="messageListener" ref="myTextListener" />  
 </bean>  
  
 <!-- jms 模板 -->  
 <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">  
 <property name="connectionFactory">  
 <ref local="jmsFactory" />  
 </property>  
 </bean>  
  
  
 <!-- 消息发布器 -->  
 <bean id="springPublisher" class="jms.activemq.myexample.spring.SpringPublisher">  
 <property name="template">  
 <ref local="jmsTemplate" />  
 </property>  
 <property name="topic">  
 <ref local="myTopic" />  
 </property>  
 </bean>  
</beans>

### [Spring+JMS例子](http://www.blogjava.net/wangbing/archive/2010/03/31/317093.html)

第一: 在applicationContext.xml

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>    
 <bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">  
   <property name="brokerURL">  
     <value>tcp://192.168.3.48:61616</value>  
   </property>  
    
 </bean>  
   
 <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">  
    <property name="connectionFactory">  
      <ref bean="connectionFactory"/>  
    </property>  
 </bean>  
   
 <bean id="destination" class="org.apache.activemq.command.ActiveMQQueue">  
   <constructor-arg index="0">  
     <value>HelloWorldQueue</value>  
   </constructor-arg>  
 </bean>  
</beans>

2.写发送方

package ch13.JMS;

import javax.jms.Destination;  
import javax.jms.JMSException;  
import javax.jms.Message;  
import javax.jms.Session;

import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import org.springframework.jms.core.JmsTemplate;  
import org.springframework.jms.core.MessageCreator;

public class HelloWorldSender  
{  
 public static void main(String args[]) throws Exception  
 {  
  ApplicationContext context = new ClassPathXmlApplicationContext(new String[] { "ch13/JMS/applicationContext.xml" });  
  JmsTemplate jmsTemplate = (JmsTemplate) context.getBean("jmsTemplate");  
  Destination destination = (Destination) context.getBean("destination");  
  jmsTemplate.send  
  (  
   destination, new MessageCreator()  
   {  
    public Message createMessage(Session session) throws JMSException  
    {  
     return session.createTextMessage("大家好这个是测试！");  
    }  
   }  
  );  
 }

}  
3.写接收方

package ch13.JMS;

import javax.jms.Destination;  
import javax.jms.TextMessage;

import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import org.springframework.jms.core.JmsTemplate;

public class HelloWorldReciver  
{

 public static void main(String args[]) throws Exception  
 {  
  ApplicationContext context = new ClassPathXmlApplicationContext(new String[] { "ch13/JMS/applicationContext.xml" });  
  JmsTemplate jmsTemplate = (JmsTemplate) context.getBean("jmsTemplate");  
  Destination destination = (Destination) context.getBean("destination");  
  System.out.println("will wait:" + jmsTemplate.getReceiveTimeout()+ " seconds for message");  
  TextMessage msg = (TextMessage) jmsTemplate.receive(destination);  
  System.out.println("reviced msg is:" + msg.getText());  
 }

}

4.启动activemq中bin 下的activemq.bat

5.先运行 HelloWorldSender

6.再运行 HelloWorld

7.结果:

will wait:-1 seconds for message  
reviced msg is:大家好 这个是测试！

## Spring IOC

1、获取Bean的方法

ApplicationContext ctx = WebApplicationContextUtils.*getWebApplicationContext*(

request.getSession().getServletContext() );

TestService testService = (TestService)ctx.getBean("testService");

testService.sayHello();

# JSP

JSP中动作与指令的区别

指令：  
 <%@include tile=""%>  
 该指令只是静态包含文件，不能为一个动态的地址，而且不带任何参数，服务器将对包含近来的文件进行编译。  
  
动作：  
 <JSP:include page="">  
 可以包含静态或动态文件，包含静态文件时服务器不对包含的文件编译，而在包含动态文件时服务器会编译包含动态文件，会自动判断包含文件是动态还是静态。  
 与include指令不同，该动作元素在JSP页面被用户请求时，才将指定文件插入服务器的响应输出中。