# 1. WEB-INF目录

■WEB-INF

WEB-INF是Java的WEB应用的**安全目录**。所谓安全就是客户端无法访问，只有服务端可以访问的目录。

如果想在页面中直接访问其中的文件，必须通过 web.xml 文件对要访问的文件进行相应映射才能访问。

■META-INF

相当于一个信息包，目录中的文件和目录获得Java平台的认可与解释，用来**配置应用程序、扩展程序、类加载器和服务manifest.mf文件**，在用jar打包时自动生成。

# 2.SpringMVC实现 如何通过TOMCAT访问WEB-INF下的JSP文件

楼主是maven项目，平时若将jsp文件放入webapp(未放入WEB-INF下)，tomcat访问时只需http://127.0.0.1:8080/myHadoop/login.jsp

即http://主机地址：端口号/项目名/文件名.jsp   即可访问

然而，若将jsp文件放在WEB-INF目录下，则访问无法通过地址栏的输入进行访问因此本文旨在介绍如何用springMVC实现

访问WEB-INF下的JSP文件

springMVC的导包什么的就不细说了

项目结构如下：

首先在spring-servlet.xml配置文件中进行内部视图解析器的配置：

这里配置的含义为加载视图解析器，如当发起login请求时添加前缀"/WEB-INF/views/" 后缀“.jsp”

配置完成后在web.xml文件中进行加载：

接着，写一个控制跳转的controller（重要！）

简单写一下代码：

@RequestMapping("/") 含义 请求会调用userLogin()方法 返回的 login字符串 会被上文提到的内部视图解析器

进行前后缀拼接成 /WEB-INF/views/login.jsp 最终返回登录页面

tomcat地址栏输入为：(即主机号：端口号/项目名) 即可

最后显示页面：

大功告成！！

# 3.springmvc-servlet.xml,applicationContext.xml 哪个先被加载

加载顺序：

web.xml(ContextLoaderListener) → applicationContext.xml

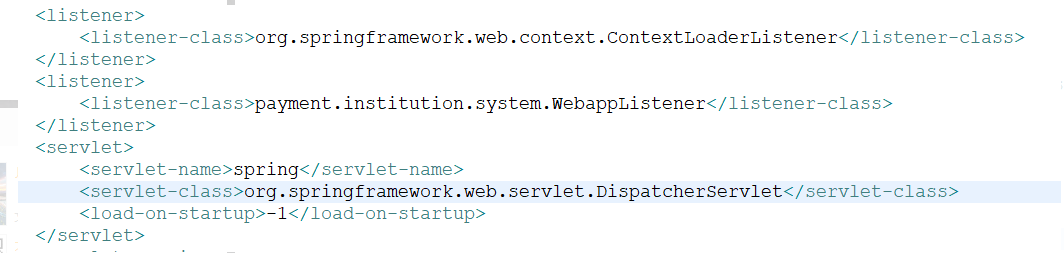
web.xml(DispatcherServlet)→xxx-servlet.xml

1、applicationContext.xml是随ContextLoaderListener的加载而执行的，而xxx-servlet.xml是随DispatcherServlet的加载而执行的,在web.xml中；

2、加载顺序是listener>filter>servlet；

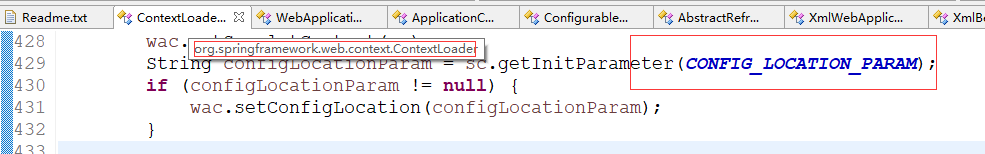
3、applicationContext.xml先加载！

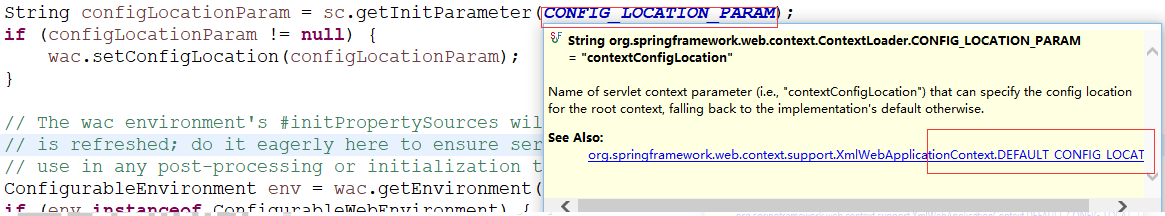
先看web.xml配置

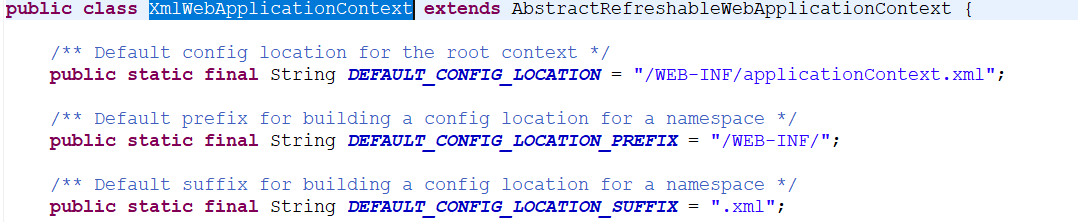


默认去找一个applicationContext.xml 的文件

为什么？

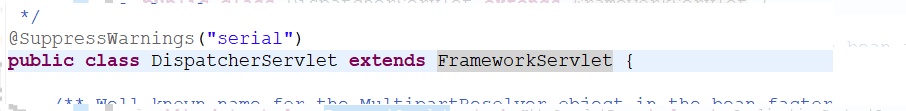


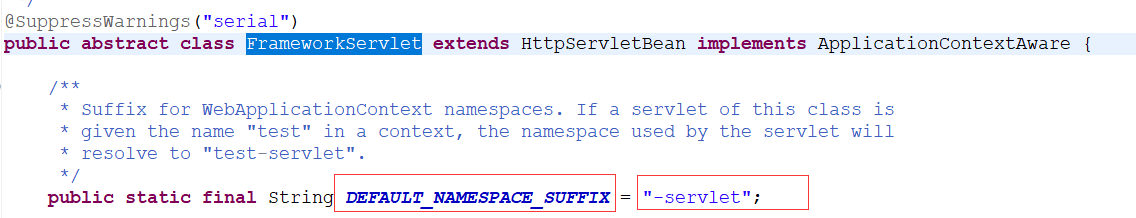


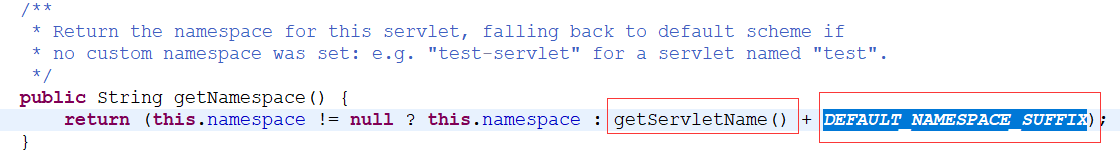


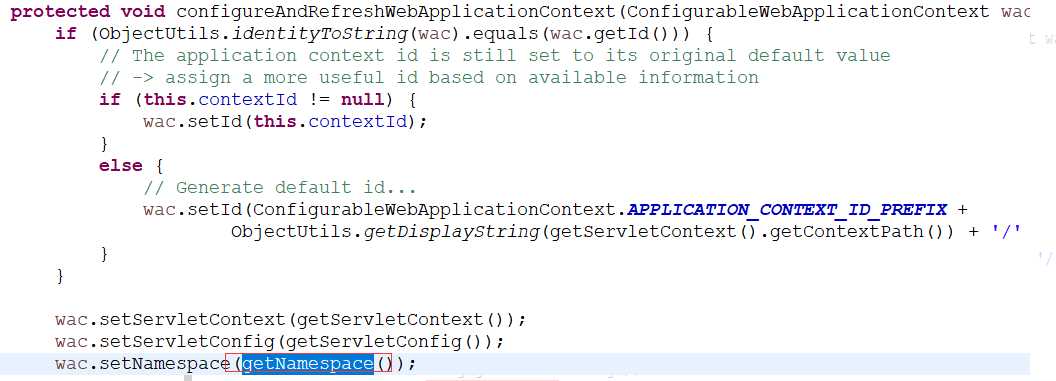
然后默认去找一个\*-servlet.xml的文件

为什么？

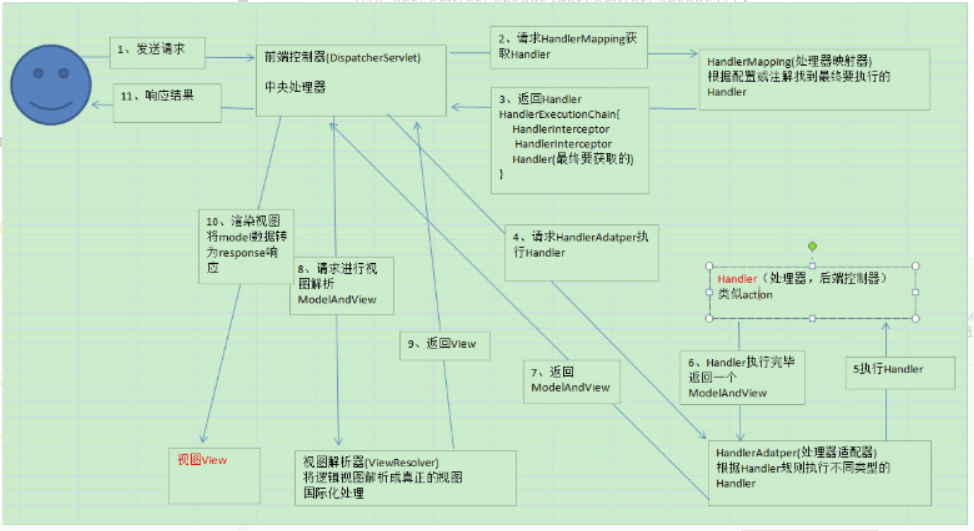






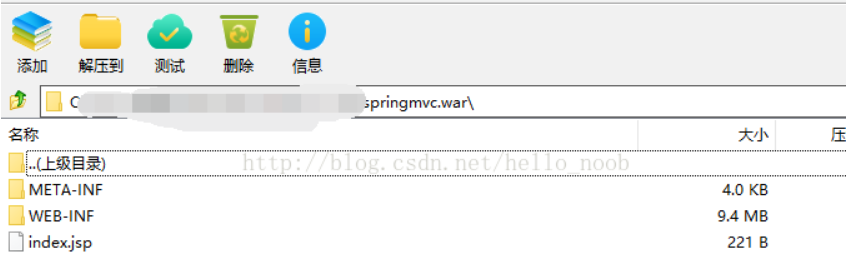


Spring MVC 原理

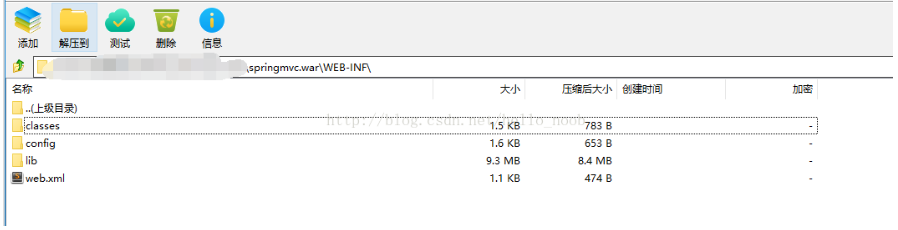


# 4. web.xml中的classpath指项目中的哪个路径

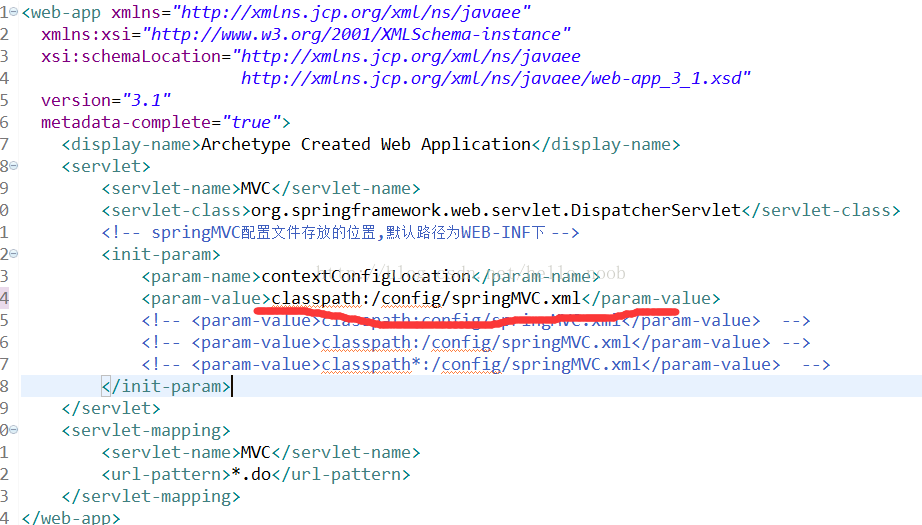
以springmvc项目为例,将其打包成springmvc.war文件



发现底下有3个文件，进去WEB-INF



我们通常说的classpath指的是WEB-INF下的classes文件(classpath对应的路径其实就是项目中的src文件中。)



因为config文件和classes文件是同级目录所以报错

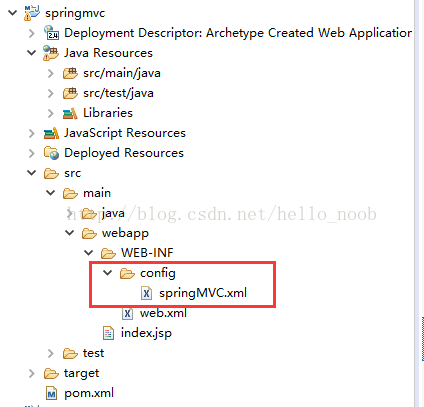
报错信息：parsing XML document from class path resource

[config/springMVC.xml]; nested exception is

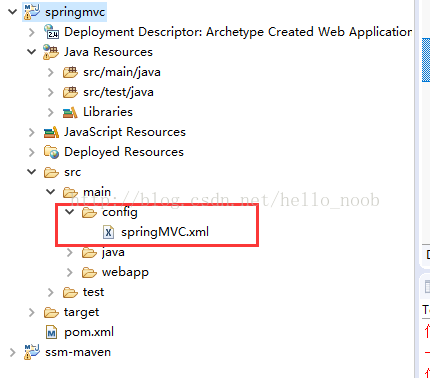
java.io.FileNotFoundException: class path resource

[config/springMVC.xml] cannot be opened because it does not exist

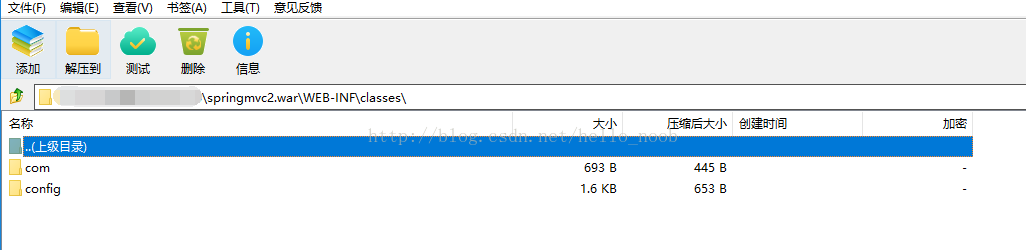
说是springMVC.xml没找到



而我的项目中config文件是放在webapp->WEB-INF下的,所以我要将config文件移到src->main文件夹下



发现在.war文件下config文件就在WEB-INF->classes文件下了



说明springMVC.xml配置文件可以被找到,也就是说classpath对应的路径其实就是项目中的src文件中。

后面测试：如果你将config文件放在src->main下也是可以被找到的，放在src->main->source也是可以

# 5. Spring MVC的web.xml配置详解

1、[spring](http://lib.csdn.net/base/javaee) 框架解决字符串编码问题：过滤器 **CharacterEncodingFilter**（filter-name）   
2、在web.xml配置监听器**ContextLoaderListener**（listener-class）   
ContextLoaderListener的作用就是启动Web容器时，自动装配ApplicationContext的配置信息。因为它实现了ServletContextListener这个接口，在web.xml配置这个监听器，启动容器时，就会默认执行它实现的方法。   
3、部署applicationContext的xml文件：**contextConfigLocation**（context-param下的param-name）   
4、DispatcherServlet是前置控制器，配置在web.xml文件中的。拦截匹配的请求，Servlet拦截匹配规则要自已定义，把拦截下来的请求，依据某某规则分发到目标Controller(我们写的Action)来处理。   
**DispatcherServlet**（servlet-name、servlet-class、init-param、param-name(contextConfigLocation)、param-value）   
在DispatcherServlet的初始化过程中，框架会在web应用的 WEB-INF文件夹下寻找名为[servlet-name]-servlet.xml 的配置文件，生成文件中定义的bean

Web.xml：

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

<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"

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

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd">

<!-- 在Spring框架中是如何解决从页面传来的字符串的编码问题的呢？

下面我们来看看Spring框架给我们提供过滤器CharacterEncodingFilter

这个过滤器就是针对于每次浏览器请求进行过滤的，然后再其之上添加了父类没有的功能即处理字符编码。

其中encoding用来设置编码格式，forceEncoding用来设置是否理会 request.getCharacterEncoding()方法，设置为true则强制覆盖之前的编码格式。-->

<filter>

<filter-name>characterEncodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>characterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 项目中使用Spring 时，applicationContext.xml配置文件中并没有BeanFactory，要想在业务层中的class 文件中直接引用Spring容器管理的bean可通过以下方式-->

<!--1、在web.xml配置监听器ContextLoaderListener-->

<!--ContextLoaderListener的作用就是启动Web容器时，自动装配ApplicationContext的配置信息。因为它实现了ServletContextListener这个接口，在web.xml配置这个监听器，启动容器时，就会默认执行它实现的方法。

在ContextLoaderListener中关联了ContextLoader这个类，所以整个加载配置过程由ContextLoader来完成。

它的API说明

第一段说明ContextLoader可以由 ContextLoaderListener和ContextLoaderServlet生成。

如果查看ContextLoaderServlet的API，可以看到它也关联了ContextLoader这个类而且它实现了HttpServlet这个接口

第二段，ContextLoader创建的是 XmlWebApplicationContext这样一个类，它实现的接口是WebApplicationContext->ConfigurableWebApplicationContext->ApplicationContext->

BeanFactory这样一来spring中的所有bean都由这个类来创建

IUploaddatafileManager uploadmanager = (IUploaddatafileManager) ContextLoaderListener.getCurrentWebApplicationContext().getBean("uploadManager");

-->

<listener>

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

</listener>

<!--2、部署applicationContext的xml文件-->

<!--如果在web.xml中不写任何参数配置信息，默认的路径是"/WEB-INF/applicationContext.xml，

在WEB-INF目录下创建的xml文件的名称必须是applicationContext.xml。

如果是要自定义文件名可以在web.xml里加入contextConfigLocation这个context参数：

在<param-value> </param-value>里指定相应的xml文件名，如果有多个xml文件，可以写在一起并以“,”号分隔。

也可以这样applicationContext-\*.xml采用通配符，比如这那个目录下有applicationContext-ibatis-base.xml，

applicationContext-action.xml，applicationContext-ibatis-dao.xml等文件，都会一同被载入。

在ContextLoaderListener中关联了ContextLoader这个类，所以整个加载配置过程由ContextLoader来完成。-->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/applicationContext.xml</param-value>

</context-param>

<!--如果你的DispatcherServlet拦截"/"，为了实现REST风格，拦截了所有的请求，那么同时对\*.js,\*.jpg等静态文件的访问也就被拦截了。-->

<!--方案一：激活Tomcat的defaultServlet来处理静态文件-->

<!--要写在DispatcherServlet的前面， 让 defaultServlet先拦截请求，这样请求就不会进入Spring了，我想性能是最好的吧。-->

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.css</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.swf</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.gif</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.jpg</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.png</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.js</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.html</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.xml</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.json</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.map</url-pattern>

</servlet-mapping>

<!--使用Spring MVC,配置DispatcherServlet是第一步。DispatcherServlet是一个Servlet,,所以可以配置多个DispatcherServlet-->

<!--DispatcherServlet是前置控制器，配置在web.xml文件中的。拦截匹配的请求，Servlet拦截匹配规则要自已定义，把拦截下来的请求，依据某某规则分发到目标Controller(我们写的Action)来处理。-->

<servlet>

<servlet-name>DispatcherServlet</servlet-name><!--在DispatcherServlet的初始化过程中，框架会在web应用的 WEB-INF文件夹下寻找名为[servlet-name]-servlet.xml 的配置文件，生成文件中定义的bean。-->

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

<!--指明了配置文件的文件名，不使用默认配置文件名，而使用dispatcher-servlet.xml配置文件。-->

<init-param>

<param-name>contextConfigLocation</param-name>

<!--其中<param-value>\*\*.xml</param-value> 这里可以使用多种写法-->

<!--1、不写,使用默认值:/WEB-INF/<servlet-name>-servlet.xml-->

<!--2、<param-value>/WEB-INF/classes/dispatcher-servlet.xml</param-value>-->

<!--3、<param-value>classpath\*:dispatcher-servlet.xml</param-value>-->

<!--4、多个值用逗号分隔-->

<param-value>classpath:spring/dispatcher-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup><!--是启动顺序，让这个Servlet随Servletp容器一起启动。-->

</servlet>

<servlet-mapping>

<!--这个Servlet的名字是dispatcher，可以有多个DispatcherServlet，是通过名字来区分的。每一个DispatcherServlet有自己的WebApplicationContext上下文对象。同时保存的ServletContext中和Request对象中.-->

<!--ApplicationContext是Spring的核心，Context我们通常解释为上下文环境，我想用“容器”来表述它更容易理解一些，ApplicationContext则是“应用的容器”了:P，Spring把Bean放在这个容器中，在需要的时候，用getBean方法取出-->

<servlet-name>DispatcherServlet</servlet-name>

<!--Servlet拦截匹配规则可以自已定义，当映射为@RequestMapping("/user/add")时，为例,拦截哪种URL合适？-->

<!--1、拦截\*.do、\*.htm， 例如：/user/add.do,这是最传统的方式，最简单也最实用。不会导致静态文件（jpg,js,css）被拦截。-->

<!--2、拦截/，例如：/user/add,可以实现现在很流行的REST风格。很多互联网类型的应用很喜欢这种风格的URL。弊端：会导致静态文件（jpg,js,css）被拦截后不能正常显示。 -->

<url-pattern>/</url-pattern> <!--会拦截URL中带“/”的请求。-->

</servlet-mapping>

<welcome-file-list><!--指定欢迎页面-->

<welcome-file>login.html</welcome-file>

</welcome-file-list>

<error-page> <!--当系统出现404错误，跳转到页面nopage.html-->

<error-code>404</error-code>

<location>/nopage.html</location>

</error-page>

<error-page> <!--当系统出现java.lang.NullPointerException，跳转到页面error.html-->

<exception-type>java.lang.NullPointerException</exception-type>

<location>/error.html</location>

</error-page>

<session-config><!--会话超时配置，单位分钟-->

<session-timeout>360</session-timeout>

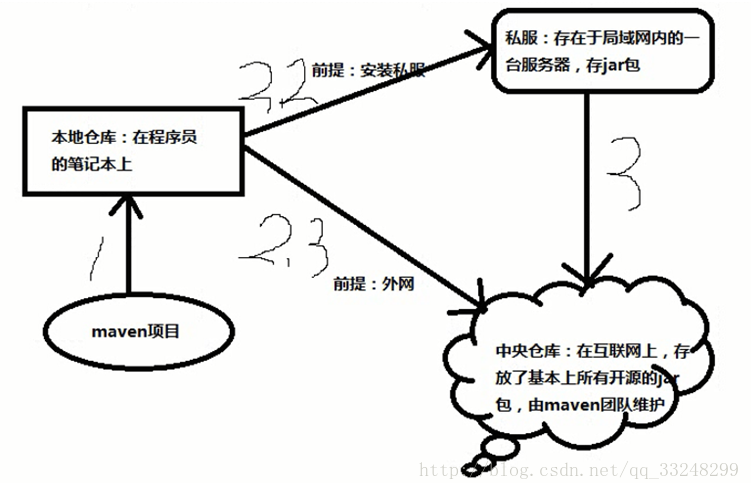
</session-config>

</web-app>

# 6. maven本地仓库及项目标准目录结构

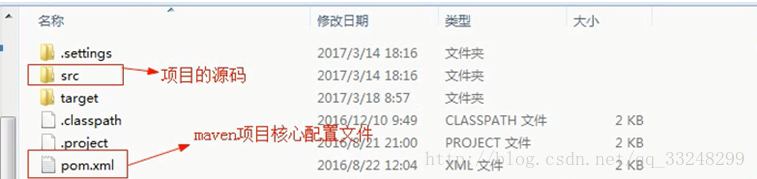
## 6.1.maven仓库类型

本地仓库 私服 中央仓库



## 6.2maven项目标准目录结构

### 1. maven文件夹目录



**target:**

src源码编译完之后存到这个目录下

**src下面**

main:项目运行需要  
java:主要源码

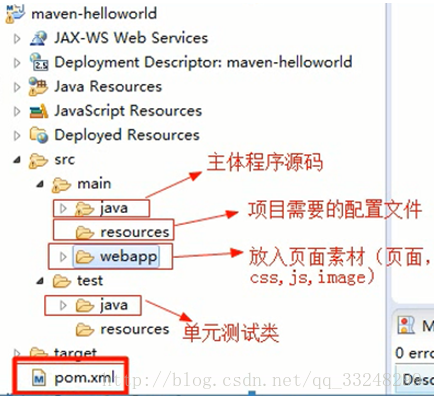
resources:配置文件

webapp:类似于动态web项目的webContent 页面素材 jsp

WEB-INF index.jsp

test:作单元测试  
java resources:配置文件 (一般不会存)

### 2.对项目中文件进行细分



## 6.3 Introduction to the Standard Directory Layout

来自官网：

<http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>

Having a common directory layout would allow for users familiar with one Maven project to immediately feel at home in another Maven project. The advantages are analogous to adopting a site-wide look-and-feel.

The next section documents the directory layout expected by Maven and the directory layout created by Maven. Please try to conform to this structure as much as possible; however, if you can't these settings can be overridden via the project descriptor.

|  |  |
| --- | --- |
| src/main/java | Application/Library sources |
| src/main/resources | Application/Library resources |
| src/main/filters | Resource filter files |
| src/main/webapp | Web application sources |
| src/test/java | Test sources |
| src/test/resources | Test resources |
| src/test/filters | Test resource filter files |
| src/it | Integration Tests (primarily for plugins) |
| src/assembly | Assembly descriptors |
| src/site | Site |
| LICENSE.txt | Project's license |
| NOTICE.txt | Notices and attributions required by libraries that the project depends on |
| README.txt | Project's readme |

At the top level, files descriptive of the project: a pom.xml file. In addition, there are textual documents meant for the user to be able to read immediately on receiving the source: README.txt, LICENSE.txt, etc.

There are just two subdirectories of this structure: src and target. The only other directories that would be expected here are metadata like CVS, .git or .svn, and any subprojects in a multiproject build (each of which would be laid out as above).

The target directory is used to house all output of the build.

The src directory contains all of the source material for building the project, its site and so on. It contains a subdirectory for each type: main for the main build artifact, test for the unit test code and resources, site and so on.

Within artifact producing source directories (ie. main and test), there is one directory for the language java (under which the normal package hierarchy exists), and one for resources (the structure which is copied to the target classpath given the default resource definition).

If there are other contributing sources to the artifact build, they would be under other subdirectories: for example src/main/antlr would contain Antlr grammar definition files.

# 7. Spring security

### 7.1创建maven项目

对于maven项目的创建，大家可以参考以下文章，这里不再赘述。[《maven 实战》笔记](http://blog.csdn.net/yin380697242/article/details/51786347)

pom.xml添加如下依赖:

<properties>

<spring.version>4.1.0.RELEASE</spring.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>4.1.0.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>4.1.0.RELEASE</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1-b02</version>

<type>jar</type>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

<scope>runtime</scope>

</dependency>

</dependencies>

### 7.2项目配置文件

1 spring 配置文件 applicationContext.xml

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

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

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

xmlns:c="http://www.springframework.org/schema/c" xmlns:cache="http://www.springframework.org/schema/cache"

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

xmlns:jdbc="http://www.springframework.org/schema/jdbc" xmlns:jee="http://www.springframework.org/schema/jee"

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

xmlns:p="http://www.springframework.org/schema/p" xmlns:task="http://www.springframework.org/schema/task"

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

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

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd

http://www.springframework.org/schema/cache http://www.springframework.org/schema/cache/spring-cache.xsd

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

http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc.xsd

http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd

http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang.xsd

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

http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd

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

</beans>

2 spring mvc  配置文件 springmvc-servlet.xml

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

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

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

xmlns:c="http://www.springframework.org/schema/c" xmlns:cache="http://www.springframework.org/schema/cache"

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

xmlns:jdbc="http://www.springframework.org/schema/jdbc" xmlns:jee="http://www.springframework.org/schema/jee"

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

xmlns:p="http://www.springframework.org/schema/p" xmlns:task="http://www.springframework.org/schema/task"

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

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

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd

http://www.springframework.org/schema/cache http://www.springframework.org/schema/cache/spring-cache.xsd

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

http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc.xsd

http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd

http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang.xsd

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

http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd

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

<!-- 注解驱动 -->

<mvc:annotation-driven />

<!-- 使Spring支持自动检测组件,自动扫描controller -->

<context:component-scan base-package="com.yin.myproject.security.controller" />

<!-- 定义视图解析器 -->

<bean id="viewResolver"

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/views/" />

<property name="suffix" value=".jsp"></property>

</bean>

</beans>

3 spring security配置文件 spring-security.xml

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

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

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

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

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

<!-- spring配置文件位置 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

<!-- spring核心监听器 -->

<listener>

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

</listener>

<!-- springMVC分发处理器 -->

<servlet>

<servlet-name>spring</servlet-name>

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

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

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

</servlet-mapping>

<!-- spring security -->

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy

</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring-security.xml</param-value>

</context-param>

<!-- 配置默认过滤器 作用是像类似于css,jpg,js等的文件不被springMVC核心过滤器拦截 -->

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.css</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.html</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.gif</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.jpg</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.js</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.png</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.json</url-pattern>

</servlet-mapping>

<!-- 配置默认过滤器 作用是像类似于css,jpg,js等的文件不被springMVC核心处理器拦截 -->

<!-- 编码过滤器，解决中文乱码 -->

<filter>

<filter-name>SpringEncoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>SpringEncoding</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- session超时定义,单位为分钟 -->

<session-config>

<session-timeout>30</session-timeout>

</session-config>

<!-- 出错页面定义 -->

<error-page>

<exception-type>java.lang.Throwable</exception-type>

<location>/common/500.jsp</location>

</error-page>

<error-page>

<error-code>500</error-code>

<location>/common/500.jsp</location>

</error-page>

<error-page>

<error-code>404</error-code>

<location>/common/404.jsp</location>

</error-page>

<error-page>

<error-code>403</error-code>

<location>/common/403.jsp</location>

</error-page>

<!-- welcome page -->

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

# 8.Eclipse中修改Web项目的URL访问路径

1 tomcat启动的URL

在自己的项目名称上，点右键，properties，web project settings，然后修改右边的context root，一般默认是项目名，但是，如果是复制别的项目工程，还是原来复制过来的项目名

2 maven工程：项目名称就是artifactId，即f2pdj，在哪里启动，就是哪里，不管是不是父子工程。还有，改的时候要小心，别把  <parent>标签里面的artifactId改错了，那就直接报错了

<groupId>com.thinkgem.jeesite</groupId>

<!-- <artifactId>jeesite</artifactId> -->

<artifactId>f2pdj</artifactId>

<version>1.2.6</version>

<packaging>war</packaging>

---------------------------------------------------------------------------------------

下面是copy别人的

背景

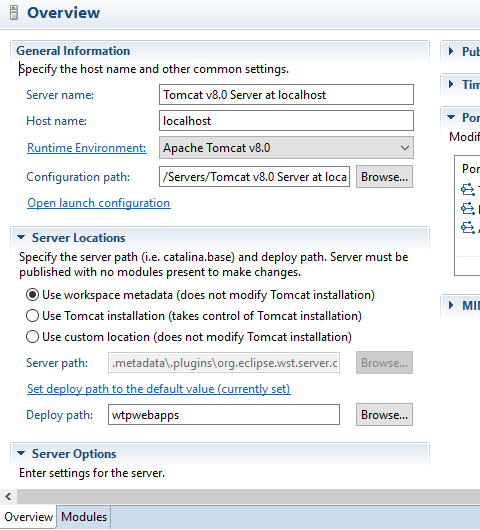
访问路径，也就是指在浏览器中访问该web系统时的根路径，比如http://localhost:8080/xxxx/index.jsp  这里的xxxx，也就是request.getContextPath()得到的值。

我们在做系统开发的时候，在本地工作区可能会有同一个系统的多个版本存在，比如上面所说的xxxx系统，这里可能会有xxxx1、xxxx1\_1、xxxx1\_2代表该系统的1.0 、1.1、 1.2版本，对应的工程名字默认就是系统的访问路径，但是，我们可能会希望这些项目能够有一个统一的访问路径xxxx，这样向服务器部署的时候不管部署的是哪个项目，访问路径都是统一的。

方法

在服务器部署的时候，比如tomcat，我们只需要修改对应的context.xml文件即可，<Context path="">属性对应的就是访问路径。

在Eclipse中，双击Server视图的Tomcat实例，如下：



我们可以看到在Server locations中有3个选项，在上图中选择的是第一个选项，这由Eclipse对tomcat的配置进行管理，对应的虚拟配置目录在图片中的Server Path一栏中。

我们可以去直接修改那个路径下的conf/server.xml文件，或者点击下面的Modules，在里面的web modules里进行修改各个项目的Path。其效果最终都是修改server.xml文件里Context的path内容