## 简介以及原理

[Swagger](http://swagger.io/) 是一个规范和完整的框架，用于生成、描述、调用和可视化 RESTful 风格的 Web 服务。总体目标是使客户端和文件系统作为服务器以同样的速度来更新。文件的方法，参数和模型紧密集成到服务器端的代码，允许API来始终保持同步。Swagger 让部署管理和使用功能强大的API从未如此简单。

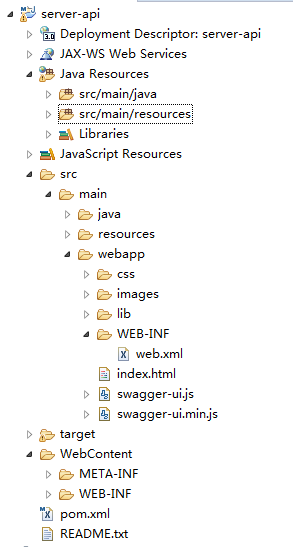
**原理：**

**后台：后端部分与[Java](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "_blank)集成，后最终会产生一个json文件。**

**前台：前台部分就是html、css、js文件，js利用后台产生的json文件构造api；**

## 一、新建工程

我们新建一个Maven工程，并添加Web Facet，工程结构如下图所示：



## 二、添加Maven依赖

**在pom文件中添加：**

<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/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>cn.fansunion</groupId>

<artifactId>server-api</artifactId>

<version>1.0</version>

<packaging>war</packaging>

<properties>

<java-version>1.7</java-version>

<org.springframework-version>4.1.6.RELEASE</org.springframework-version>

<maven.compiler.encoding>UTF-8</maven.compiler.encoding>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>javax.servlet</groupId>

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

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.1.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.1.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>4.1.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>4.1.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.1.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

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

<version>4.1.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>4.1.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>4.1.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-oxm</artifactId>

<version>4.1.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>4.1.6.RELEASE</version>

<scope>test</scope>

</dependency>

<!-- swagger-mvc -->

<dependency>

<groupId>com.mangofactory</groupId>

<artifactId>swagger-springmvc</artifactId>

<version>1.0.2</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.6.6</version>

</dependency>

<!-- swagger-mvc -->

<!-- slfj4 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.7</version>

</dependency>

<!-- slfj4 -->

</dependencies>

<build>

<!-- 指定maven编译的jdk版本,如果不指定,maven3默认用jdk 1.5 maven2默认用jdk1.3 -->

<plugins>

<plugin>

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

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

<encoding>UTF8</encoding>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<warSourceDirectory>src/main/webapp</warSourceDirectory>

</configuration>

</plugin>

<!-- <plugin>

<groupId>org.eclipse.jetty</groupId>

<artifactId>jetty-maven-plugin</artifactId>

<version>9.3.0.M1</version>

<configuration>

<scanIntervalSeconds>3</scanIntervalSeconds>

<webApp>

<contextPath>/server-api</contextPath>

</webApp>

<connectors>

work around file locking on windows

<connector implementation="org.mortbay.jetty.bio.SocketConnector">

<port>8080</port>this connector defaults to 1300 for some reason

</connector>

</connectors>

</configuration>

</plugin> -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>tomcat-maven-plugin</artifactId>

<configuration>

<server>tomcat</server>

<url>http://localhost/manager/text</url>

<path>/server-api</path>

</configuration>

</plugin>

</plugins>

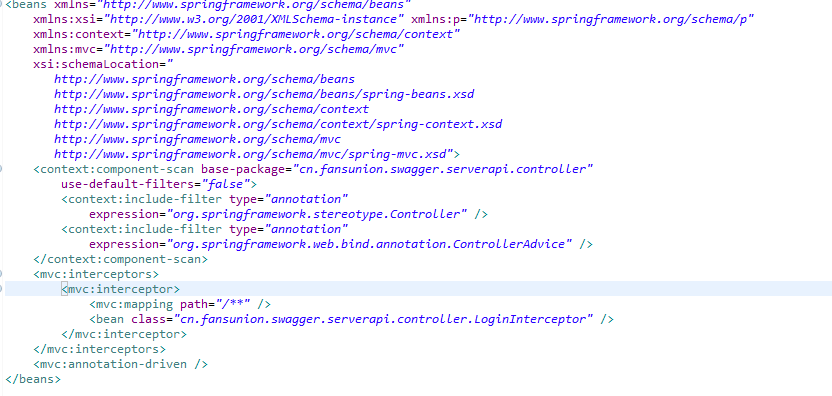
<finalName>server-api</finalName>

</build>

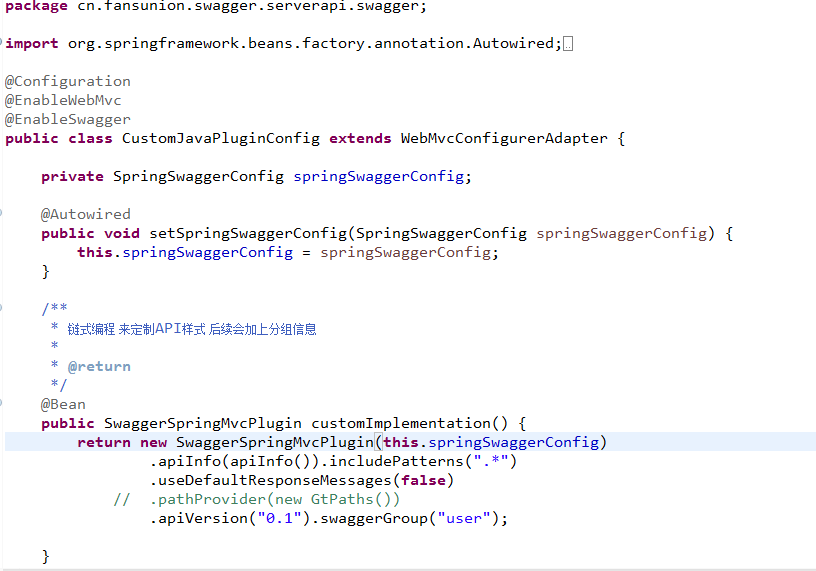
</project>

## 三、添加配置

在工程中的resources文件夹下新建一个spring的文件夹，并新建一个spring-mvc.xml l的spring mvc配置文件，添加如下内容：



**添加一个SwaggerConfig定制类，如果不配置此类，则swagger会扫描 base-package包下的所有类，生成api文档，用于配置Swagger定制类的说明如下：**





## 四、新建接口类

新建一个接口类，如下：

/\*\*

\*

\* 用户管理

\*

\*/

@Api(value = "user", description = "用户管理", produces = MediaType.***APPLICATION\_JSON\_VALUE***)

@Controller

@RequestMapping("user")

**public** **class** UserController {

// 列出某个类目的所有规格

@ApiOperation(value = "获得用户列表", notes = "列表信息", httpMethod = "GET", produces = MediaType.***APPLICATION\_JSON\_VALUE***)

@ResponseBody

@RequestMapping(value = "list", method = RequestMethod.***GET***)

**public** Result<User> list(

@ApiParam(value = "分类ID", required = **true**) @RequestParam Long categoryId,

@ApiParam(value = "分类ID", required = **true**) @RequestParam Long categoryId2,

@ApiParam(value = "token", required = **true**) @RequestParam String token) {

Result<User> result = **new** Result<User>();

User user = **new** User();

user.setName("zhangsan");

user.setPassword("password");

user.setSex(1);

user.setToken("aastewetwewe97wewesf7w8");

result.setData(user);

**return** result;

}

@RequestMapping(value = "getUserById", method = RequestMethod.***GET***)

@ResponseBody

@ApiOperation(value = "根据id获得用户信息", notes = "用户信息", httpMethod = "GET", produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** User getUserById(

@ApiParam(value = "user ID", required = **true**) @RequestParam String id

){

User user = **new** User();

user.setName("lisi");

user.setPassword("password");

user.setSex(1);

user.setToken("aastewetwewe97wewesf7w8");

**return** user;

}

@ApiOperation(value = "添加用户", notes = "添加用户(用于数据同步)", httpMethod = "POST", produces = MediaType.***APPLICATION\_JSON\_VALUE***)

@ResponseBody

@RequestMapping(value = "add", method = RequestMethod.***POST***)

// @RequestBody只能有1个

// 使用了@RequestBody，不能在拦截器中，获得流中的数据，再json转换，拦截器中，也不清楚数据的类型，无法转换成java对象

// 只能手动调用方法

**public** Result<String> add(@RequestBody User user) {

String u = findUser(user);

System.***out***.println(u);

**return** **null**;

}

@ApiOperation(value = "update用户", notes = "update user", httpMethod = "POST", produces = MediaType.***APPLICATION\_JSON\_VALUE***)

@ResponseBody

@RequestMapping(value = "update", method = RequestMethod.***GET***)

**public** Result<String> update(User user) {

String u = findUser(user);

System.***out***.println(u);

**return** **null**;

}

**private** String findUser(User user) {

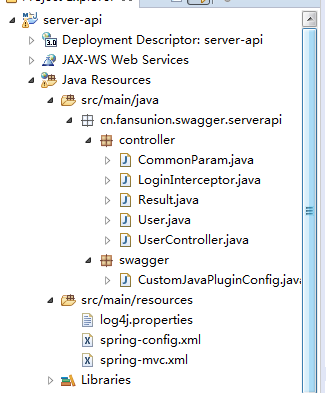
String token = user.getToken();

**return** token;

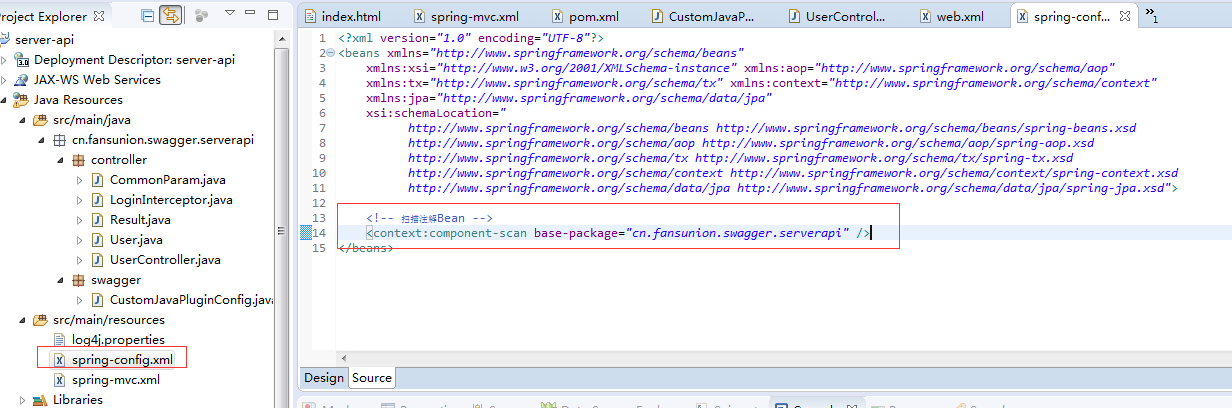
}

}

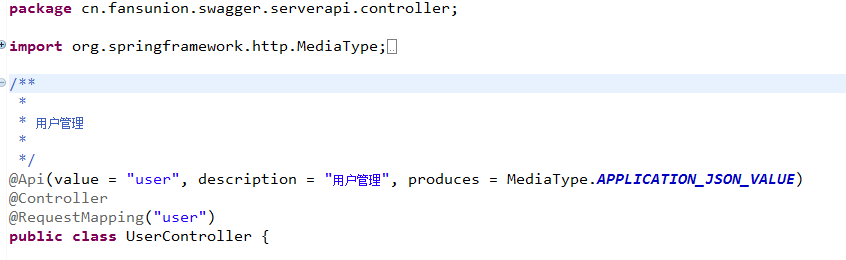
好，目前为止我们的代码已经编写完成，整个工程的目录结构如下：

s

为了让Swagger能够扫描Spring mvc中定义的Controller，我们需要在mvc的配置文件里面定义扫描的路径和相关的bean：



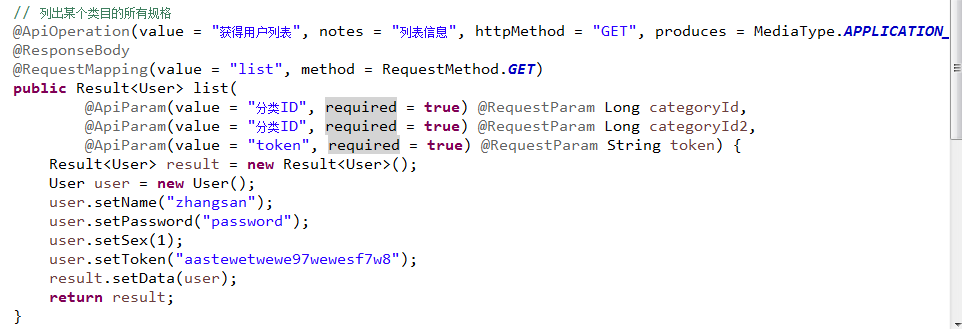
### Controller类注释



@Api(value = "user", description = "用户管理", produces = MediaType.APPLICATION\_JSON\_VALUE)

#### 类注释解释

**@API，用来描述Controller类的注释，value表示Controller的链接地址，访问此类的URL中需要加上/user，description表示描述，produces表示返回数据结构**



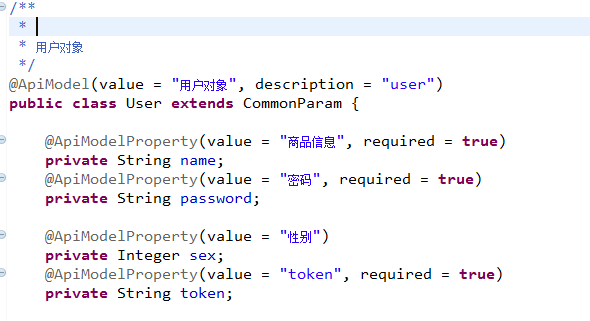
#### 方法注释解释

**@APIOperation，用来描述方法的注释，value表示访问此方法的链接地址，如在user类下的list方法，如/user/list访问即可，notes表示方法的描述，httpMethod表示请求方式，produces表示返回数据结构**

**@ApiParam，用来描述参数的注释，value表示参数名，required表示是否必填。**

**@RequestParam，用来表示参数类型。**

### 实体类注释



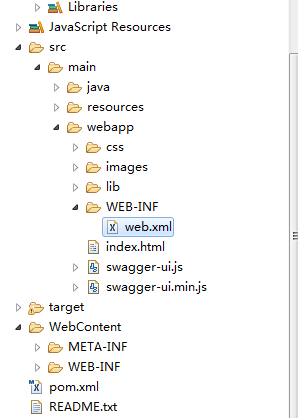
#### 实体类注释解释

@ApiModel，表示实体类注释，value表示对象名称，description表示描述

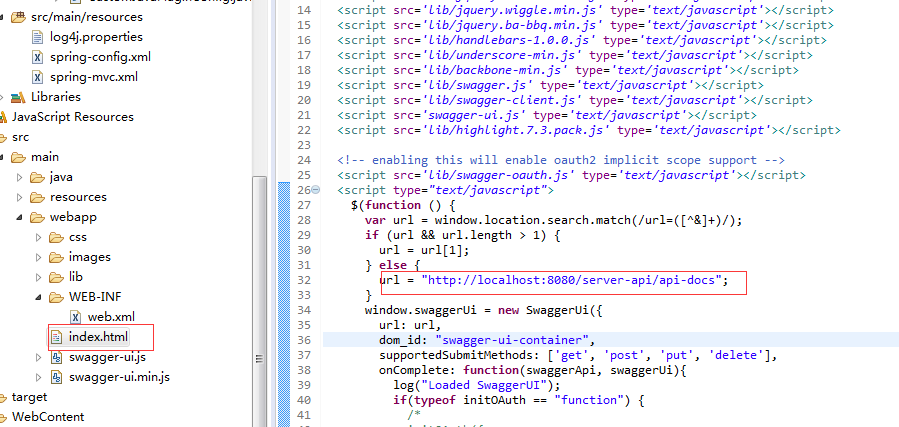
@ApiModeProperty，表示描述字段的注释，value表示字段名,required表示是否必填。

## 五、添加Swagger ui

在GitHub上下载[SwaggerUI](https://github.com/swagger-api/swagger-ui)项目，将dist下所有内容拷贝到本地项目server-api/webapp下面，结果目录如下图所示：



打开目录下的index.html文件，如图：



**将url修改成格式如[http://localhost:8080/{project-name}/api-docs](http://localhost:8080/%7bproject-name%7d/api-docs)的形式**

好，现在我们启动tomcat来看看效果：



## 六、常见swagger注解一览与使用

### 注解浏览

@Api : 在类上注解表名这个类是一个Swagger资源

@ApiImplicitParam 在API Operation中代表只有一个参数

@ApiImplicitParams  包装多个ApiImplicitParam对象

@ApiModel 关于Swagger模块提供额外的信息

@ApiModelProperty 添加和操作模型属性数据

@ApiOperation 描述一个操作或者通常针对特定路径的HTTP方法

@ApiParam  添加操作参数的额外元数据

@ApiResponse 描述一个操作可能的响应

@ApiResponses 包装多个ApiResponse对象

@Authorization声明对一个资源或者操作的授权

@AuthorizationScope 声明OAuth2授权范围

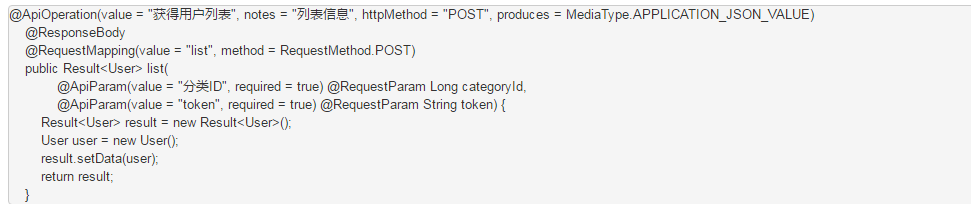
@ResponseHeader声明可以作为response的头部

详情请见：

<http://jakubstas.com/spring-jersey-swagger-create-documentation/>

### 注解使用

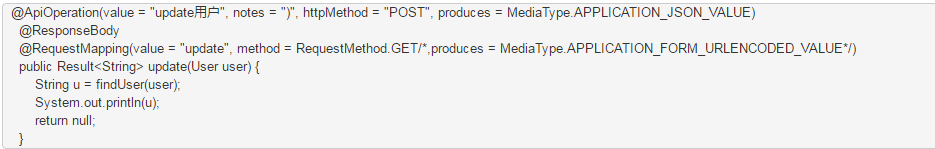
#### 例子1：



**@ApiParam(value = "token", required = true) @RequestParam String token**

**Web前端/移动端HTTP请求方式：直接把参数附带到URL后面，或者用AJAX方法，表单提交。**

#### 例子2：



当参数太多的时候，需要定义太多的参数，排版看起来很不舒服。

这个时候，可以使用对象来接收。

@ApiModel(value = "用户对象",description="user2")

public class User extends CommonParam{

}

Web前端/移动端HTTP请求方式：直接把参数附带到URL后面，或者用AJAX方法，表单提交。

这里面存在一个小问题，当后端用对象User来接收参数的时候，Swagger自带的工具是这样的：

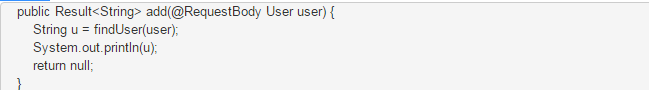


这种形式，并不是表单提交，或者把参数附加到URL的后面。

我们只能手动构造URL，附带参数去提交。

如果需要测试的话！

#### 例子3：



使用了@RequestBody，不能在拦截器中，获得流中的数据，再json转换，拦截器中，也不清楚数据的类型，无法转换成java对象

Web前端/移动端HTTP请求方式：必须把参数，放到request请求的body中去。

后端不能直接用request.getParam("token")这种。

获得request body中的数据，手动转换成目标数据。

    String charReader(HttpServletRequest request) throws IOException {

        BufferedReader br = request.getReader();

        String str, wholeStr = "";

        while ((str = br.readLine()) != null) {

            wholeStr += str;

        }

        return wholeStr;

    }

#### 经验总结：

**1.参数不多的时候，用例子1，用@ApiParam注解生成文档。**

**swagger可视化界面，可以直接设置参数，发送请求来测试**

**2.参数比较多的时候，用例子2，用对象来接收参数，在对象里针对每个字段，@ApiModelProperty注解生成文档。**

**swagger可视化界面，可以直接设置参数，但是无法接收到。**

**因此，推荐使用其它HTTP请求或POST模拟工具，发送请求，模拟测试。**

**不推荐例子3，不通用，局限性比较大。**

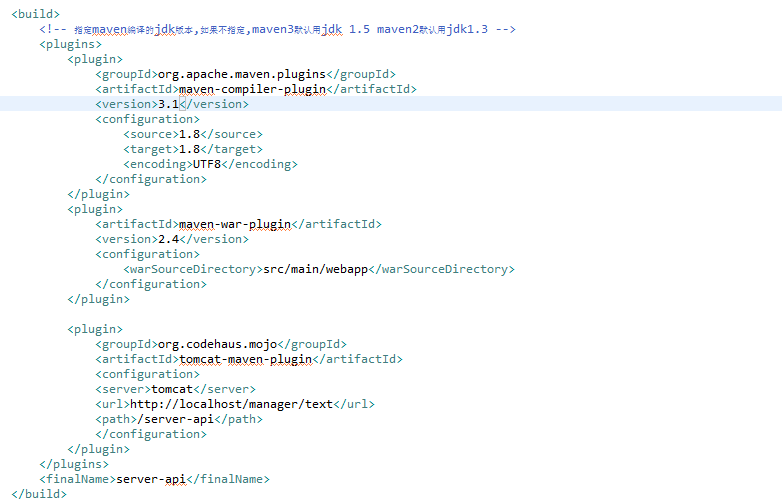
## 七、参考源码

**见：svn://210.51.17.141/svndata/eCommerce/ 下的 swagger-demo项目**

****

## 八、关于Swagger

1、Swagger内嵌了一个tomcat服务器，与maven整合时，只需在pom文件中配置tomcat的版本以及jdk等其他信息即可，如下：



其中plugin中的path表示的是访问的根目录，如上所配置，访问时对应

