# 今日大纲

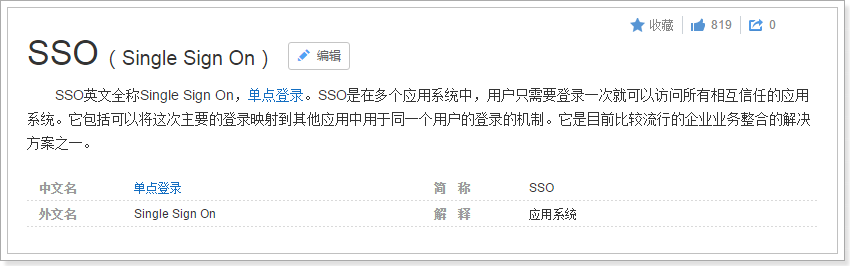
1. 实现单点登录系统
2. 基于单点登录系统实现，用户的注册和登录
3. 讲解订单系统

# 单点登录系统

## 为什么要使用单点登录系统？

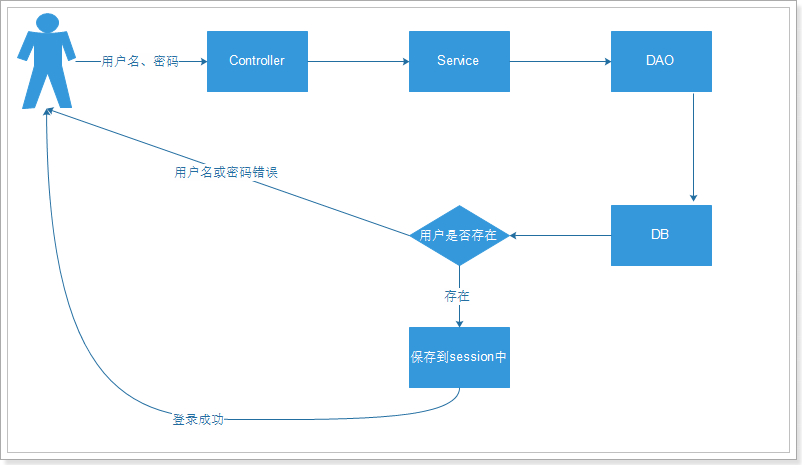
之前实现的登录和注册是在同一个tomcat内部完成，我们现在的系统架构是每一个系统都是由一个团队进行维护，每个系统都是单独部署运行一个单独的tomcat，所以，不能将用户的登录信息保存到session中（多个tomcat的session是不能共享的），所以我们需要一个单独的系统来维护用户的登录信息。

## 单点登录系统（SSO）

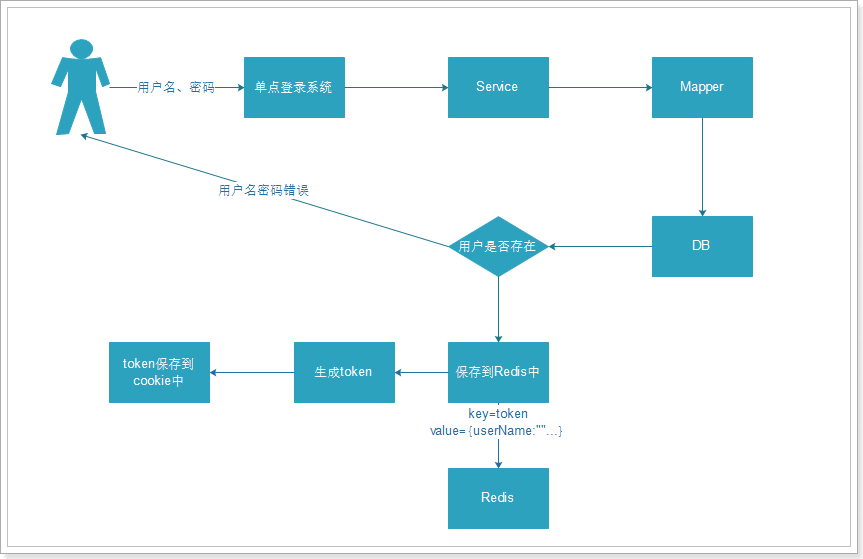


## 登录流程

### 之前的登录流程

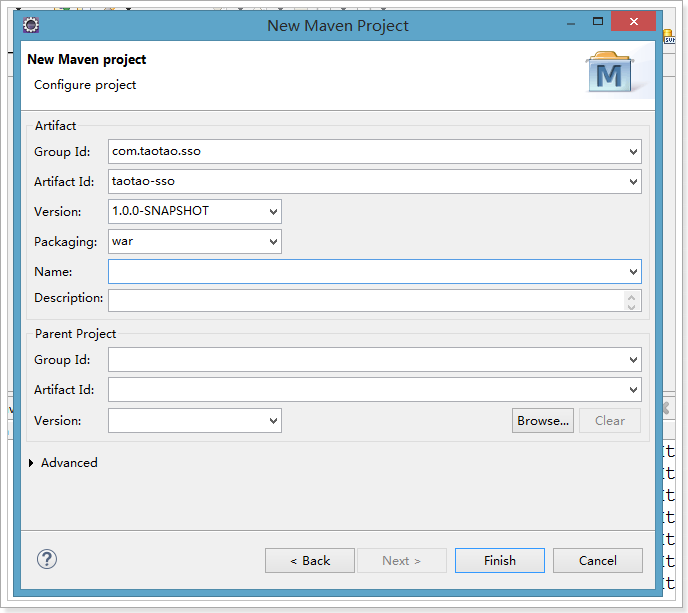


### 现在的登录流程



## 创建工程（taotao-sso）

### 创建工程



### 导入依赖

<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"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.taotao.parent</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao.sso</groupId>

<artifactId>taotao-sso</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>com.taotao.common</groupId>

<artifactId>taotao-common</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

</dependency>

<!-- Mybatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

</dependency>

<!-- 通用Mapper -->

<dependency>

<groupId>com.github.abel533</groupId>

<artifactId>mapper</artifactId>

</dependency>

<!-- MySql -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</dependency>

<!-- Jackson Json处理工具包 -->

<dependency>

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

<artifactId>jackson-databind</artifactId>

</dependency>

<!-- 连接池 -->

<dependency>

<groupId>com.jolbox</groupId>

<artifactId>bonecp-spring</artifactId>

</dependency>

<!-- JSP相关 -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<scope>provided</scope>

</dependency>

<!-- Apache工具组件 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

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

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

<configuration>

<port>8083</port>

<path>/</path>

</configuration>

</plugin>

</plugins>

</build>

</project>

### Web.xml

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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

id=*"WebApp\_ID"* version=*"2.5"*>

<display-name>taotao-sso</display-name>

<context-param>

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

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

</context-param>

<!--Spring的ApplicationContext 载入 -->

<listener>

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

</listener>

<!-- 编码过滤器，以UTF8编码 -->

<filter>

<filter-name>encodingFilter</filter-name>

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

<init-param>

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

<param-value>UTF8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

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

</filter-mapping>

<!-- 配置SpringMVC框架入口 -->

<servlet>

<servlet-name>taotao-sso</servlet-name>

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

<init-param>

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

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

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>taotao-sso</servlet-name>

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

</servlet-mapping>

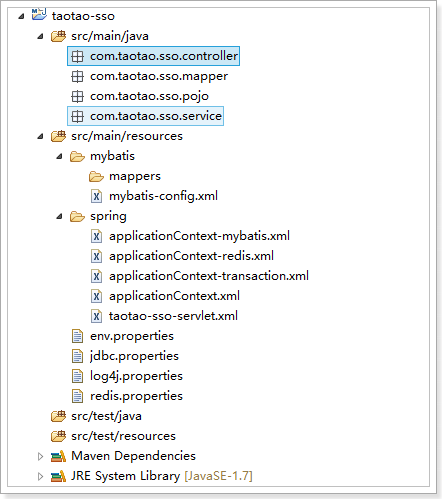
<welcome-file-list>

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

</welcome-file-list>

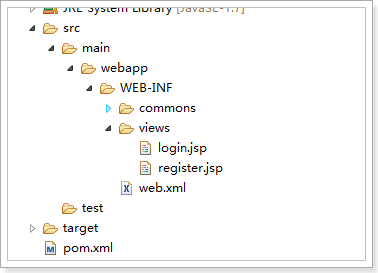
</web-app>

### SSM配置文件

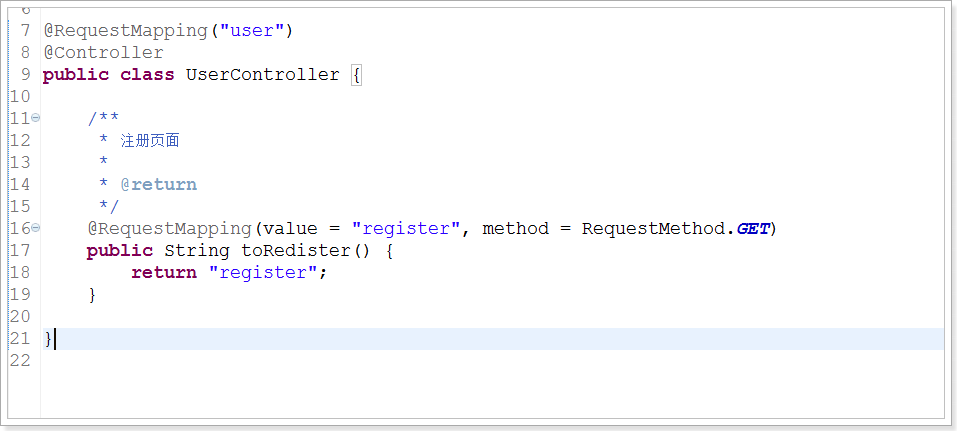


### 导入静态页面

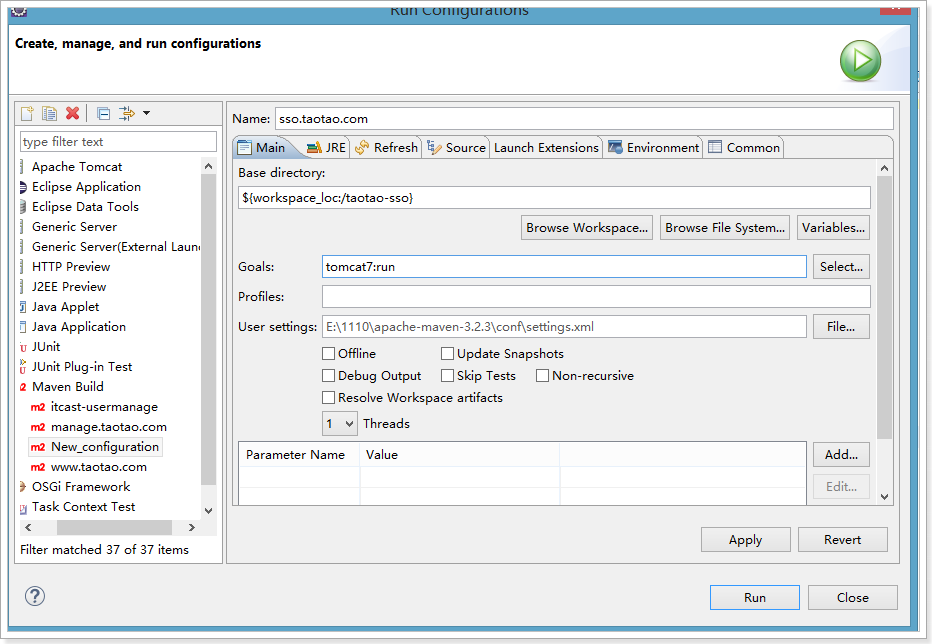
从taotao-web中拷贝页面：



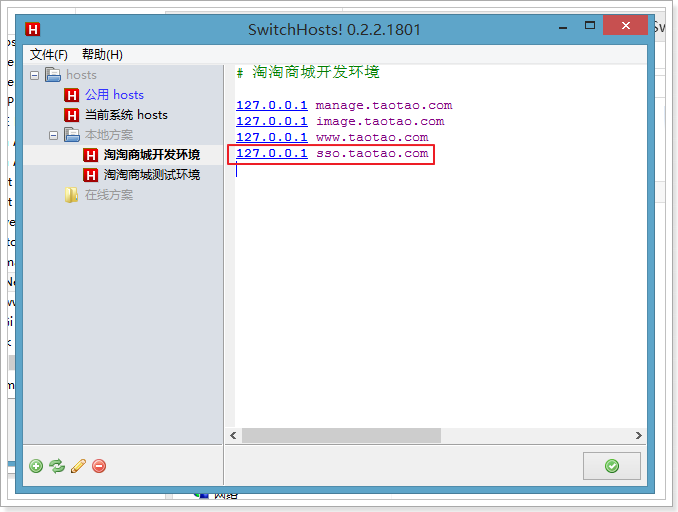
### 编写UserController



### 启动tomcat



### 配置hosts



### 配置nginx



### 测试



发现，JS、CSS没有加载。

## 解决静态资源加载的问题

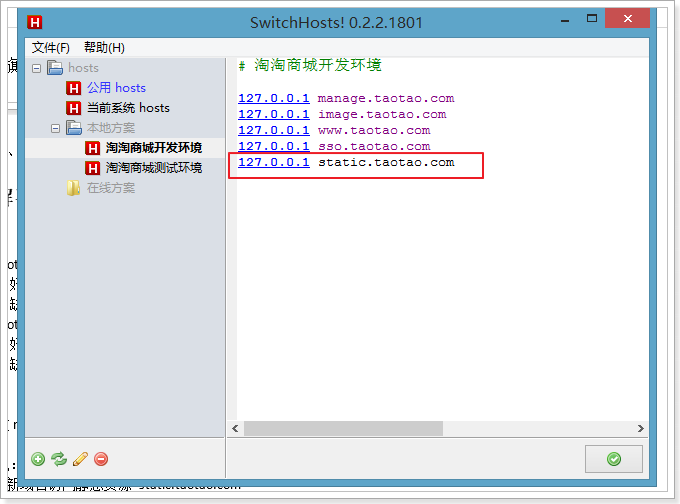
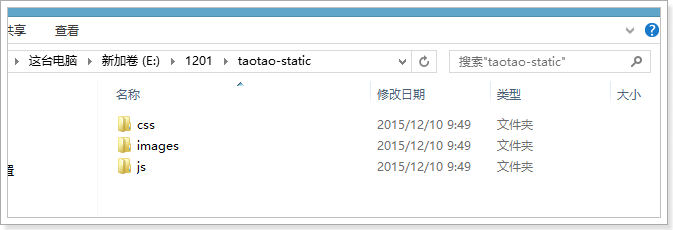
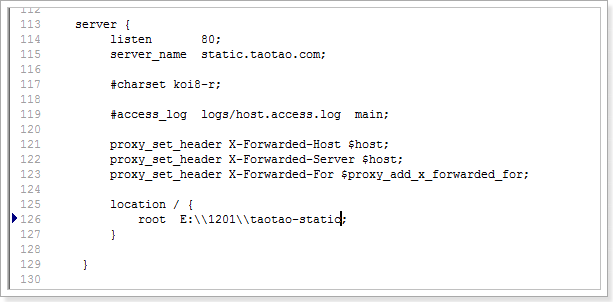
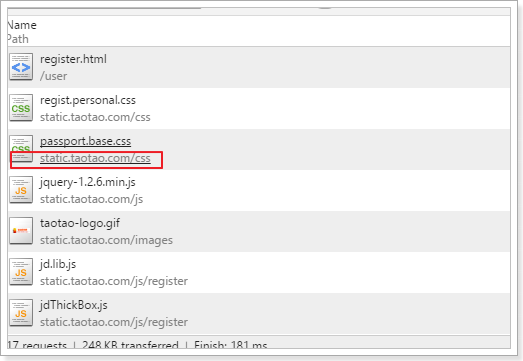
方案：

1. 将taotao-web中的js和css拷贝到taotao-sso中
   1. 好处：简单、方便
   2. 缺点：重复、对用户而言需要重复加载
2. 将taotao-sso中的引用指向[www.taotao.com](http://www.taotao.com)
   1. 好处：对用户而言只需要加载一次即可
   2. 缺点：修改页面

方案三：

需要通过nginx访问静态资源，JS、CSS、Image。

具体实现：

1. 使用新域名访问静态资源 static.taotao.com
   1. 好处：避免携带一些无用的cookie  
      
2. 拷贝JS和CSS到磁盘路径中：  
   
3. 配置nginx  
   
4. 测试：  
   

TODO: 前台系统改造静态资源的访问。

## 表结构

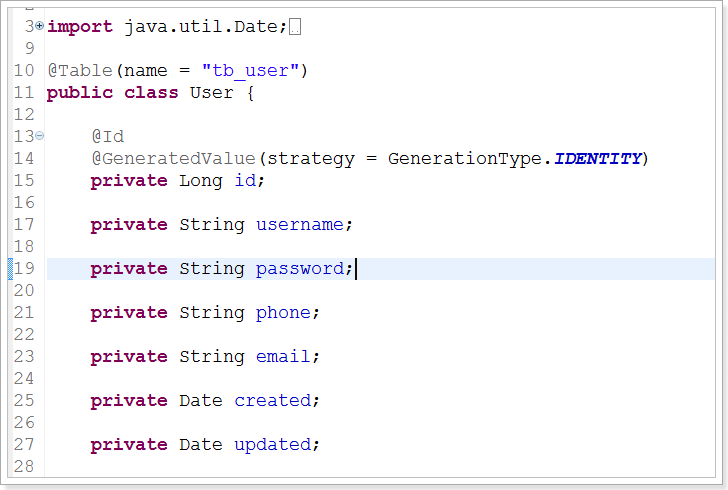


## 接口文档（模板）

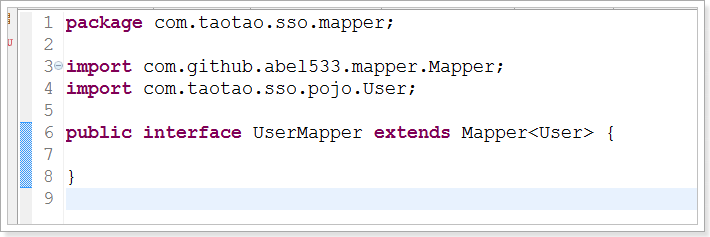


## 基础代码

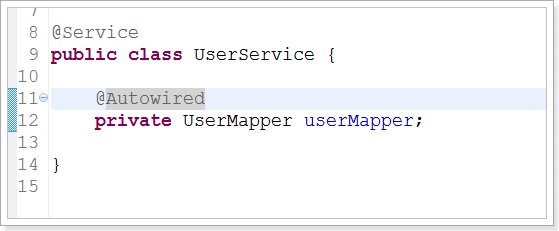
### 创建POJO



### 创建Mapper

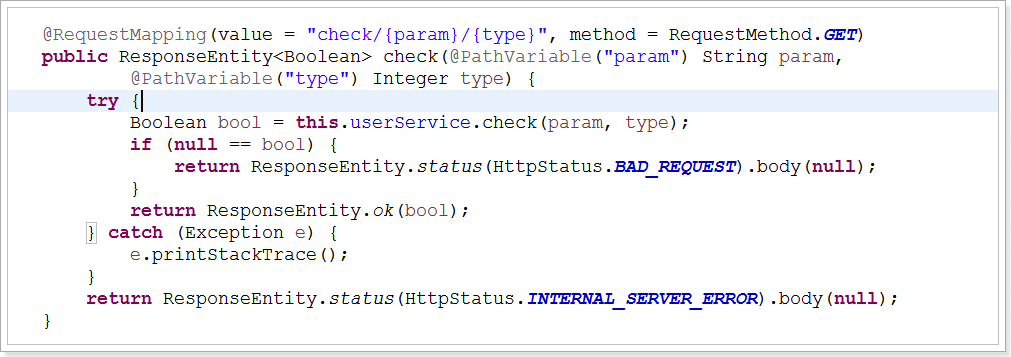


### 创建UserService



## 检测数据是否可用

### Controller



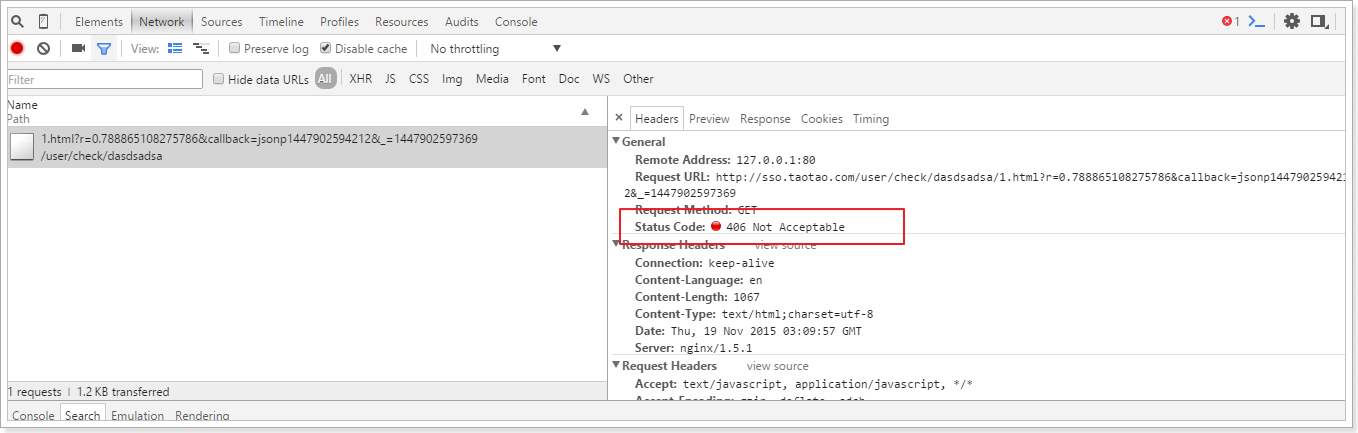
### Service



### 修改JS



测试：



### 日志

2015-11-19 11:09:57,870 [http-bio-8083-exec-2] [org.springframework.jdbc.datasource.DataSourceTransactionManager]-[DEBUG] Releasing JDBC Connection [ConnectionHandle{url=jdbc:mysql://127.0.0.1:3306/taotao?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true, user=root, debugHandle=null, lastResetAgoInSec=0, lastUsedAgoInSec=0, creationTimeAgoInSec=0}] after transaction

2015-11-19 11:09:57,870 [http-bio-8083-exec-2] [org.springframework.jdbc.datasource.DataSourceUtils]-[DEBUG] Returning JDBC Connection to DataSource

2015-11-19 11:09:57,891 [http-bio-8083-exec-2] [org.springframework.web.servlet.mvc.method.annotation.ExceptionHandlerExceptionResolver]-[DEBUG] Resolving exception from handler [public org.springframework.http.ResponseEntity<java.lang.Boolean> com.taotao.sso.controller.UserController.check(java.lang.String,java.lang.Integer)]: org.springframework.web.HttpMediaTypeNotAcceptableException: Could not find acceptable representation

2015-11-19 11:09:57,893 [http-bio-8083-exec-2] [org.springframework.web.servlet.mvc.annotation.ResponseStatusExceptionResolver]-[DEBUG] Resolving exception from handler [public org.springframework.http.ResponseEntity<java.lang.Boolean> com.taotao.sso.controller.UserController.check(java.lang.String,java.lang.Integer)]: org.springframework.web.HttpMediaTypeNotAcceptableException: Could not find acceptable representation

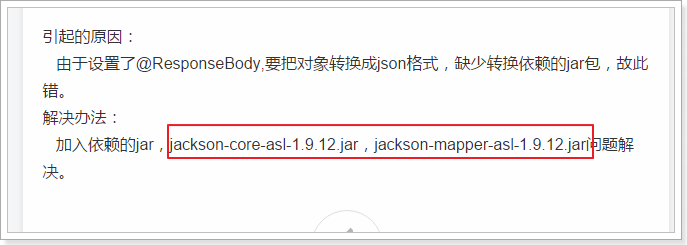
2015-11-19 11:09:57,893 [http-bio-8083-exec-2] [org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver]-[DEBUG] Resolving exception from handler [public org.springframework.http.ResponseEntity<java.lang.Boolean> com.taotao.sso.controller.UserController.check(java.lang.String,java.lang.Integer)]: org.springframework.web.HttpMediaTypeNotAcceptableException: Could not find acceptable representation

2015-11-19 11:09:57,893 [http-bio-8083-exec-2] [org.springframework.web.servlet.DispatcherServlet]-[DEBUG] Null ModelAndView returned to DispatcherServlet with name 'taotao-sso': assuming HandlerAdapter completed request handling

2015-11-19 11:09:57,893 [http-bio-8083-exec-2] [org.springframework.web.servlet.DispatcherServlet]-[DEBUG] Successfully completed request

百度之后的结果：

<http://www.tuicool.com/articles/q2eEfe>



但是，不能解决我们的问题。

究竟是什么问题呢？

SpringMVC的规定：**在SpringMVC中如果请求以html结尾，那么就不会返回JSON数据**。

解决方案：（多条路进入SpringMVC）



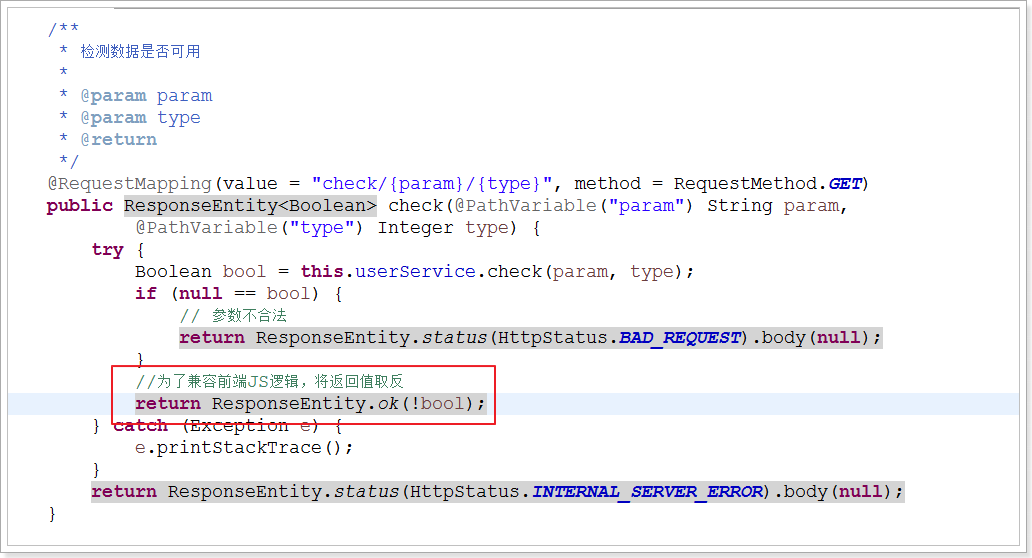
### 测试





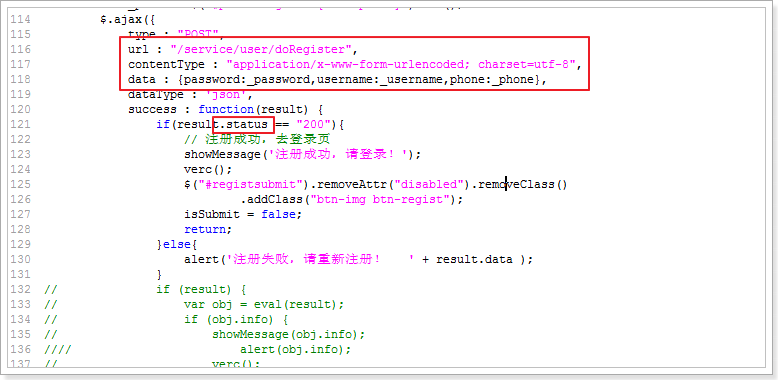


Controller：

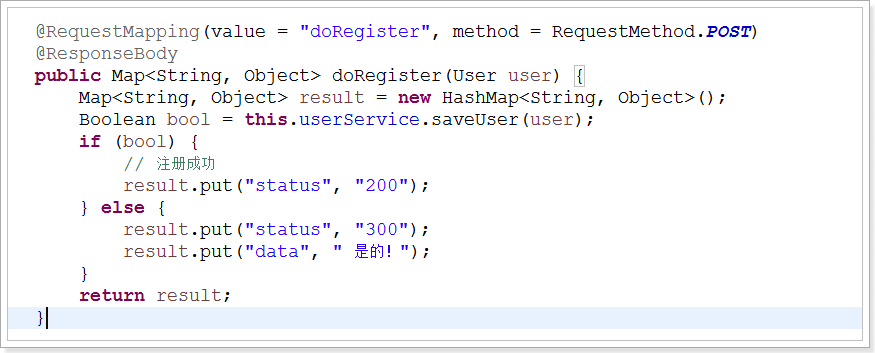


# 注册

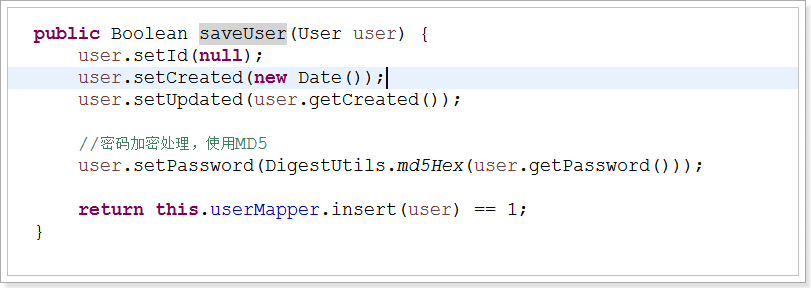
## 前端JS



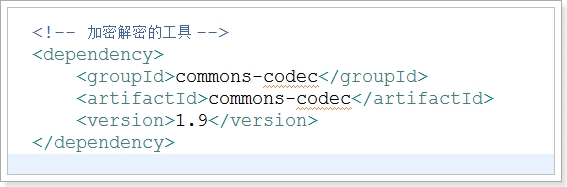
## Controller



## Service

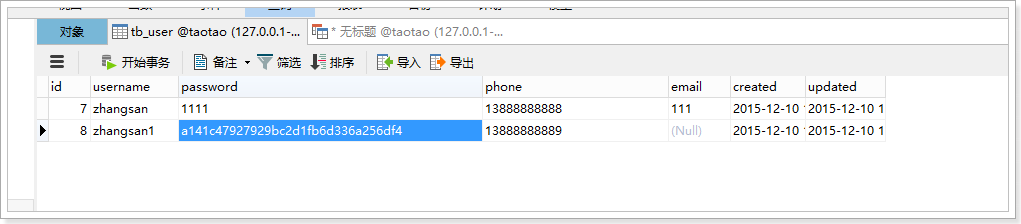


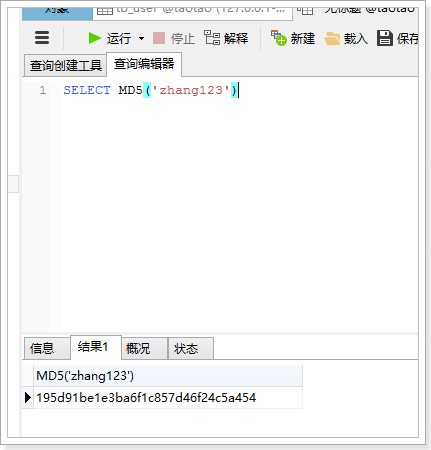
使用Apache提供的加密解密工具进行MD5加密:



## 测试

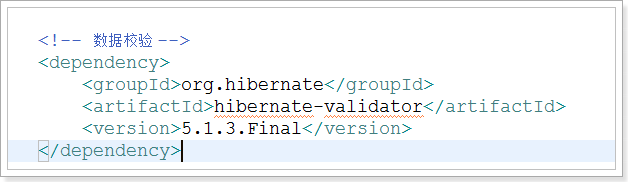




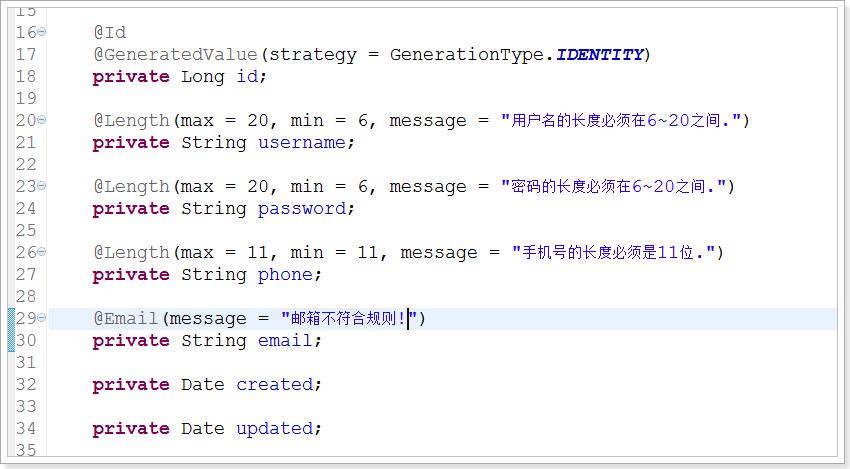


## 数据校验

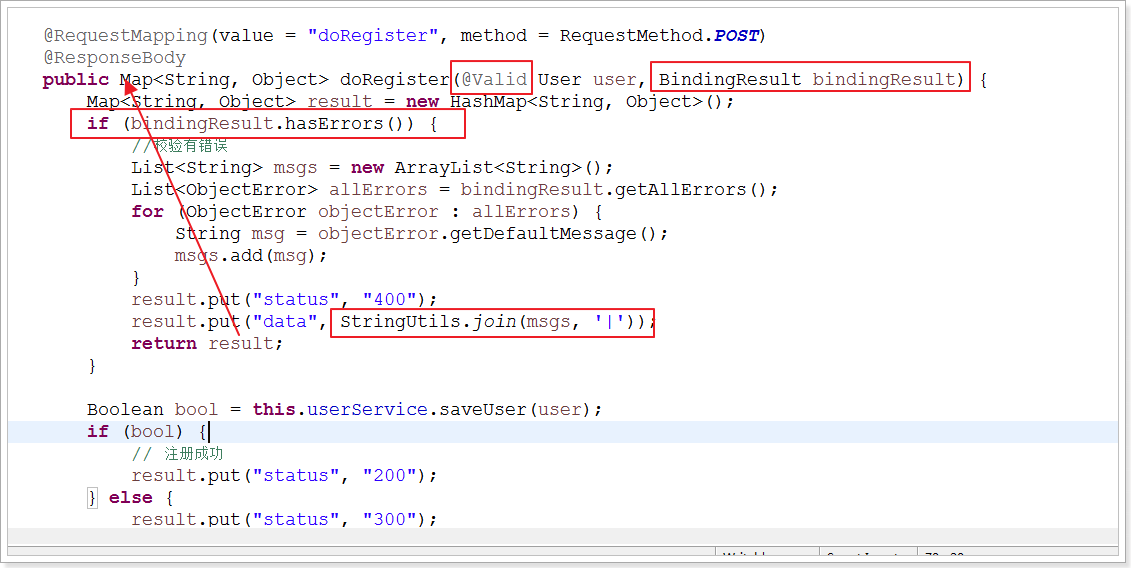
在SpringMVC中实现数据校验：



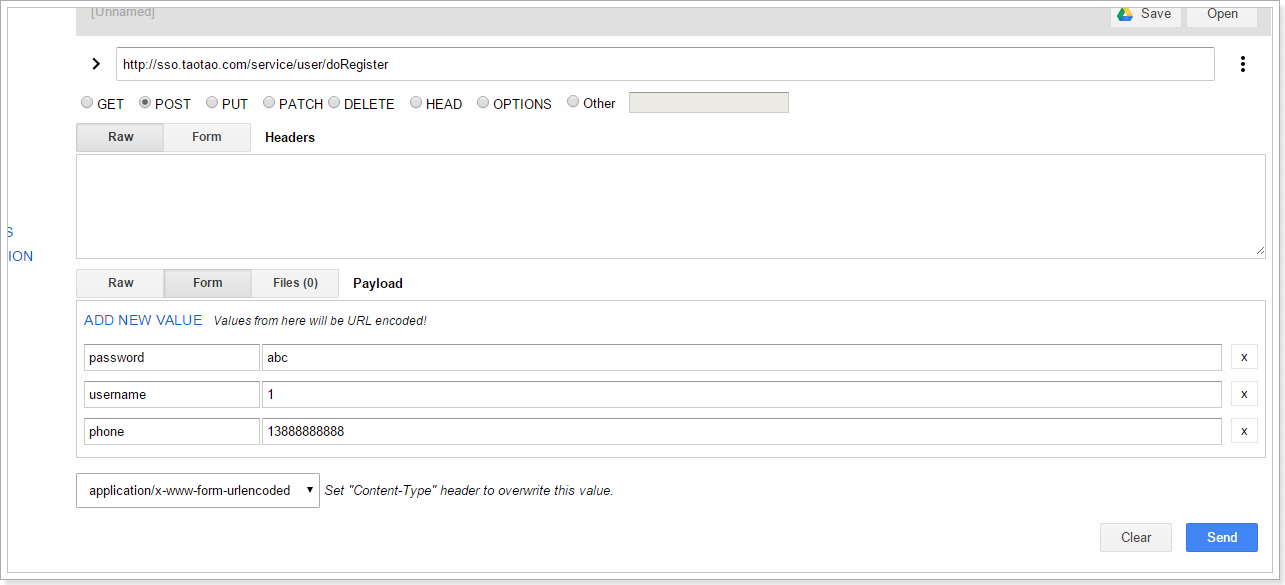
### 在POJO中添加校验规则

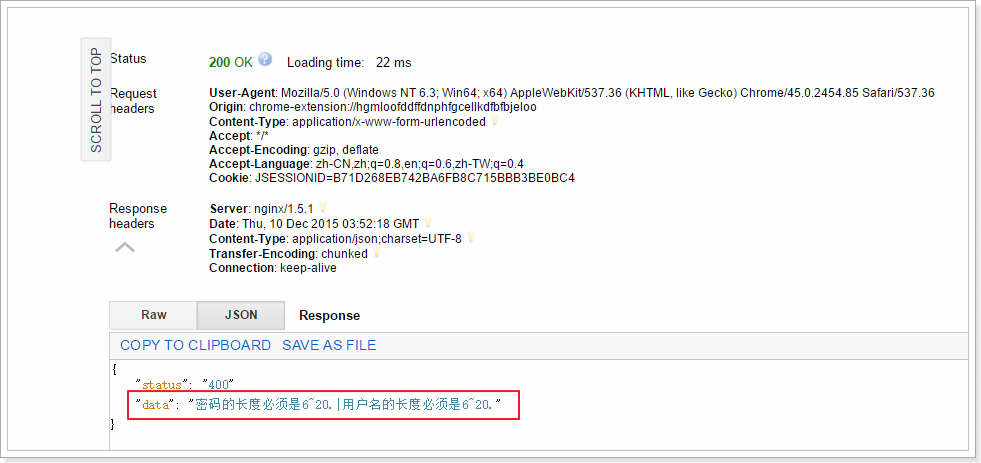


### 在Controller中添加校验注解



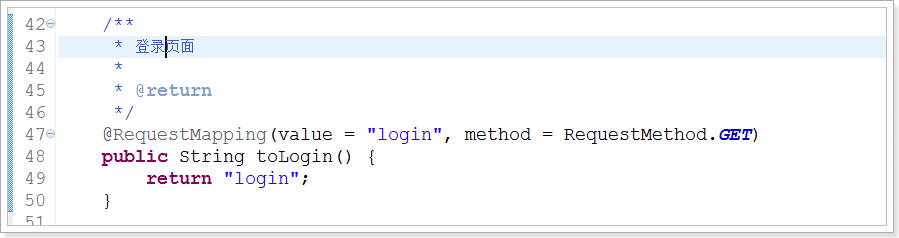
### 测试



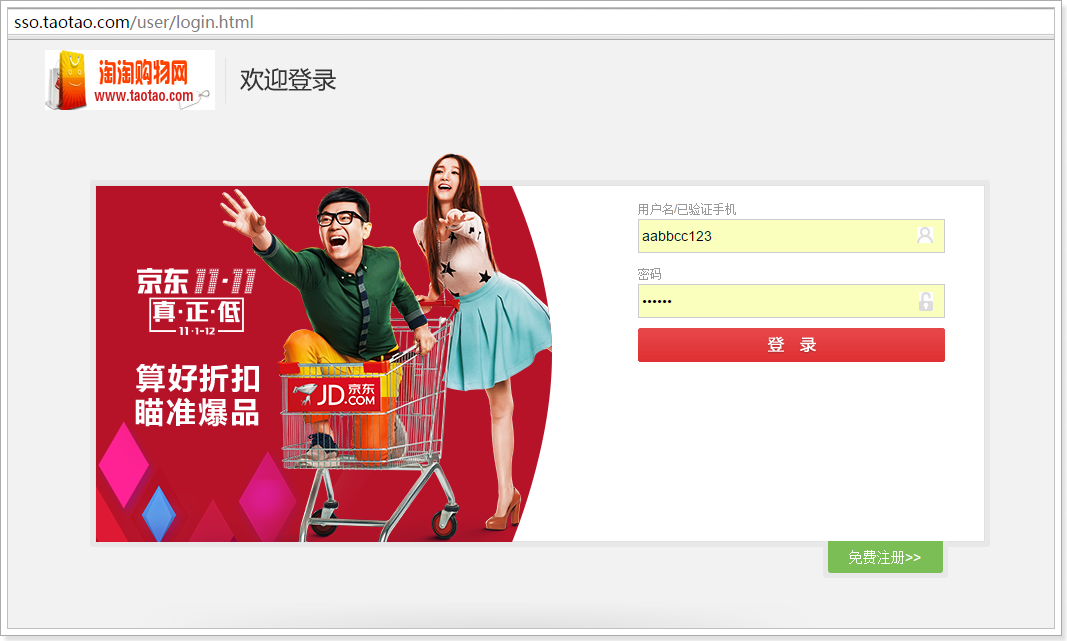


# 登录

## 跳转到登录页面



## 测试

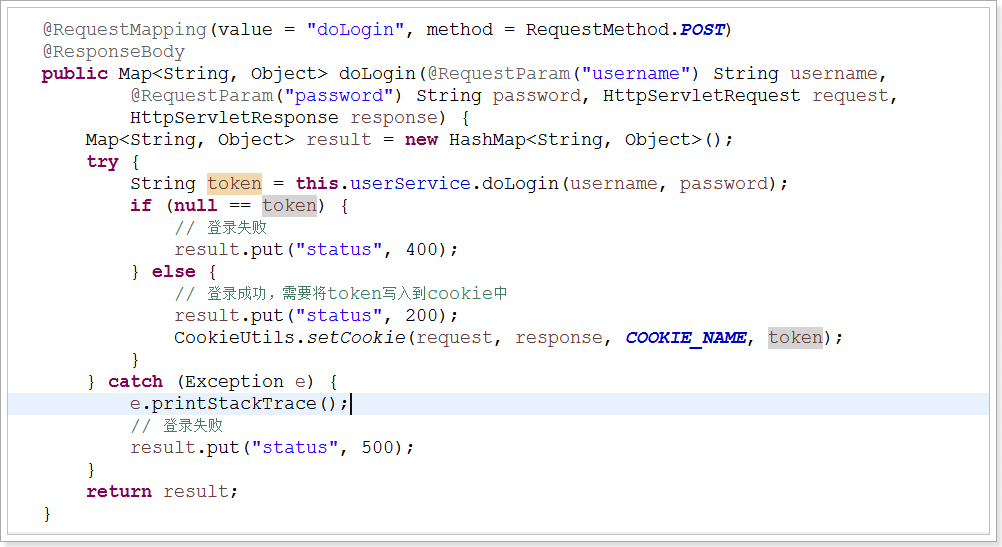


## 提交事件



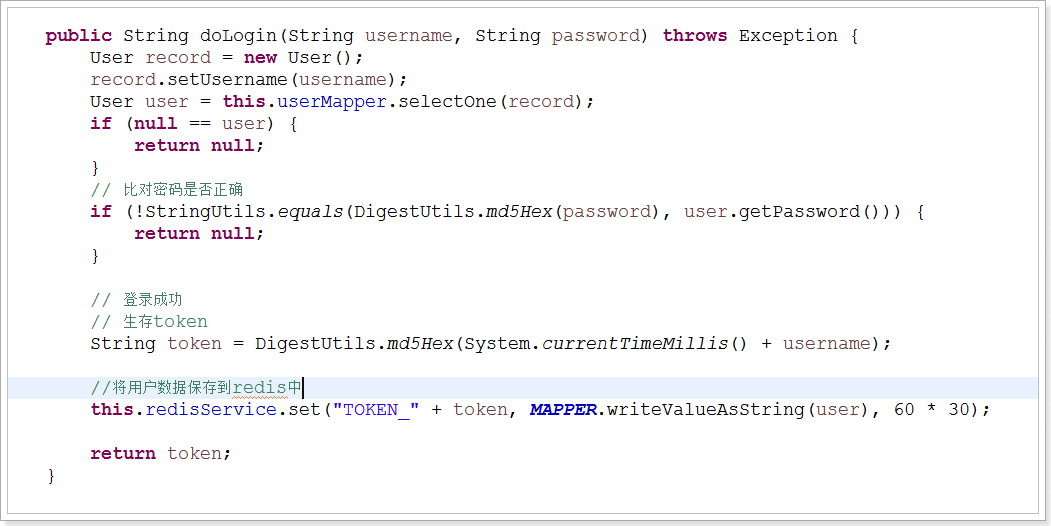
## 后台处理提交逻辑

### Controller

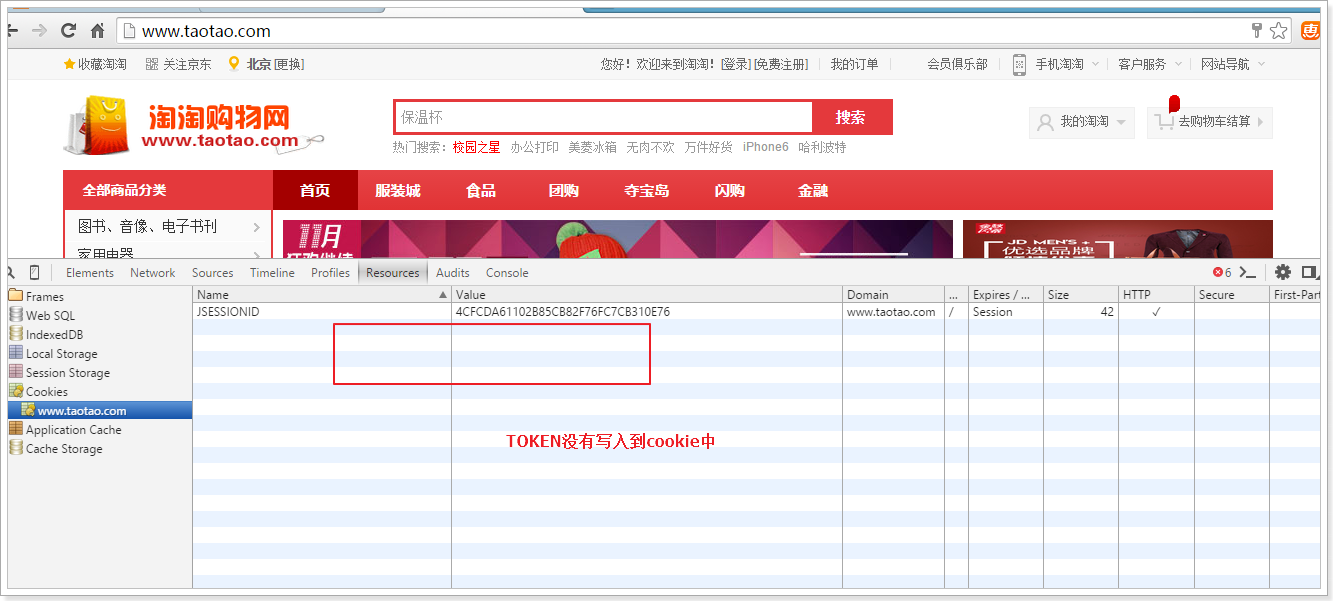


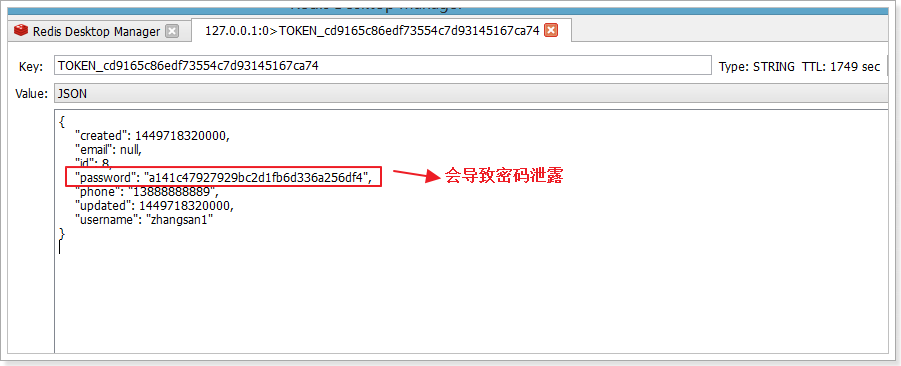
### Service

先通过用户名查询User对象，再对比密码是否相同。



### 测试

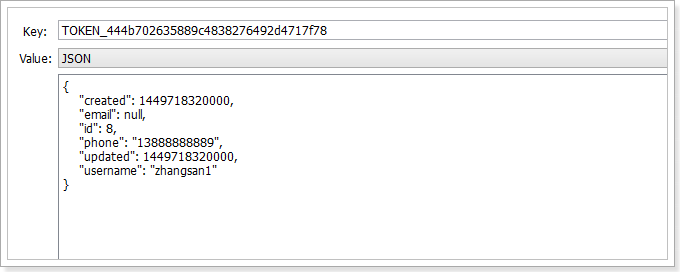




### 忽略保存数据中的password

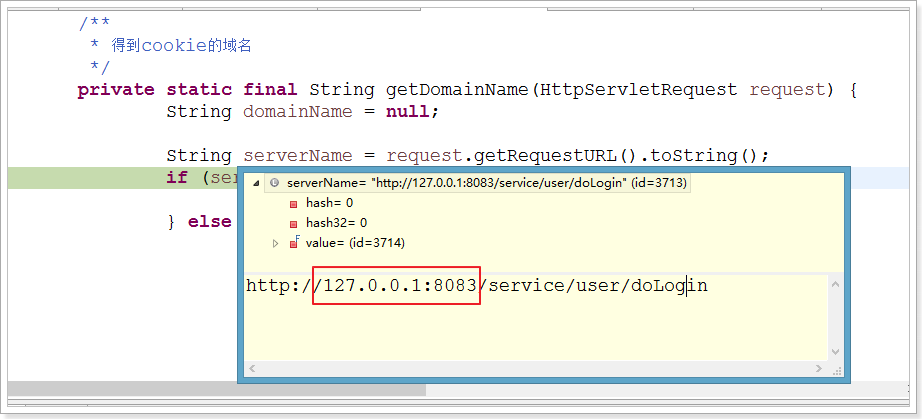


测试：



## 登录成功后没有写入cookie

问题：



获取到到的127.0.0.1，需要将cookie写入到taotao.com中，这样违反了浏览的安全的原则，导致写入失败。

解决方案：

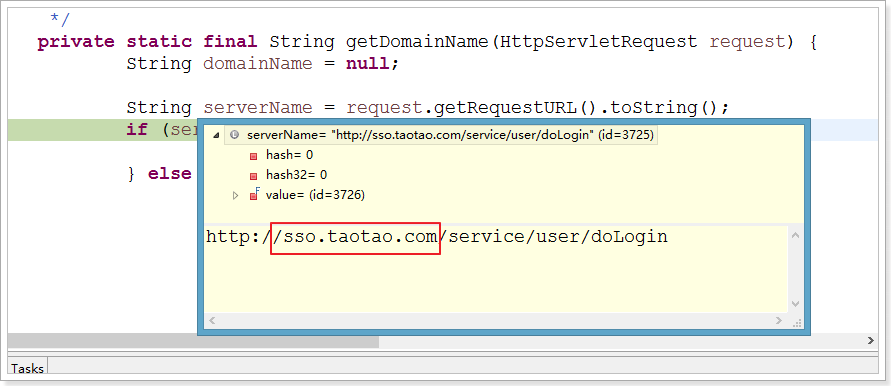
只需要通过request对象获取到正确的地址（xxx.taotao.com）即可。

请求：用户请求 🡺 nginx 🡺 tomcat，tomcat根本不知道真正的请求来源。

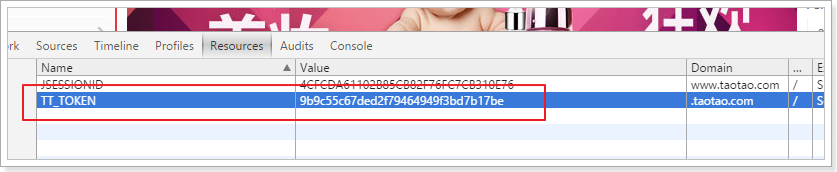
解决：

在nginx的配置文件中添加Host的代理头信息：





效果：

、

总结：

二级域名可以将cookie写入到主域名下。 A.taotao.com 🡺 taotao.com

二级域名之间不能互相写入，a.taotao.com 不能写入到 b.taotao.com

## 显示当前登录人的用户名

### 前端JS



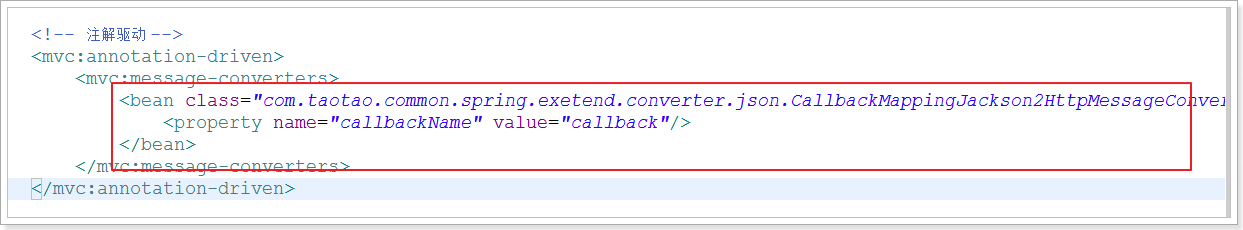
### 后端Controller



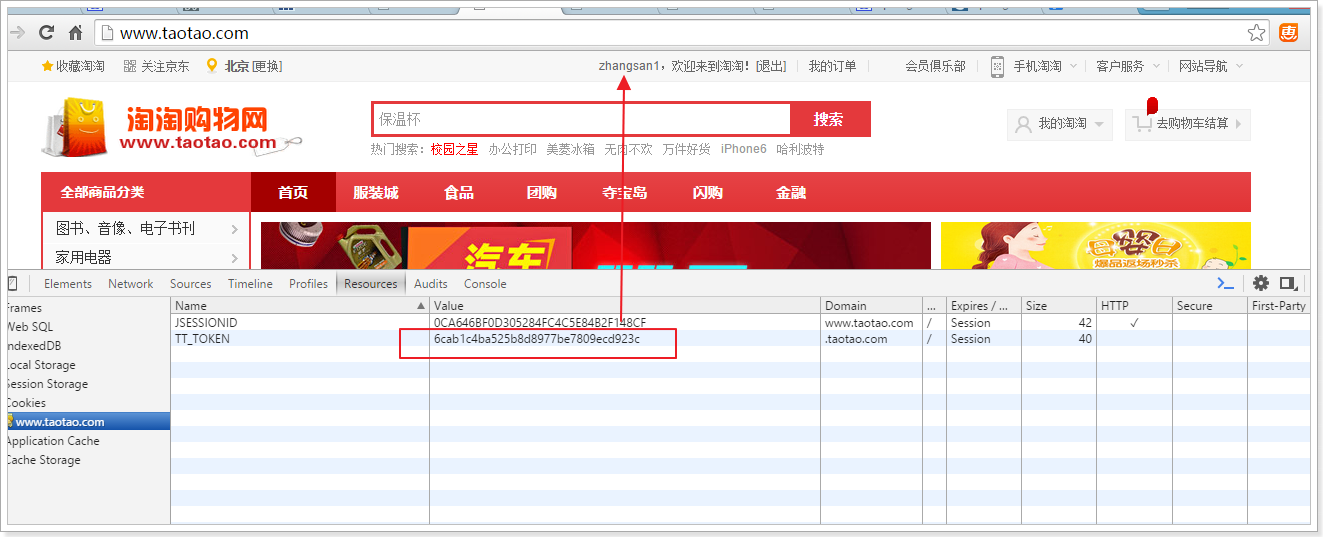
### Service



### 添加跨域支持



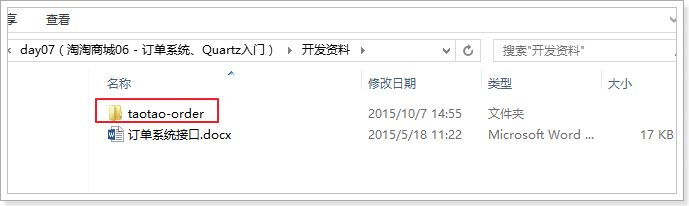
### 测试

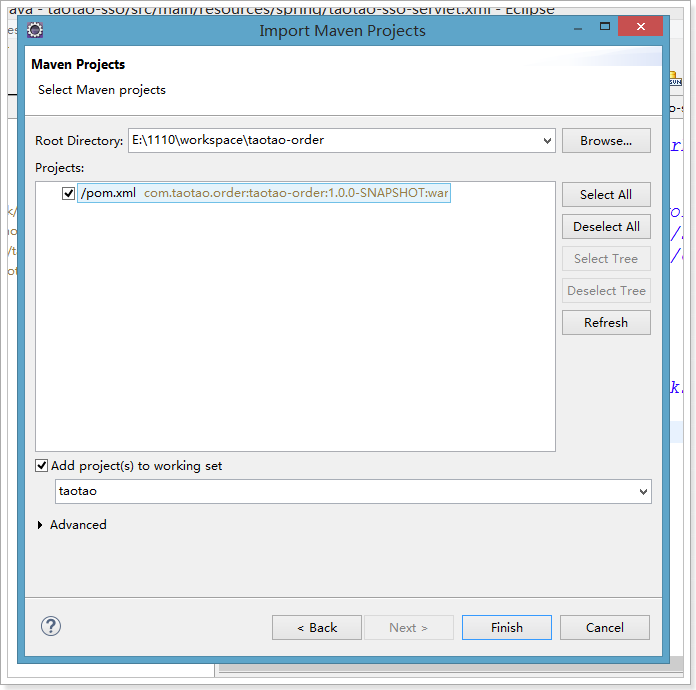


# 订单系统

说明：订单系统只是做讲解，不做开发。

## 导入taotao-order



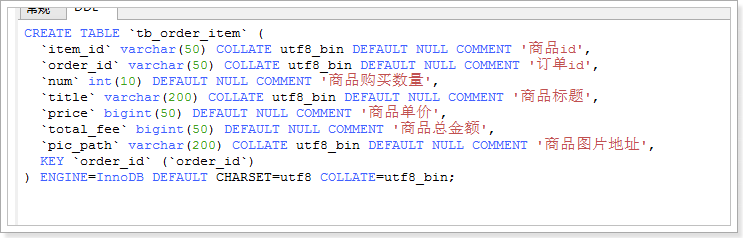


## 表结构

订单表：



订单商品表：



疑问：为什么要冗余存储商品的基本信息？

订单物流表（收货人信息）：



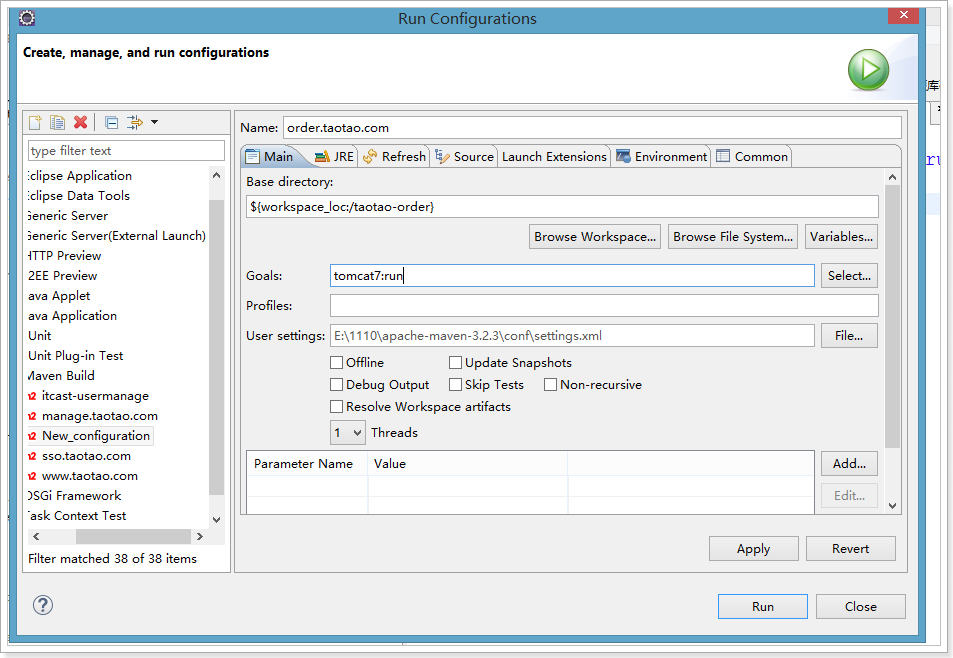
## 开放平台



<http://open.taobao.com/doc2/apiDetail?spm=0.0.0.0.MiNiKt&apiId=47&docType>=：



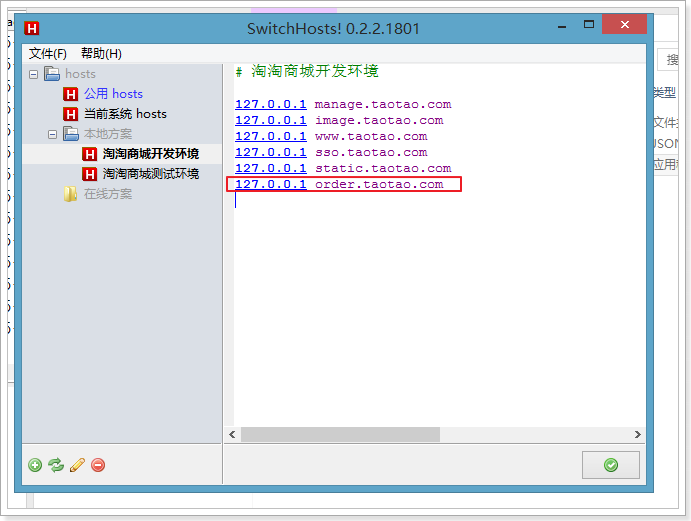
## 配置tomcat插件



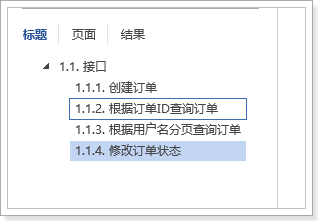
## 配置nginx



## 配置hosts



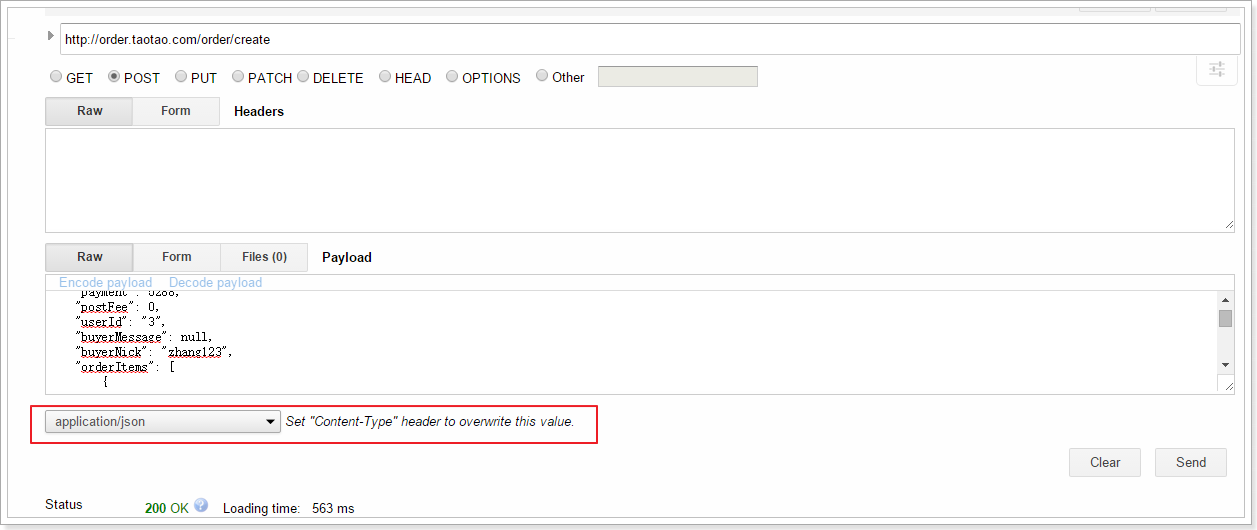
## 接口文档



## 接口

### 创建订单接口

使用：



### 如何生成订单号

订单号需求是什么？

1. 唯一
2. 可读性高
   1. 纯数字
3. 长度
   1. 不长于20位

如何生成：

1. 时间戳
   1. 可能重复
2. 手机号
3. 时间戳 + 随机数
   1. 可能重复
4. 时间戳 + 自增id
   1. 可行
5. 用户id + 时间戳
6. 用户id + 店铺id + 时间戳

### 具体实现

Controller： 接收提交的json数据

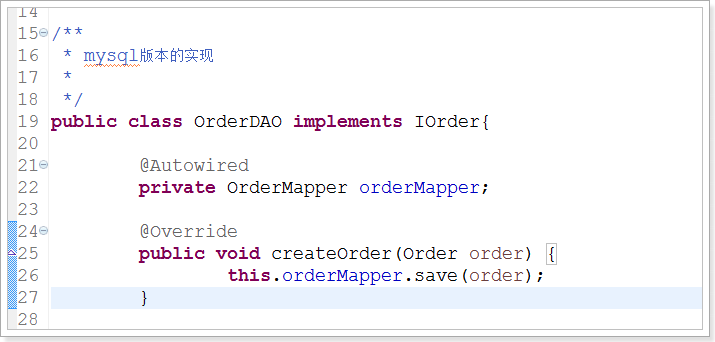


Service：将json反序列化为Order对象，对Order对象做校验，生成订单ID：

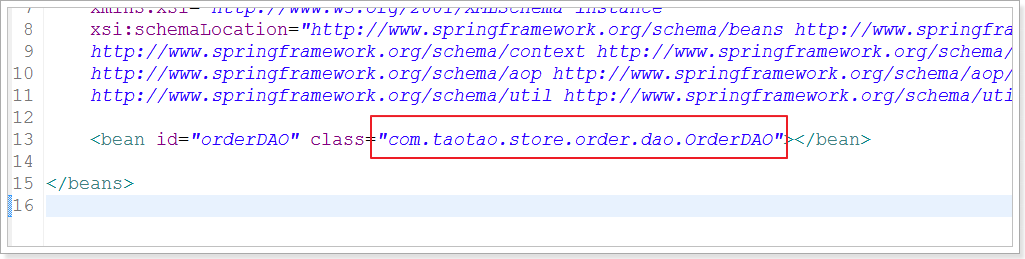




OrderDAO是一个接口的实现类，后期如果对订单数据的存储做改造时，保存订单就会有多套实现：

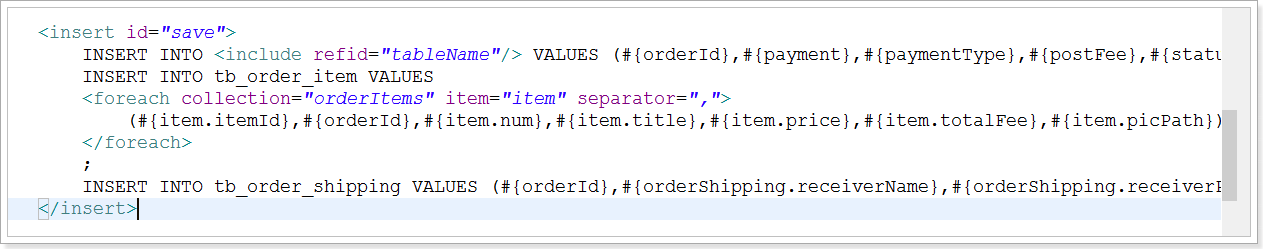


通过配置将OrderDAO加入到Spring容器：



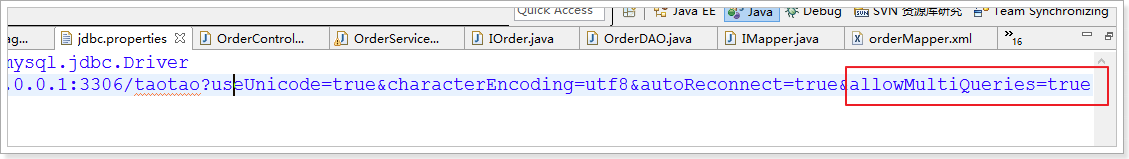
使用配置文件配置bean，后期的修改更加灵活。

Mapper.xml:



一个statement执行多条SQL语句，默认情况下支持吗？ -- 不支持的。

在连接字符串中设置：



是在同一个事务中吗？ -- 是的。