**淘淘商城**

**第十三天**

# 课程计划

1. 实现跳转订单结算页
2. 创建订单服务
   1. 数据库表分析
   2. 创建订单服务
3. 实现订单功能
   1. 实现创建订单
   2. 跳转订单创建成功页

4. Quartz回顾

5. 实现清理无效订单

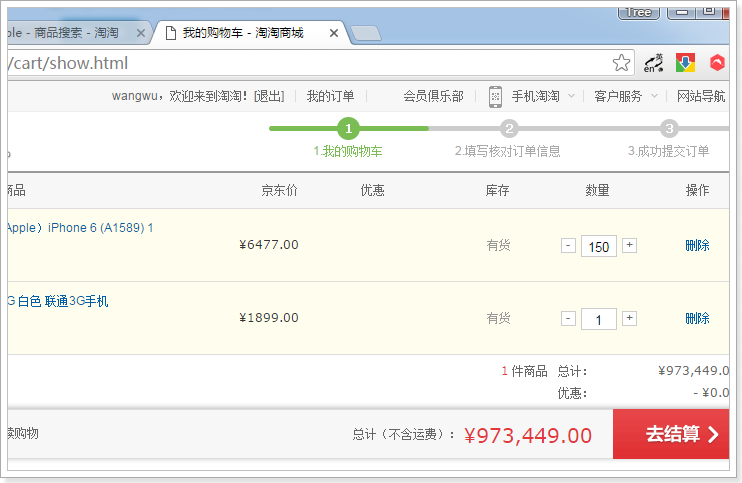
# 实现跳转订单结算页

## 分析

### 页面跳转

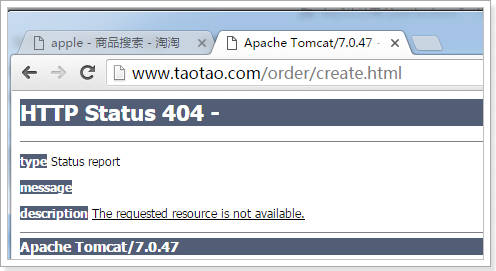
在购物车有“去结算按钮”

当点击“去结算”时，应该跳转到订单结算页面

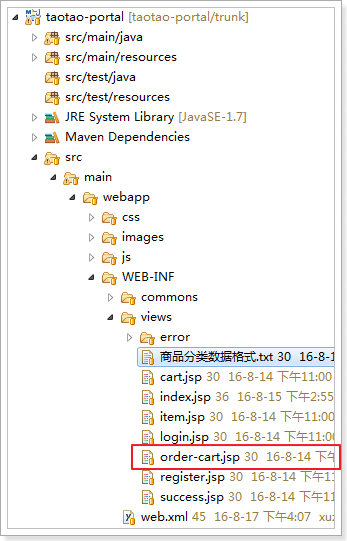


点击“去结算”按钮，访问页面

http://www.taotao.com/order/create.html



订单页面是taotao-portal的order-cart.jsp



### 页面展示数据

这里是把购物车所有的商品都添加到订单中

页面需要的是carts，

需要登陆用户所属的购物车查询出来，放到页面中



## 实现跳转订单结算页

在taotao-portal实现下单功能，需要跳转到订单页面

在taotao-portal实现Controller

@Controller

@RequestMapping("order")

**public** **class** OrderController {

@Value("${TT\_TICKET}")

**private** String TT\_TICKET;

@Autowired

**private** UserService userService;

@Autowired

**private** CartService cartService;

// http://www.taotao.com/order/create.html

/\*\*

\* 跳转到订单结算页

\* **@param** request

\* **@param** model

\* **@return**

\*/

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

**public** String create(HttpServletRequest request, Model model) {

// 查询用户信息

String ticket = CookieUtils.*getCookieValue*(request, **this**.TT\_TICKET, **true**);

User user = **this**.userService.queryUserByTicket(ticket);

**if** (user != **null**) {

// 根据用户id查询购物车

List<Cart> carts = **this**.cartService.queryCartByUserId(user.getId());

// 把查询到的购物车放到Model中

model.addAttribute("carts", carts);

}

**return** "order-cart";

}

}

实现效果



## 实现订单拦截器

用户只有在登录的时候，才能够创建订单

所以需要添加订单拦截器，如果用户登录，就可以进入订单结算页面，

用户未登录，跳转到登录页面

### 实现拦截器

在taotao-portal实现

**public** **class** OrderHandlerInterceptor **implements** HandlerInterceptor {

@Value("${TT\_TICKET}")

**private** String TT\_TICKET;

@Autowired

**private** UserService userService;

@Override

**public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)

**throws** Exception {

// 获取cookie中的ticket

String ticket = CookieUtils.*getCookieValue*(request, **this**.TT\_TICKET);

**if** (StringUtils.*isBlank*(ticket)) {

// 用户未登录

response.sendRedirect("/page/login.html");

**return** **false**;

}

// 根据ticket查询用户信息

User user = **this**.userService.queryUserByTicket(ticket);

**if** (user == **null**) {

// 用户登录超

response.sendRedirect("/page/login.html");

**return** **false**;

}

**return** **true**;

}

@Override

**public** **void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,

ModelAndView modelAndView) **throws** Exception {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,

Exception ex) **throws** Exception {

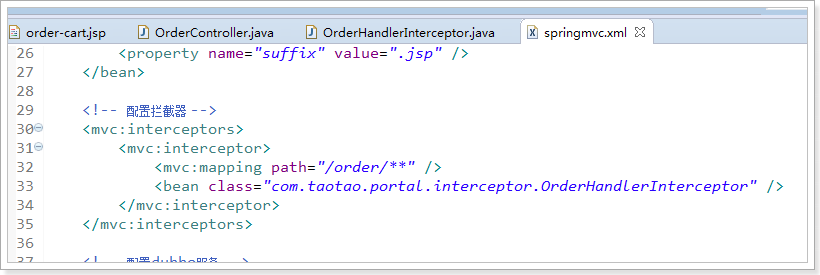
// **TODO** Auto-generated method stub

}

}

### 配置拦截器

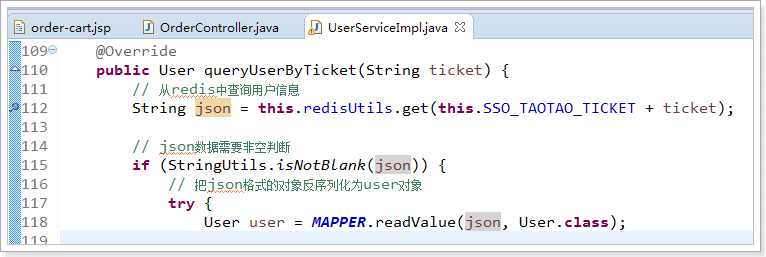
在taotao-portal配置



## 加入拦截器存在的问题

加入拦截器后，发现有两个问题：

1. 跳转订单结算页面，会调用taotao-sso服务层的queryUserByTicket方法两次。
2. 用户登录成功后，跳转到的是首页，应该是跳转到订单结算页面



### 重复调用单点登录服务的问题

调用两次服务查询用户信息的原因

第一次：拦截器查询

第二次：Controller里面查询

解决方案：

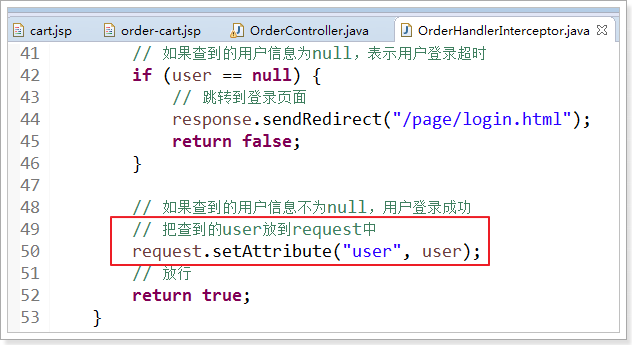
可以把查询信息找地方放起来，第一次查询调用服务获取用户信息并保存起来，

第二次和第二次之后查询，就从之前保存的地方获取

可以使用request进行保存

#### 修改拦截器

在taotao-portal中修改



#### 修改Controller方法

不用判断user是否为null，因为拦截器已经处理过了

在taotao-portal中修改

/\*\*

\* 跳转订单结算页

\*

\* **@param** model

\* **@param** request

\* **@return**

\*/

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

**public** String create(Model model, HttpServletRequest request) {

// 查询用户登录信息

// String ticket = CookieUtils.getCookieValue(request, this.TT\_TICKET,

// true);

// User user = userService.queryUserByTicket(ticket);

// 从request中获取user信息

User user = (User) request.getAttribute("user");

// if (user != null) {

// 根据用户id查询购物车数据

List<Cart> carts = **this**.cartService.queryListByUserId(user.getId());

// 把查到的购物车放到模型中

model.addAttribute("carts", carts);

// }

**return** "order-cart";

}

### 登录成功跳转到首页的问题

用户在跳转订单结算页面，如果没有登录，需要进行登录操作，但是登陆成功是跳转到首页，应该跳转到原来的请求，就是订单结算页.

解决分析：

1. 拦截器在跳转到登陆页面时候，把之前的请求地址保存下来，作为参数进行跳转

2. 在登录页面登录成功后，查看是否有请求参数，如果有，就跳转到这参数所对应的url，如果参数的值为空，则跳转到首页

#### 修改拦截器

修改拦截器，在跳转到登录页面的时候，携带原来的请求url，

修改taotao-portal的拦截器

@Override

**public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)

**throws** Exception {

// 从cookie中获取ticket

String ticket = CookieUtils.*getCookieValue*(request, **this**.TT\_TICKET, **true**);

// 判断cookie中是否有ticket，如果cookie中没有表示用户未登录

**if** (StringUtils.*isBlank*(ticket)) {

// 如果cookie中没有表示用户未登录

// 跳转到登录页面,需要保存用户现在的请求地址，如果登录成功，应该跳转到这个地址上

String redirectURL = request.getRequestURL().toString();

response.sendRedirect("/page/login.html?redirectURL=" + redirectURL);

**return** **false**;

}

// 根据ticket查询用户信息

User user = **this**.userService.queryUserByTicket(ticket);

// 如果查到的用户信息为null，表示用户登录超时

**if** (user == **null**) {

// 跳转到登录页面

response.sendRedirect("/page/login.html");

**return** **false**;

}

// 如果查到的用户信息不为null，用户登录成功

// 把查到的user放到request中

request.setAttribute("user", user);

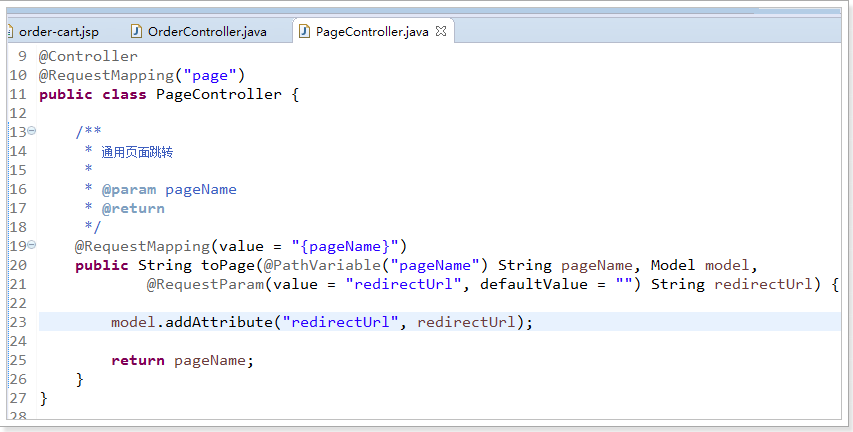
// 放行

**return** **true**;

}

#### 修改通用跳转方法

修改通用跳转方法，把请求参数放到页面中



#### 修改登录按钮逻辑

在taotao-portal的login.jsp修改点击登录按钮逻辑

$("#loginsubmit").click(**function** () {

**var** loginUrl = "../uc/loginService";

**var** redirectUrl = "${redirectUrl}";

**var** flag = validateFunction.FORM\_validate();

**if** (flag) {

**var** uuid = $("#uuid").val();

$(**this**).attr({ "disabled": "disabled" });

**var** \_username = $("#formlogin [name=username]").val();

**var** \_password = $("#formlogin [name=password]").val();

$.ajax({

type: "POST",

url: "/service/user/doLogin?r=" + Math.random(),

contentType: "application/x-www-form-urlencoded; charset=utf-8",

data: {username:\_username,password:\_password},

dataType : "json",

error: **function** () {

$("#nloginpwd").attr({ "class": "text highlight2" });

//请求失败

$("#loginpwd\_error").html("网络超时，请稍后再试").show().attr({ "class": "error" });

$("#loginsubmit").removeAttr("disabled");

$this.removeAttr("disabled");

},

success: **function** (result) {

**if** (result) {

**var** obj = eval(result);

**if** (obj.status == 200) {

**if**(redirectUrl){

//当redirectURL不为空的时候执行

obj.success = redirectUrl;

}**else**{

//当redirectURL为空跳转首页

obj.success = "http://www.taotao.com/";

}

**var** isIE = !-[1,];

**if** (isIE) {

**var** link = document.createElement("a");

link.href = obj.success;

link.style.display = 'none';

document.body.appendChild(link);

link.click();

} **else** {

window.location = obj.success;

}

**return**;

}**else**{

$("#loginsubmit").removeAttr("disabled");

verc();

$("#nloginpwd").attr({ "class": "text highlight2" });

$("#loginpwd\_error").html("账号或密码错误!").show().attr({ "class": "error" });

}

}

}

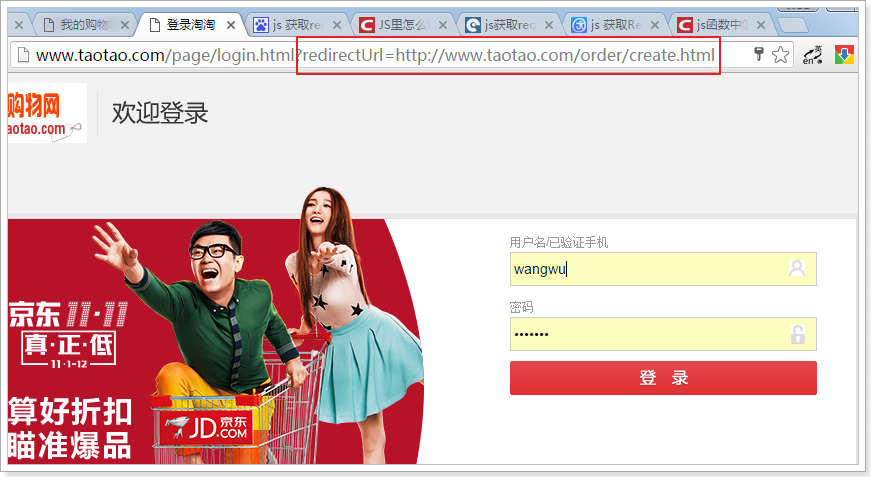
});

}

});

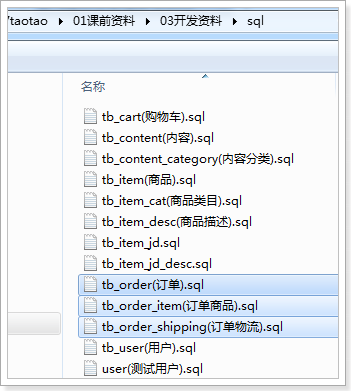
#### 效果

跳转登录页面



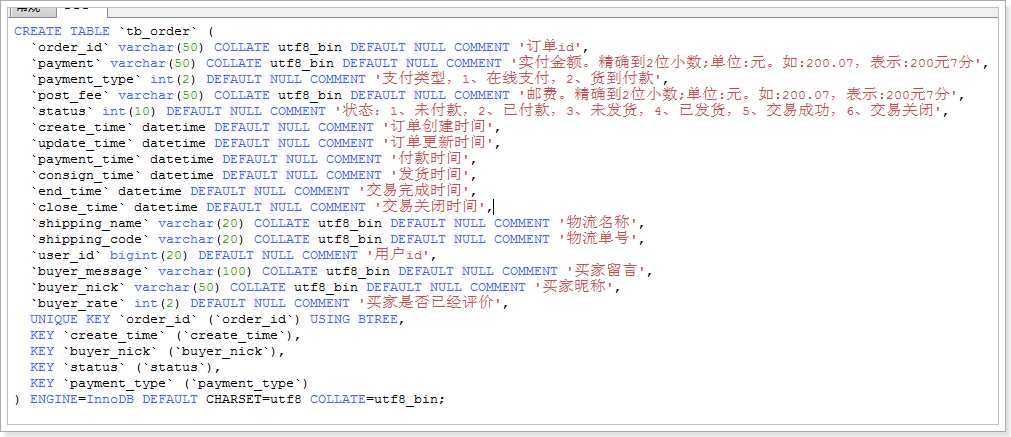
登录成功返回订单结算页

# 创建订单服务



## 数据库表分析

### 订单表



1. 发现订单id不是自增，需要设置

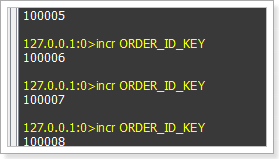
订单id需求：

* 1. 唯一的
  2. 可读性高
  3. 不能太长（不超过20位）

创建的方式：

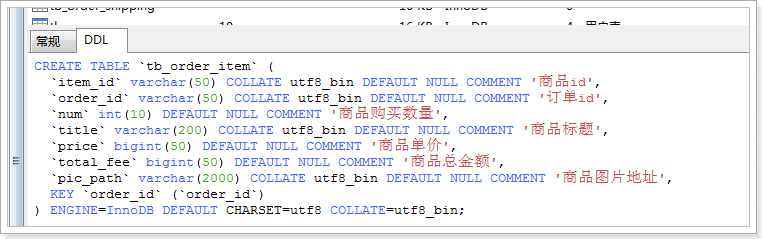
可以使用用户id+Redis的唯一数，作为订单id

redis是单线程的，可以使用incr命令，每次都加一，这样次取得数字就是唯一的



1. 没有设置主键，但是订单id是唯一约束，可以使用他作为主键

### 订单商品表



使用冗余设计，展示订单商品信息，不需要做关联查询（优化点）

订单表和订单商品表关系：一对多

### 订单物流表



订单表和订单物流表关系：一对一

用户表和订单表：一对多

## 创建工程

需要创建taotao-order，提供订单服务

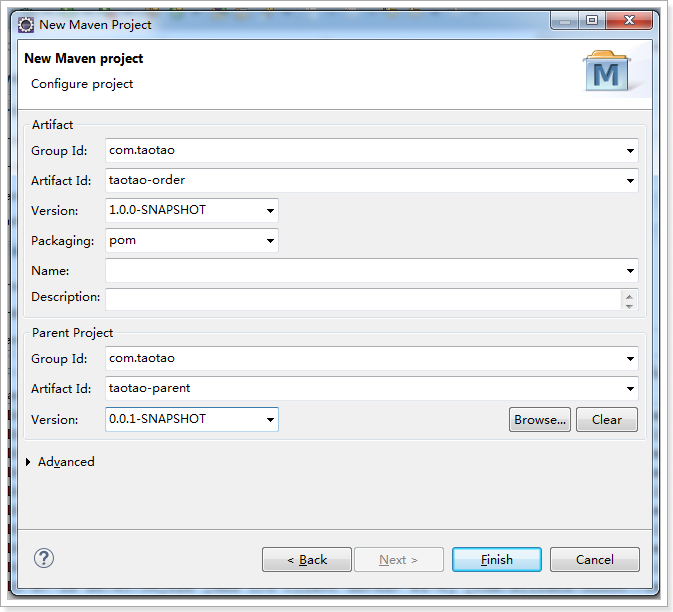
创建taotao-order

创建taotao-order-interface接口

创建taotao-order-service

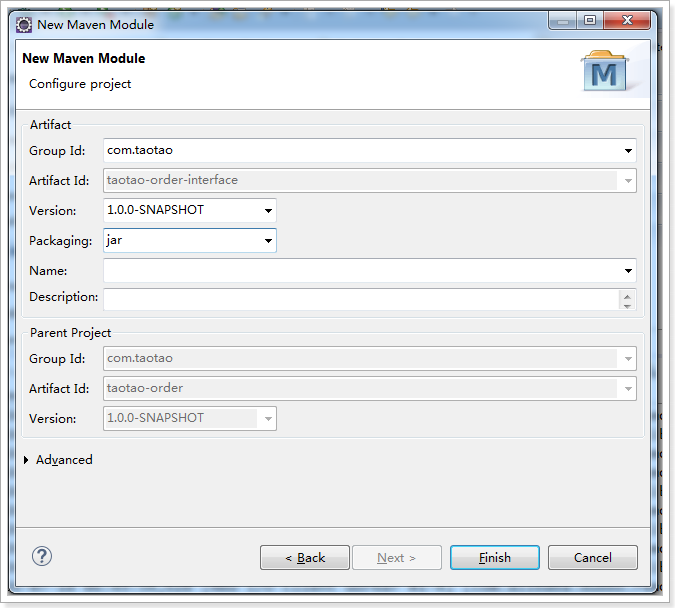
### 创建taotao-order

聚合工程父工程



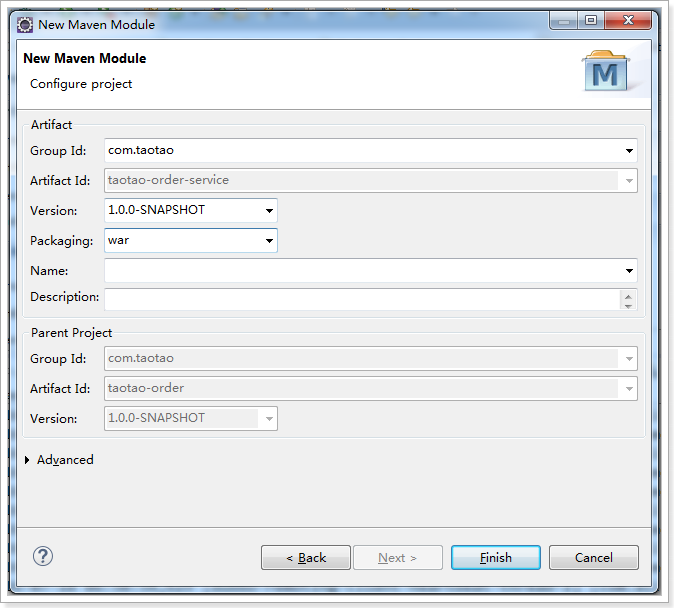
### 创建taotao-order-interface

聚合工程子工程



### 创建taotao-order-service

聚合工程子工程



## 加入依赖

### taotao-order

<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</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-order</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>pom</packaging>

<modules>

<module>taotao-order-interface</module>

<module>taotao-order-service</module>

</modules>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</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>8088</port>

<path>/</path>

</configuration>

</plugin>

</plugins>

</build>

</project>

### taotao-order-interface

<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</groupId>

<artifactId>taotao-order</artifactId>

<version>1.0.0-SNAPSHOT</version>

</parent>

<artifactId>taotao-order-interface</artifactId>

<dependencies>

<!-- 加入taotao-manager-pojo依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-manager-pojo</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

</dependencies>

</project>

### taotao-order-service

<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</groupId>

<artifactId>taotao-order</artifactId>

<version>1.0.0-SNAPSHOT</version>

</parent>

<artifactId>taotao-order-service</artifactId>

<packaging>war</packaging>

<dependencies>

<!-- 加入taotao-manager-mapper依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-manager-mapper</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- 加入taotao-order-interface依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-order-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- spring -->

<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>

<dependency>

<groupId>org.springframework</groupId>

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

</dependency>

<!-- 分页助手 -->

<dependency>

<groupId>com.github.pagehelper</groupId>

<artifactId>pagehelper</artifactId>

</dependency>

<dependency>

<groupId>com.github.jsqlparser</groupId>

<artifactId>jsqlparser</artifactId>

</dependency>

<!-- MySql -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</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>

<!-- dubbo相关 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

</dependency>

<!-- Redis客户端 -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

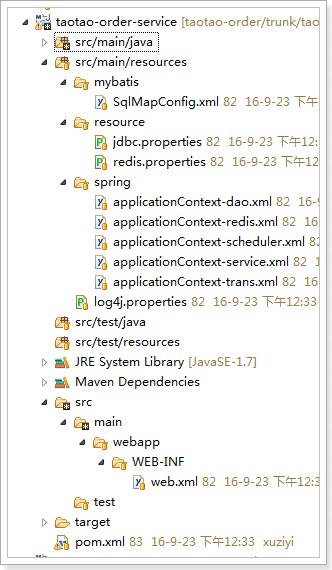
</dependency>

</dependencies>

</project>

## 加入配置文件

配置文件结构



### 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=*"MyWebApp"* version=*"2.5"*>

<display-name>taotao-order</display-name>

<welcome-file-list>

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

</welcome-file-list>

<!-- spring -->

<context-param>

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

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

</context-param>

<!-- spring监听器 -->

<listener>

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

</listener>

</web-app>

### applicationContext-dao.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:p=*"http://www.springframework.org/schema/p"*

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

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

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

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

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

<!-- 配置 读取properties文件 jdbc.properties -->

<context:property-placeholder location=*"classpath:\*.properties"* />

<!-- 配置数据源 -->

<bean id=*"dataSource"* class=*"com.jolbox.bonecp.BoneCPDataSource"*

destroy-method=*"close"*>

<!-- 数据库驱动 -->

<property name=*"driverClass"* value=*"${jdbc.driver}"* />

<!-- 相应驱动的jdbcUrl -->

<property name=*"jdbcUrl"* value=*"${jdbc.url}"* />

<!-- 数据库的用户名 -->

<property name=*"username"* value=*"${jdbc.username}"* />

<!-- 数据库的密码 -->

<property name=*"password"* value=*"${jdbc.password}"* />

<!-- 检查数据库连接池中空闲连接的间隔时间，单位是分，默认值：240，如果要取消则设置为0 -->

<property name=*"idleConnectionTestPeriod"* value=*"60"* />

<!-- 连接池中未使用的链接最大存活时间，单位是分，默认值：60，如果要永远存活设置为0 -->

<property name=*"idleMaxAge"* value=*"30"* />

<!-- 每个分区最大的连接数 -->

<property name=*"maxConnectionsPerPartition"* value=*"150"* />

<!-- 每个分区最小的连接数 -->

<property name=*"minConnectionsPerPartition"* value=*"5"* />

</bean>

<!-- 配置SqlSession会话工厂 -->

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!-- 配置数据源 -->

<property name=*"dataSource"* ref=*"dataSource"* />

<!-- 配置mybatis全局配置文件 -->

<property name=*"configLocation"* value=*"classpath:mybatis/SqlMapConfig.xml"* />

</bean>

<!-- 配置Mapper扫描 -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<!-- 配置扫描的包 -->

<property name=*"basePackage"* value=*"com.taotao.manager.mapper"* />

</bean>

</beans>

### applicationContext-redis.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:p=*"http://www.springframework.org/schema/p"*

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

xmlns:mvc=*"http://www.springframework.org/schema/mvc"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

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

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

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

<!-- 配置JedisPool -->

<bean id=*"jedisPool"* class=*"redis.clients.jedis.JedisPool"*>

<!-- 配置redis服务的ip -->

<constructor-arg name=*"host"* value=*"${jedis.host}"* />

<!-- 配置redis服务的端口号 -->

<constructor-arg name=*"port"* value=*"${jedis.port}"* />

</bean>

<!-- 配置操作单机版Redis的实现类RedisPool -->

<bean class=*"com.taotao.order.redis.RedisPool"* />

<!-- 配置JedisCluster -->

<bean id=*"jedisCluster"* class=*"redis.clients.jedis.JedisCluster"*>

<!-- 配置redis集群信息 -->

<constructor-arg name=*"nodes"*>

<set>

<!-- Redis集群节点的信息 -->

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host1}"* />

<constructor-arg name=*"port"* value=*"${cluster.port1}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host2}"* />

<constructor-arg name=*"port"* value=*"${cluster.port2}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host3}"* />

<constructor-arg name=*"port"* value=*"${cluster.port3}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host4}"* />

<constructor-arg name=*"port"* value=*"${cluster.port4}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host5}"* />

<constructor-arg name=*"port"* value=*"${cluster.port5}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host6}"* />

<constructor-arg name=*"port"* value=*"${cluster.port6}"* />

</bean>

</set>

</constructor-arg>

</bean>

<!-- 配置操作集群版的Redis的实现类RedisCluster -->

<!-- <bean class="com.taotao.order.redis.RedisCluster" /> -->

</beans>

复制taotao-sso-service的Redis工具类到taotao-cart-service

### applicationContext-service.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:p=*"http://www.springframework.org/schema/p"*

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

xmlns:mvc=*"http://www.springframework.org/schema/mvc"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

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

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

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

<!-- 配置service扫描 -->

<context:component-scan base-package=*"com.taotao.order.service"* />

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name=*"taotao-order-service"* />

<!-- 使用multicast广播注册中心暴露服务地址 -->

<!-- <dubbo:registry address="multicast://224.5.6.7:1234" /> -->

<!-- <dubbo:registry address="N/A" /> -->

<dubbo:registry protocol=*"zookeeper"* address=*"192.168.37.161:2181"* />

<!-- 用dubbo协议在20880端口暴露服务 -->

<dubbo:protocol name=*"dubbo"* port=*"20885"* />

<!-- 声明需要暴露的服务接口 -->

<!-- <dubbo:service interface="com.taotao.content.service.ContentCategoryService" -->

<!-- ref="contentCategoryServiceImpl" /> -->

</beans>

### applicationContext-trans.xml

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

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

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

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-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>

<!-- 定义事务管理器 -->

<bean id=*"transactionManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"* />

</bean>

<!-- 定义事务策略 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<!-- 传播行为 -->

<tx:method name=*"save\*"* propagation=*"REQUIRED"* />

<tx:method name=*"insert\*"* propagation=*"REQUIRED"* />

<tx:method name=*"add\*"* propagation=*"REQUIRED"* />

<tx:method name=*"create\*"* propagation=*"REQUIRED"* />

<tx:method name=*"delete\*"* propagation=*"REQUIRED"* />

<tx:method name=*"update\*"* propagation=*"REQUIRED"* />

<tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"query\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

</tx:attributes>

</tx:advice>

<aop:config>

<!--pointcut元素定义一个切入点，execution中的第一个星号 用以匹配方法的返回类型，

这里星号表明匹配所有返回类型。 com.abc.dao.\*.\*(..)表明匹配cn.itcast.mybatis.service包下的所有类的所有

方法 -->

<aop:pointcut id=*"myPointcut"* expression=*"execution(\* com.taotao.order.service.\*.\*(..))"* />

<!--将定义好的事务处理策略应用到上述的切入点 -->

<aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"myPointcut"* />

</aop:config>

</beans>

### SqlMapConfig.xml

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

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 开启驼峰自动映射 -->

<settings>

<setting name=*"mapUnderscoreToCamelCase"* value=*"true"* />

</settings>

<plugins>

<!-- 配置分页插件 -->

<!-- com.github.pagehelper为PageHelper类所在包名 -->

<plugin interceptor=*"com.github.pagehelper.PageHelper"*>

<property name=*"dialect"* value=*"mysql"* />

<!-- 设置为true时，使用RowBounds分页会进行count查询 -->

<property name=*"rowBoundsWithCount"* value=*"true"* />

</plugin>

<!-- 配置通用Mapper插件 -->

<plugin interceptor=*"com.github.abel533.mapperhelper.MapperInterceptor"*>

<!--主键自增回写方法,默认值MYSQL,详细说明请看文档 -->

<property name=*"IDENTITY"* value=*"MYSQL"* />

<!--通用Mapper接口，多个通用接口用逗号隔开 -->

<property name=*"mappers"* value=*"com.github.abel533.mapper.Mapper"* />

</plugin>

</plugins>

</configuration>

## 开发准备

taotao-order需要提供订单服务

需要有OrderService接口 和 OrderServiceImpl实现类

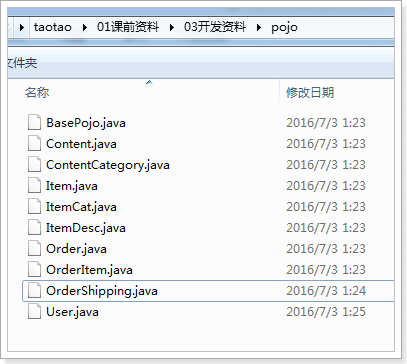
创建两个类，并发布发布服务

需要操作订单表、订单商品表和物流表，

需要在taotao-manager中创建pojo和Mapper

### 创建pojo

pojo地址



在taotao-manager-pojo创建

Order

@Table(name = "tb\_order")

**public** **class** Order **implements** Serializable{

@Id

**private** String orderId;

**private** String payment;

**private** Integer paymentType;

**private** String postFee;

**private** Integer status;

**private** Date createTime;

**private** Date updateTime;

**private** Date paymentTime;

**private** Date consignTime;

**private** Date endTime;

**private** Date closeTime;

**private** String shippingName;

**private** String shippingCode;

**private** Long userId;

**private** String buyerMessage;

**private** String buyerNick;

**private** Integer buyerRate;

// 订单商品

// @Transient和数据库表映射的时候，忽略该属性

@Transient

**private** List<OrderItem> orderItems;

// 订单物流

@Transient

**private** OrderShipping orderShipping;

get/set。。。。。。

}

OrderItem

@Table(name = "tb\_order\_item")

**public** **class** OrderItem **implements** Serializable{

@Id

**private** String itemId;

@Id

**private** String orderId;

**private** Integer num;

**private** String title;

**private** Long price;

**private** Long totalFee;

**private** String picPath;

get/set。。。。。。

}

OrderShipping.

@Table(name = "tb\_order\_shipping")

**public** **class** OrderShipping **implements** Serializable {

@Id

**private** String orderId;

**private** String receiverName;

**private** String receiverPhone;

**private** String receiverMobile;

**private** String receiverState;

**private** String receiverCity;

**private** String receiverDistrict;

**private** String receiverAddress;

**private** String receiverZip;

**private** Date created;

**private** Date updated;

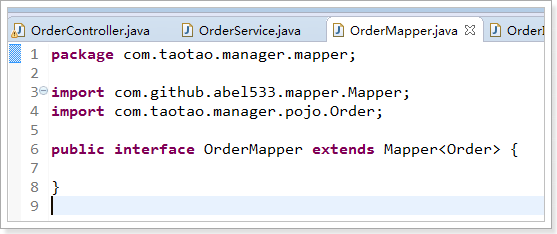
get/set。。。。。。

}

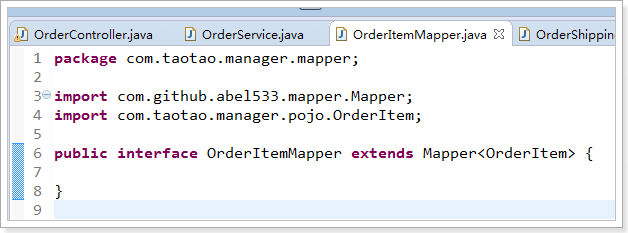
### 创建Mapper

在taotao-manager-mapper创建

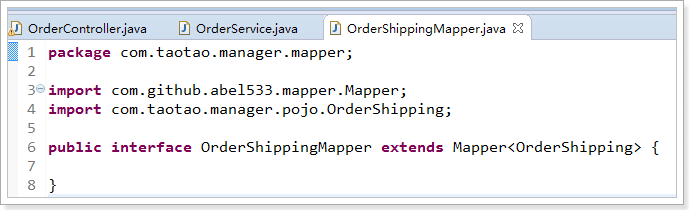
OrderMapper



OrderItemMapper

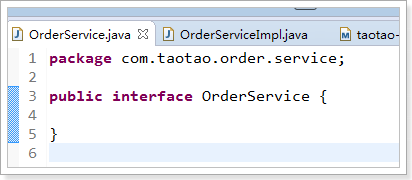


OrderShippingMapper



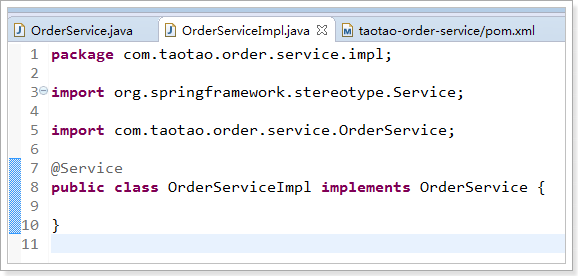
### 创建OrderService

在taotao-order-interface创建



### 创建OrderServiceImpl

在taotao-order-service创建



### 服务声明

#### 服务发布

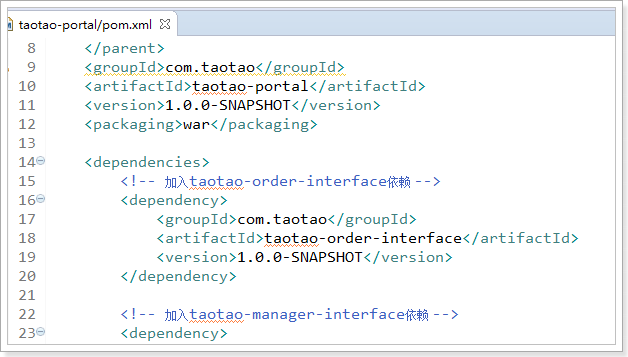
在taotao-order-service声明服务发布



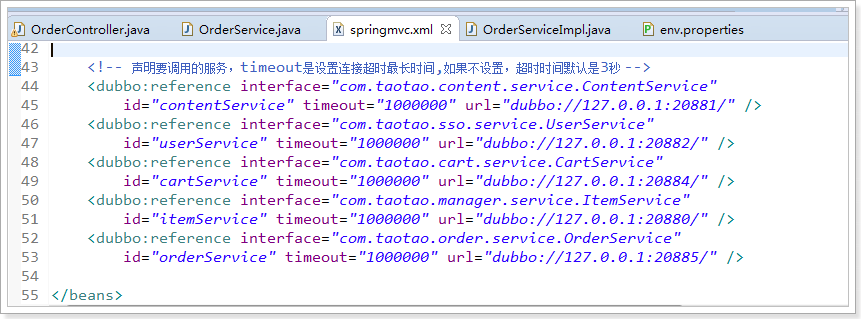
#### 服务调用

在taotao-portal声明服务调用

加入依赖



调用服务



# 实现订单功能

## 实现创建订单

### 分析

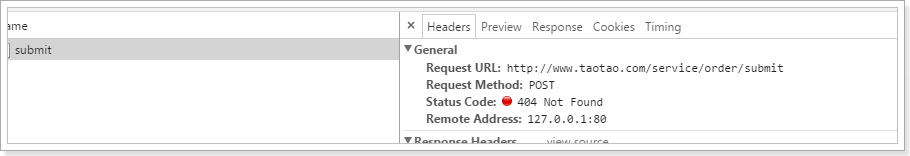
在订单结算页面，当点击提交订单按钮

进入taotao-portal的OrderController，调用order服务进行订单的创建

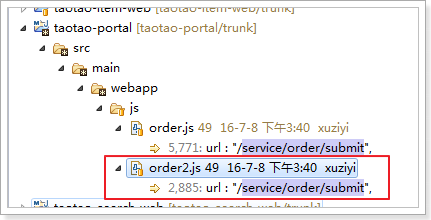


点击按钮发起的请求是POST请求

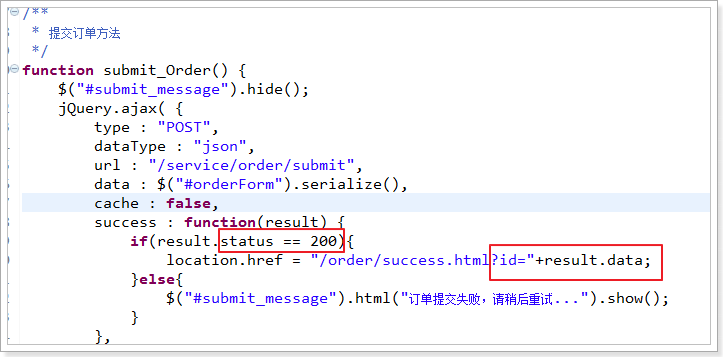
url是：http://www.taotao.com/service/order/submit



在taotao-portal中



\webapp\js\order2.js中的提交逻辑：



### 实现Controller

在taotao-portal实现

// type : "POST",

// dataType : "json",

// url : "/service/order/submit",

/\*\*

\* 创建订单

\*

\* **@param** order

\* **@return**

\*/

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

@ResponseBody

**public** Map<String, Object> saveOrder(Order order) {

// 给订单设置用户信息

User user = UserThreadLocal.*get*();

order.setUserId(user.getId());

order.setBuyerNick(user.getUsername());

//创建订单并返回订单号

String orderId = **this**.orderService.saveOrder(order);

// 创建订单成功，封装返回的数据

Map<String, Object> map = **new** HashMap<String, Object>();

map.put("status", 200);

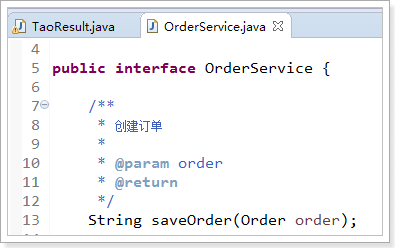
map.put("data", orderId);

**return** map;

}

### 实现Service接口

在taotao-order-interface实现



### 实现Service接口实现类

在taotao-order-service实现

@Service

**public** **class** OrderServiceImpl **implements** OrderService {

@Value("{ORDER\_TAOTAO\_ORDERID\_INCR}")

**private** String ORDER\_TAOTAO\_ORDERID\_INCR;

@Autowired

**private** RedisUtils redisUtils;

@Autowired

**private** OrderMapper orderMapper;

@Autowired

**private** OrderItemMapper orderItemMapper;

@Autowired

**private** OrderShippingMapper orderShippingMapper;

@Override

**public** String saveOrder(Order order) {

// 创建订单号,用户id+唯一数

String orderId = "" + order.getUserId() + **this**.redisUtils.incr(**this**.ORDER\_TAOTAO\_ORDERID\_INCR);

// 保存订单数据

order.setOrderId(orderId);

order.setStatus(1);

order.setCreateTime(**new** Date());

order.setUpdateTime(order.getCreateTime());

**this**.orderMapper.insertSelective(order);

// 保存订单商品数据

List<OrderItem> items = order.getOrderItems();

**for** (OrderItem orderItem : items) {

orderItem.setOrderId(orderId);

**this**.orderItemMapper.insertSelective(orderItem);

}

// 保存订单物流数据

OrderShipping orderShipping = order.getOrderShipping();

orderShipping.setOrderId(orderId);

orderShipping.setCreated(order.getCreateTime());

orderShipping.setUpdated(orderShipping.getCreated());

**this**.orderShippingMapper.insertSelective(orderShipping);

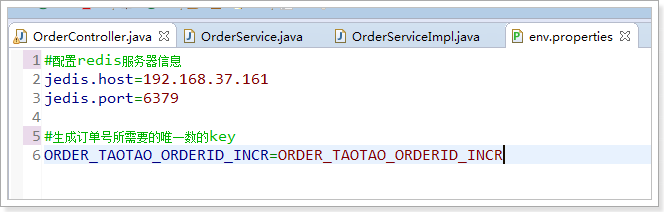
// 返回订单id

**return** orderId;

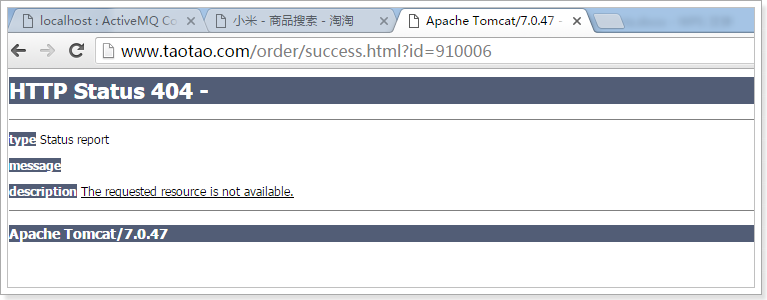
}

}

加入配置文件



### 效果

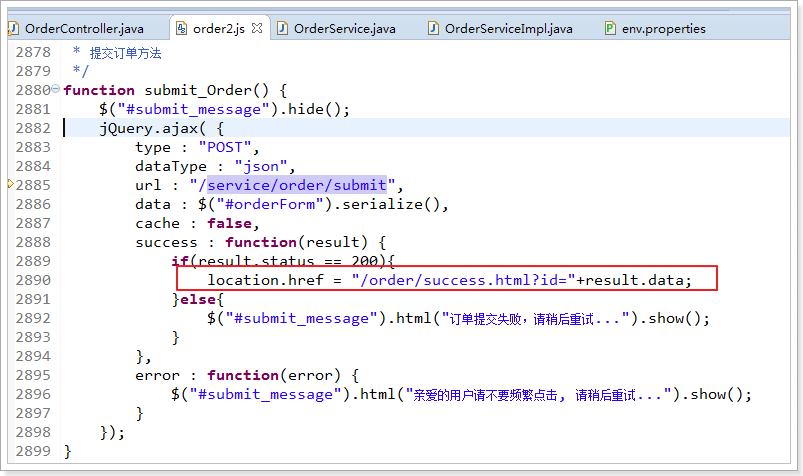


## 实现跳转订单成功页

### 分析

当订单创建成功后，自动跳转成功页面

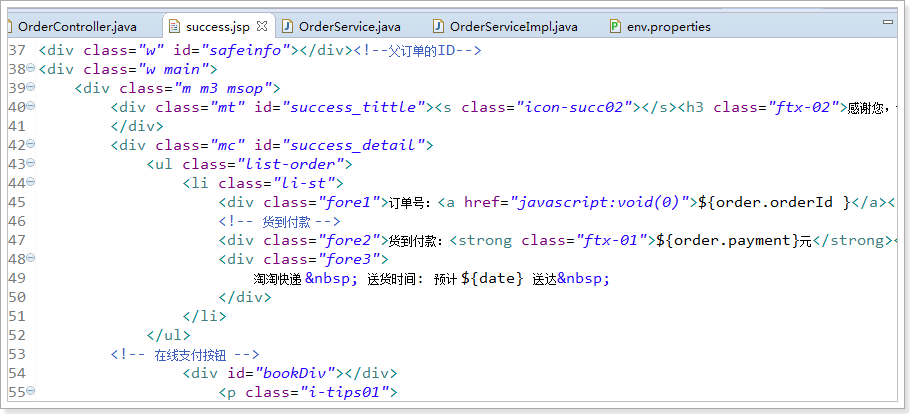
携带请求参数是订单id



成功页面taotao-portal是\webapp\WEB-INF\views\success.jsp

需要传递订单数据和成功送达时间

订单数据调用服务获取，送达时间为当前时间后两天



### 加入时间组件依赖

送达时间可以使用joda-time时间组件

在taotao-portal加入依赖



### 实现Controller

在taotao-portal实现

// http://www.taotao.com/order/success.html?id=910006

/\*\*

\* 跳转到订单成功页

\*

\* **@param** orderId

\* **@param** model

\* **@return**

\*/

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

**public** String success(@RequestParam("id") String orderId, Model model) {

// 根据orderid查询订单，使用订单服务

Order order = **this**.orderService.queryOrderByOrderId(orderId);

// 获取当前时间的两天后，即时送达时间

String date = **new** DateTime().plusDays(2).toString("yyyy年MM月dd日HH时mm分ss秒SSS毫秒");

// 把订单数据放到模型中

model.addAttribute("order", order);

// 把送达时间放到模型中

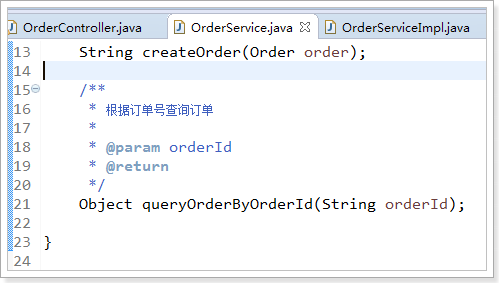
model.addAttribute("date", date);

**return** "success";

}

### 实现Service接口

在taotao-order-interface实现



### 实现Service接口实现类

在taotao-order-service实现

@Override

**public** Object queryOrderByOrderId(String orderId) {

// 查询订单数据

Order order = **this**.orderMapper.selectByPrimaryKey(orderId);

// 查询订单商品数据，并设置到订单中

OrderItem param = **new** OrderItem();

// 设置查询条件

param.setOrderId(orderId);

List<OrderItem> items = **this**.orderItemMapper.select(param);

// 设置到订单中

order.setOrderItems(items);

// 查询订单物流数据，并设置到订单中

OrderShipping orderShipping = **this**.orderShippingMapper.selectByPrimaryKey(orderId);

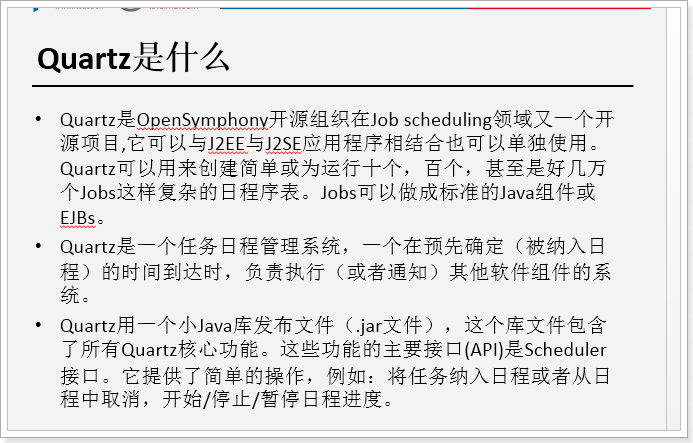
order.setOrderShipping(orderShipping);

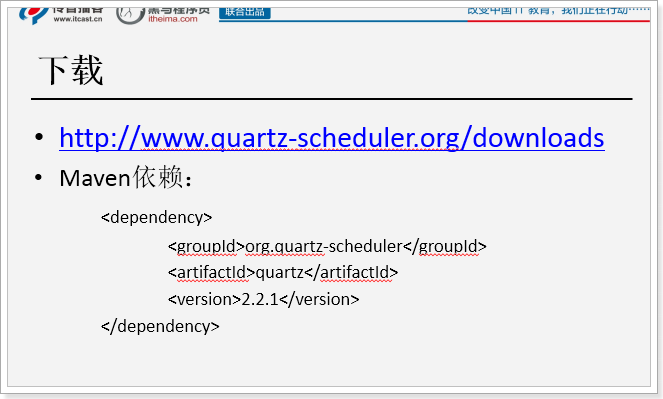
// 返回订单

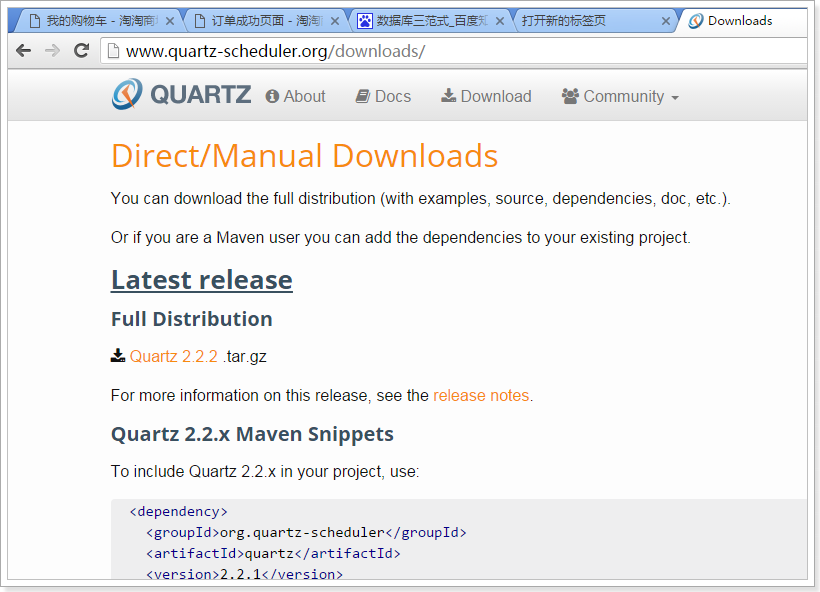
**return** order;

}

# Quartz定时任务

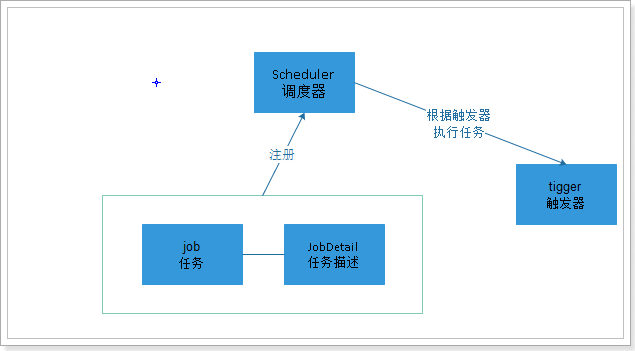






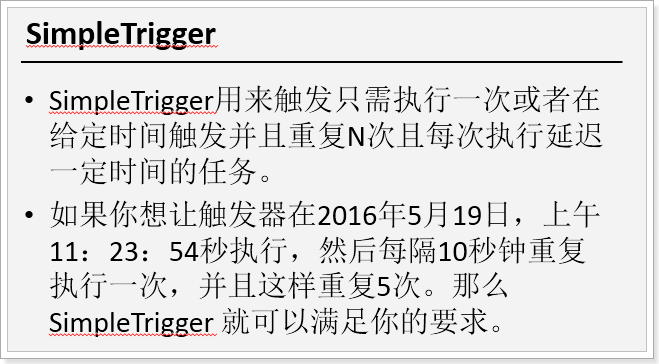
## 核心接口





## tigger触发器

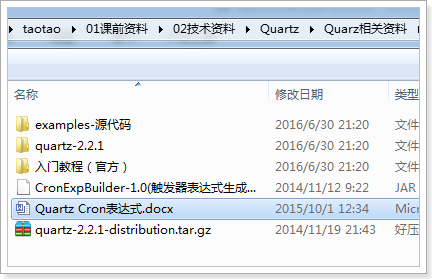
* SimpleTrigger
* CronTrigger



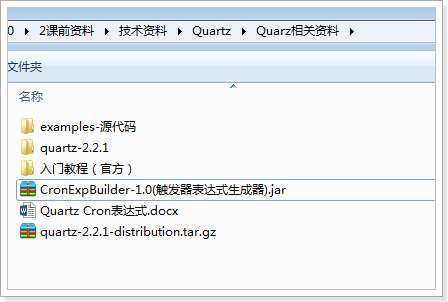


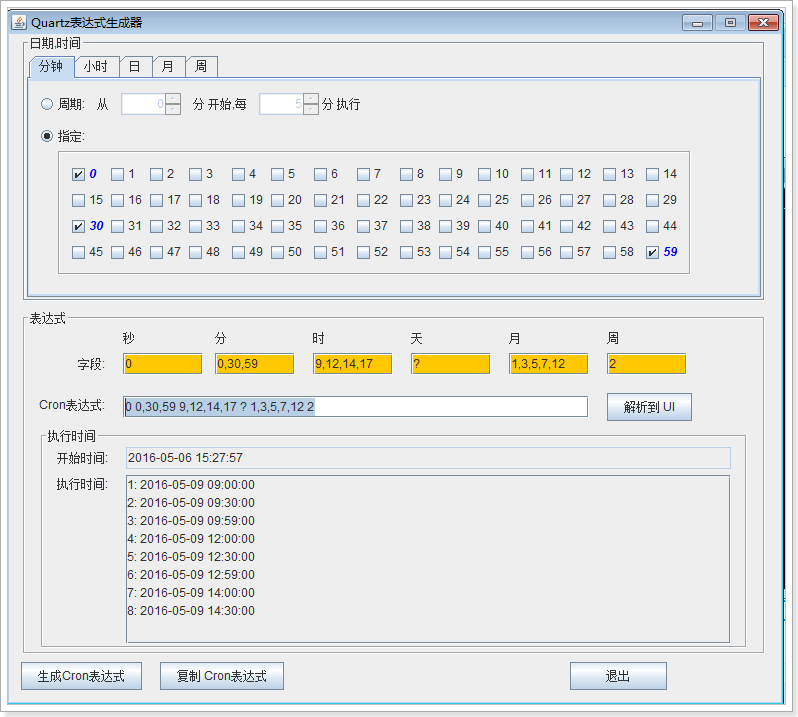
## 表达式



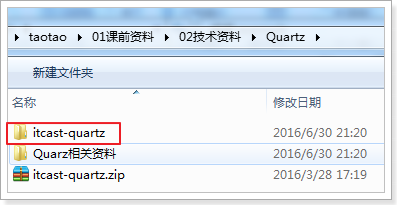


表达式生成器



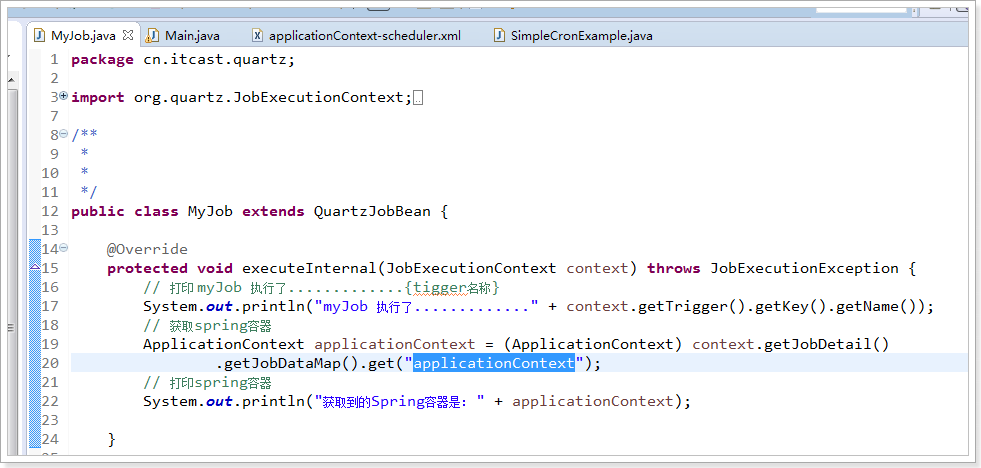


## Quartz案例学习



## Quartz和spring整合

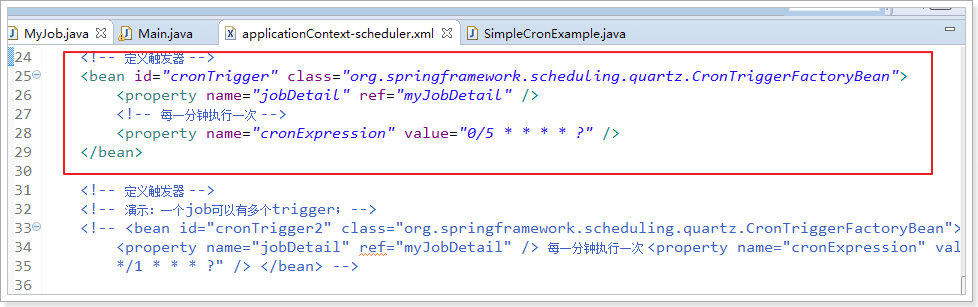
### 定义任务



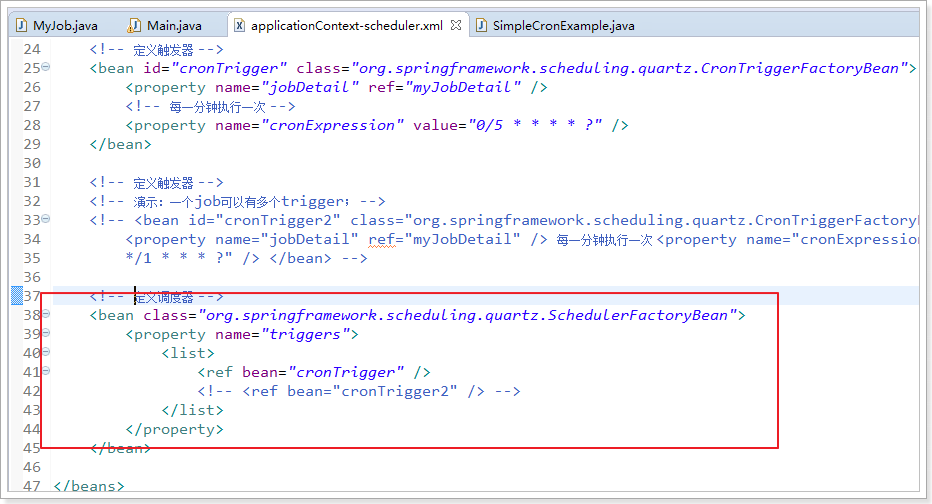
### 配置任务由spring管理



### 配置tigger由spring管理



### 配置调度器



# 实现定时删除无效订单

## 分析

无效订单

支付方式是在线支付

创建时间是两天前

订单状态未付款状态

如何删除：

核心数据不能做物理删除，只能逻辑删除，意思就是，修改订单状态

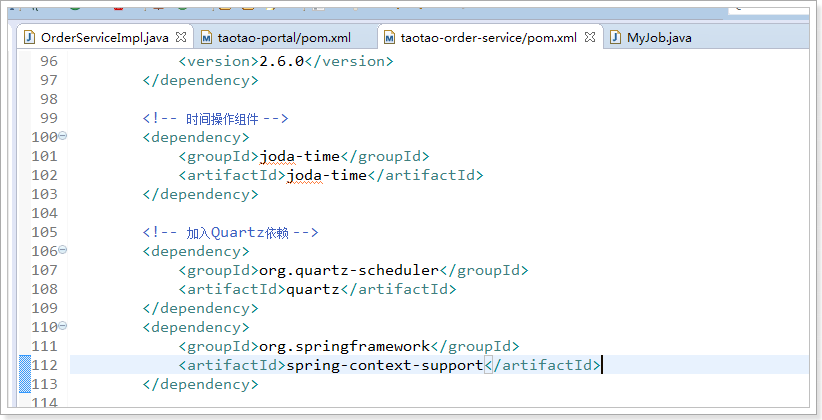
把订单状态修改为关闭订单

把订单的关闭时间修改为当前时间

每隔一分钟删除一次

## 加入依赖

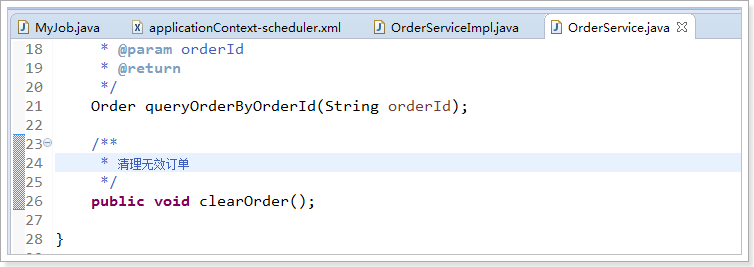
taotao-order-service加入



## 实现清理无效订单方法

### 实现service接口

在taotao-order-interface实现



### 实现service接口实现类

在taotao-order-service实现



## 整合Quartz到项目中

### 编写配置文件

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

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

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

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

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-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>

<!-- 定义任务bean -->

<bean name=*"myJobDetail"*

class=*"org.springframework.scheduling.quartz.JobDetailFactoryBean"*>

<!-- 指定具体的job类 -->

<property name=*"jobClass"* value=*"com.taotao.order.service.quartz.MyJob"* />

<!-- 指定job的名称 -->

<property name=*"name"* value=*"myJob"* />

<!-- 指定job的分组 -->

<property name=*"group"* value=*"jobs"* />

<!-- 必须设置为true，如果为false，当没有活动的触发器与之关联时会在调度器中删除该任务 -->

<property name=*"durability"* value=*"true"* />

<!-- 指定spring容器的key，如果不设定在job中的jobmap中是获取不到spring容器的 -->

<property name=*"applicationContextJobDataKey"* value=*"applicationContext"* />

</bean>

<!-- 定义触发器 -->

<bean id=*"cronTrigger"*

class=*"org.springframework.scheduling.quartz.CronTriggerFactoryBean"*>

<property name=*"jobDetail"* ref=*"myJobDetail"* />

<!-- 每一分钟执行一次 -->

<property name=*"cronExpression"* value=*"0 0/1 \* \* \* ?"* />

</bean>

<!-- 定义调度器 -->

<bean class=*"org.springframework.scheduling.quartz.SchedulerFactoryBean"*>

<property name=*"triggers"*>

<list>

<ref bean=*"cronTrigger"* />

</list>

</property>

</bean>

</beans>

### 编写任务bean

编写任务，执行清理方法

