# 单点登录SSO

## 用户登出操作

### url请求地址

<http://www.jt.com/user/logout.html>

当用户登出之后,应该重定向到系统首页 业务必然

### 编辑Controller

/\*\*

\* 实现用户登出操作

\* 1.删除redis request对象~~cookie中~~~JT\_TICKET~~key

\* 2.删除cookie

\* **@return**

\*/

@RequestMapping("/logout")

**public** String logout(HttpServletRequest request,HttpServletResponse response) {

Cookie[] cookies = request.getCookies();

**if**(cookies.length !=0) {

String token = **null**;

**for** (Cookie cookie : cookies) {

**if**("JT\_TICKET".equals(cookie.getName())) {

token = cookie.getValue();

**break**;

}

}

//判断token数据有值 删除redis/删除cookie

**if**(!StringUtils.*isEmpty*(token)) {

jedisCluster.del(token);

Cookie cookie = **new** Cookie("JT\_TICKET","");

cookie.setMaxAge(0);//立即删除cookie

cookie.setPath("/");

cookie.setDomain("jt.com");

response.addCookie(cookie);

}

}

//当用户登出时,页面重定向到系统首页

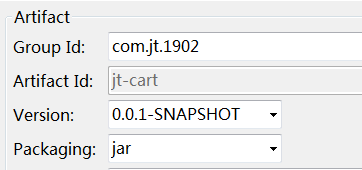
**return** "redirect:/";

}

# 京淘购物车实现

## 创建项目jt-cart

### 创建项目



### 编辑POM.xml文件

编辑jt-cart中的pom.xml文件.注意添加继承/依赖/maven启动项等

<parent>

<groupId>com.jt.1902</groupId>

<artifactId>jt</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>jt-cart</artifactId>

<dependencies>

<dependency>

<groupId>com.jt.1902</groupId>

<artifactId>jt-common</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>

</dependencies>

<build>

<plugins>

<!--跳过测试类打包 -->

<plugin>

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

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<skip>true</skip>

</configuration>

</plugin>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

### 编辑YML配置文件

注意修改:提供者服务名称/服务端口号

server:

port: 8094

servlet:

context-path: /

spring:

datasource:

#引入druid数据源

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/jtdb?serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true

username: root

password: root

#Springboot整合dubbo

dubbo:

scan:

basePackages: com.jt #包扫描dubbo的service注解

application:

name: provider-cart #提供者服务名称

registry: #链接zookeeper

address: zookeeper://192.168.175.129:2181?backup=192.168.175.129:2182,192.168.175.129:2183

protocol:

name: dubbo

port: 20881 #服务通信端口 消费者链接提供者端口

#mybatis-plush配置

mybatis-plus:

type-aliases-package: com.jt.pojo

mapper-locations: classpath:/mybatis/mappers/\*.xml

configuration:

map-underscore-to-camel-case: true

#引入日志信息

logging:

level:

com.jt.mapper: debug

### 编辑POJO对象

说明:在jt-common中添加pojo对象

@Data

@Accessors(chain = **true**)

@TableName("tb\_cart")

**public** **class** Cart **extends** BasePojo{

@TableId(type = IdType.***AUTO***)

**private** Long id;

**private** Long userId;

**private** Long itemId;

**private** String itemTitle;

**private** String itemImage;

**private** Long itemPrice;

**private** Integer num;

}

### 编辑Dubbo接口

说明:在jt-common中添加dubbo接口

//在jt-common中添加接口

**public** **interface** DubboCartService {

}

### 编辑提供者实现类

说明:编辑提供者实现类.

//编辑提供者jt-cart中的实现类

@Service(timeout = 3000)

**public** **class** DubboCartServiceImpl **implements** DubboCartService {

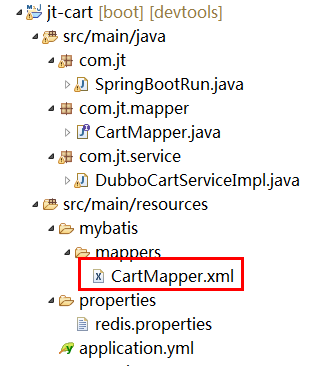
@Autowired

**private** CartMapper cartMapper;

}

### 编辑Mybatis-映射文件名称

修改namespace路径



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

<!DOCTYPE mapper

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

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

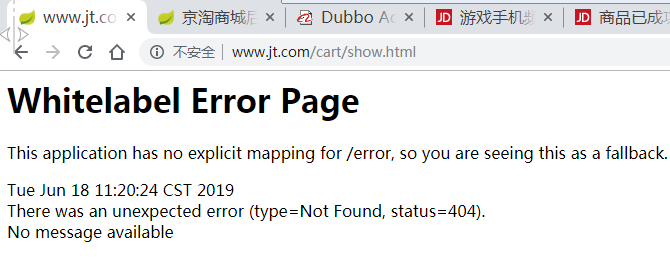
<mapper namespace=*"com.jt.mapper.CartMapper"*>

</mapper>

## 购物车列表展现

### 业务说明

当用户点击购物车按钮时,发起url请求,要求跳转到用户购物车展现页面



2.页面分析

当跳转页面时,跳转到cart.jsp中.将购物车信息进行循环展现

<c:forEach items=*"${cartList}"* var=*"cart"*>

### 编辑Controller

说明:在消费者jt-web中编辑CartController

//因为需要跳转页面 所以不能使用restController

//如果后期返回值是json串.则在方法上添加@ResponseBody注解

@Controller

@RequestMapping("/cart")

**public** **class** CartController {

@Reference(timeout=3000,check=**false**)

**private** DubboCartService cartService;

/\*\*

\* 1.实现商品列表信息展现

\* 2.页面取值: ${cartList}

\*/

@RequestMapping("/show")

**public** String findCartList(Model model) {

Long userId = 7L; //暂时写死

List<Cart> cartList =

cartService.findCartListByUserId(userId);

model.addAttribute("cartList", cartList);

**return** "cart";//返回页面逻辑名称

}

}

### 编辑Service实现类

//编辑提供者jt-cart中的实现类

@Service(timeout = 3000)

**public** **class** DubboCartServiceImpl **implements** DubboCartService {

@Autowired

**private** CartMapper cartMapper;

@Override

**public** List<Cart> findCartListByUserId(Long userId) {

QueryWrapper<Cart> queryWrapper =

**new** QueryWrapper<Cart>();

queryWrapper.eq("user\_id", userId);

**return** cartMapper.selectList(queryWrapper);

}

}

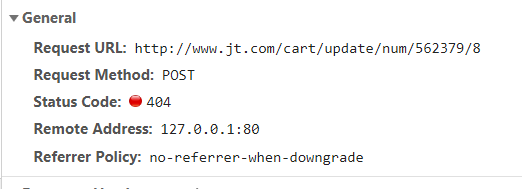
### 页面效果



## 商品数量修改

### 页面分析

1. url分析



1. 页面js分析

$(".increment").click(**function**(){//＋

**var** \_thisInput = $(**this**).siblings("input");

\_thisInput.val(eval(\_thisInput.val()) + 1);

$.post("/cart/update/num/"+\_thisInput.attr("itemId")+"/"+\_thisInput.val(),**function**(data){

TTCart.refreshTotalPrice();

});

});

### 编辑Controller

/\*\*

\* 实现购物车数量的修改

\* url地址:http://www.jt.com/cart/update/num/562379/8

\* 规定:如果url参数中使用restFul风格获取数据时

\* 介绍参数是对象并且属性匹配.则可以使用对象接收

\*/

@RequestMapping("/update/num/{itemId}/{num}")

@ResponseBody

**public** SysResult updateCartNum(Cart cart) {

**try** {

Long userId = 7L;

cart.setUserId(userId);

cartService.updateCartNum(cart);

**return** SysResult.*ok*();

} **catch** (Exception e) {

e.printStackTrace();

**return** SysResult.*fail*();

}

}

### 编辑提供者Service

/\*\*

\* 更新数据信息: num/updated

\* 判断条件:where user\_id and item\_id

\*/

@Transactional //事务控制

@Override

**public** **void** updateCartNum(Cart cart) {

Cart tempCart = **new** Cart();

tempCart.setNum(cart.getNum())

.setUpdated(**new** Date());

UpdateWrapper<Cart> updateWrapper =

**new** UpdateWrapper<Cart>();

updateWrapper.eq("user\_id", cart.getUserId())

.eq("item\_id", cart.getItemId());

cartMapper.update(tempCart, updateWrapper);

}

## 购物车删除

### 页面分析

1. url分析

<http://www.jt.com/cart/delete/562379.html>

说明请求成功之后,需要**重定向**到购物车展现页面.

### 编辑消费者Controller

/\*\*

\* 实现购物车删除操作

\*/

@RequestMapping("/delete/{itemId}")

**public** String deleteCart(Cart cart) {

Long userId = 7L;

cart.setUserId(userId);

cartService.deleteCart(cart);

//重定向到购物车列表页面

**return** "redirect:/cart/show.html";

}

### 编辑提供者Service

/\*\*

\* 将对象中不为null的属性当做where条件

\* 前提:保证cart中只能有2个属性不为null.

\*/

@Override

**public** **void** deleteCart(Cart cart) {

QueryWrapper<Cart> queryWrapper =

**new** QueryWrapper<Cart>(cart);

cartMapper.delete(queryWrapper);

}

## 购物车新增

### 业务说明

当用户通过商品详情页点击按钮式.执行添加购物车操作.



### 页面分析

url: <http://www.jt.com/cart/add/1474391972.html>

页面跳转:跳转到购物车展现页面.

页面js分析: item.jsp中

//利用post传值

**function** addCart(){

**var** url = "http://www.jt.com/cart/add/${item.id}.html";

document.forms[0].action = url; //js设置提交链接

document.forms[0].submit(); //js表单提交

}

Form表单信息:

<form id=*"cartForm"* method=*"post"*>

<input class=*"text"* id=*"buy-num"* name=*"num"* value=*"1"* onkeyup="setAmount.modify('#buy-num');"/>

<input type=*"hidden"* class=*"text"* name=*"itemTitle"* value=*"${item.title }"*/>

<input type=*"hidden"* class=*"text"* name=*"itemImage"* value=*"${item.images[0]}"*/>

<input type=*"hidden"* class=*"text"* name=*"itemPrice"* value=*"${item.price}"*/>

</form>

### 编辑消费者Controller

编辑购物车新增,新增完成之后跳转购物车展现页面.

/\*\*

\* 新增购物车

\* 页面表单提交 发起post请求

\* 携带购物车参数

\*/

@RequestMapping("/add/{itemId}")

**public** String insertCart(Cart cart) {

Long userId = 7L;

cart.setUserId(userId);

cartService.insertCart(cart);

//新增数据之后.展现购物车列表信息

**return** "redirect:/cart/show.html";

}

### 编辑提供者Service

/\*\*

\* 新增业务实现

\* 1.用户第一次新增可以直接入库

\* 2.用户不是第一次入库.应该只做数量的修改.

\*

\* 根据:itemId和userId查询数据

\*/

@Override

@Transactional

**public** **void** insertCart(Cart cart) {

QueryWrapper<Cart> queryWrapper = **new** QueryWrapper<>();

queryWrapper.eq("user\_id", cart.getUserId())

.eq("item\_id", cart.getItemId());

Cart cartDB = cartMapper.selectOne(queryWrapper);

**if**(cartDB == **null**) {

//用户第一次购买商品 可以直接入库

cart.setCreated(**new** Date())

.setUpdated(cart.getCreated());

cartMapper.insert(cart);

}**else** {

//表示用户多次添加购物车 只做数量的修改

**int** num = cart.getNum() + cartDB.getNum();

cartDB.setNum(num)

.setUpdated(**new** Date());

cartMapper.updateById(cartDB);

/\*\*

\* 衍生知识

\* 修改操作时,除了主键之外都要更新不为null的数据

\* sql: update tb\_cart

\* set num=#{num},updated=#{updated},

\* user\_id=#{userId},item\_id=#{itemId},

\* xxxxxxxxxxxx

\* where id = #{id}

\* 目的:

\* 有时自己手写sql可能效果更好!!!

\*/

}

}

### 页面效果



## 购物车权限实现

### 业务需求

当用户没有登录条件下.不允许访问购物车/订单/支付/物流等相关涉密操作.

如何实现:

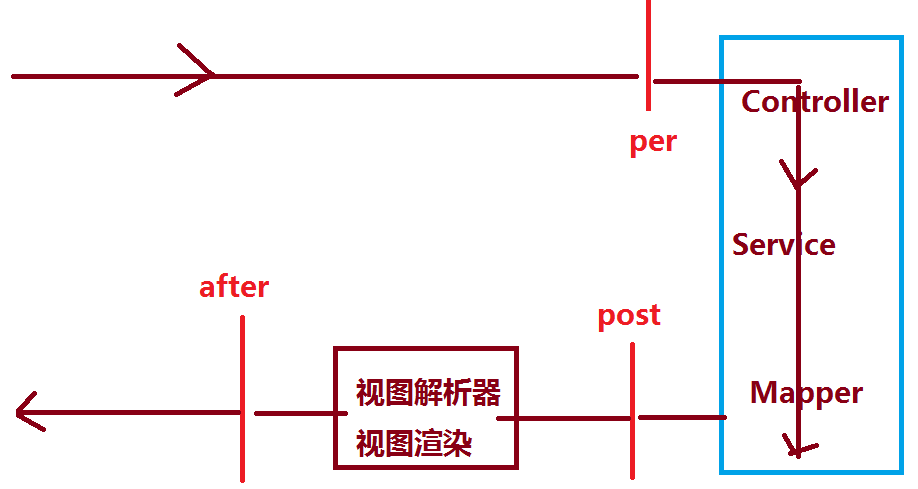
AOP: 主要实现了业务层解耦.但是无法直接获取request/response对象

拦截器: 由于需要通过url地址判断用户是否有权限.所有拦截器是首选.

### 实现思路

利用SpringMVC的拦截器.判断用户是否登录.如果用户已经登录则拦截器放行用户请求.如果用户没有登录,应该重定向到用户登录页面.

### 拦截器工作原理



### 用户拦截器定义

说明:在jt-web中添加拦截器操作类.

@Component //将拦截器交给spring容器管理

**public** **class** UserInterceptor **implements** HandlerInterceptor{

@Autowired

**private** JedisCluster jedisCluster;

/\*\*

\* 在spring4版本中要求必须重写3个方法,不管是否需要

\* 在spring5版本中在接口中添加default属性,则省略不写.

\*/

/\*\*

\* 返回值结果:

\* true: 拦截放行.

\* false: 请求拦截. 重定向到登录页面

\*

\* 业务逻辑:

\* 1.获取Cookie数据

\* 2.从cookie中获取token(TICKET)

\* 3.判断redis缓存服务器中是否有数据.

\*/

@Override

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

**throws** Exception {

String token = **null**;

//1.获取Cookie信息

Cookie[] cookies = request.getCookies();

**for** (Cookie cookie : cookies) {

**if**("JT\_TICKET".equals(cookie.getName())) {

token = cookie.getValue();

**break**;

}

}

//2.判断token是否有效

**if**(!StringUtils.*isEmpty*(token)) {

//4.判断redis中是否有数据

String userJSON = jedisCluster.get(token);

**if**(!StringUtils.*isEmpty*(userJSON)) {

//redis中有用户数据.拦截放行

**return** **true**;

}

}

//3.重定向到用户登录页面

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

**return** **false**;//表示拦截

}

}

### 拦截器配置

说明:编辑jt-web中配置类 MVCConfigurer.添加拦截器配置

@Configuration

**public** **class** MvcConfigurer **implements** WebMvcConfigurer{

@Autowired

**private** UserInterceptor userInterceptor;

//开启匹配后缀型配置

@Override

**public** **void** configurePathMatch(PathMatchConfigurer configurer) {

configurer.setUseSuffixPatternMatch(**true**);

}

//新增拦截器配置

//url:http://www.jt.com/cart/xxxxx

//url:http://www.jt.com/order/xxxxx

// /\* 只能拦截下一级路径

// /\*\* 拦截所有的路径

@Override

**public** **void** addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(userInterceptor)

.addPathPatterns("/cart/\*\*","/order/\*\*");

}

}

### 作业:

要求:动态获取当前登录用户id信息!!!!

提示:拦截器 request对象/ThreadLocal