# 单点登录

## 面试问题

### COOKIE禁用怎么办???

例子:例如京东网站,当cookie禁用时展现会有问题.

原因:如果使用Cookie可以简化服务端数据的处理的方式.但是如果需要解决cookie禁用的问题.则后台的业务逻辑处理需要改变后台的传参方式.

1.解决策略一:**URL重写技术**. 每次用户请求都需要拼接?sessionId=100100

<http://www.jt.com?sessionId=1231123>

<http://wwww.cart.com?sessionId=12312123>

一般大型公司为了运行的稳定一般不会进行处理.另外用户禁用COOKIE的的人数较少.不能因为少部分人影响整个网站的运行.

2.解决策略二:实现SessionId的共享.利用Nginx实现.具体操作我找找资料

### SSO实现必须使用Redis吗????

实现单点核心:将用户信息保存到第三方.并且第三方服务器都能访问的.

软件种类:**1.redis** 2.**zookeeper** 3.mysql数据库

zookeeper中有树形结构:

一级目录

二级目录

三级目录…..

## 实现用户登出操作

### 业务需求

当用户点击登出按钮时,应该将redis数据删除.Cookie数据清空.并且重定向到系统首页.

### 编辑Controller

@RequestMapping("/logout")

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

//1.删除redis??? key~~~~jt\_ticket~~~Cookie

Cookie[] cookies = request.getCookies();

String token = **null**;

**if**(cookies.length>0) {

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

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

//获取指定数据的值

token = cookie.getValue();

**break**;

}

}

}

//要求token不为null才做处理

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

jedisCluster.del(token);

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

cookie.setMaxAge(0);//立即删除

cookie.setPath("/");

cookie.setDomain("jt.com");

response.addCookie(cookie);

}

**return** "redirect:/";//重定向到系统首页

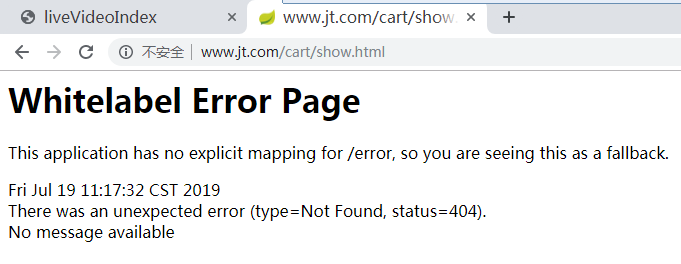
}

# 京淘购物车实现

## 购物车展现

### 页面分析

1. url分析



### 编辑Controller

@Controller

@RequestMapping("/cart")

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

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

**private** DubboCartService cartService;

/\*\*

\* 当用户点击购物车按钮时.应该展现用户的购物记录

\* 业务实现:

\* 查询用户购物行为 userId 暂时写死

\* **@return**

\*/

@RequestMapping("/show")

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

Long userId = 7L;

List<Cart> cartList =

cartService.findCartListByUserId(userId);

model.addAttribute("cartList", cartList);

**return** "cart"; //跳转购物车页面

}

}

### 编辑Service

@Service(timeout = 3000)

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

@Autowired

**private** CartMapper cartMapper;

//根据userId查询cartList集合

@Override

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

QueryWrapper<Cart> queryWrapper =

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

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

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

}

}

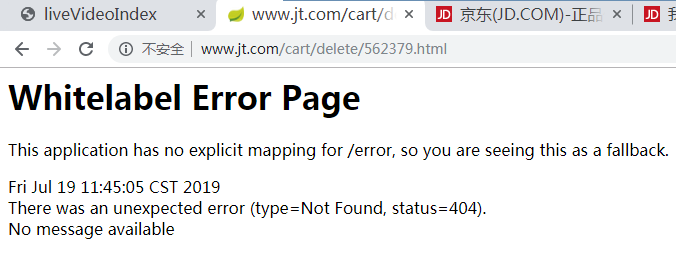
### 页面效果展现



## 实现购物车删除

### 业务需求

当用户点击删除按钮时.应该按照user\_id和item\_id删除数据.应该重定向到用户的购物车页面.



### 编辑Controller

/\*\*

\* 如果rest风格接收参数时与对象的属性名称一致

\* 则可以使用对象接收

\* **@param** itemId

\* **@return**

\*/

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

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

Long userId = 7L;

cart.setUserId(userId);

cartService.deleteCart(cart);

**return** "redirect:/cart/show.html";//应该采用重定向方式获取数据

}

### 编辑Service

/\*\*

\* 利用对象中不为null的属性充当where条件

\*/

@Override

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

//delete from tb\_cart where user\_id=7 and item\_id=562379

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

cartMapper.delete(queryWrapper);

}

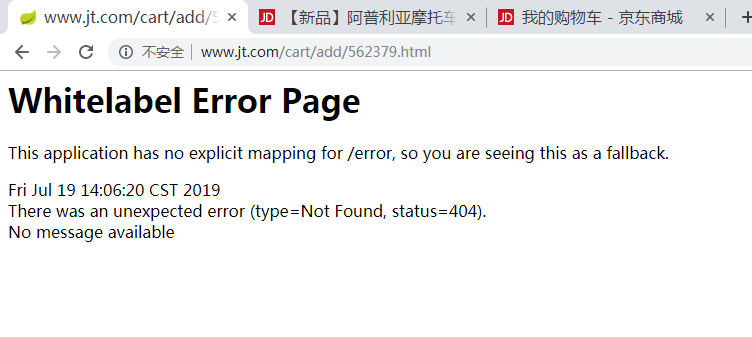
## 商品购物车新增

### 业务需求

当用户通过商品详情添加购物车时,应该跳转购物车列表页面.如果同一件商品被用户新增多次购物车.应该只做数量的更新.不做记录的新增.

### 页面分析

1.页面url跳转



1. 页面js分析

<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>

页面表单提交方式

//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

}

### 编辑Controller

/\*\*

\* 该方法利用页面的表单提交获取参数.

\* 之后应该跳转购物车展现页面

\* **@param** cart

\* **@return**

\*/

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

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

Long userId = 7L;

cart.setUserId(userId);

cartService.saveCart(cart);

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

}

### 编辑Service

/\*\*

\* 思路:

\* 根据user\_id item\_id查询数据库.

\* null 表示第一次购买 新增入库

\* !null 表示已经购买过 做数量的更新.

\*/

@Override

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

QueryWrapper<Cart> queryWrapper =

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

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

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

Cart cartDB = cartMapper.selectOne(queryWrapper);

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

//表示第一次新增

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

.setUpdated(cart.getCreated());

cartMapper.insert(cart);

}**else** {

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

Cart cartTemp = **new** Cart();

cartTemp.setId(cartDB.getId()) //设定主键

.setNum(num)

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

//sql:update tb\_cart set num=3,updated=now() where id=100

cartMapper.updateById(cartTemp);

}

}

### 页面效果



## 购物车数量的修改

### 页面分析

1. 页面URL地址



2.页面JS

$(".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){

//ajax正确页面返回

TTCart.refreshTotalPrice();

});

});

### 编辑Controller

//http://www.jt.com/cart/update/num/562379/10

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

@ResponseBody

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

Long userId = 7L;

cart.setUserId(userId);

cartService.updateCartNum(cart);

**return** SysResult.*success*();

}

### 编辑Service

//应该更新数量 set num = 100,updated=时间 where user\_id=xx and item\_id=xxx

/\*\*

\* entity:要修改的数据

\* updateWrapper:修改的where条件

\*/

@Override

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

Cart cartTemp = **new** Cart();

cartTemp.setNum(cart.getNum())

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

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

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

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

cartMapper.update(cartTemp, updateWrapper);

}

## 实现京淘项目权限控制

### 业务说明

需求:如果用户进行涉密操作时,要求必须用户先登录.

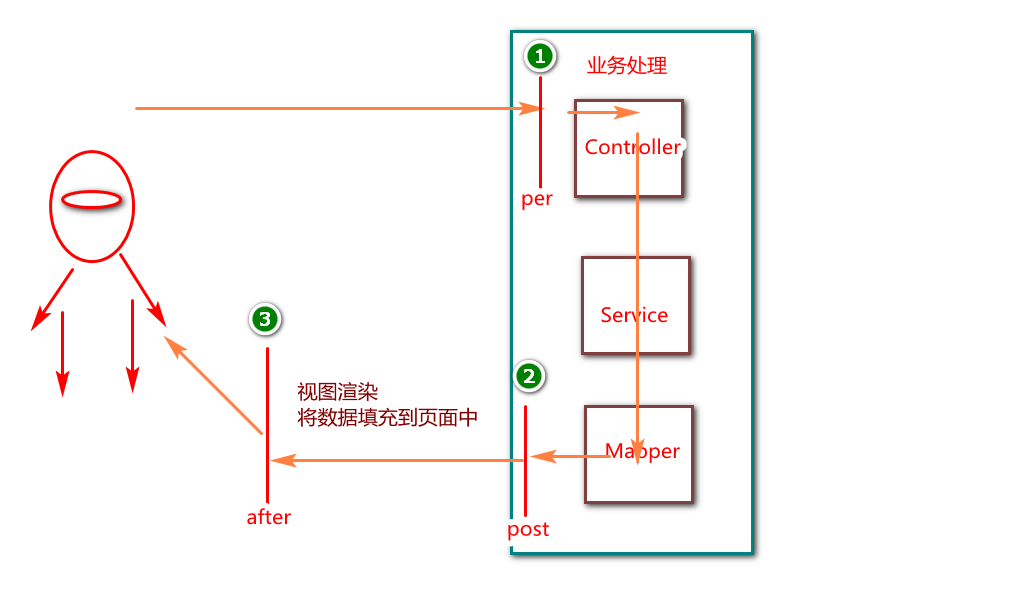
例子:购物车操作,订单操作,物流操作….

解决方案:

1.shiro:一般不会使用在分布式系统中!!! 一般使用shiro在单独的业务系统CRM OA 拦截器封装了….

2.一般在分布式系统中使用传统的**拦截器**的方式手动的控制权限.

### 拦截器调用执行过程



### 传参方式

#### 利用request对象传参

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

User user = ObjectMapperUtil.*toObject*(userJSON, User.**class**);

//将user数据通过request对象封装.

request.setAttribute("JT\_USER",user);

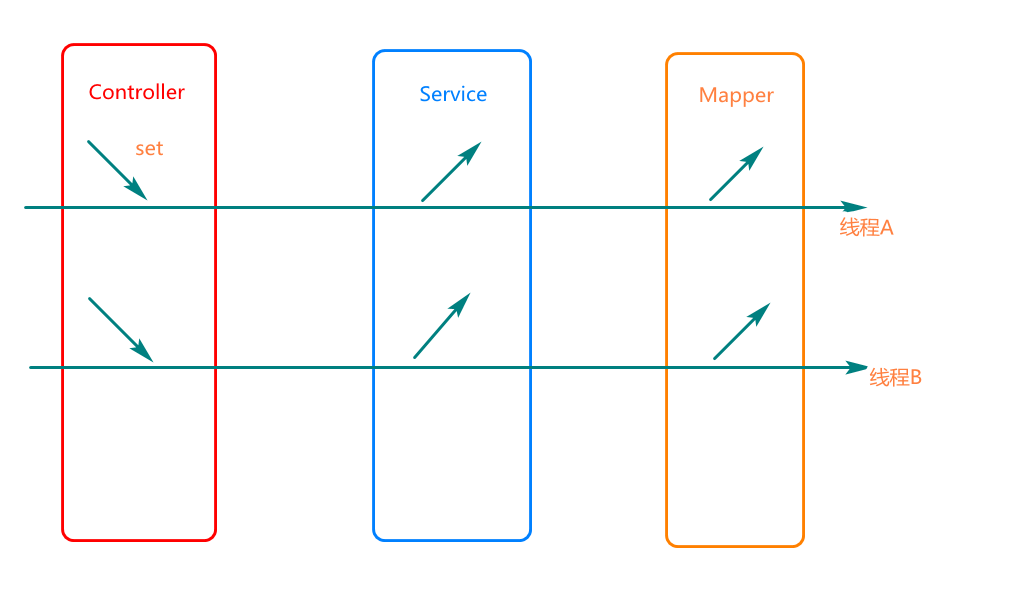
**return** **true**;

}

#### 利用ThreadLocal传参

名称:**本地线程变量**

作用:可以在同一个线程内实现数据的共享!!!



### 封装ThreadLocal工具类

**public** **class** UserThreadLocal {

**private** **static** ThreadLocal<User> *threadLocal* = **new** ThreadLocal<>();

**public** **static** **void** set(User user) {

*threadLocal*.set(user);

}

**public** **static** User get() {

**return** *threadLocal*.get();

}

//防止内存泄漏remove()

**public** **static** **void** remove() {

*threadLocal*.remove();

}

}

### 编辑拦截器

/\*\*

\*

\* **@author** Administrator

\* Spring4版本 HandlerInterceptor必须重写3个方法

\* Spring5版本 HandlerInterceptor都有默认值

\*/

@Component

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

@Autowired

**private** JedisCluster jediscluster;

/\*\*

\* boolean:

\* true: 表示放行

\* false:表示拦截 一般配合重定向使用

\*

\* 业务实现步骤

\* 1.获取用户Cookie中的token信息.

\* 2.校验数据是否有效

\* 3.校验redis中是否有数据

\* 如果上述操作正确无误.返回true

\* 否则return false 重定向到登录页面

\*

\*/

@Override

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

**throws** Exception {

//1.获取cookie数据

Cookie[] cookies = request.getCookies();

String token = **null**;

**if**(cookies.length>0) {

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

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

//获取指定数据的值

token = cookie.getValue();

**break**;

}

}

}

//2.校验token的有效性

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

//2.1校验redis中是否有数据

String userJSON = jediscluster.get(token);

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

User user = ObjectMapperUtil.*toObject*(userJSON, User.**class**);

//将user数据通过request对象封装.

request.setAttribute("JT\_USER",user);

//利用threadLocal实现数据共享

UserThreadLocal.*set*(user);

**return** **true**;

}

}

//必须重定向

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

**return** **false**;

}

@Override

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

**throws** Exception {

UserThreadLocal.*remove*();

}

}

### 编辑拦截器配置文件

@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**);

}

//定义拦截器 添加需要匹配的路径

/\*\*

\* /\*\* 拦截请求的多级目录 /cart/aa/bb/ccc

\* /\* 拦截请求的一级目录 /cart/aa

\*/

@Override

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

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

//如果有多个拦截器,可以addInterceptor多次

}

}

### 动态获取UserId

@RequestMapping("/show")

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

//User user = (User) request.getAttribute("JT\_USER");

**Long userId = UserThreadLocal.*get*().getId();**

List<Cart> cartList =

cartService.findCartListByUserId(userId);

model.addAttribute("cartList", cartList);

**return** "cart"; //跳转购物车页面

}