# 知识回顾

## 重定向和转发问题

### 重定向

1. 请求多次.发起多次request请求.
2. URL的请求地址必然发生改变.
3. 请求速度变慢了.中间经过多次请求.
4. 使用重定向时,request参数将不能传递.
5. **重定向是发生在用户与服务之间的.**

### 转发

1. 用户向服务器请求一次.就可以得到响应
2. URL地址不会发生变化.
3. 转发是发生在**服务与服务之间的请求**.
4. **使用转发.因为request对象是同一个.所以可以携带参数.**

**以业务出发:**

1. 请求要不要跳转url地址.

如果请求不需要修改url地址,使用转发

如果url地址需要与业务同步,需要修改.使用重定向.

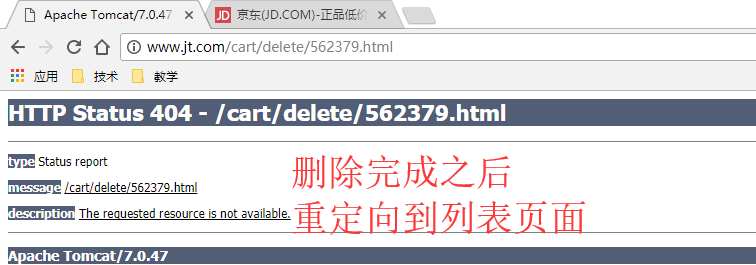
1. 是否需要携带参数

如果请求需要携带参数,使用转发

如果请求业务已经执行完成.没有后续的任务,同时不需要携带参数,.则使用重定向.

## 购物车删除业务

### 购物车页面分析



### 编辑Web Controller

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

**public** String deleteCart(@PathVariable Long itemId,HttpServletRequest request) {

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

**long** userId = user.getId();

cartService.deleteCartById(userId,itemId);

//重定向到列表页面

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

}

### 编辑web Service

@Override

**public** **void** deleteCartById(Long userId, Long itemId) {

String url = "http://cart.jt.com/cart/delete/"+userId+"/"+itemId;

httpClient.doGet(url);

}

### 编辑Cart Controller

//实现购物车删除操作

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

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

cartService.deleteCartById(cart);

**return** SysResult.*oK*();

}

### 编辑Cart Service

@Override

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

QueryWrapper<Cart> queryWrapper =

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

cartMapper.delete(queryWrapper);

//System.out.println("删除成功!!!!");

}

## 关于406报错说明

### 报错说明

浏览器解析异常.浏览器要求回显的数据是**HTML页面**.但是用户回显的数据不满足要求(JSON串)

### 案例分析

url请求地址:

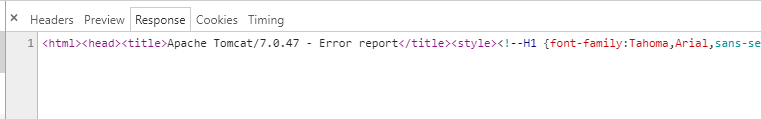
[http://www.jt.com**/service**/cart/update/num/562379/6](http://www.jt.com/service/cart/update/num/562379/6)

修改url请求路径:

<http://www.jt.com/cart/update/num/562379/6.html>

回显数据的结果:要求展现时必须为html





### 问题描述说明

说明:根据分析,如果使用XXXX.html进行访问时.首先该请求满足了伪静态的要求.增强了搜索引擎的友好性.但是该请求必须返回html页面.

为了同时满足伪静态和正常的ajax操作.则在web.xml中开辟了第二条访问路径.

<!--第一条请求路径 伪静态操作 -->

<servlet-mapping>

<servlet-name>springmvc-web</servlet-name>

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

</servlet-mapping>

<!-- 防止springMVC框架返回json时和html冲突报 406 错误 -->

<!--第二条请求路径 -->

<servlet-mapping>

<servlet-name>springmvc-web</servlet-name>

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

</servlet-mapping>

访问请求确定:

如果访问时需要跳转页面则url写\*.html,如果发起的请求要求返回JSON.则以/service/开头.否则必然报406报错!!!.

## ThreadLocal介绍

### 需求说明

在业务层处理时,如果需要获取当前的用户信息,则必须在Controller方法中传递一个Request对象.如果我们业务层有200个方法,则需要传递200次.

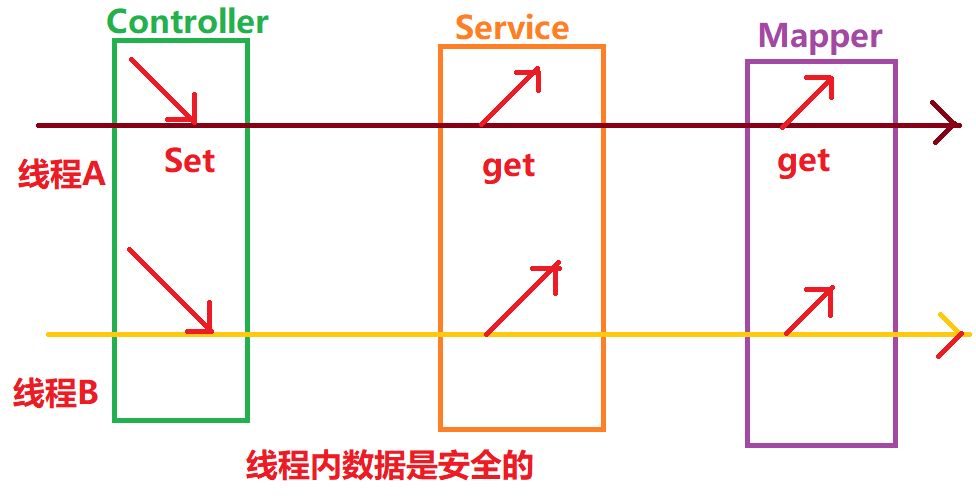
如果在业务调用时,需要在Service层中动态获取用户信息.则必须通过Controller中的方法将用户的User对象进行传递.

如果某些参数不是这个方法的主要业务.但是必须传递该参数.这样会导致业务层参数较多.维护繁琐.

### ThreadLocal说明

名称:本地线程变量

作用: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> *userThreadLocal* =

**new** ThreadLocal<>();

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

*userThreadLocal*.set(user);

}

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

**return** *userThreadLocal*.get();

}

//使用ThreadLocal切记删除对象.防止内存泄漏.

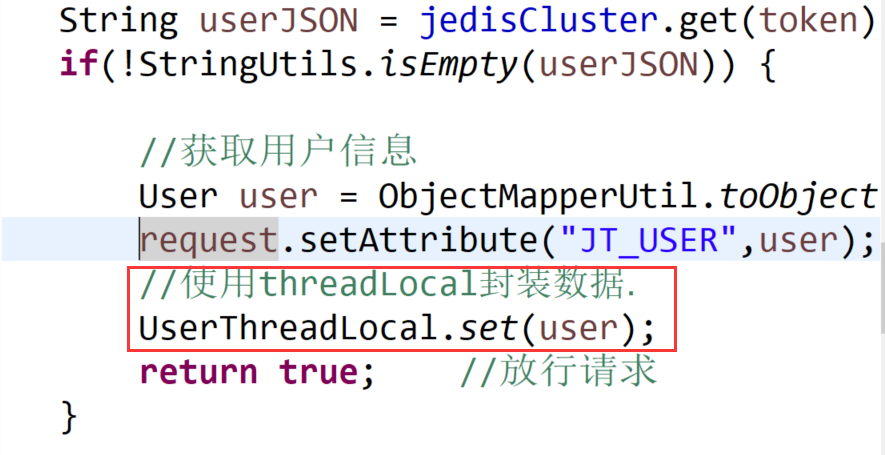
**public** **static** **void** remove() {

*userThreadLocal*.remove();

}

}

### ThreadLocal使用



业务调用:

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

**public** String deleteCart(@PathVariable Long itemId,HttpServletRequest request) {

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

//long userId = user.getId();

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

cartService.deleteCartById(userId,itemId);

//重定向到列表页面

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

}

### 防止内存溢出

//视图渲染完成之后执行 关闭链接/关闭流(大文件写入写出BIO

//NIO 非阻塞式IO )

//删除某些对象防止内存泄漏

@Override

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

**throws** Exception {

**UserThreadLocal.*remove*();**

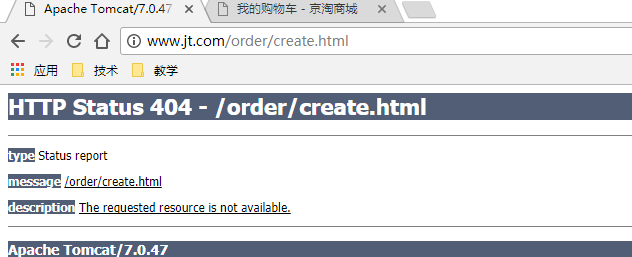
//方法执行完成后,删除对象

}

# 实现订单业务

## 实现订单确认页面跳转

### 页面分析



### 编辑Web Controller

@Controller

@RequestMapping("/order")

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

@Autowired

**private** CartService cartService;

//实现订单页面跳转

@RequestMapping("/create")

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

**long** userId =

UserThreadLocal.*get*().getId();

List<Cart> cartList =

cartService.findCartByUserId(userId);

model.addAttribute("carts", cartList);

//跳转订单确认 转发操作

**return** "order-cart";

}

}

### 编辑web Service

该方法之前已经完成,现在直接调用.

@Override

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

String url = "http://cart.jt.com/cart/query/"+userId;

String result = httpClient.doGet(url);

SysResult sysResult =

ObjectMapperUtil.*toObject*(result, SysResult.**class**);

**return** (List<Cart>) sysResult.getData();

}

### 页面效果



## 实现订单入库

### 接口文档规范

|  |  |
| --- | --- |
| 请求方法 | POST |
| URL | http://order.jt.com/order/create |
| 参数 | 以json格式封装所有的参数，提交到controller@RequestBody  {  “payment”:520,  “postFee”:0,  “userId”:”3”,  “buyerMessage”:null,  “buyerNick”:”chenchen”,  “orderItems”:[  {  “itemId”:”36”,  “num”:1,  “title”:”苹果（APPLE）iPhone 6 A14847”  “price”:5288,  “totalFee”:5288,  “picPath”:http://image.jt.com/images/2015/10/11/20151011045028374892.jpg  }  ],  “orderShipping”:{  “receiverName”:”陈晨”,  “receiverPhone”:””,  “receiverMobile”:”13579009830”,  “receiverState”:”北京”,  “receiverCity”:” 北京”,  “receiverDistrict”:” 昌平区”,  “receiverAddress”:”东小口镇天字第一号隔壁”,  “receiverZip”:” 110120”  }  } |
| 返回值 | orderId 订单编号 使用String类型接收返回值  采用SysResult对象返回 |

### 编辑order Controller

@RestController

@RequestMapping("/order")

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

@Autowired

**private** OrderService orderService;

//实现订单信息新增 同时入库3张表

@RequestMapping("/create")

**public** SysResult saveOrder(String orderJSON) {

Order order

= ObjectMapperUtil.*toObject*(orderJSON, Order.**class**);

String orderId = orderService.saveOrder(order);

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

**return** SysResult.*oK*(orderId);

}**else** {

**return** SysResult.*build*(201,"新增订单失败");

}

}

}

### 编辑Order Service

@Service

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

@Autowired

**private** OrderMapper orderMapper;

@Autowired

**private** OrderShippingMapper orderShippingMapper;

@Autowired

**private** OrderItemMapper orderItemMapper;

@Override

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

String orderId = ""+ order.getUserId()

+System.*currentTimeMillis*();

Date date = **new** Date();

//1.实现订单入库

order.setOrderId(orderId);

order.setStatus(1); //表示未付款

order.setCreated(date);

order.setUpdated(date);

orderMapper.insert(order);

System.***out***.println("订单入库成功!!!");

OrderShipping orderShipping =

order.getOrderShipping();

orderShipping.setOrderId(orderId);

orderShipping.setCreated(date);

orderShipping.setUpdated(date);

orderShippingMapper.insert(orderShipping);

System.***out***.println("订单物流入库成功!!!");

//订单商品入库

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

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

orderItem.setOrderId(orderId);

orderItem.setCreated(date);

orderItem.setUpdated(date);

orderItemMapper.insert(orderItem);

}

System.***out***.println("订单商品入库成功!!!");

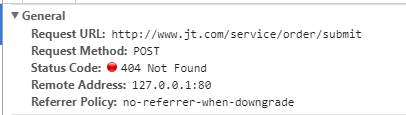
**return** orderId;

}

}

### 前台页面分析

1.页面分析



2.查看页面JS

jQuery.ajax( {

type : "POST",

dataType : "json",

url : "/service/order/submit",

data : $("#orderForm").serialize(),

cache : **false**,

success : **function**(result) {

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

location.href = "/order/success.html?id="+result.data;

}**else**{

$("#submit\_message").html("订单提交失败，请稍后重试...").show();

}

},

error : **function**(error) {

$("#submit\_message").html("亲爱的用户请不要频繁点击, 请稍后重试...").show();

}

});

### 编辑web Controller

//实现订单新增

@RequestMapping("/submit")

@ResponseBody

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

String orderId =

orderService.saveOrder(order);

**if**(!StringUtils.*isEmpty*(orderId))

**return** SysResult.*oK*(orderId);

**else**

**return** SysResult.*build*(201,"订单新增失败");

}

### 编辑web Service

@Service

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

@Autowired

**private** HttpClientService httpClient;

@Override

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

String url = "http://order.jt.com/order/create";

String orderJSON = ObjectMapperUtil.*toJSON*(order);

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

params.put("orderJSON", orderJSON);

String result = httpClient.doPost(url, params);

SysResult sysResult

= ObjectMapperUtil.*toObject*(result, SysResult.**class**);

String orderId = **null**;

**if**(sysResult.getStatus() == 200) {

orderId = (String) sysResult.getData();

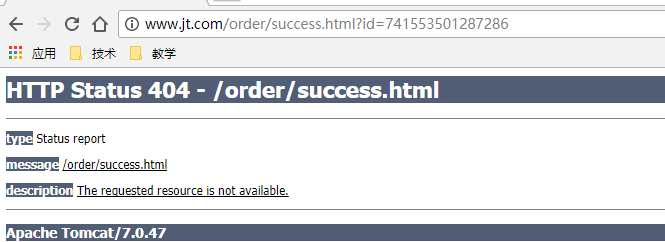
}

**return** orderId;

}

}

### 页面效果



## 关于SpringMVC值封装问题

### SpringMVC接收简单类型

页面描述:

<input type=*"hidden"* name=*"paymentType"* value=*"1"*/>

<input type=*"hidden"* name=*"num"* value=*"101"*/>

参数接收:

Public void params(String *paymentType,Integer num*){…}

### SpringMVC接收对象信息

页面描述:

<input type=*"hidden"* name=*"paymentType"* value=*"1"*/>

<input type=*"hidden"* name=*"num"* value=*"101"*/>

参数接收:

Public void params(Order order){….}

前台条件:order对象中必须*paymentType/num属性.以及get/set方法*

### SpringMVC为对象的引用赋值

页面描述:

<input type=*"hidden"* name=*"dog.dogName"* value=*"阿拉斯加"*/>

<input type=*"hidden"* name=*"dog.age"* value=*"3"*/>

<input type=*"hidden"* name=*"dog.food.foodName"* value=*"狗粮"*/>

参数接收:

Public void params(Person person){}

定义对象代码

Public class Person{

Private String username;

……..

Private Dog dog; //对象的引用!!!

}

Public class Dog{

Private String dogName;

Private Integer age;

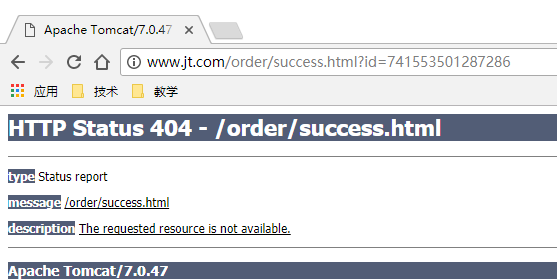
Get/set…

}

## 实现订单查询

### 前台页面分析

1.页面url地址



2.页面取值分析

<li class=*"li-st"*>

<div class=*"fore1"*>订单号：<a href=*"javascript:void(0)"*>${order.orderId }</a></div>

<!-- 货到付款 -->

<div class=*"fore2"*>货到付款：<strong class=*"ftx-01"*>${order.payment}元</strong></div>

<div class=*"fore3"*>

京淘快递 &nbsp; 送货时间: 预计 ${date} 送达&nbsp;

</div>

</li>

### 编辑web Controller

//实现订单的查询

@RequestMapping("/success")

**public** String findOrderById(String id,Model model) {

//根据orderId获取订单信息

Order order = orderService.findOrderById(id);

model.addAttribute("order", order);

**return** "success";

}

### 编辑web Service

//通过httpClient获取orderJSON数据

@Override

**public** Order findOrderById(String id) {

String url = "http://order.jt.com/order/query/"+id;

String orderJSON = httpClient.doGet(url);

**return** ObjectMapperUtil.*toObject*(orderJSON, Order.**class**);

}

### 编辑Order Controller

//根据orderId查询order信息

@RequestMapping("/query/{orderId}")

**public** Order findOrderById(@PathVariable String orderId) {

**return** orderService.findOrderById(orderId);

}

### 编辑Order Service

@Override

**public** Order findOrderById(String orderId) {

Order order = orderMapper.selectById(orderId);

OrderShipping orderShipping =

orderShippingMapper.selectById(orderId);

QueryWrapper<OrderItem> queryWrapper =

**new** QueryWrapper<OrderItem>().eq("order\_id", orderId);

List<OrderItem> lists =

orderItemMapper.selectList(queryWrapper);

order.setOrderShipping(orderShipping);

order.setOrderItems(lists);

**return** order;

}

### 页面效果展现

