# 京淘订单实现

## 动态获取用户信息

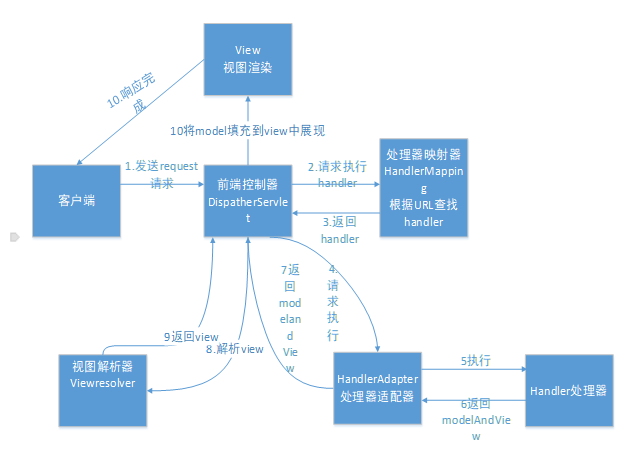
### 解决问题

1. 如何动态的获取userId
2. 如何实现用户的校验.当用户没有登陆时,则不能直接访问购物车.先跳转到登陆页面.

核心:**每次**用户访问都需要进行登录判断.(拦截器)

## SpringMVC配置拦截器

### MVC调用原理图



### 编辑业务处理类

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

@Autowired

**private** JedisCluster jedisCluster;

/\*\*

\* 执行业务之前拦截 返回值true请求放行 false请求拦截,添加重定向地址

\* 1.通过request对象获取Cookie.

\* 2.从Cookie中获取token数据.

\* 如果没有数据.则证明用户没有登陆.

\* 3.根据token数据获取redis中的信息

\* 如果redis缓存中,没有数据,则需要用户重新登陆.

\* 4.如果token中有数据.redis中有记录.则表示用户

\* 已经登陆.页面正确跳转.

\*/

@Override

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

**throws** Exception {

//1.获取cookie

Cookie[] cookies = request.getCookies();

String token = **null**;

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

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

token = cookie.getValue();

**break**;

}

}

//2.判断是否有记录

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

//2.2配置redis中是否有记录

String userJSON =

jedisCluster.get(token);

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

**return** **true**;

}

}

//证明用户没有登陆,需要重定向到登陆页面

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

**return** **false**;

}

@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

}

}

### 编辑拦截器配置

<!--配置拦截器

(mapping+, exclude-mapping\*, (bean | ref))

/cart/\*\* 表示拦截多级路径

/cart/\* 表示只拦截cart下的一级路径

-->

<mvc:interceptors>

<mvc:interceptor>

<mvc:mapping path=*"/cart/\*\*"*/>

<bean class=*"com.jt.web.interceptor.UserInterceptor"*/>

</mvc:interceptor>

</mvc:interceptors>

### 动态获取userID(一)

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

@Autowired

**private** JedisCluster jedisCluster;

/\*\*

\* 执行业务之前拦截 返回值true请求放行 false请求拦截,添加重定向地址

\* 1.通过request对象获取Cookie.

\* 2.从Cookie中获取token数据.

\* 如果没有数据.则证明用户没有登陆.

\* 3.根据token数据获取redis中的信息

\* 如果redis缓存中,没有数据,则需要用户重新登陆.

\* 4.如果token中有数据.redis中有记录.则表示用户

\* 已经登陆.页面正确跳转.

\*/

@Override

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

**throws** Exception {

//1.获取cookie

Cookie[] cookies = request.getCookies();

String token = **null**;

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

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

token = cookie.getValue();

**break**;

}

}

//2.判断是否有记录

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

//2.2配置redis中是否有记录

String userJSON =

jedisCluster.get(token);

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

User user = MapperUtil

.*toObject*(userJSON, User.**class**);

Long userId = user.getId();

request

.getSession()

.setAttribute("JT\_WEB\_USER", userId);

**return** **true**; //表示放行

}

}

//证明用户没有登陆,需要重定向到登陆页面

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

**return** **false**;

}

@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

}

}

2.获取id的步骤

//实现购物车列表展现

@RequestMapping("/show")

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

Long userId =

(Long) request

//.getSession()

.getAttribute("JT\_WEB\_USER");

//Long userId = 7L; //暂时写死 7L

List<Cart> cartList =

cartService.findCartListByUserId(userId);

model.addAttribute("cartList", cartList);

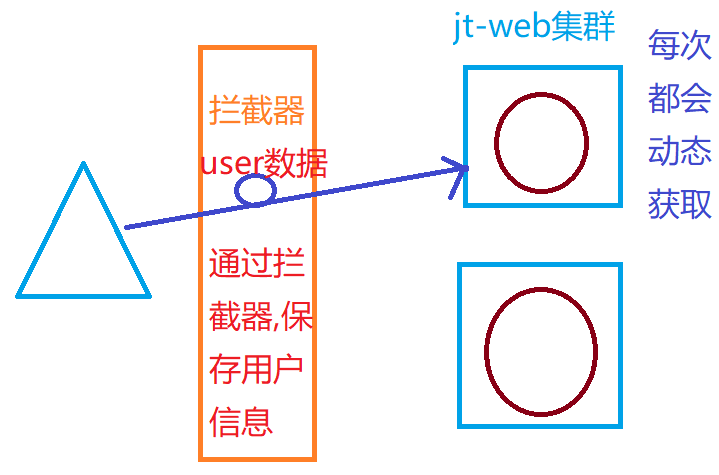
**return** "cart";

}

### Session到底共享不共享(难点)

答案:一定不共享

问题:为什么上课时,通过session域获取用户信息,没错呢???



说明:Session是不共享的.但是通过拦截器之后,用户每次操作,都能获取用户信息.而用户信息,通过request请求,进行携带,所以后续的操作能够获取

用户信息.

### 通过request对象获取数据缺点?

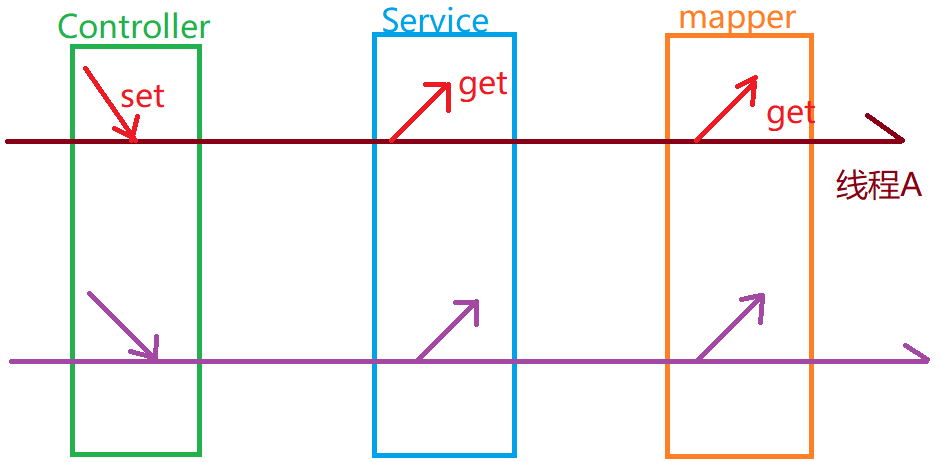
说明:如果想要在程序执行的任何位置获取用户信息,不方便…

问题:如果使用user信息,.则必须通过方法传参动态获取.

升级:ThreadLocal(本地线程变量)

### ThreadLocal介绍

作用:可以在一个线程内的任何位置,动态的获取数据.(**线程安全的**)



### 编辑工具类

**public** **class** UserThreadLocalUtil {

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

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

*thread*.set(user);

}

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

**return** *thread*.get();

}

//为了防止内存泄漏,添加删除方法

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

*thread*.remove();

}

}

工具类编辑完成后,将common打包.

### 编辑拦截器配置

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

User user = MapperUtil

.*toObject*(userJSON, User.**class**);

Long userId = user.getId();

request

//.getSession()

.setAttribute("JT\_WEB\_USER", userId);

**UserThreadLocalUtil.*set*(user);**

**return** **true**; //表示放行

}

### 业务层调用

将cart中CRUD操作中userId都换为动态获取的方式.

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

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

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

cart.setUserId(userId);

cartService.saveCart(cart);

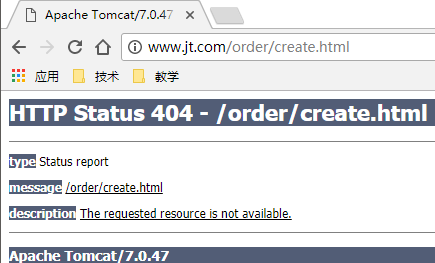
**return** "redirect:/cart/show.html";

}

## 京淘订单确认跳转

### 页面分析

1.url分析



2.js分析

<a class="checkout" href="/order/create.html" clstag="clickcart|keycount|xincart|gotoOrderInfo" id="toSettlement">去结算<b></b></a>

当用户页面跳转时,需要获取userId,查询用户购买的记录,之后进行展现.

### 编辑前台Controller

@Controller

@RequestMapping("/order")

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

@Autowired

**private** CartService cartService;

//跳转到订单确认页面

@RequestMapping("/create")

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

//准备用户的购物车记录.

Long userId = UserThreadLocalUtil

.*get*().getId();

//根据userId获取购物车记录

List<Cart> cartList =

cartService.findCartListByUserId(userId);

model.addAttribute("carts", cartList);

**return** "order-cart";

}

}

### 编辑springmvc配置文件

<mvc:interceptors>

<mvc:interceptor>

<mvc:mapping path=*"/cart/\*\*"*/>

<mvc:mapping path=*"/order/\*\*"*/>

<bean class=*"com.jt.web.interceptor.UserInterceptor"*/>

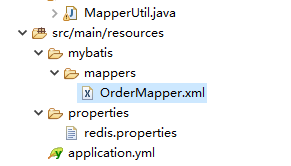
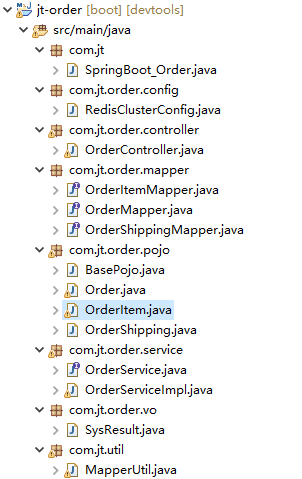
</mvc:interceptor>

</mvc:interceptors>

## 订单项目创建

### 创建项目

1. 项目图解



### 编辑yml配置文件

server:

port: 8095

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

#mybatis-plush配置

mybatis-plus:

type-aliases-package: com.jt.order.pojo

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

configuration:

map-underscore-to-camel-case: **true** #自动开启驼峰规则映射

### 编辑主启动类

@SpringBootApplication

@MapperScan("com.jt.order.mapper")

**public** **class** SpringBoot\_Order{

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBoot\_Order.**class**, args);

}

}

### 修改nginx

#编辑购物车服务器

server {

listen 80;

server\_name order.jt.com;

location / {

#反向代理到url

proxy\_pass http://localhost:8095;

}

}

## 订单新增

### 页面分析

1.页面url



1. 页面传参



1. 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();

}

});

### 编辑前台Controller

//实现订单新增 http://www.jt.com/service/order/submit

@RequestMapping("/submit")

@ResponseBody

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

**try** {

String orderId = orderService.saveOrder(order);

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

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

}

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"订单提交失败");

}

### 编辑前台Service方法

@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 = MapperUtil.*toJSON*(order);

Map<String,String> params =

**new** HashMap<String, String>();

params.put("orderJSON", orderJSON);

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

SysResult sysResult =

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

String orderId = **null**;

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

orderId = (String) sysResult.getData();

}

**return** orderId;

}

}

### 编辑后台Controller

@RestController

@RequestMapping("/order")

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

@Autowired

**private** OrderService orderService;

@RequestMapping("/create")

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

**try** {

Order order =

MapperUtil.*toObject*(orderJSON, Order.**class**);

String orderId =

orderService.saveOrder(order);

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

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

}

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"订单入库失败");

}

}

### 编辑后台Service

@Service

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

@Autowired

**private** OrderMapper orderMapper;

@Autowired

**private** OrderItemMapper orderItemMapper;

@Autowired

**private** OrderShippingMapper orderShippingMapper;

/\*\*

\* 一次入库三张表数据

\*

\*/

@Override

@Transactional

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

//拼串时注意 运算法则

String orderId = ""+order.getUserId() +

System.*currentTimeMillis*();

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

order.setOrderId(orderId);

order.setStatus(1); //未付款状态

order.setCreated(date);

order.setUpdated(date);

orderMapper.insert(order);

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

//入库订单物流信息

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!!!");

//入库订单商品

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;

}

}

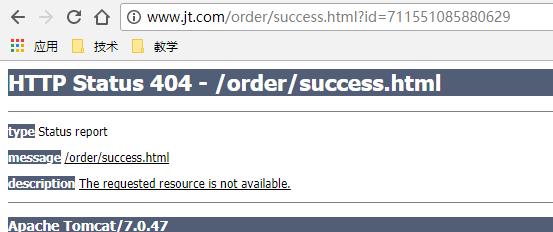
### 页面效果



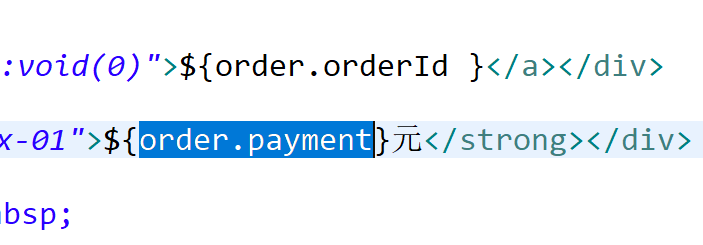
## 实现订单回显

### 页面分析

1.页面分析



2.页面取值



### 编辑Controller

//实现订单成功页面跳转

@RequestMapping("/success")

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

Order order = orderService.findOrderById(id);

model.addAttribute("order", order);

**return** "success";

}

### 编辑前台Service

@Override

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

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

String orderJSON =

httpClient.doGet(url);

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

}

### 编辑后台Controller

//根据orderId查询order对象

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

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

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

}

### 编辑后台Service

/\*\*

\* 根据orderId查询订单的全部信息 3张表

\*/

@Override

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

Order order = orderMapper.selectById(orderId);

OrderShipping orderShipping =

orderShippingMapper.selectById(orderId);

QueryWrapper<OrderItem> queryWrapper =

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

queryWrapper.eq("order\_id", orderId);

List<OrderItem> orderItems =

orderItemMapper.selectList(queryWrapper);

//数据库封装数据

order.setOrderShipping(orderShipping);

order.setOrderItems(orderItems);

**return** order;

}

### 页面访问效果

