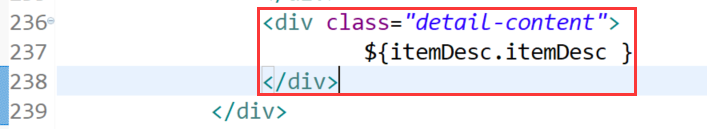
# 实现商品详情展现

## 实现步骤

### 页面分析



### 编辑Controller

@RequestMapping("/{itemId}")

**public** String findItemById(@PathVariable Long itemId,Model model){

//获取远程商品信息

Item item = itemService.findItemById(itemId);

ItemDesc itemDesc = itemService.findItemDescById(itemId);

model.addAttribute("item", item);

model.addAttribute("itemDesc", itemDesc);

**return** "item";

}

### 编辑Service

@Override

**public** ItemDesc findItemDescById(Long itemId) {

String url = "http://manage.jt.com/web/item/findItemDescById";

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

params.put("itemId", itemId+"");

ItemDesc itemDesc = **null**;

//返回的Item的JSON

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

**try** {

itemDesc = ***objectMapper***.readValue(resultJSON, ItemDesc.**class**);

} **catch** (Exception e) {

e.printStackTrace();

**throw** **new** RuntimeException();

}

**return** itemDesc;

}

### 编辑后台Controller

//http://manage.jt.com/web/item/findItemDescById

@RequestMapping("/findItemDescById")

@ResponseBody

**public** ItemDesc findItemDescById(Long itemId){

**return** itemService.findItemDescById(itemId);

}

### 编辑后台Service

@Override

**public** ItemDesc findItemDescById(Long itemId) {

**return** itemDescMapper.selectByPrimaryKey(itemId);

}

# 单点登录实现

## SSO单点登录介绍

### SSO介绍

SSO英文全称Single Sign On，[单点登录](https://baike.baidu.com/item/%E5%8D%95%E7%82%B9%E7%99%BB%E5%BD%95/4940767" \t "_blank)。SSO是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。它包括可以将这次主要的登录映射到其他应用中用于同一个用户的登录的机制。它是目前比较流行的企业业务整合的解决方案之一。

### 实现单点登录设想

问题:用户的登陆信息只保存到一台业务服务中,其他的服务器需要使用用户信息时,数据无法实现共享.

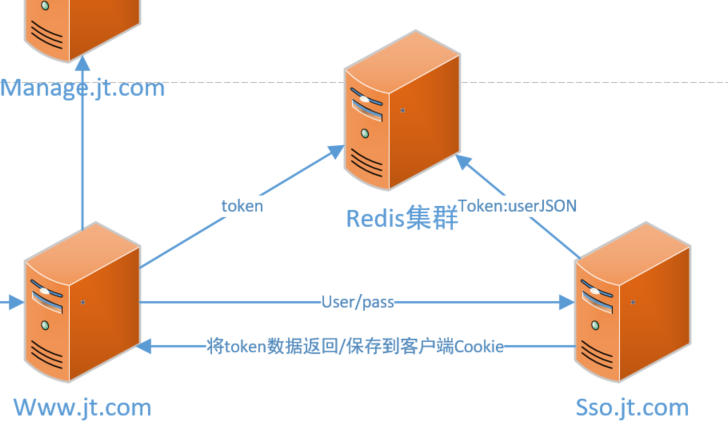
回忆:

1. url重写技术 动态拼接sessionId 效率太低

解决Cookie禁用问题.大型公司一般不会考虑该问题.

1. IP\_hash 将用户IP绑定到指定的一台服务器上.

### 单点登录实现

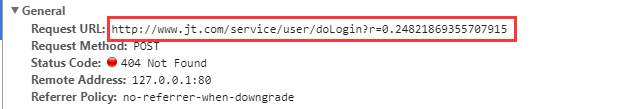


1. 当用户输入用户名和密码时将数据发送到单点登录系统中.
2. 单点登录系统接收到用户名和密码后,进行校验.如果用户名和密码不正确,直接错误返回即可.如果用户名和密码是正确的.将用户的信息转化为JSON串,并且生成加密的秘钥token.将数据保存到redis中.token当做key.userJSON当做value.将用户的token数据返回给客户端.
3. 获取到服务器返回值之后,如果用户名和密码不正确,则给用户进行提示.如果用户登陆成功.需要将token数据保存到客户端的Cookie中.并且实现页面跳转.同时实现数据的回显.

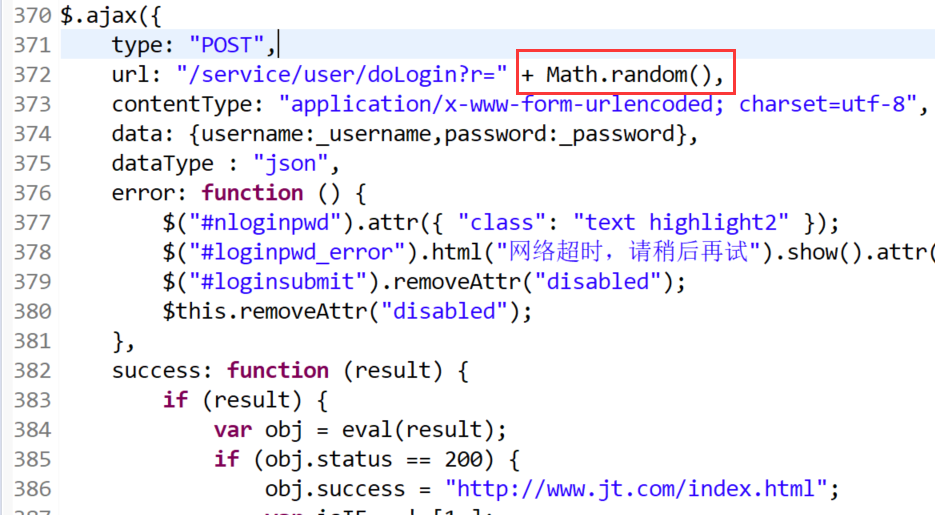
## 用户登陆实现

### 页面分析

1. url分析



1. ajax分析



### 编辑webController

//实现用户登陆操作

@RequestMapping("/doLogin")

@ResponseBody

**public** SysResult doLogin(User user,HttpServletResponse response){

**try** {

String token = userService.findUserByUP(user);

//判断登陆是否有效

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

**return** SysResult.*build*(201, "用户登陆失败");

}

//将token数据保存到Cookie中

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

cookie.setPath("/"); //cookie保存路劲 一般都是/

/\*\*

\* cookie的声明周期 单位是秒

\* >0 生命的存活时间

\* =0 表示立即删除cookie

\* -1 表示当会话结束删除cookie

\*/

cookie.setMaxAge(3600 \* 24 \* 7);

response.addCookie(cookie);

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

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201, "用户查询失败");

}

### 接口文档定义

|  |  |
| --- | --- |
| 请求方法 | POST |
| URL | http://sso.jt.com/user/login |
| 参数 | u 用户名  p 密码 |
| 示例 | <http://sso.jt.com/user/login>   |  |  | | --- | --- | | **u：** | chenchen | | **p：** | 123456 | |
| 返回值 | {  status: 200  msg: “OK”  data:” e10adc3949ba59abbe56e057f20f883e” //登录成功，返回ticket  } |
| 备注 | 登录完成，返回ticket，前台系统写入cookie |
| ticket算法 | 唯一标识每个用户：动态唯一  安全：混淆md5加密  md5（“JT\_TICKET\_” + System.currentTime + username） |

### 定义webService

/\*\*

\* 1.定义远程url

\* 2.封装参数

\* 3.发起httpClient请求 resultJSON sysResultJSON

\* 4.判断返回值的状态是否为200,如果不是200 抛出异常

\* 5.如果状态码为200.获取token数据之后返回.

\*/

@Override

**public** String findUserByUP(User user) {

String token = **null**;

String url = "http://sso.jt.com/user/login";

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

params.put("username", user.getUsername());

params.put("password", user.getPassword());

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

**try** {

SysResult sysResult =

***objectMapper***.readValue(resultJSON, SysResult.**class**);

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

//获取token数据

token = (String) sysResult.getData();

}

} **catch** (Exception e) {

e.printStackTrace();

**throw** **new** RuntimeException();

}

**return** token;

}

### 编辑SSOController

@RequestMapping("/login")

@ResponseBody

**public** SysResult findUserByUP(User user){

**try** {

String token = userService.findUserByUP(user);

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

**return** SysResult.*build*(201,"用户查询失败");

}

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

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"用户查询失败");

}

### 编辑SSOService

/\*\*

\* 编辑sso 业务实现

\* 1.根据用户名和密码查询数据 检查数据是否正确

\* 无数据: 用户名和密码不正确 直接返回 null throw

\* 有数据: 用户名和密码正确

\* 2.根据加密算法 生成tokenmd5

\* （“JT\_TICKET\_” + System.currentTime + username）

\* 3.将用户信息转化为userJSON数据.将token:userJSON保存到redis中

\* 4.将token数据返回.

\*

\*/

@Override

**public** String findUserByUP(User user) {

user.setPassword(DigestUtils.*md5Hex*(user.getPassword()));

User userDB = userMapper.findUserByUP(user);

String returnToken = **null**;

**if**(userDB == **null**){

**throw** **new** RuntimeException();

}

String token = "JT\_TICKET\_" + System.*currentTimeMillis*() + user.getUsername();

returnToken = DigestUtils.*md5Hex*(token);

**try** {

String userJSON = ***objectMapper***.writeValueAsString(userDB);

jedisCluster.setex(returnToken, 3600 \* 24 \* 7, userJSON);

System.***out***.println("用户单点登录成功!!!");

} **catch** (Exception e) {

e.printStackTrace();

**throw** **new** RuntimeException();

}

**return** returnToken;

}

### 编辑SSOMapper

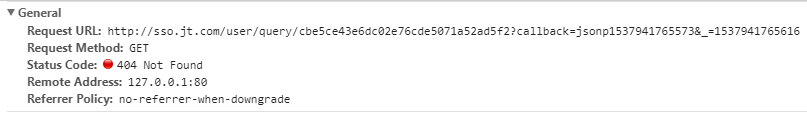
@Select("select \* from tb\_user where username = #{username} and password = #{password}")

User findUserByUP(User user);

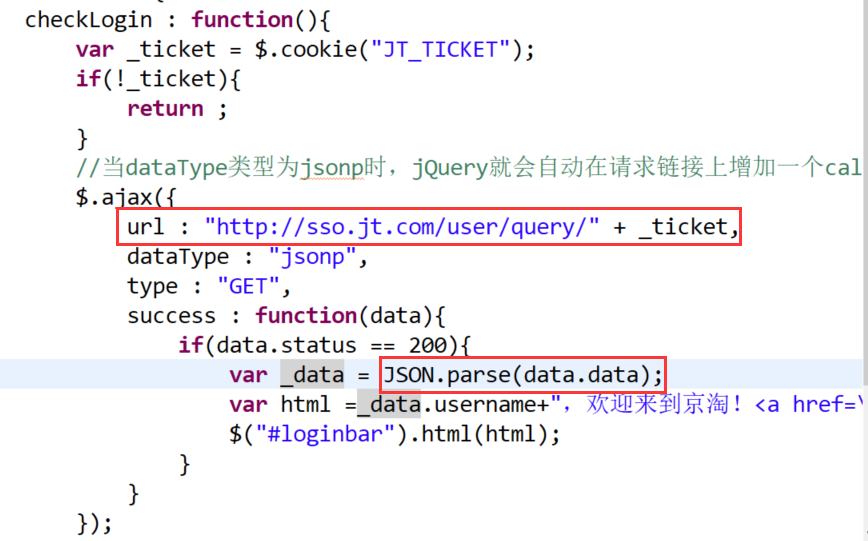
## 用户数据回显

### 页面分析

1.页面url



2.页面js



### 编辑SSOController

//根据用户Cookie信息查询用户

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

@ResponseBody

**public** MappingJacksonValue findUserByTicket(

@PathVariable String token,

String callback){

String userJSON = jedisCluster.get(token);

MappingJacksonValue jacksonValue = **null**;

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

//redis中的数据不为null

jacksonValue = **new** MappingJacksonValue(SysResult.*oK*(userJSON));

}**else**{

jacksonValue = **new** MappingJacksonValue(SysResult.*build*(201,"用户查询失败"));

}

jacksonValue.setJsonpFunction(callback);

**return** jacksonValue;

}

## 用户登出操作

### 编辑webController

/\*\*

\* 用户登出操作

\* 1.删除redis

\* 1.从cookie中获取token

\* 2.从redis删除key

\* 2.删除cookie

\* JT\_TICKET

\* **@return**

\*/

@RequestMapping("/logout")

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

//获取cookie

Cookie[] cookies = request.getCookies();

String token = **null**;

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

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

token = cookie.getValue();

**break**;

}

}

//删除redis中的数据

jedisCluster.del(token);

//删除Cookie

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

cookie.setPath("/");

cookie.setMaxAge(0); //立即删除Cookie

response.addCookie(cookie);

//重定向到系统首页

**return** "redirect:/index.html";

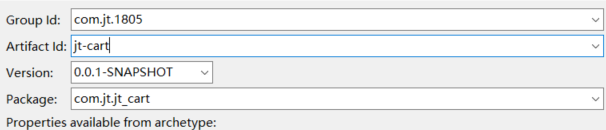
}

# 京淘购物车实现

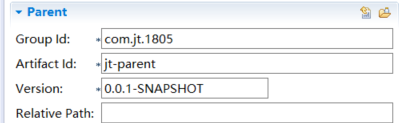
## 搭建项目

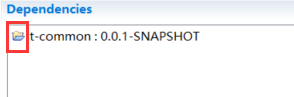
### 创建项目

1.构建项目



2.添加继承和依赖





1. 添加tomcat插件

<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.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<port>8093</port>

<!--项目的发布路径 -->

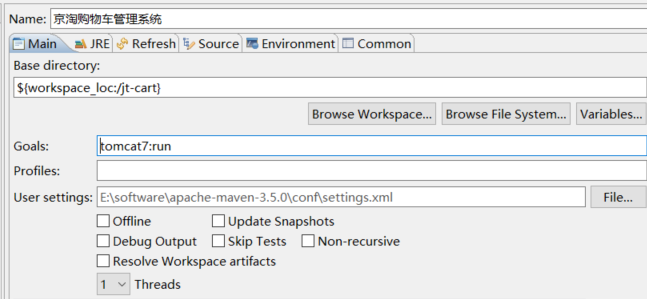
<path>/</path>

</configuration>

</plugin>

</plugins>

</build>



1. 配置Nginx实现反向代理

#单点登录管理系统

server {

listen 80;

server\_name cart.jt.com;

location / {

#实现服务器代理

proxy\_pass http://localhost:8094;

proxy\_connect\_timeout 3;

proxy\_read\_timeout 3;

proxy\_send\_timeout 3;

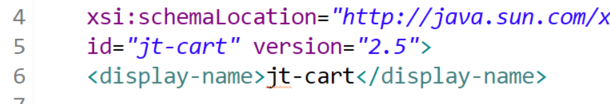
}

}

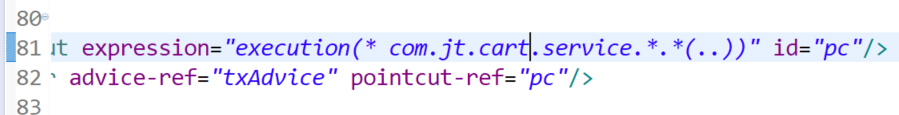
修改hosts文件

### 添加配置文件

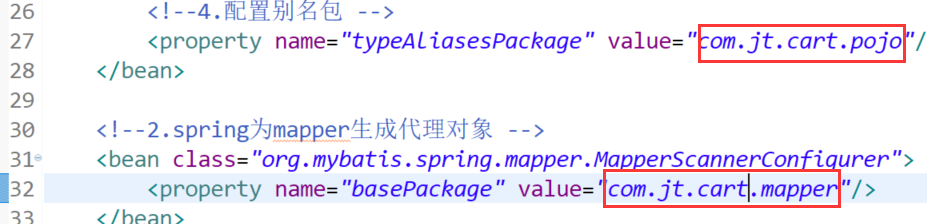
1. 修改web.xml



1. 修改Spring配置文件



1. 修改Mycatis配置文件

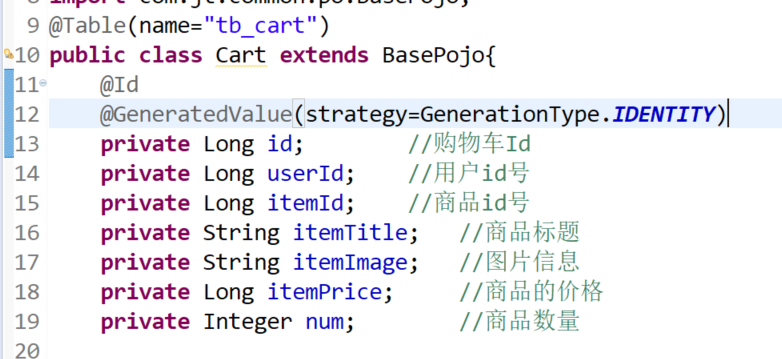


1. 修改mybatis映射文件

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

</mapper>

### 编辑pojo文件



## 购物车展现

### 编辑HttpClient中doGet方法

/\*\*

\* 实现get请求

\* 说明:

\* get请求中的参数是经过拼接形成的.

\* localhost:8091/addUser?id=1&name=tom

\* 1 www.jt.com?id=1&name=tom&

String basicUrl = url + "?";

for (Map.Entry<String, String> entry : params.entrySet()) {

basicUrl = basicUrl + entry.getKey() + "="

+ entry.getValue() + "&";

}

url = basicUrl.substring(0, basicUrl.length()-1);\*/

**public** String doGet(String url,Map<String,String> params,

String charset){

String result = **null**;

//1.判断字符集编码是否为空

**if**(StringUtils.*isEmpty*(charset)){

charset = "UTF-8";

}

**try** {

//2.判断是否有参数 url www.jt.com

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

//通过工具类的写法实现路径的自动的拼接

URIBuilder builder = **new** URIBuilder(url);

**for** (Map.Entry<String,String> entry : params.entrySet()) {

builder.addParameter(entry.getKey(), entry.getValue());

}

//将路径进行拼接 addUser?id=1&name=tom

url = builder.build().toString();

System.***out***.println("获取请求路径:"+url);

}

//定义请求类型

HttpGet httpGet = **new** HttpGet(url);

httpGet.setConfig(requestConfig);

//发起请求

CloseableHttpResponse httpResponse = httpClient.execute(httpGet);

**if**(httpResponse.getStatusLine().getStatusCode() == 200){

result = EntityUtils.*toString*(httpResponse.getEntity(),charset);

}

} **catch** (Exception e) {

e.printStackTrace();

}

**return** result;

}

**public** String doGet(String url,Map<String,String> params){

**return** doGet(url, params, **null**);

}

**public** String doGet(String url){

**return** doGet(url,**null**,**null**);

}

编辑完成后将项目打包

### 页面分析



### 编辑webController

@Controller

@RequestMapping("/cart")

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

@Autowired

**private** CartService cartService;

//http://www.jt.com/cart/show.html

@RequestMapping("/show")

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

Long userId = 7L; //暂时写死

//获取购物车列表信息

List<Cart> cartList = cartService.findCartByUserId(userId);

model.addAttribute("cartList", cartList);

**return** "cart"; //返回购物车页面

}

}

### 编辑前台CartService

@Controller

@RequestMapping("/cart")

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

@Autowired

**private** CartService cartService;

//http://www.jt.com/cart/show.html

@RequestMapping("/show")

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

Long userId = 7L; //暂时写死

//获取购物车列表信息

List<Cart> cartList = cartService.findCartByUserId(userId);

model.addAttribute("cartList", cartList);

**return** "cart"; //返回购物车页面

}

}

### 编辑后台CartController

@Controller

@RequestMapping("/cart")

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

@Autowired

**private** CartService cartService;

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

@ResponseBody

**public** SysResult findCartByUserId(@PathVariable Long userId){

**try** {

List<Cart> cartList = cartService.findCartByUserId(userId);

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

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"购物车查询失败");

}

}

### 编辑后台CartService

@Service

**public** **class** CartServiceImpl **implements** CartService {

@Autowired

**private** CartMapper cartMapper;

@Override

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

Cart cart = **new** Cart();

cart.setUserId(userId);

**return** cartMapper.select(cart);

}

}

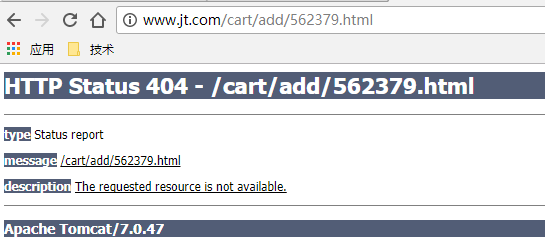
### 页面效果



## 购物车新增

### 页面分析

1.页面url



2.页面JS

<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]}*"*/><!--获取第一张图片 -->

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

</form>

//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

}

### 编辑前台Controller

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

**public** String saveCart(@PathVariable Long itemId,Cart cart){

Long userId = 7L;

cart.setUserId(userId);

cart.setItemId(itemId);

cartService.saveCart(cart);

//跳转到购物车列表页面

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

}

### 编辑前台Service

//编辑前台 Service业务 将cart数据入库

@Override

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

String url = "http://cart.jt.com/cart/save";

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

**try** {

//为了传参简单 将数据转化为json

String cartJSON = ***objectMapper***.writeValueAsString(cart);

params.put("cartJSON", cartJSON);

httpClient.doPost(url, params);

} **catch** (Exception e) {

e.printStackTrace();

}

}

### 编辑后台Controller

//编辑后台购物车新增

@RequestMapping("/save")

@ResponseBody

**public** SysResult saveCart(String cartJSON){

**try** {

Cart cart = ***objectMapper***.readValue(cartJSON,Cart.**class**);

cartService.saveCart(cart);

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

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"购物车新增失败");

}

### 编辑后台Service

/\*\*

\* 新增购物车业务逻辑

\* 如果用户购买相同的商品应该做数量的更新

\* 如果购买新的商品则做新增操作

\* 业务实现:

\* 通过 userId和ItemId查询数据库,检查用户是否有购买行为

\*

\*/

@Override

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

//1.根据itemId和userId查询数据库

Cart cartDB = cartMapper.findCartByUI(cart);

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

//表示用户没有购买过该商品 做新增操作

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

cart.setUpdated(cart.getCreated());

cartMapper.insert(cart);

}**else**{

//表示用户购买过商品 做数量的更新

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

cartDB.setNum(num);

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

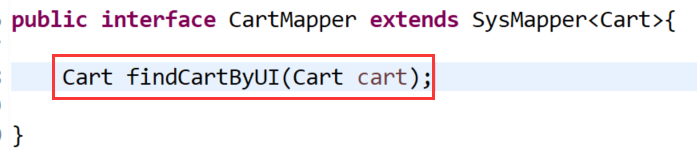
cartMapper.updateByPrimaryKeySelective(cartDB);

}

}

### 编辑Mapper接口

1.编辑mapper接口



2.编辑mapper映射文件

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

<select id="findCartByUI" resultType="Cart">

select \* from tb\_cart where user\_id = #{userId} and item\_id = #{itemId}

</select>

</mapper>

### 页面效果

