# 今日任务

1. 购物车分析
2. 购物车方案讨论
3. 代码实现（前端）
4. 拦截器实现

# 购物车

## 简介

购物车相当于现实中超市的购物车，不同的是一个是实体车，一个是虚拟车而已。用户可以在购物网站的不同页面之间跳转，以选购自己喜爱的商品，点击购买时，该商品就自动保存到你的购物车中，重复选购后，最后将选中的所有商品放在购物车中统一到付款台结账，这也是尽量让客户体验到现实生活中购物的感觉。服务器通过追踪每个用户的行动，以保证在结账时每件商品都物有其主。

## 功能

1. 把商品添加到购物车，即订购
2. 删除购物车中已定购的商品
3. 修改购物车中某个商品的订购数量

4） 清空购物车

5） 显示购物车中商品清单及数量、价格

## 实现

实现购物车的关键在于服务器识别每一个用户并维持与他们的联系。但是HTTP协议是一种“**无状态(Stateless)**”的协议，因而服务器不能记住是谁在购买商品，当把商品加入购物车时，服务器也不知道购物车里原先有些什么，使得用户在不同页面间跳转时购物车无法“随身携带”，这都给购物车的实现造成了一定的困难。下面就让我们来看看目前企业常用哪些方式来解决购物车的问题的。

# 方案讨论

目前购物车的实现主要是通过cookie、session或结合数据库的方式。下面分析一下它们的机制及作用。

1. **Cookie**

**优点:**

1.cookie是存储在客户端的,且占用很少的资源,一般cookie中可以存储300个cookie,每个cookie为4KB,既可以满足购物车的需求,还可以减轻服务器的压力.

2.cookie是浏览器内置,只要在cookie定义的有效期内,数据都不会丢失.

3.cookie不是可执行文件,所以不会给用户带来病毒或攻击用户系统

**缺点:**

1.基于cookie开发的购物车要求用户浏览器必须支持并设置为启用cookie,否则购物车则失效.

　 2.存在着关于cookie侵犯访问者隐私的争论,因此有些用户会禁止本机cookie的功能.

3.如果换一台机器在去登录的话,就会丢失购物车信息。

4.不能跨设备（终端）使用

1. **session**

**优点:**

1. session可以与客户端保持同步,不依赖与客户端的设置。

　　　 2.与cookie相比,session是存储在服务器端的信息,因此比较安全。

**缺点:**

1.session会占用服务器资源,加大服务器的负载,尤其当并发用户很多时,会生成大量的session,影响服务器的性能。

1. **结合数据库**

**优点:**

1.数据库与cookie分别负责记录数据和维持回话,能发挥各自的优势,使安全性和服务器性能都得到了提高。

2.不论换到哪一个机器上,购物车信息都不丢失。

**缺点:**

1.每个购物的行为 ,都要与数据库进行连接,直至对表的操作完成后,连接才释放.当并发用户很多时,会影响数据库的性能 ,因此,这对数据库的性能提出了更高的要求。

**总结**

综合以上三种情况，各有优缺点，我们再结合具体业务情况，根据用户是否是登录状态，设计方案如下：

1. 用户未登录，购物车存入cookie中
2. 用户登录，购物车存入redis中

# 接口

|  |  |  |  |
| --- | --- | --- | --- |
| **业务类型** | **method** | **url** | **return body** |
| 添加 | post | carts/json? | 返回一个统一的json对象，  {  “code”: xxx,  “message”:”xxxx”,  “data”:{} or []  }  返回逻辑视图 |
| 修改 | put | carts/json? |
| 删除 | delete | carts/json? |
| 查询 | get | cart.html |

# Cookie版

**添加思路（都由前端实现）**

1、用户未登录状态下，用户添加购物车，首先从cookie中查询购物车中的商品列表

2、 判断cookie的商品列表中是否有要添加的商品信息

3、如果cookie中有该商品信息，将商品的数量相加

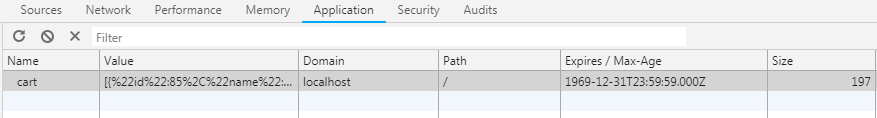
4、如果没有，根据商品的id值查询商品信息

5、将商品添加到购物车列表中

6、将购物车列表写入cookie中，设置cookie的过期时间



## 属性

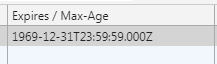


Cookie的值：中文会被编码。

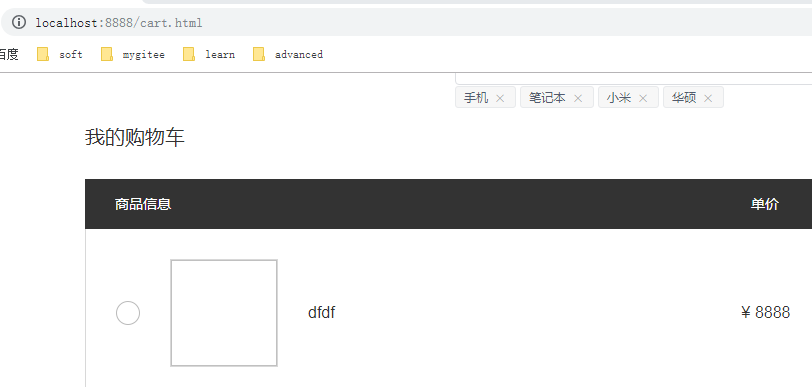
域名和path: 默认根路径



过期时间：



购物车列表：



# 问题？？

需要登录的页面，如何统一判断？？？

答案：使用拦截器。

# 拦截器

## 实现接口

HandlerInterceptor

|  |
| --- |
| **default** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)  **throws** Exception {  **return** **true**;  }  **default** **void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,  @Nullable ModelAndView modelAndView) **throws** Exception {  }  **default** **void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,  @Nullable Exception ex) **throws** Exception {  } |

## LoginInterceptor

|  |
| --- |
| **package** com.go9.mall.interceptor;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.util.StringUtils;  **import** org.springframework.web.servlet.HandlerInterceptor;  **public** **class** LoginInterceptor **implements** HandlerInterceptor {  @Value("${LOGIN\_URL}")  **private** String LOGIN\_URL;  /\*\*  \* 拦截的业务逻辑，判断有咩有登录，如果没有token就重定向到登录服务器上登录  \* 如果有token，需向sso服务器求证信息，1）登录信息过期，就不放行，并且重定向到服务器登录。  \* 1）如果验证通过，就返回登录的用户信息，并且放行，访问想访问的页面。  \*/  @Override  **public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)  **throws** Exception {  String token = request.getParameter("token");  StringBuffer lastAccessUrl = request.getRequestURL();  String params = request.getQueryString();  **if**(!StringUtils.*isEmpty*(params)) {  lastAccessUrl.append("?").append(params);  }  **if**(StringUtils.*isEmpty*(token)) {  response.sendRedirect(LOGIN\_URL+"?returnUrl="+lastAccessUrl);  **return** **false**;  }**else** {  //by token get user info  /\*Member member = ssoService.getUserInfo(token);  if(member==null) {  response.sendRedirect(LOGIN\_URL+"?returnUrl="+lastAccessUrl);  return false;  }else {  request.setAttribute("member", member);  return true;  }\*/  **return** **true**;  }  }    } |

## 拦截配置类

|  |
| --- |
| **package** com.go9.mall.config;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.web.servlet.config.annotation.InterceptorRegistry;  **import** org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;  **import** org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport;  **import** com.go9.mall.interceptor.LoginInterceptor;  @Configuration  **public** **class** MallMvcConfig **extends** WebMvcConfigurationSupport {  @Bean  **public** LoginInterceptor getLoginInterceptor() {  **return** **new** LoginInterceptor();  }    /\*\*  \* 拦截器  \*/  @Override  **protected** **void** addInterceptors(InterceptorRegistry registry) {  registry.addInterceptor(**this**.getLoginInterceptor())  .addPathPatterns("/order.html","/pay.html");  **super**.addInterceptors(registry);  }    /\*\*  \* 放行静态资源,是因为我们添加了拦截器  \*/  @Override  **protected** **void** addResourceHandlers(ResourceHandlerRegistry registry) {  registry.addResourceHandler("/\*\*").addResourceLocations("classpath:/static/");  **super**.addResourceHandlers(registry);  }  } |

## orderController

|  |
| --- |
| **package** com.go9.mall.controller;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.GetMapping;  @Controller  **public** **class** OrderController {    @GetMapping("/order.html")  **public** String order() {  **return** "order";  }  } |

## 测试拦截



结算时需要登录，用户登录如何处理？？预习单点登录。