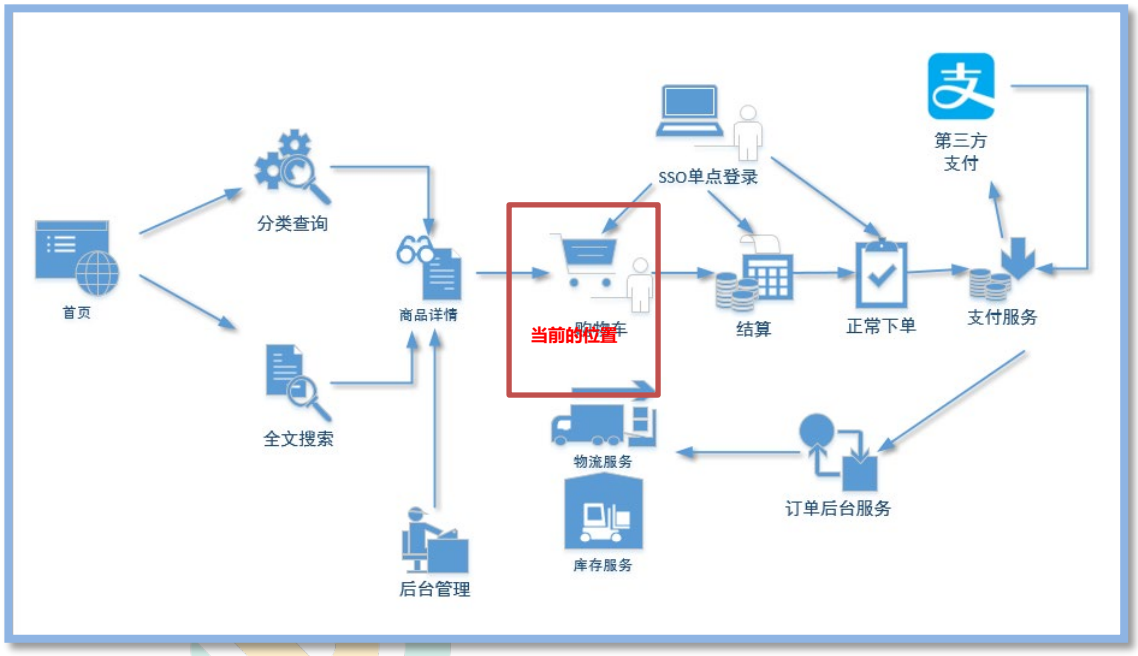
尚品汇商城

一、购物车业务简介



购物车模块要能够存储顾客所选的的商品，记录下所选商品，还要能随时更新，当用户决定购买时，用户可以选择决定购买的商品进入结算页面。

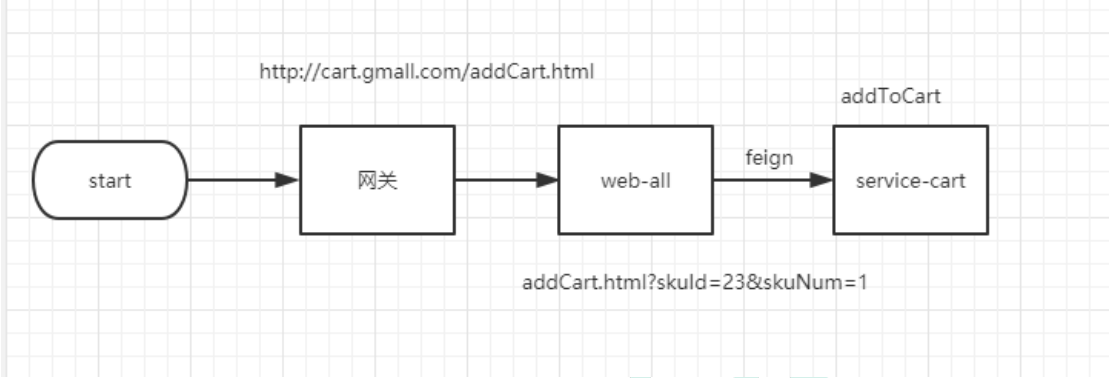
功能要求：

1) 利用缓存提高性能。

2) 未登录状态也可以存入购物车，一旦用户登录要进行合并操作。

二、 购物车模块搭建

购物车添加展示流程：



2.1 搭建 service-cart 服务

搭

2.2 修改配置 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>com.atguigu.gmall</groupId>
      <artifactId>service</artifactId>
      <version>1.0</version>
   </parent>

   <version>1.0</version>
   <artifactId>service-cart</artifactId>
   <packaging>jar</packaging>
   <name>service-cart</name>
   <description>service-cart</description>
```

```
    <dependencies>
        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>service-product-client</artifactId>
            <version>1.0</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>service-cart</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

2.3 添加配置文件

bootstrap.properties

```
spring.application.name=service-cart
spring.profiles.active=dev
spring.cloud.nacos.discovery.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.prefix=${spring.application.name}
spring.cloud.nacos.config.file-extension=yaml
spring.cloud.nacos.config.shared-configs[0].data-id=common.yaml
```

2.4 启动类

```
package com.atguigu.gmall.cart;


import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.ComponentScan;

@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)
@ComponentScan(basePackages = "com.atguigu.gmall")
@EnableDiscoveryClient
@EnableFeignClients(basePackages = "com.atguigu.gmall")
```

```
public class ServiceCartApplication {
    public static void main(String[] args) {
        SpringApplication.run(ServiceCartApplication.class,args);
    }
}
```

三、功能一添加入购物车

3.1 功能解析：

1、商品详情页添加购物车

2、添加购物车，用户可以不需要登录，如果用户没有登录，则生成临时用户 id，购物车商品与临时用户 id 关联，当用户登录后，将临时用户 id 的购物车商品与登录用户 id 的商品合并

3、商品详情添加购物车时，先判断用户是否登录，如果没登录，再判断是否存在临时用户，如果 cookie 中也没有临时用户，则生成临时用户

3.2 处理临时用户

3.2.1 商品详情页

商品详情添加购物车页面方法（/item/index.html）：

```
addToCart() {
    // 判断是否登录和是否存在临时用户，如果都没有，添加临时用户
  if(!auth.isTokenExist() && !auth.isUserTempIdExist()) {
        auth.setUserTempId()
    }
    window.location.href =
'http://cart.gmall.com/addCart.html?skuId=' + this.skuId +
'&skuNum=' + this.skuNum
}
```

3.2.2 服务网关处理

思路：既然 userId 是从服务网关统一传递过来的，那么临时用户 id 我们也可以从网关传递过来，改造网关

网关中获取临时用户 id

在 server-gateway 项目中添加

```
/**
 * 获取当前用户临时用户 id
 * @param request
 * @return
 */
private String getUserTempId(ServerHttpRequest request) {
    String userTempId = "";
    List<String> tokenList = request.getHeaders().get("userTempId");
    if(null  != tokenList) {
        userTempId = tokenList.get(0);
    } else {
        MultiValueMap<String,  HttpCookie>  cookieMultiValueMap  =
request.getCookies();
        HttpCookie                          cookie                =
cookieMultiValueMap.getFirst("userTempId");
        if(cookie != null){
            userTempId = URLDecoder.decode(cookie.getValue());
        }
    }
    return userTempId;
}
```

将 userTempId 添加 header 请求头

```
//设置网关请求头
String userTempId = this.getUserTempId(request);


if(!StringUtils.isEmpty(userId) || !StringUtils.isEmpty(userTempId))
{
    if(!StringUtils.isEmpty(userId)) {
        request.mutate().header("userId", userId).build();
    }
    if(!StringUtils.isEmpty(userTempId)) {
        request.mutate().header("userTempId", userTempId).build();
    }
    //将现在的 request 变成 exchange 对象
  return chain.filter(exchange.mutate().request(request).build());
```

```
}
```

AuthContextHolder 类添加公共方法

```
/**
 * 获取当前未登录临时用户 id
 * @param request
 * @return
 */
public static String getUserTempId(HttpServletRequest request) {
    String userTempId = request.getHeader("userTempId");
    return StringUtils.isEmpty(userTempId) ? "" : userTempId;
}
```

3.3 功能开发：

3.3.1 创建实体

```
@Data
@ApiModel(description = "购物车")
public class CartInfo extends BaseEntity {
    private static final long serialVersionUID = 1L;

    @ApiModelProperty(value = "用户 id")
    private String userId;

    @ApiModelProperty(value = "skuid")
    private Long skuId;

    @ApiModelProperty(value = "放入购物车时价格")
    private BigDecimal cartPrice;

    @ApiModelProperty(value = "数量")
    private Integer skuNum;

    @ApiModelProperty(value = "图片文件")
    private String imgUrl;

    @ApiModelProperty(value = "sku 名称 (冗余)")
```

```
    private String skuName;

    @ApiModelProperty(value = "isChecked")
    private Integer isChecked = 1;

    //实时价格 skuInfo.price
    BigDecimal skuPrice;

}
```

3.3.2 创建添加购物车接口

```
package com.atguigu.gmall.cart.service;

public interface CartService {
    // 添加购物车 用户Id，商品Id，商品数量。
  void addToCart(Long skuId, String userId, Integer skuNum);

}
```

3.3.3 添加购物车实现类

定义业务需要使用的常量，RedisConst 类

```
public static final String USER_KEY_PREFIX = "user:";
public static final String USER_CART_KEY_SUFFIX = ":cart";
public static final long USER_CART_EXPIRE = 30000;
```

```
@Service
public class CartServiceImpl implements CartService {

    @Autowired
    private ProductFeignClient productFeignClient;

    @Autowired
    private RedisTemplate redisTemplate;

}
```

```
@Override
public void addToCart(Long skuId, String userId, Integer skuNum) {
    //  获取缓存 key
    String cartKey = getCartKey(userId);

    BoundHashOperations<String, String, CartInfo> boundHashOps =
this.redisTemplate.boundHashOps(cartKey);
    CartInfo cartInfo = null;
    //包含的话更新数量
    if(boundHashOp

s.hasKey(skuId.toString())) {
        cartInfo = boundHashOps.get(skuId.toString());
        cartInfo.setSkuNum(cartInfo.getSkuNum()+skuNum);
        cartInfo.setIsChecked(1);
        cartInfo.setSkuPrice(productFeignClient.getSkuPrice(skuId));
        cartInfo.setUpdateTime(new Date());
    } else {
        cartInfo = new CartInfo();
        //  给 cartInfo 赋值!
        SkuInfo skuInfo = productFeignClient.getSkuInfo(skuId);

        //  给表的字段赋值!
        cartInfo.setUserId(userId);
        cartInfo.setSkuId(skuId);
        cartInfo.setCartPrice(skuInfo.getPrice());
        cartInfo.setSkuNum(skuNum);
        cartInfo.setImgUrl(skuInfo.getSkuDefaultImg());
        cartInfo.setSkuName(skuInfo.getSkuName());
        cartInfo.setCreateTime(new Date());
        cartInfo.setUpdateTime(new Date());
        cartInfo.setSkuPrice(skuInfo.getPrice());
    }
    boundHashOps.put(skuId.toString(), cartInfo);

}
```

```
    // 获取购物车的key

    private String getCartKey(String userId) {
        //定义key user:userId:cart
        return RedisConst.USER_KEY_PREFIX + userId +
RedisConst.USER_CART_KEY_SUFFIX;
    }
```

3..3.4 添加购物车控制器

```
package com.atguigu.gmall.cart.controller;

@RestController
@RequestMapping("api/cart")
public class CartApiController {

    @Autowired
    private CartService cartService;

    /**
     * 添加购物车
     * @param skuId
     * @param skuNum
     * @param request
     * @return
     */
    @RequestMapping("addToCart/{skuId}/{skuNum}")
    public Result addToCart(@PathVariable("skuId") Long skuId,
                            @PathVariable("skuNum") Integer skuNum,
                            HttpServletRequest request) {
        // 如何获取 userId
        String userId = AuthContextHolder.getUserId(request);
        if (StringUtils.isEmpty(userId)) {
            // 获取临时用户 Id
            userId = AuthContextHolder.getUserTempId(request);
        }
        cartService.addToCart(skuId, userId, skuNum);
        return Result.ok();
    }

}
```

四、功能一展示购物车列表

4.1 功能解析

4.2 购物车列表接口：CartService

```
/**
 * 通过用户 Id 查询购物车列表
 * @param userId
 * @param userTempId
 * @return
 */
List<CartInfo> getCartList(String userId, String userTempId);
```

4.3 实现类：CartServiceImpl

```
@Override
public List<CartInfo> getCartList(String userId, String userTempId) {
    //获取临时用户购物车数据
    List<CartInfo> cartInfoList = null;
    if(!StringUtils.isEmpty(userTempId)){
        BoundHashOperations<String, String, CartInfo> boundHashOps = this.redisTemplate.boundHashOps(this.getCartKey(userTempId));
        cartInfoList = boundHashOps.values();
    }

    //获取用户购物车数据
    if(!StringUtils.isEmpty(userId)){
        BoundHashOperations<String, String, CartInfo> boundHashOps = this.redisTemplate.boundHashOps(this.getCartKey(userId));
        cartInfoList = boundHashOps.values();
    }

    if(!CollectionUtils.isEmpty(cartInfoList)){
        // 展示购物车列表的时候应该有顺序！ 京东：按照更新时间！ 苏宁：创建时间！
        cartInfoList.sort((o1,o2)->{
            // 使用时间进行比较
            return
```

```
DateUtil.truncatedCompareTo(o2.getUpdateTime(),o1.getUpdateTime(),
Calendar.SECOND);
        });
    }
    return cartInfoList;
}
```

4.4 控制器：CartApiController

```
/**
 * 查询购物车
 *
 * @param request
 * @return
 */
@GetMapping("cartList")
public Result cartList(HttpServletRequest request) {
    // 获取用户 Id
    String userId = AuthContextHolder.getUserId(request);
    // 获取临时用户 Id
    String userTempId = AuthContextHolder.getUserTempId(request);
    List<CartInfo>  cartInfoList  =  cartService.getCartList(userId,
userTempId);
    return Result.ok(cartInfoList);
}
```

五、功能--合并购物车

功能分析：

1. 当用户登录以后，先判断未登录的时候，用户是否购买了商品。
   a) 如果用户购买了商品，则找到对应的商品 Id，对数量进行合并。
   b) 没有找到的商品,则直接添加到数据。
2. 合并完成之后，删除未登录数据。

5.1 更改实现类：CartServiceImpl

```
@Override
    public List<CartInfo> cartList(String userId, String userTempId) {
        /*
            1.  判断是否登录，根据判断结果查询不同的购物车!
            2.  查询的结果需要排序!
            3.  有可能需要合并!
          在登录的情况下
                    .  未登录 ---> 登录合并!
            合并完成之后，需要删除未登录购物车数据!
                    case1: 有userId ,没有userTempId
                    case2:  没 有  userId  ,  有  userTempId      return
noLoginCartInfoList
                    case3: 有userId ,有userTempId
                       登录情况下合并购物车:
              先判断未登录购物车集合有数据!
                                true: 有数据
                  合并
                                false: 没有数据
                  只需要登录购物车数据
              删除未登录购物车!
         */
        //  声明一个集合来存储未登录数据
    List<CartInfo> noLoginCartInfoList = null;

        //  完成 case2 业务逻辑
    //  属于未登录
    if (!StringUtils.isEmpty(userTempId)){
            String cartKey = this.getCartKey(userTempId);
            //  获取登录的购物车集合数据!
      //                              noLoginCartInfoList                =
this.redisTemplate.boundHashOps(cartKey).values();
            noLoginCartInfoList                                            =
this.redisTemplate.opsForHash().values(cartKey);
        }
        //  这个是集合的排序
    if (StringUtils.isEmpty(userId)){
            if (!CollectionUtils.isEmpty(noLoginCartInfoList)){
                noLoginCartInfoList.sort((o1,o2)->{
                    //  按照更新时间:
          return
DateUtil.truncatedCompareTo(o2.getUpdateTime(),o1.getUpdateTime(),
Calendar.SECOND);
                });
            }
            //  返回未登录数据!
      return noLoginCartInfoList;
        }
        //  ----------------------case 1 and case3 -----------------
```

```
        /*
        demo:
            登录:
                17  1
                18  1

            未登录:
                17  1
                18  1
                19  2

             合并:
                17  2
                18  2
                19  2
         */
        //  属于登录
    List<CartInfo> LoginCartInfoList = null;
        //  先获取到登录购物车的key
        String cartKey = this.getCartKey(userId);
        //  hset key field value;  hget key field;  hvals key ; hmset key field value field value;  hmset key map;
        //  合并思路二:
    BoundHashOperations<String, String, CartInfo>  boundHashOperations = this.redisTemplate.boundHashOps(cartKey);
        //  判断购物车中的field
        //  boundHashOperations.hasKey(skuId.toString);
        if (!CollectionUtils.isEmpty(noLoginCartInfoList)){
            //  循环遍历未登录购物车集合
      noLoginCartInfoList.stream().forEach(cartInfo -> {
                //  在未登录购物车中的 skuId 与登录的购物车 skuId 相对 skuId = 17 18
                if (boundHashOperations.hasKey(cartInfo.getSkuId().toString())){
                    //  合并业务逻辑 : skuNum + skuNum 更新时间
          CartInfo loginCartInfo = boundHashOperations.get(cartInfo.getSkuId().toString());

loginCartInfo.setSkuNum(loginCartInfo.getSkuNum()+cartInfo.getSkuNum());
                    loginCartInfo.setUpdateTime(new Date());
                    //  最新价格
loginCartInfo.setSkuPrice(productFeignClient.getSkuPrice(cartInfo.getSkuId()));

                    //  选中状态合并!
          if (cartInfo.getIsChecked().intValue()==1){
//                                                                  if (loginCartInfo.getIsChecked().intValue()==0){
//                              loginCartInfo.setIsChecked(1);
//                          }
                        loginCartInfo.setIsChecked(1);
                    }
                    //  修改缓存的数据:     hset key field value
```

```
boundHashOperations.put(cartInfo.getSkuId().toString(),loginCartInfo);
                }else {
                    //  直接添加到缓存!      skuId = 19
                    cartInfo.setUserId(userId);
                    cartInfo.setCreateTime(new Date());
                    cartInfo.setUpdateTime(new Date());

boundHashOperations.put(cartInfo.getSkuId().toString(),cartInfo);
                }
            });
            //  删除未登录购物车数据!
      this.redisTemplate.delete(this.getCartKey(userTempId));
        }

        //  获取到合并之后的数据:
    LoginCartInfoList                                                    =
this.redisTemplate.boundHashOps(cartKey).values();
        if (CollectionUtils.isEmpty(LoginCartInfoList)){
            return new ArrayList<>();
        }
        //  设置合并之后的排序结果!
    LoginCartInfoList.sort(((o1, o2) -> {
            return
DateUtil.truncatedCompareTo(o2.getUpdateTime(),o1.getUpdateTime(),
Calendar.SECOND);
        }));
        return LoginCartInfoList;
    }
```

六、选中状态的变更

用户每次勾选购物车的多选框，都要把当前状态保存起来。由于可能会涉及更频繁的操作，所以这个勾选状态不必存储到数据库中。保留在缓存状态即可。

6.1 编写业务接口与实现

接口

```
/**
 * 更新选中状态
 *
```

```
 * @param userId
 * @param isChecked
 * @param skuId
 */
void checkCart(String userId, Integer isChecked, Long skuId);
```

实现类

```
@Override
public void checkCart(String userId, Integer isChecked, Long skuId) {
    String cartKey = this.getCartKey(userId);
    BoundHashOperations<String, String, CartInfo> boundHashOps =
this.redisTemplate.boundHashOps(cartKey);
    CartInfo cartInfo = boundHashOps.get(skuId.toString());
    if(null != cartInfo) {
        cartInfo.setIsChecked(isChecked);
        boundHashOps.put(skuId.toString(), cartInfo);
    }
}
```

6.2 编写控制器

```
// 选中状态
@GetMapping("checkCart/{skuId}/{isChecked}")
public Result checkCart(@PathVariable Long skuId,
                        @PathVariable Integer isChecked,
                        HttpServletRequest request){

    String userId = AuthContextHolder.getUserId(request);
    // 判断
    if (StringUtils.isEmpty(userId)){
        userId = AuthContextHolder.getUserTempId(request);
    }
    // 调用服务层方法
    cartService.checkCart(userId,isChecked,skuId);
    return Result.ok();
}
```

七、删除购物车

7.1 封装业务接口与实现

接口

```
void deleteCart(Long skuId, String userId);
```

实现类

```
@Override
public void deleteCart(Long skuId, String userId) {
    BoundHashOperations<String, String, CartInfo> boundHashOps = this.redisTemplate.boundHashOps(this.getCartKey(userId));
    //  判断购物车中是否有该商品！
    if (boundHashOps.hasKey(skuId.toString())){
        boundHashOps.delete(skuId.toString());
    }
}
```

7.2 编写控制器

```
/**
 * 删除
 *
 * @param skuId
 * @param request
 * @return
 */
@DeleteMapping("deleteCart/{skuId}")
public Result deleteCart(@PathVariable("skuId") Long skuId,
                         HttpServletRequest request) {
    // 如何获取 userId
    String userId = AuthContextHolder.getUserId(request);
    if (StringUtils.isEmpty(userId)) {
        // 获取临时用户 Id
        userId = AuthContextHolder.getUserTempId(request);
    }
    cartService.deleteCart(skuId, userId);
    return Result.ok();
}
```

八、前端实现

8.1 在 web-all 添加前端实现

8.1.1 添加依赖和配置网关

```
    <dependency>
       <groupId>com.atguigu.gmall</groupId>
       <artifactId>service-cart-client</artifactId>
       <version>1.0</version>
    </dependency>
```

```
- id: web-cart
  uri: lb://web-all
  predicates:
  - Host=cart.gmall.com
- id: service-cart
  uri: lb://service-cart
  predicates:
  - Path=/*/cart/**
```

8.1.2 controller 实现

```
package com.atguigu.gmall.all.controller;

/**
 * <p>
 * 购物车页面
 * </p>
 *
 */
@Controller
public class CartController {

    @Autowired
    private CartFeignClient cartFeignClient;

    @Autowired
```

```
    private ProductFeignClient productFeignClient;

    /**
     * 查看购物车
   * @param request
     * @return
     */
    @RequestMapping("cart.html")
    public String index(){
        return "cart/index";
    }

    /**
     * 添加购物车
   * @param skuId
     * @param skuNum
     * @param request
     * @return
     */
    @RequestMapping("addCart.html")
    public String addCart(@RequestParam(name = "skuId") Long skuId,
                          @RequestParam(name = "skuNum") Integer
skuNum,
                          HttpServletRequest request){
        SkuInfo skuInfo = productFeignClient.getSkuInfo(skuId);
        request.setAttribute("skuInfo",skuInfo);
        request.setAttribute("skuNum",skuNum);
        return "cart/addCart";
    }
}
```