

# 秒杀系统（高并发+安全）

秒杀系统（高并发+安全）

效果展示

项目架构

快速启动

数据库

部署步骤

项目细节

安全

登录

验证码

秒杀接口隐藏

性能优化

秒杀引入RabbitMQ

秒杀引入Redis

其他

页面静态化，前后端分离

超卖问题

项目与课程总结

## 效果展示

- 用户登录界面

用户登录

请输入手机号码

18600000000

请输入密码

.....

请输入验证码

-9|

3-6-6

重置

登录

考虑到张老师提到的安全规则"网页上登录必须加入验证码"，将用户页面引入验证码，暂时没实现老师提到的"3次登录失败后，加入验证码"，验证码的实现具体细节在下面的源码介绍；另外对用户的手机号码进行正则化表达式的校验；

- 秒杀商品列表

秒杀商品列表					
商品名称	商品图片	商品原价	秒杀价	库存数量	详情
iphone12		6999.0	0.01	0	<a href="#">详情</a> <a href="#">详情（静态化）</a>
华为Mate40		5999.0	0.01	50	<a href="#">详情</a> <a href="#">详情（静态化）</a>
iphon11		4999.0	0.01	98	<a href="#">详情</a> <a href="#">详情（静态化）</a>
小米11		3999.0	0.01	99	<a href="#">详情</a> <a href="#">详情（静态化）</a>

放置四个秒杀商品链接，具体信息可以进入“详情”页、“详情(静态化)”页参与秒杀，详情(静态化)是对前后端分离的一种尝试，将页面直接缓存到客户端。

- 秒杀时间未到

秒杀商品详情			
没有收货地址的提示.....			
商品名称	小米11		
商品图片			
秒杀开始时间	2021-01-15 15:00:00	秒杀倒计时: 188秒	<a href="#">立即秒杀</a>
商品原价	3999		
秒杀价	0.01		
库存数量	99		

商品设定的时间未到，立即秒杀按钮是无法点击的，这样可以阻止部分用户的疯狂服务器请求。

- 秒杀开始

没有收货地址的提示.....			
商品名称	小米11		
商品图片			
秒杀开始时间	2021-01-15 15:00:00	秒杀进行中	<div>8*6-0</div> <a href="#">立即秒杀</a>
商品原价	3999		
秒杀价	0.01		
库存数量	99		

秒杀开始，加入数学公式验证码，这样可以避免用户通过明文地址将秒杀请求不停地发送到服务端，同时也有效的防止机器人等手段参与秒杀。

- 秒杀结果


秒杀商品详情

没有收货地址的提示.....

商品名称

小米11

商品图片



秒杀开始时间

2021-01-15 15:00:00

秒杀进行中

3-1\*0

3

商品原价

3999

秒杀价

0.01

库存数量

99

信息

恭喜你，秒杀成功！查看订单?


确定

取消

商品名称

小米11

商品图片



订单价格

0.01

下单时间

2021-01-15 15:22:07

订单状态

未支付

立即支付

收货人

XXX 18600000000

收货地址


北京市海淀区颐和园路5号

秒杀订单详情

商品名称

小米11

商品图片



订单价格

0.01

下单时间

2021-01-15 15:22:07

订单状态

未支付

立即支付

收货人

XXX 18600000000

收货地址


北京市海淀区颐和园路5号

完成秒杀请求，进入商品支付界面。

- 避免重复秒杀

秒杀商品详情

没有收货地址的提示.....


商品名称	iphon11		
商品图片	 <div>不能重复秒杀</div>		
秒杀开始时间	2021-01-15 14:00:00	秒杀进行中	<div>5+3=7</div> <div>1</div> <div>立即秒杀</div>
商品原价	4999		
秒杀价	0.01		
库存数量	98		

同一个用户秒杀到了两个一样的商品，这种情形也是超卖，应当避免，为此我们将利用数据库本身自带的特性进行防止。

- 秒杀活动结束

秒杀商品详情

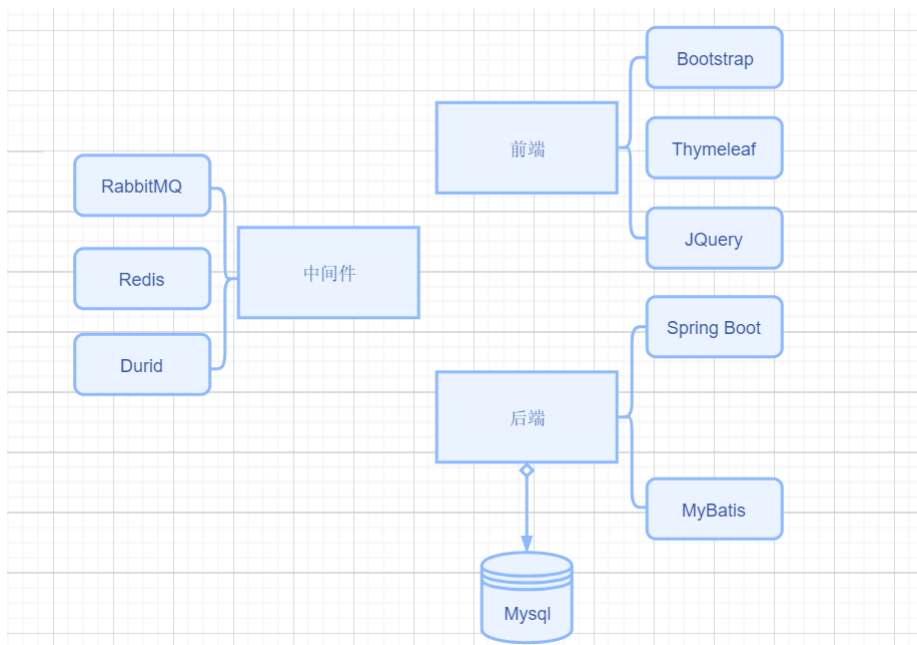
没有收货地址的提示.....

商品名称	iphon11		
商品图片			
秒杀开始时间	2021-01-15 14:00:00	秒杀已经结束	<div>立即秒杀</div>
商品原价	4999		
秒杀价	0.01		
库存数量	98		

秒杀活动结束，不再提供秒杀接口。

## 项目架构

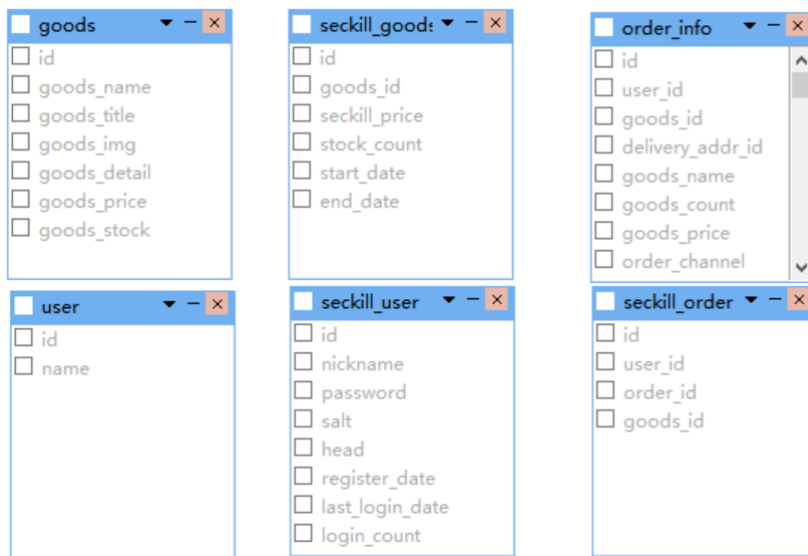
本项目模拟高并发的场景，完成商品的秒杀，同时针对相关的安全规则进行项目优化



## 快速启动

### 数据库

数据设计如下：



- user、goods、order\_info这三张表是正常系统存在的数据表
- seckill\_user、seckill\_goods、seckill\_order三张表是针对秒杀进行设计的

具体的表信息为：

seckill\_user表

对象	seckill_user @seckill (localh...	seckill_goods @seckill (loca...	seckill_order @seckill (local...
	开始事务	备注	筛选
	排序	导入	导出
id	nickname	password	salt
18600000000	Charlie	5e7b3a9754c2777f96174c1a2b3c4d	(Null)
head	register_date	last_login_date	login_count
(Null)	(Null)	(Null)	0

seckill\_goods表

对象 seckill_user @seckill (localh... seckill_goods @seckill (loca... seckill_order @seckill (loca...						
开始事务 备注 筛选 排序 导入 导出						
id	goods_id	seckill_price	stock_count	start_date	end_date	
1	1	0.01	0	2021-01-15 10:58:00	2021-01-15 12:05:00	
2	2	0.01	50	2021-01-15 13:00:00	2021-01-15 13:05:00	
3	3	0.01	98	2021-01-15 14:00:00	2021-01-15 15:00:00	
4	4	0.01	98	2021-01-15 15:00:00	2021-01-15 16:00:00	

seckill\_order表

对象 seckill_user @seckill (localh... seckill_goods @seckill (loca... seckill_order @seckill (local...			
开始事务 备注 筛选 排序 导入 导出			
id	user_id	order_id	goods_id
1	18600000000	1	1
2	18600000000	2	3
3	18600000000	3	4

## 部署步骤

- 1、克隆仓库到本地

```
1 | git clone https://github.com/charliehxl/seckill.git
```

- 2、导入IDEA
- 3、项目的访问入口：[http://localhost:8080/login/to\\_login](http://localhost:8080/login/to_login)

初始账号和密码为：18600000000 / 000000

## 项目细节

### 安全

#### 登录

将明文密码进行两次MD5加密

客户端：Client\_Password=MD5(明文+固定salt)

服务端：Server\_Password=MD5(Client\_Password+随机salt)

第一次（在前端加密，客户端）：密码加密是（明文密码+固定盐值）生成md5用于传输，由于http是明文传输，当输入密码若直接发送服务端验证，此时被截取将直接获取到明文密码，获取用户信息。加salt是为了混淆密码，原则就是明文密码不能在网络上传输。

第二次：在服务端再次加密，当获取到前端发送来的密码后。通过MD5（密码+随机盐值）再次生成密码后存入数据库。防止数据库被盗的情况下，通过md5反查，查获用户密码。方法是盐值会在用户登陆的时候随机生成，并存在数据库中，这个时候就会获取到。

可以看到我们的seckill\_user表中也存在一个salt字段，用户登录生成的。

- 前端代码

```

1  function doLogin() {
2
3      // 获取用户输入密码
4      var inputPass = $("#password").val();
5      // 获取salt g_password_salt = "1a2b3c4d";
6      var salt = g_password_salt;
7      // md5+salt, 与服务器端的第一次MD5规则一致
8      var str = "" + salt.charAt(0) + salt.charAt(2) + inputPass +
salt.charAt(5) + salt.charAt(4);
9      var password = md5(str);
10
11     $.ajax({
12         url: "/login/do_login",
13         type: "POST",
14         data: {
15             mobile: $("#mobile").val(),
16             password: password
17         },
18         success: function (data) {
19             layer.closeAll();
20             if (data.code == 0) {
21                 layer.msg("成功");
22                 window.location.href = "/goods/to_list";
23             } else {
24                 layer.msg(data.msg);
25             }
26             console.log(data);
27         },
28         error: function () {
29             layer.closeAll();
30         }
31     });
32 }

```

- 后端代码

```

1  public String login(HttpServletResponse response, LoginVo loginVo) {
2      if (loginVo == null) {
3          throw new GlobalException(CodeMsg.SERVER_ERROR);
4      }
5      // 获取用户提交的手机号码和密码
6      String mobile = loginVo.getMobile();
7      String password = loginVo.getPassword();
8      // 判断手机号是否存在
9      SeckillUser user = this.getMiaoshaUserById(Long.parseLong(mobile));
10     if (user == null)
11         throw new GlobalException(CodeMsg.MOBILE_NOT_EXIST);
12
13     // 判断手机号对应的密码是否一致
14     String dbPassword = user.getPassword();
15     String dbSalt = user.getSalt();
16     String calcPass = MD5Util.formPassToDbPass(password, dbSalt);
17     if (!calcPass.equals(dbPassword))
18         throw new GlobalException(CodeMsg.PASSWORD_ERROR);
19
20     String token = UUIDUtil.uuid();

```

```

21     redisService.set(SeckillUserKeyPrefix.token, token, user);
22     Cookie cookie = new Cookie(COOKIE_NAME_TOKEN, token);
23     cookie.setMaxAge(SeckillUserKeyPrefix.token.expireSeconds());
24     cookie.setPath("/");
25     response.addCookie(cookie);
26     return token;
27 }

```

- 手机号正则化
  - 简单的正则应用

```

1  // 手机号正则
2  private static final Pattern mobilePattern = Pattern.compile("1\\d{10}");
3
4  public static boolean isMobile(String mobile) {
5      if (StringUtils.isEmpty(mobile))
6          return false;
7
8      Matcher matcher = mobilePattern.matcher(mobile);
9      return matcher.matches();
10 }

```

## 验证码

这种方式主要是防止客户端通过明文地址+goodsId将秒杀请求不停地发送到服务端，也有效防止机器人等手段参与秒杀。

- 生成验证码代码

```

1  @RequestMapping(value = "/verifyCode", method = RequestMethod.GET)
2  @ResponseBody
3  public Result<String> getMiaoshaVerifyCode(HttpServletRequest response,
4      SeckillUser user, @RequestParam("goodsId") long goodsId) {
5      if (user == null)
6          return Result.error(CodeMsg.SESSION_ERROR);
7
8      // 创建验证码
9      try {
10         BufferedImage image = seckillService.createVerifyCode(user,
11             goodsId);
12         ServletOutputStream out = response.getOutputStream();
13         // 将图片写入到resp对象中
14         ImageIO.write(image, "JPEG", out);
15         out.close();
16         out.flush();
17         return null;
18     } catch (Exception e) {
19         e.printStackTrace();
20         return Result.error(CodeMsg.SECKILL_FAIL);
21     }
22 }

```

```

1  public BufferedImage createverifyCode(SeckillUser user, long goodsId) {

```



```

2     if (user == null || goodsId <= 0) {
3         return null;
4     }
5
6     // 验证码的宽高
7     int width = 80;
8     int height = 32;
9
10    BufferedImage image = new BufferedImage(width,
11    height,BufferedImage.TYPE_INT_RGB);
12    Graphics g = image.getGraphics();
13    g.setColor(new Color(0xDCDCDC));
14    g.fillRect(0, 0, width, height);
15    g.setColor(Color.black);
16    g.drawRect(0, 0, width - 1, height - 1);
17    Random rdm = new Random();
18    for (int i = 0; i < 50; i++) {
19        int x = rdm.nextInt(width);
20        int y = rdm.nextInt(height);
21        g.drawOval(x, y, 0, 0);
22    }
23    String verifyCode = generateVerifyCode(rdm);
24    g.setColor(new Color(0, 100, 0));
25    g.setFont(new Font("Candara", Font.BOLD, 24));
26    g.drawString(verifyCode, 8, 24);
27    g.dispose();
28
29    // 计算表达式值，并把把验证码值存到redis中
30    int expResult = calc(verifyCode);
31    redisService.set(SeckillKeyPrefix.seckillVerifyCode, user.getId() + ","
32    + goodsId, expResult);
33    //输出图片
34    return image;
35 }
36
37 private String generateVerifyCode(Random rdm) {
38     int num1 = rdm.nextInt(10);
39     int num2 = rdm.nextInt(10);
40     int num3 = rdm.nextInt(10);
41     char op1 = ops[rdm.nextInt(3)];
42     char op2 = ops[rdm.nextInt(3)];
43     String exp = "" + num1 + op1 + num2 + op2 + num3;
44     return exp;
45 }

```

```

1 // 使用ScriptEngine计算验证码中的数学表达式的值
2 private int calc(String exp) {
3     try {
4         ScriptEngineManager manager = new ScriptEngineManager();
5         ScriptEngine engine = manager.getEngineByName("JavaScript");
6         return (Integer) engine.eval(exp); // 表达式计算
7     } catch (Exception e) {
8         e.printStackTrace();
9         return 0;
10    }
11 }
12

```

```

13 public boolean checkVerifyCode(SeckillUser user, long goodsId, int
    verifyCode) {
14     if (user == null || goodsId <= 0) {
15         return false;
16     }
17
18     Integer oldCode = redisService.get(SeckillKeyPrefix.seckillVerifyCode,
        user.getId() + "," + goodsId, Integer.class);
19     if (oldCode == null || oldCode - verifyCode != 0) { // !!!!!!!
20         return false;
21     }
22
23     redisService.delete(SeckillKeyPrefix.seckillVerifyCode, user.getId() +
        "," + goodsId);
24     return true;
25 }
26

```

在服务端计算出验证码的表达式的值，存储在服务端，客户端输入验证码的表达式值，传入服务端进行验证。

- 点击秒杀之前，向让用户输入验证码，分散用户的请求；
- 添加生成验证码的接口；
- 在获取秒杀路径的时候，验证验证码；
- ScriptEngine的使用（用于计算验证码上的表达式）。

当秒杀未开始时，商品详情页异步地向服务端发出获取商品详细信息的请求，同时，获取验证码。服务端收到获取验证码的请求后，生成验证码返回给客户端，同时，将验证码的结果存储在redis中，以便客户端发起秒杀请求时做验证码的校验。

## 秒杀接口隐藏

假使我们将秒杀地址写为静态地址，首先看客户端的秒杀操作逻辑：

```

1 <button type="button" id="buyButton" onclick="doMiaosha()">立即秒杀</button>
2
3 function doMiaosha() {
4     $.ajax({
5         url: "/miaosha/do_miaosha_static",
6         type: "POST",
7         data: {
8             goodsId: $("#goodsId").val(),
9         },
10        success: function (data) {
11            if (data.code == 0) {
12                getMiaoshaResult($("#goodsId").val());
13            } else {
14                layer.msg(data.msg);
15            }
16        },
17        error: function () {
18            layer.msg("客户端请求有误");
19        }
20    });
21 }

```

这样的话，用户点击秒杀按钮后，会向服务端请求秒杀商品的秒杀信息，客户端的POST请求是以明文的方式发送给服务器的，如果使用一种工具将POST请求体中的数据和请求的URL组合起来，构成一个完整的POST请求，然后不停地向服务器请求资源，则会给服务器带来很大的压力，同时，这样一种作弊的方式带来的用户体验也是极差的，这样一种设计缺陷会被别有用心的人用于不正当交易，因此，需要一种方式克服这种缺陷，这就引出了秒杀接口的隐藏。

### 实现秒杀接口地址的隐藏

在秒杀开始之前，秒杀接口地址不要写到客户端，而是在秒杀开始之后，将秒杀地址动态地在客户端和服务器间进行交互完成拼接。这样一来，秒杀开始之前，秒杀地址对客户端不可见。

- 后端代码

```
1 public Result<String> getMiaoshaPath(Model model, SeckillUser
  user, @RequestParam("goodsId") long goodsId, @RequestParam(value =
    "verifyCode", defaultValue = "0") int verifyCode) {
2
3     model.addAttribute("user", user);
4
5     if (user == null) {
6         return Result.error(CodeMsg.SESSION_ERROR);
7     }
8
9     // 校验验证码
10    boolean check = seckillService.checkVerifyCode(user, goodsId,
      verifyCode);
11    if (!check)
12        return Result.error(CodeMsg.REQUEST_ILLEGAL); // 检验不通过，请求非法
13
14    // 检验通过，获取秒杀路径
15    String path = seckillService.createSeckillPath(user, goodsId);
16    // 向客户端回传随机生成的秒杀地址
17    return Result.success(path);
18 }
```

```
1 //goodsId和verifyCode两个参数进行
2 public String createSeckillPath(SeckillUser user, long goodsId) {
3     if (user == null || goodsId <= 0) {
4         return null;
5     }
6     // 随机生成秒杀地址
7     String path = MD5Util.md5(UUIDUtil.uuid() + "123456");
8     redisService.set(SeckillKeyPrefix.seckillPath, "" + user.getId() + "_" +
      goodsId, path);
9     return path;
10 }
```

```
1 public boolean checkPath(SeckillUser user, long goodsId, String path) {
2     if (user == null || path == null)
3         return false;
4     // 从redis中读取出秒杀的path变量是否为本次秒杀操作执行前写入redis中的path
5     String oldPath = redisService.get(SeckillKeyPrefix.seckillPath, "" +
      user.getId() + "_" + goodsId, String.class);
6     return path.equals(oldPath);
7 }
```

- 前端代码

```
1  /*秒杀接口隐藏*/
2  function getSeckillPath() {
3      var goodsId = $("#goodsId").val();
4      g_showLoading();
5      $.ajax({
6          url: "/miaosha/path",
7          type: "GET",
8          data: {
9              goodsId: goodsId,
10             verifyCode: $("#verifyCode").val()
11         },
12         success: function (data) {
13             if (data.code == 0) {
14                 var path = data.data;
15                 doMiaosha(path);
16             } else {
17                 layer.msg(data.msg);
18             }
19         },
20         error: function () {
21             layer.msg("客户端请求有误");
22         }
23     });
24 }
25
26 /*真正做秒杀的接口，path为服务端返回的秒杀接口地址*/
27 function doMiaosha(path) {
28     $.ajax({
29         url: "/miaosha/" + path + "/do_miaosha_static",
30         type: "POST",
31         data: {
32             goodsId: $("#goodsId").val()
33         },
34         success: function (data) {
35             if (data.code == 0) {
36                 getMiaoshaResult($("#goodsId").val());
37             } else {
38                 layer.msg(data.msg);
39             }
40         },
41         error: function () {
42             layer.msg("客户端请求有误");
43         }
44     });
45 }
46
47 /*获取秒杀的结果*/
48 function getMiaoshaResult(goodsId) {
49     $.ajax({
50         url: "/miaosha/result",
51         type: "GET",
52         data: {
53             goodsId: $("#goodsId").val(),
54         },
55         success: function (data) {
```

```

56         if (data.code == 0) {
57             var result = data.data;
58             if (result < 0) {
59                 layer.msg("对不起，秒杀失败");
60             } else if (result == 0) {
61                 setTimeout(function () {
62                     getMiaoshaResult(goodsId);
63                 }, 200);
64             } else {
65                 layer.confirm("恭喜你，秒杀成功！查看订单？", {btn: ["确定",
"取消"]},
66                     function () {
67                         window.location.href = "/order_detail.htm?orderId="
+ result;
68                     },
69                     function () {
70                         layer.closeAll();
71                     });
72             }
73         } else {
74             layer.msg(data.msg);
75         }
76     },
77     error: function () {
78         layer.msg("客户端请求有误");
79     }
80 });
81 }

```

## 性能优化

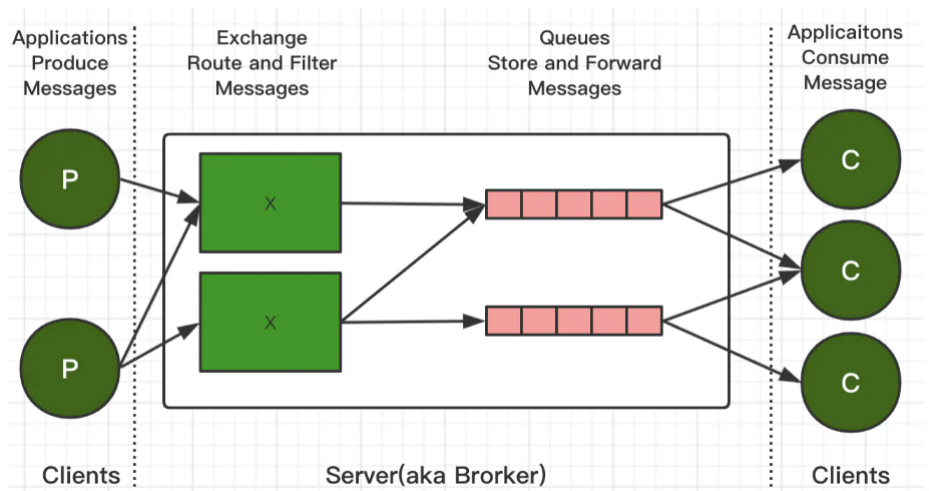
### 秒杀引入RabbitMQ

针对秒杀系统，最大的挑战就是请求的骤增。假设系统A在某一段时间请求数暴增，有5000个请求发送过来，系统A这时就会发送5000条SQL进入MySQL进行执行，MySQL对于如此庞大的请求当然处理不过来，MySQL就会崩溃，导致系统瘫痪。

使用消息队列主要有三个作用：

1、解耦 2、异步 3、削峰

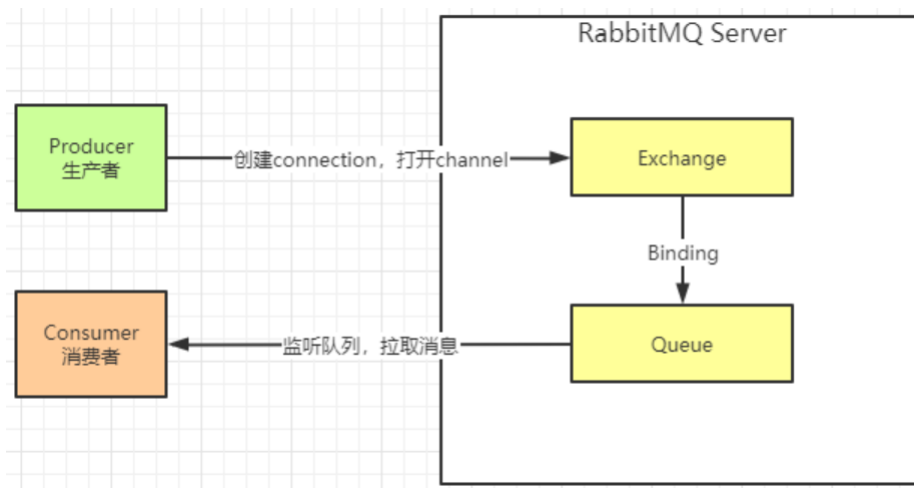
RabbitMQ的整体架构



RabbitMQ的组成:

- Broker: 消息队列服务进程。此进程包括两个部分: Exchange和Queue。
- Exchange: 消息队列交换机。按一定的规则将消息路由转发到某个队列。
- Queue: 消息队列, 存储消息的队列。
- Producer: 消息生产者。生产方客户端将消息同交换机路由发送到队列中。
- Consumer: 消息消费者。消费队列中存储的消息。

协同工作大概的流程为:



在项目的具体应用:

```

1 //在MQ中传递的秒杀信息 -- 包含参与秒杀的用户和商品的id
2 //将用户秒杀信息投递到MQ中 (使用direct模式的exchange)
3 public void sendMiaoshaMessage(SeckillMessage message) {
4     String msg = RedisService.beanToString(message);
5     logger.info("MQ send message: " + msg);
6     // 第一个参数为消息队列名, 第二个参数为发送的消息
7     amqpTemplate.convertAndSend(MQConfig.SECKILL_QUEUE, msg);
8 }

```

```

1 //处理收到的秒杀成功信息
2 @RabbitListener(queues = MQConfig.SECKILL_QUEUE)
3 public void receiveMiaoshaInfo(String message) {
4     logger.info("MQ: message: " + message);
5     SeckillMessage seckillMessage =
6     RedisService.stringToBean(message, SeckillMessage.class);
7     // 获取秒杀用户信息与商品id
8     SeckillUser user = seckillMessage.getUser();

```

```

8      long goodsId = seckillMessage.getGoodsId();
9
10     // 获取商品的库存
11     GoodsVo goods = goodsService.getGoodsVoByGoodsId(goodsId);
12     Integer stockCount = goods.getStockCount();
13     if (stockCount <= 0)
14         return;
15
16     SeckillOrder order =
orderservice.getSeckillOrderByUserIdAndGoodsId(user.getId(), goodsId);
17     if (order != null)
18         return;
19
20     // 减库存 下订单 写入秒杀订单
21     seckillService.seckill(user, goods);
22 }

```

## 秒杀引入Redis

Redis是什么？

Redis: Remote Dictionary Server(远程字典服务器)

是完全开源免费的，用C语言编写的，遵守BSD协议，是一个高性能的(key/value)分布式内存数据库，基于内存运行并支持持久化的NoSQL数据库，是当前最热门的NoSql数据库之一,也被人们称为数据结构服务器。

Redis 与其他 key - value 缓存产品有以下三个特点：

- Redis支持数据的持久化，可以将内存中数据保持在磁盘中，重启的时候可以再次加载进行使用
- Redis不仅仅支持简单的key-value类型的数据，同时还提供list, set, zset, hash等数据结构的存储。
- Redis支持数据的备份，即master-slave模式的数据备份

具体的使用功能有以下：

- 内存存储和持久化：redis支持异步将内存中的数据写到硬盘上，同时不影响继续服务
- 取最新N个数据的操作，如：可以将最新的10条评论的ID放在Redis的List集合里面
- 模拟类似于HttpSession这种需要设定过期时间的功能
- 发布、订阅消息系统
- 定时器、计数器

针对具体的项目操作：

### 1、Redis预减库存减少数据库的访问

在做秒杀时，需要先查询数据库中的商品库存，确保逻辑正确，在本项目中，我们将库存信息信息存储在redis中，从而可以减少对数据库的访问。

```

1 //redis 的get操作，通过key获取存储在redis中的对象
2 public <T> T get(KeyPrefix prefix, String key, Class<T> clazz) {
3     Jedis jedis = null;// redis连接
4
5     try {
6         jedis = jedisPool.getResource();

```

```

7         String realKey = prefix.getPrefix() + key;
8         String strValue = jedis.get(realKey);
9         T objValue = stringToBean(strValue, clazz);
10        return objValue;
11    } finally {
12        returnToPool(jedis);
13    }
14 }

```

```

1 // redis的set操作
2 public <T> boolean set(KeyPrefix prefix, String key, T value) {
3     Jedis jedis = null;
4     try {
5         jedis = jedisPool.getResource();
6         String strValue = beanToString(value);
7         if (strValue == null || strValue.length() <= 0)
8             return false;
9         String realKey = prefix.getPrefix() + key;
10        int seconds = prefix.expireSeconds();
11
12        if (seconds <= 0) {
13            jedis.set(realKey, strValue);
14        } else {
15            jedis.setex(realKey, seconds, strValue);
16        }
17        return true;
18    } finally {
19        returnToPool(jedis);
20    }
21 }

```

```

1 // 判断key是否存在于redis中
2 public <T> boolean exists(KeyPrefix keyPrefix, String key) {
3     Jedis jedis = null;
4     try {
5         jedis = jedisPool.getResource();
6         String realKey = keyPrefix.getPrefix() + key;
7         return jedis.exists(realKey);
8     } finally {
9         returnToPool(jedis);
10    }
11 }

```

```

1 //自增
2 public <T> Long incr(KeyPrefix keyPrefix, String key) {
3     Jedis jedis = null;
4     try {
5         jedis = jedisPool.getResource();
6         String realKey = keyPrefix.getPrefix() + key;
7         return jedis.incr(realKey);
8     } finally {
9         returnToPool(jedis);
10    }
11 }
12

```



```

13 //自减
14 public <T> Long decr(KeyPrefix keyPrefix, String key) {
15     Jedis jedis = null;
16     try {
17         jedis = jedisPool.getResource();
18         String realKey = keyPrefix.getPrefix() + key;
19         return jedis.decr(realKey);
20     } finally {
21         returnToPool(jedis);
22     }
23 }

```

```

1 //删除缓存中的用户数据
2 public boolean delete(KeyPrefix prefix, String key) {
3     Jedis jedis = null;
4     try {
5         jedis = jedisPool.getResource();
6         String realKey = prefix.getPrefix() + key;
7         Long del = jedis.del(realKey);
8         return del > 0;
9     } finally {
10        returnToPool(jedis);
11    }
12 }

```

```

1 //将对象转换为对应的json字符串
2 public static <T> String beanToString(T value) {
3     if (value == null)
4         return null;
5
6     Class<?> clazz = value.getClass();
7     /*首先对基本类型处理*/
8     if (clazz == int.class || clazz == Integer.class)
9         return "" + value;
10    else if (clazz == long.class || clazz == Long.class)
11        return "" + value;
12    else if (clazz == String.class)
13        return (String) value;
14    /*然后对Object类型的对象处理*/
15    else
16        return JSON.toJSONString(value);
17 }

```

```

1 // 根据传入的class参数，将json字符串转换为对应类型的对象
2 public static <T> T stringToBean(String strValue, Class<T> clazz) {
3
4     if ((strValue == null) || (strValue.length() <= 0) || (clazz ==
5 null))
6         return null;
7
8     // int or Integer
9     if ((clazz == int.class) || (clazz == Integer.class))
10        return (T) Integer.valueOf(strValue);
11    // long or Long
12    else if ((clazz == long.class) || (clazz == Long.class))

```

```

12         return (T) Long.valueOf(strValue);
13         // String
14     else if (clazz == String.class)
15         return (T) strValue;
16         // 对象类型
17     else
18         return JSON.toJavaObject(JSON.parseObject(strValue), clazz);
19 }
20

```

## 其他

### 页面静态化，前后端分离

页面静态化指的是将页面直接缓存到客户端。常用的技术有 `Angular.js`，`Vue.js`。

其实现方式就是通过 `ajax` 异步请求服务器获取动态数据，对于非动态数据部分缓存在客户端，客户端通过获取服务端返回的 `json` 数据解析完成相应的逻辑。

在本项目中，我们对**商品详情页**和**订单详情页**做了一个静态化处理。

对于商品详情页，异步地从服务端获取商品详情信息，然后客户端完成页面渲染工作。除此之外，对于秒杀信息的获取也是通过异步获取完成的。例如，当秒杀开始时，用户执行秒杀动作，客户端就会**轮询**服务器获取秒杀结果。而不需要服务器直接返回页面。

而对于订单详情页，实际上也是同样的思路。

```

1  //使用ajax从服务端请求页面数据，跳转到这个页面时就会执行(function有$)
2  $(function () {
3      // countdown();
4      getDetail();// 获取商品详情
5  });
6
7  function getDetail() {
8      var goodsId = g_getQueryString("goodsId");// goodsId为goods_list.html中详情url中的参数
9      $.ajax({
10         url: "/goods/to_detail_static/" + goodsId,
11         type: "GET",
12         success: function (data) { // data为edu.uestc.controller.GoodsListController#toDetailStatic的返回值
13             if (data.code == 0) {
14                 render(data.data);
15             } else {
16                 layer.msg(data.msg);
17             }
18         },
19         error: function () {
20             layer.msg("客户端请求有误");
21         }
22     });
23 }
24
25 /*渲染页面*/
26 function render(detail) {

```

```

27     var seckillStatus = detail.seckillStatus;
28     var remainSeconds = detail.remainSeconds;
29     var goods = detail.goods;
30     var user = detail.user;
31     if (user) {
32         $("#userTip").hide();
33     }
34     $("#goodsName").text(goods.goodsName);
35     $("#goodsImg").attr("src", goods.goodsImg);
36     $("#startTime").text(new Date(goods.startDate).format("yyyy-MM-dd
hh:mm:ss"));
37     $("#remainSeconds").val(remainSeconds);
38     $("#goodsId").val(goods.id);
39     $("#goodsPrice").text(goods.goodsPrice);
40     $("#seckillPrice").text(goods.seckillPrice);
41     $("#stockCount").text(goods.stockCount);
42     countdown();
43 }
44
45 function countdown() {
46     var remainSeconds = $("#remainSeconds").val();
47     var timeout;
48     if (remainSeconds > 0) { //秒杀还没开始，倒计时
49         $("#buyButton").attr("disabled", true);
50         $("#miaoshaTip").html("秒杀倒计时: " + remainSeconds + "秒");
51         timeout = setTimeout(function () {
52             $("#countDown").text(remainSeconds - 1);
53             $("#remainSeconds").val(remainSeconds - 1);
54             countdown();
55         }, 1000);
56     } else if (remainSeconds == 0) { //秒杀进行中
57         $("#buyButton").attr("disabled", false);
58         if (timeout) {
59             clearTimeout(timeout);
60         }
61         $("#miaoshaTip").html("秒杀进行中");
62         // 在倒计时结束时获取验证码（使用ajax异步向服务器请求验证码图片）
63         $("#verifyCodeImg").attr("src", "/miaosha/verifyCode?goodsId=" +
$("#goodsId").val());
64         $("#verifyCodeImg").show(); // 从服务器加载完验证码图片后，显示出来
65         $("#verifyCode").show();
66     } else { //秒杀已经结束
67         $("#buyButton").attr("disabled", true);
68         $("#miaoshaTip").html("秒杀已经结束");
69         $("#verifyCodeImg").hide();
70         $("#verifyCode").hide();
71     }
72 }

```

## 超卖问题

超卖问题实际上是两个问题：

1. 商品的库存减为负数，也就出现了超卖问题，这是不合理的；
2. 同一个用户秒杀到了两个一样的商品，这种情形也是超卖，应当避免。

来看看两个问题出现的情形：

对于**第一个问题**，我们知道，秒杀需要执行两个关键的操作，第一个是从数据库减库存，第二个是生成订单到插入到数据库，这两个操作共同构成了秒杀操作，因此，**秒杀操作是一个事务**。

```
1  @Transactional
2  public OrderInfo miaosha(MiaoshaUser user, GoodsVo goods) {
3      // 1. 减库存
4      boolean success = goodsService.reduceStock(goods);
5      if (!success) {
6          setGoodsOver(goods.getId());
7          return null;
8      }
9      // 2. 生成订单：向order_info表和maiosha_order表中写入订单信息
10     return ordersService.createOrder(user, goods);
11 }
```

如果商品是由10个，而达到 `goodsService.reduceStock()` 的请求有100个，当它们同时执行减库存操作时，会导致库存变为-90，这就引发了超卖问题。如何解决呢？

来看看减库存操作的数据库的 Mapper

```
1  @Update("UPDATE miaosha_goods SET stock_count = stock_count-1 WHERE
      goods_id=#{goodsId}")
2  int reduceStack(MiaoshaGoods miaoshaGoods);
```

因为每次 UPDATE 对于数据库来说都是原子的，如果每次减库存操作之前先判断库存是否大于零，则可以利用数据库层面的原子性来保证库存不会为负数，这也就解决了超卖的问题。

```
1  @Update("UPDATE miaosha_goods SET stock_count = stock_count-1 WHERE
      goods_id=#{goodsId} AND stock_count > 0")
2  int reduceStack(MiaoshaGoods miaoshaGoods);
```

AND stock\_count > 0 即为数据库层面解决超卖的保证。

对于**第二个问题**，考虑一种情形，如果一个未秒杀成功的用户**同时**对一个商品发出两次秒杀请求，对于两次秒杀请求，服务器层面会判断用户的两次秒杀请求为合法请求，然后完成从数据库减库存和将订单插入到数据库的操作，显然，这是不合理的。因为一个用户只能秒杀一个商品，如果执行成功，则订单表中会出现两条商品id和用户id相同的记录，一个商品的库存被同一个用户减了两次（也可能是多次），这就引发了超卖问题。

因此，为了解决这个问题，我们要充分利用事务的特性。从数据库减库存和将订单记录插入到数据库构成了事务，如果一个操作未执行成功，则事务会回滚。如果我们对 `miaosha_order` 中的 `user_id` 和 `goods_id` 字段创建一个联合**唯一索引**，则在插入两条 `user_id` 和 `goods_id` 相同的记录时，将会操作失败，从而事务回滚，秒杀不成功，这就解决了同一个用户发起对一个商品同时发起多次请求引发的超卖问题。

至此，通过上述的分析，超卖问题就可以得到解决了。总结起来如下。

**总结：**

- SQL加库存数量的判断：防止库存变为负数；
- 数据库加**唯一索引**：防止用户重复购买。

# 项目与课程总结

---

整个学期课程学下来，收获满满。因为解决课程不会的问题，认识了很多厉害的同学，同时也为一个跨考生打开了开发的“冰山一角”，在此很感谢张老师的课程细心安排，另外针对同学的实习也尽心尽力。

针对该项目，也存在很多待优化的地方，希望能得到老师的指导。