一　秒杀对于系统的特点

瞬间高并发。在瞬间超过系统负荷的请求如何处理---消息队列缓存机制。

二　创建 springboot 项目

三　导入依赖（rabbitmq）

```
<!--rabbitmq-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

四　创建交换机、队列

```
@SpringBootApplication
@EnableEurekaClient
@MapperScan("cn.shu.seckill.mapper")
public class SeckillStarter {
    public static void main(String[] args) {
        SpringApplication.run(SeckillStarter.class,args);
    }

    /**
     * 创建交换机 seckillEX
     * @return
     */
    @Bean
    public DirectExchange createEX(){
        return new DirectExchange("seckillEX");
    }

    /**
     * 创建队列seckillQueue
     * @return
     */
    @Bean
    public Queue createQueue(){
        return new Queue("seckillQueue");
    }

    /**
     * 绑定交换机、队列和路由key seckill
     * @return
```

```
     */
    @Bean
    public Binding createBinding(){
        return BindingBuilder.bind(createQueue()).to(createEX()).with("seckill");
    }
}
```

五　秒杀接口(生产者)

```
/**
 *
 * @param seckillId 商品ID
 * @return
 */
@Autowired
private RabbitTemplate rabbitTemplate;

@RequestMapping("/{seckillId}")
public SysResult seckill(@PathVariable long seckillId){
    System.out.println(seckillId);
    //模拟不同用户访问商品
    //只允许一个用户秒杀一次 可以使用redis
    //模拟用户的电话号码，实际应该从系统获取用户信息
    int userPhone=new Random().nextInt(99999);
    System.out.println(userPhone);
    //消息发送到队列
    //交换机seckillEX 路由key：seckill
    String msg=userPhone+"/"+seckillId;
    rabbitTemplate.convertAndSend("seckillEX","seckill",msg);
    return SysResult.ok();
}
```

六　消费者

```
/**
 * @作者 舒新胜
 * @项目 easymall-2002-all
 * @创建时间 2020/6/12 10:03
 */
@Component
public class SeckillConsumer {
    @Autowired
    private SkillMapper skillMapper;

    @Autowired
    private StringRedisTemplate redisTemplate;


    @RabbitListener(queues="seckillQueue")
    public void consum(String msg){
        System.out.println("接受到秒杀消息：msg);
        //消息格式 msg=电话号码/seckillId
        /*更新成功
        update seckill set number=number-1
        where seckill_id=#{seckillId}
        AND number>0
        and now()>start_time
        and now()<end_time
       */
```

```
        long phoneNumber = Long.parseLong(msg.split("/")[0]);
        long seckillId = Long.parseLong(msg.split("/")[1]);


        //更新数据库库存，10 万并发，每次都要判断
        int result=skillMapper.decrNumberById(seckillId);
        //更新库存失败
        if (result==0){
            return;
        }
        //更新库存成功
        Success suc=new Success();
        suc.setSeckillId(seckillId);
        suc.setUserPhone(phoneNumber);
        suc.setCreateTime(new Date());
        suc.setState(0);
        //插入数据库成功的信息
        skillMapper.insertSuccess(suc);
    }

}
```

1 问题点

减库存和判断条件同时在一个 sql 语句中，效率低下

- 秒杀商品即使已经为 0 了，大量消费逻辑依然在执行数据库的 update，无故消耗了数据库的资源
- 高并发时，数据库线程安全问题，导致超卖

2 解决

引入 redis 解决上述问题，在 redis 中执行减库存

decr number redis 会把减完的结果返回消费端

拿着这个结果判断当前是否有减库存的权限.

线程并发安全

redis 执行减库存数字,有没有可能 2 个 redis 的客户端执行 decr num 得到同样的结果?

? redis 是个单线程单进程的软件,性能高?为什么设计成单线程

- 单线程,非阻塞线程设计
- 内存运行

单线程,相对于多线程,性能略低.redis 吞吐量(单位时间内处理的数据量)

经过测试发现 redis 的吞吐量,无论是多线程还是单线程

瓶颈不在 redis 本身,在网络带宽,单线程足以处理,而且单线程不用考虑线程安全,不同浪费 cpu 切换资源

2.A 导入依赖 redis 依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-redis</artifactId>
</dependency>
```

2.B redis 中创建模拟数据

模拟 num_hw 商品秒杀数量 80

```
10.42.175.170:8002> set num_hw 80
OK
```

2.C 消费者代码增加 redis 逻辑

```
@Component
public class SeckillConsumer {
    @Autowired
    private SkillMapper skillMapper;

    @Autowired
    private StringRedisTemplate redisTemplate;


    @RabbitListener(queues="seckillQueue")
    public void consum(String msg){
        System.out.println("接受到秒杀消息："+msg);
        //消息格式 msg=电话号码/seckillId
        /*更新成功
        update seckill set number=number-1
        where seckill_id=#{seckillId}
        AND number>0
        and now()>start_time
        and now()<end_time

       */
        long phoneNumber = Long.parseLong(msg.split("/")[0]);
        long seckillId = Long.parseLong(msg.split("/")[1]);
//将 num_hw 商品数量减一
        Long decr=redisTemplate.opsForValue().increment("num_hw",-1);
        if(decr<0){
            //进入说明商品被其它消费端消费完了
            System.out.println("已被秒杀完了");
            return;
        }

        //更新库存
        int result=skillMapper.decrNumberById(seckillId);
        //更新库存失败
        if (result==0){
            return;
        }
        //更新库存成功
        Success suc=new Success();
        suc.setSeckillId(seckillId);
        suc.setUserPhone(phoneNumber);
        suc.setCreateTime(new Date());
        suc.setState(0);
        //插入
        skillMapper.insertSuccess(suc);
    }

}
```