1. 环境搭建

安装本节课的必备环境

RocketMQ 消息中间件安装

RocketMQ 是由阿里捐赠给 Apache 的一款分布式、队列模型的开源消息中间件

第四章
秒杀系统订单模块

欧阳修

本章导学

实现秒杀系统订单模块

本章导学

1．环境安装

- [ ] RocketMQ 消息中间件

2．订单生成

- [ ] 订单问题分析
- [ ] 设计用户表 & 订单表
- [ ] 实现创建订单功能
- [ ] 订单 ID 的生成方案
- [ ] 通过雪花算法生成订单 ID

3．整合消息中间件

- [ ] 整合 RocketMQ 到项目中
- [ ] 测试发送与接收

4．订单消息处理

- [ ] 发送创建订单信息
- [ ] 订单消息处理
- [ ] 扣减库存
- [ ] 测试订单创建流程

本章导学

使用到的技术点

1. MySQL
2. Spring Boot
3. MyBatis
4. Redis
5. 雪花算法
6. Jmeter 压测工具
7. RocketMQ 消息中间件

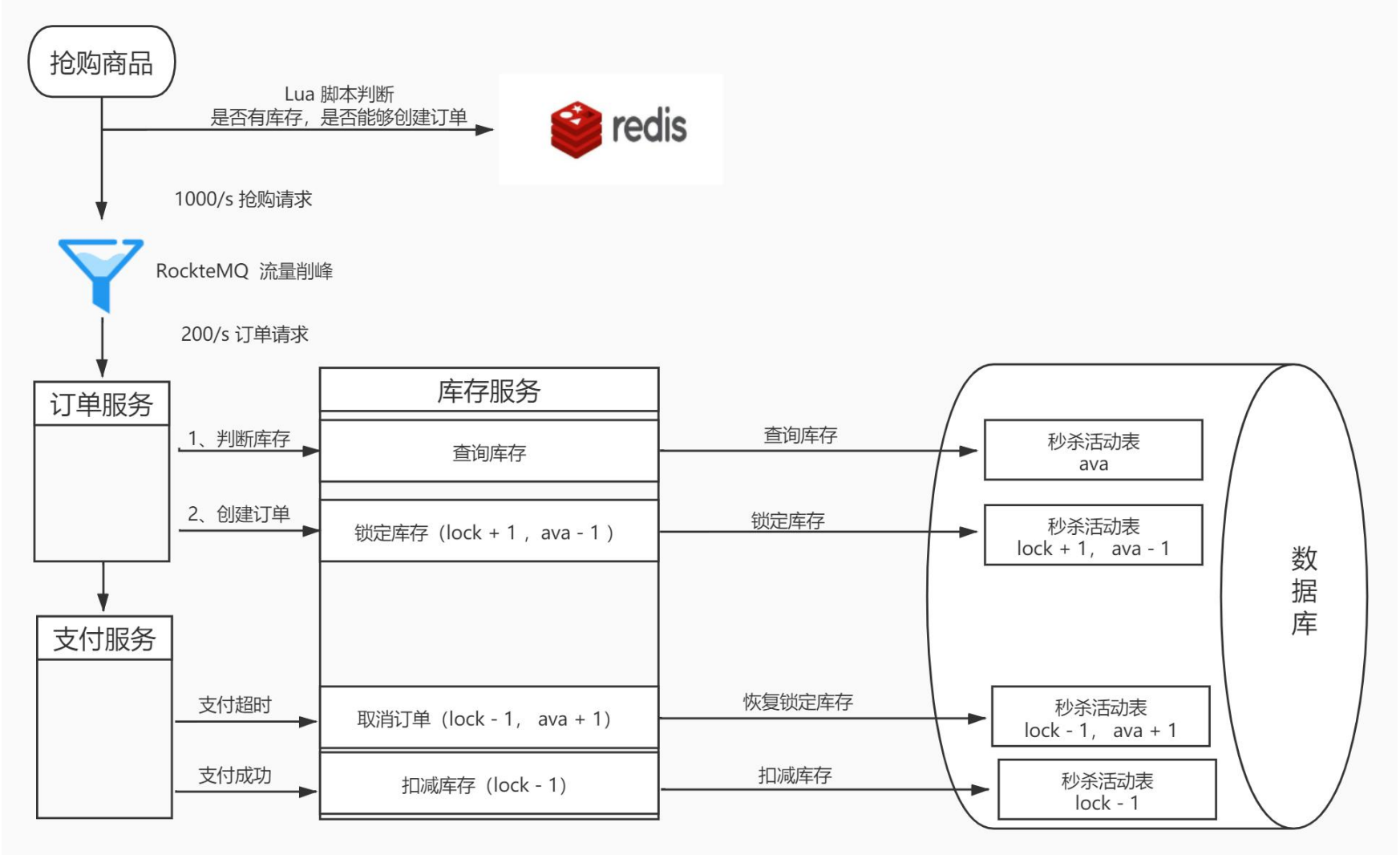
秒杀流程分析

本章将要实现的部分

秒杀活动流程分析

九章算法
www.jiuzhang.com

整体流程

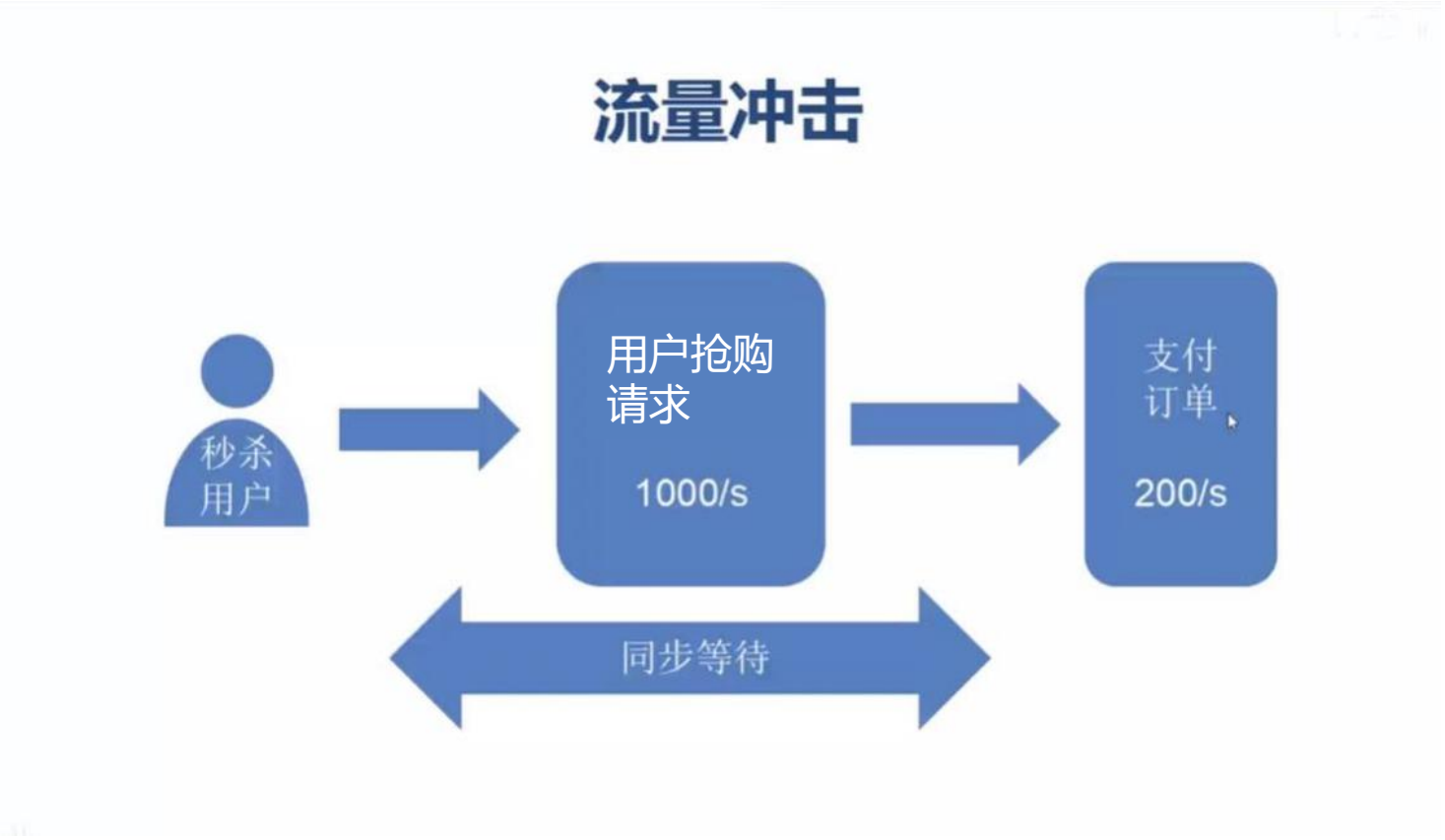


2．订单生成

生成抢购订单

1. 流量冲击从哪里来，冲击了哪些服务

流量冲击问题

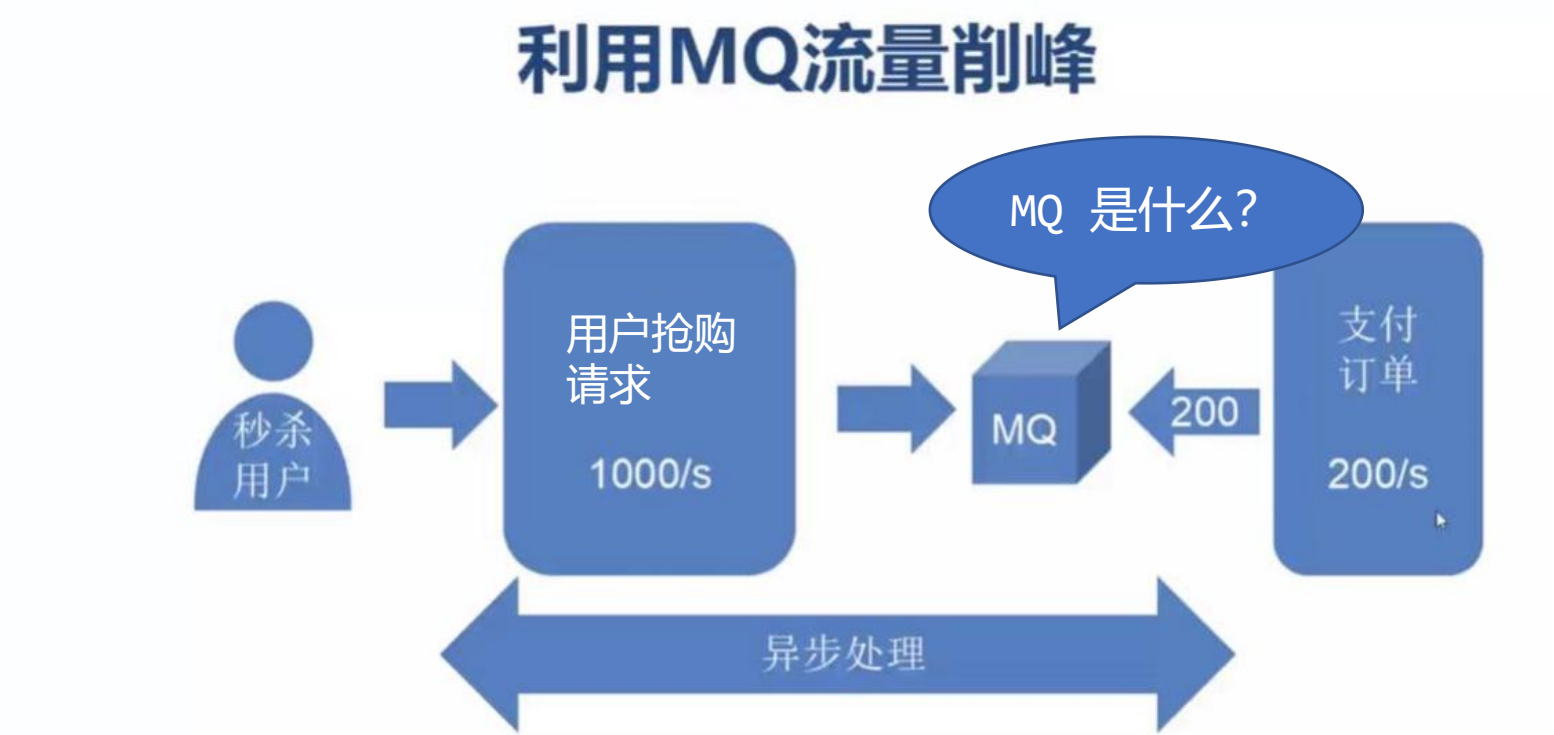


2．怎样应对流量冲击?

核心理念：削峰填谷(Peak Load Shifting)、异步（Asynchronous）解耦

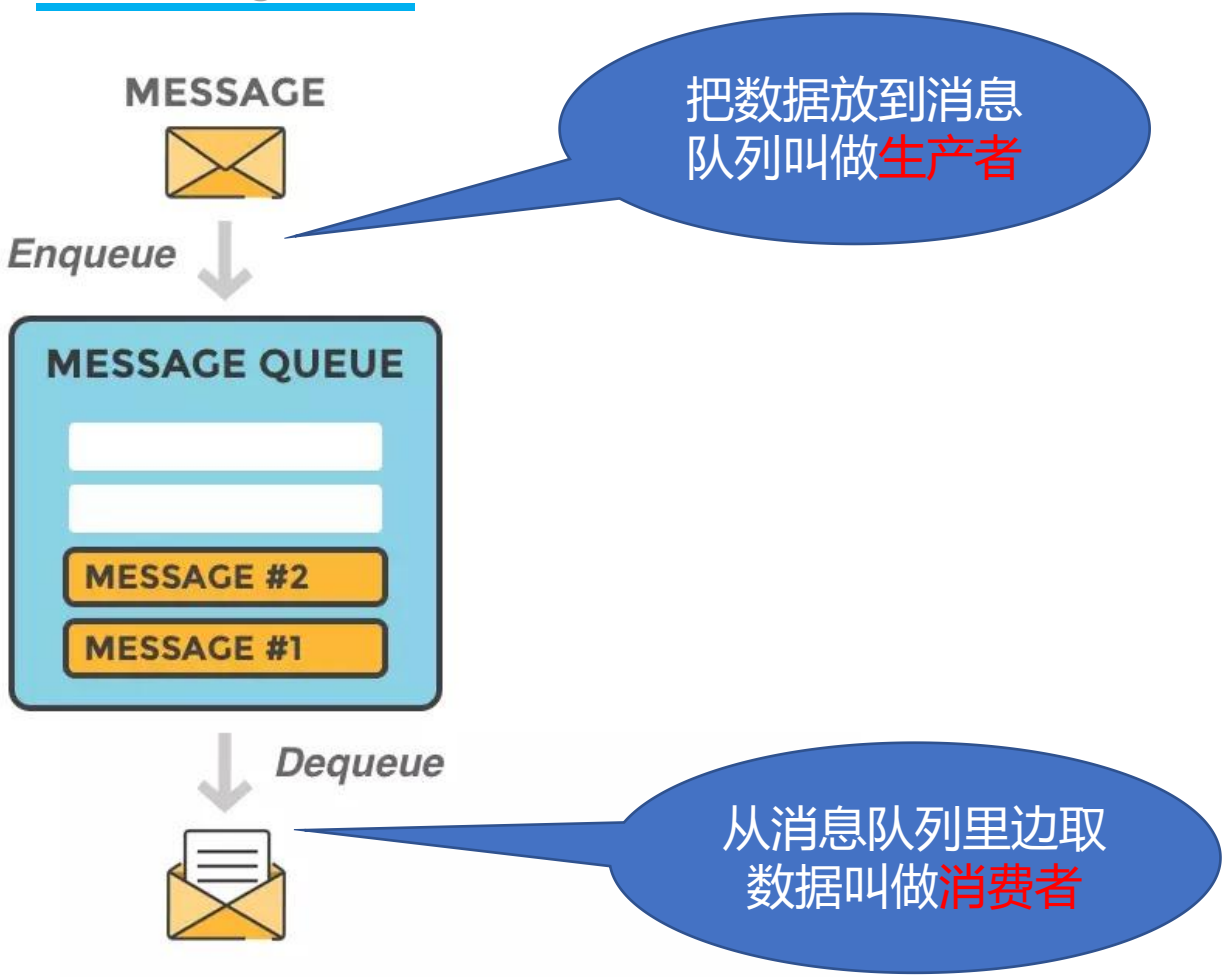
3. 瞬间的高流量冲击下，需要进行订单流量削峰填谷

通过 Message Queue 进行削峰



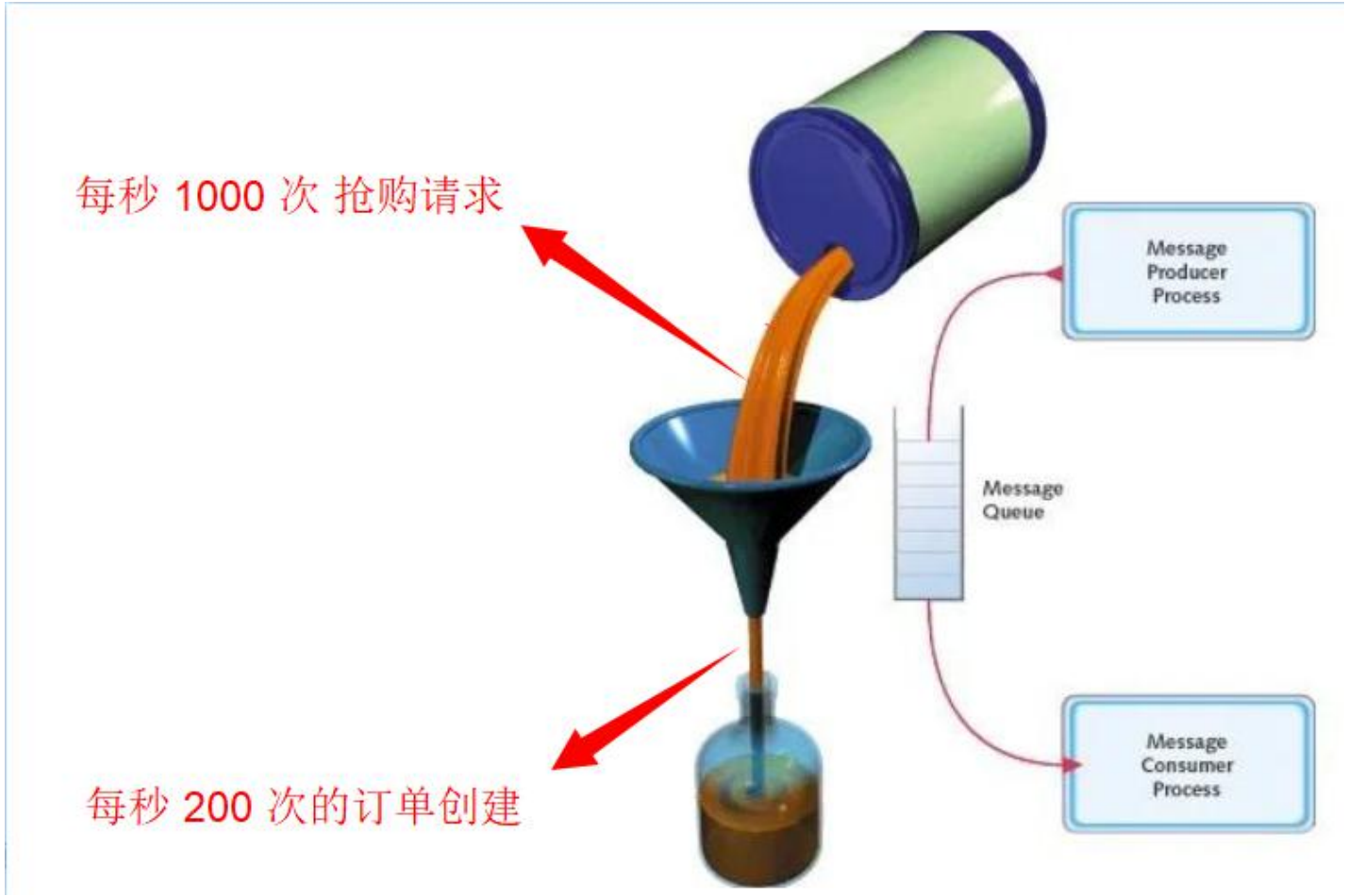
4. 消息队列是什么?

MQ = Message Queue



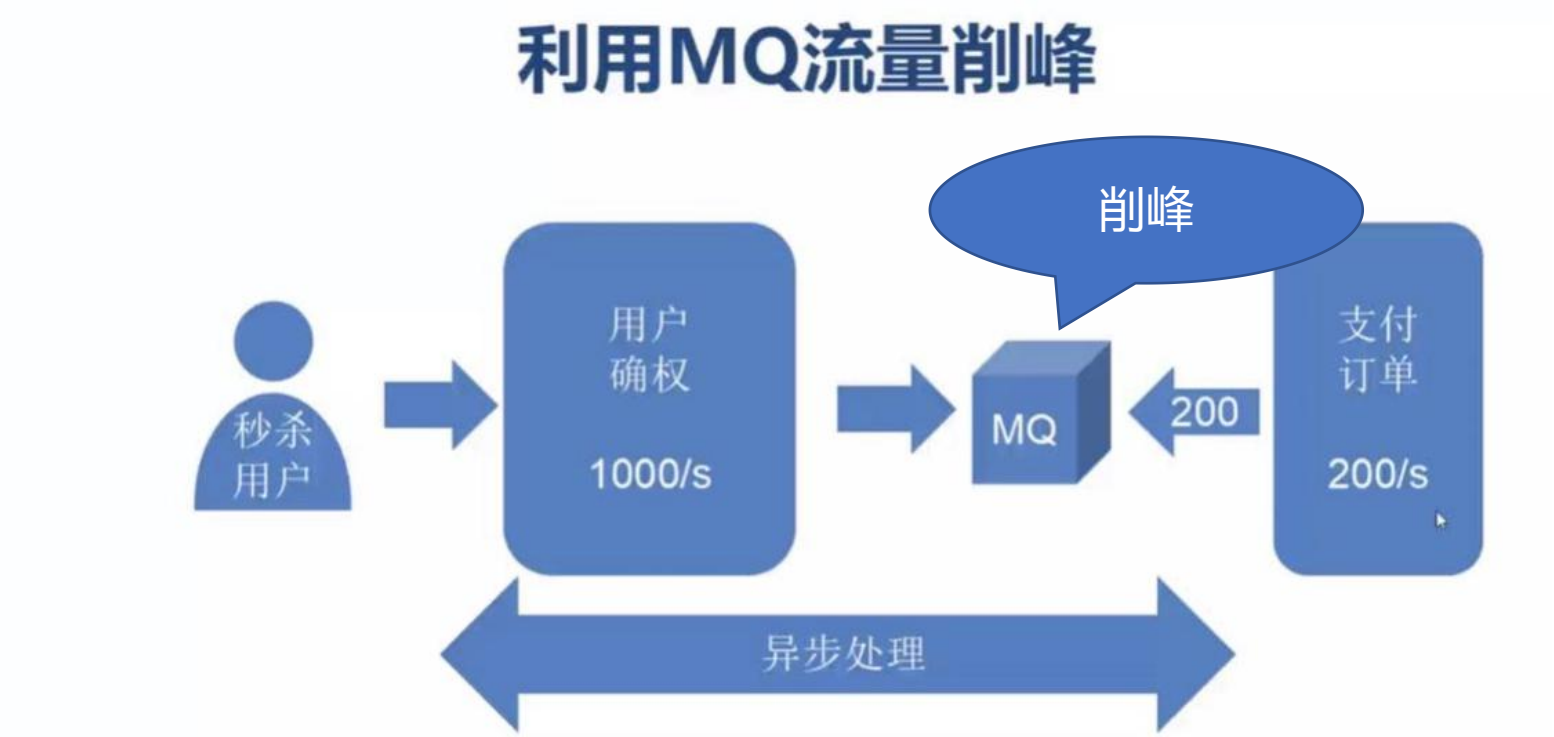
5. 使用消息队列有什么作用?

汹涌而至的消息(流量)从敞口流入，经过缩口以固定的流速输出给消费者（数据库）



6. 消息队列削峰

通过消息队列解决瞬间的高流量冲击问题



1. 哪些字段是必须的?

秒杀订单表需要哪些字段

商品ID 秒杀订单需要一个对应的商品

秒杀活动ID 秒杀订单需要对应一个秒杀活动

支付金额 秒杀订单需要支付的金额

用户ID 哪位用户进行抢购的

状态 是否已经支付、发货、收货等

支付和创建时间 什么时候下单的、什么时候付款的

2．订单表 应该有哪些对应的字段

字段 column	类型 type	解释 explain
id	BIGINT	订单ID
commodity_id	BIGINT	商品ID
seckill_activity_id	BIGINT	秒杀活动ID
money	DECIMAL	支付金额
user_id	BIGINT	用户ID
status	TINYINT	状态：0未支付，1已支付
pay_time	DATETIME	支付时间
create_time	DATETIME	创建时间

3. 用户表 应该有哪些对应的字段

字段 column	类型 type	解释 explain
id	BIGINT	用户ID
user_name	VARCHAR	用户名
address	VARCHAR	地址
phone	VARCHAR	手机号

4．表创建成功后，我们来逆向生成代码

```
<!-- targetPackage：mapper接口生成的位置 -->
<javaClientGenerator type="XMLMAPPER"
                     targetPackage="com.jiuzhang.seckill.db.mappers"
                     targetProject="./src/main/java">
    <!-- enableSubPackages:是否让schema作为包的后缀 -->
    <property name="enableSubPackages" value="true"/>
</javaClientGenerator>
<table schema="" tableName="seckill_order" domainObjectName="SeckillOrder"  enableCountByExample="false"
       enableSelectByExample="false" enableUpdateByExample="false" selectByExampleQueryId="false">
</table>
<table schema="" tableName="seckill_user" domainObjectName="SeckillUser"  enableCountByExample="false"
       enableSelectByExample="false" enableUpdateByExample="false" selectByExampleQueryId="false">
</table>
```

1. 创建订单的具体实现

```java
/**
 * 创建订单
 */
public Order createOrder(long seckillActivityId,long userId) throws Exception {
    /*
     * 1.创建订单
     */
    SeckillActivity seckillActivity = seckillActivityDao.querySeckillActivityById(seckillActivityId);
    Order order = new Order();
    //采用雪花算法生成订单ID
    order.setOrderNo("");
    order.setSeckillActivityId(seckillActivity.getId());
    order.setUserId(userId);
    order.setOrderAmount(seckillActivity.getSeckillPrice());
    /*
     *2.发送创建订单消息
     */
```

2. 传统方式 & 雪花算法

1.传统方式

1. 时间戳 + N 位随机数

这种方法在分布式系统内会产生 ID 碰撞。

2. UUID

入数据库性能差，因为UUID是无序的。

2. 雪花算法（SnowFlake）

雪花算法是 Twitter 的分布式自增 ID 算法，

经测试 SnowFlake 每秒可以产生 26 万个自增可排序的 ID。

Twitter 的 SnowFlake 生成 ID 能够按照时间有序生成。

在分布式系统内不会产生 ID 碰撞（由 DataCenter 和 WorkerID 做区分）并且效率较高。

2. 传统方式 & 雪花算法

生成方式	是否产生 ID 碰撞	ID 是否有序
时间戳 + 随机数	是	否
UUID	否	否
雪花算法（SnowFlake）	否	是

3. 雪花算法参考资料

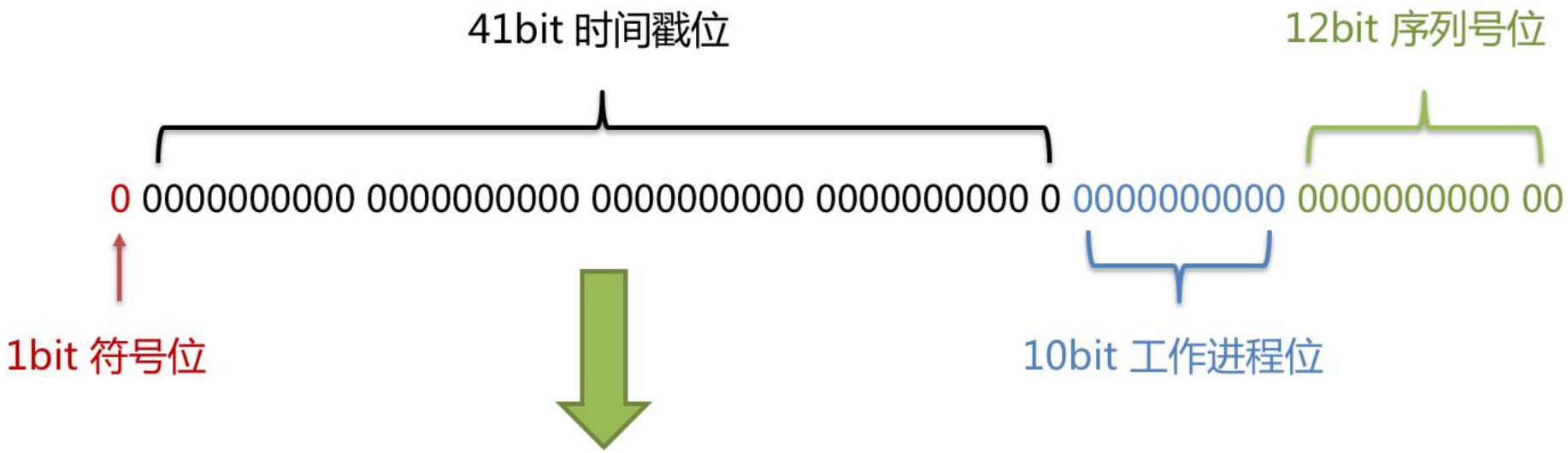
有兴趣的同学可以参考现有的开源实现

1. Scala 版详见开源项目 （Twitter 官方开源版本）

2. Java 版本中文描述

4. 雪花算法结构图

通过结构图来了解雪花算法是如何生成的



时间范围： 2^{41} / (365 * 24 * 60 * 60 * 1000L) = 69.73年
工作进程数量： 2^{10} = 1024
生成不碰撞序列的TPS： 2^{12} * 1000 = 409.6万

1. 创建雪花算法工具对象

```
/**
 * datacenterId;  数据中心
 * machineId;     机器标识
 * 在分布式环境中可以从机器配置上读取
 * 单机开发环境中先写死
 */
private SnowFlake snowFlake = new SnowFlake( datacenterId: 1,  machineId: 1);
```

2. 使用雪花算法生成 ID

```java
/**
 * 创建订单
 */
public Order createOrder(long seckillActivityId,long userId) throws Exception {
    /*
     * 1.创建订单
     */
    SeckillActivity seckillActivity = seckillActivityDao.querySeckillActivityById(seckillActivityId);
    Order order = new Order();
    //采用雪花算法生成订单ID
    order.setOrderNo(String.valueOf(snowFlake.nextId()));
    order.setSeckillActivityId(seckillActivity.getId());
    order.setUserId(userId);
    order.setOrderAmount(seckillActivity.getSeckillPrice());
    /*
     *2.发送创建订单消息
```

测试雪花算法生成订单 ID

3. 整合消息中间件

设计并实现相关数据表及功能

1. pom.xml 文件中添加依赖

引入依赖

```
<!-- rocketmq -->
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>2.1.1</version>
</dependency>
```

2. application.properties 配置文件中配置信息

配置 RocketMQ

```
### RocketMQ ###
rocketmq.name-server=139.196.48.177:9876
rocketmq.producer.group=my-group
```

1. 消息生产者开发

向消息队列中发送消息

```java
@Service
public class RocketMQService {

    @Autowired
    private RocketMQTemplate rocketMQTemplate;

    /**
     *  发送消息
     * @param topic
     * @param body
     * @throws Exception
     */
    public void sendMessage(String topic,String body) throws Exception{
        Message message = new Message(topic,body.getBytes());
        rocketMQTemplate.getProducer().send(message);
    }
```

2. 测试消息发送

测试能否发送

```
@SpringBootTest
public class MQTest {

    @Autowired
    private RocketMQService rocketMQService;

    @Test
    public void sendMQTest() throws Exception {
        rocketMQService.sendMessage( topic: "test-jiuzhang", body: "hello world "+ new Date().toString());
    }
}
```

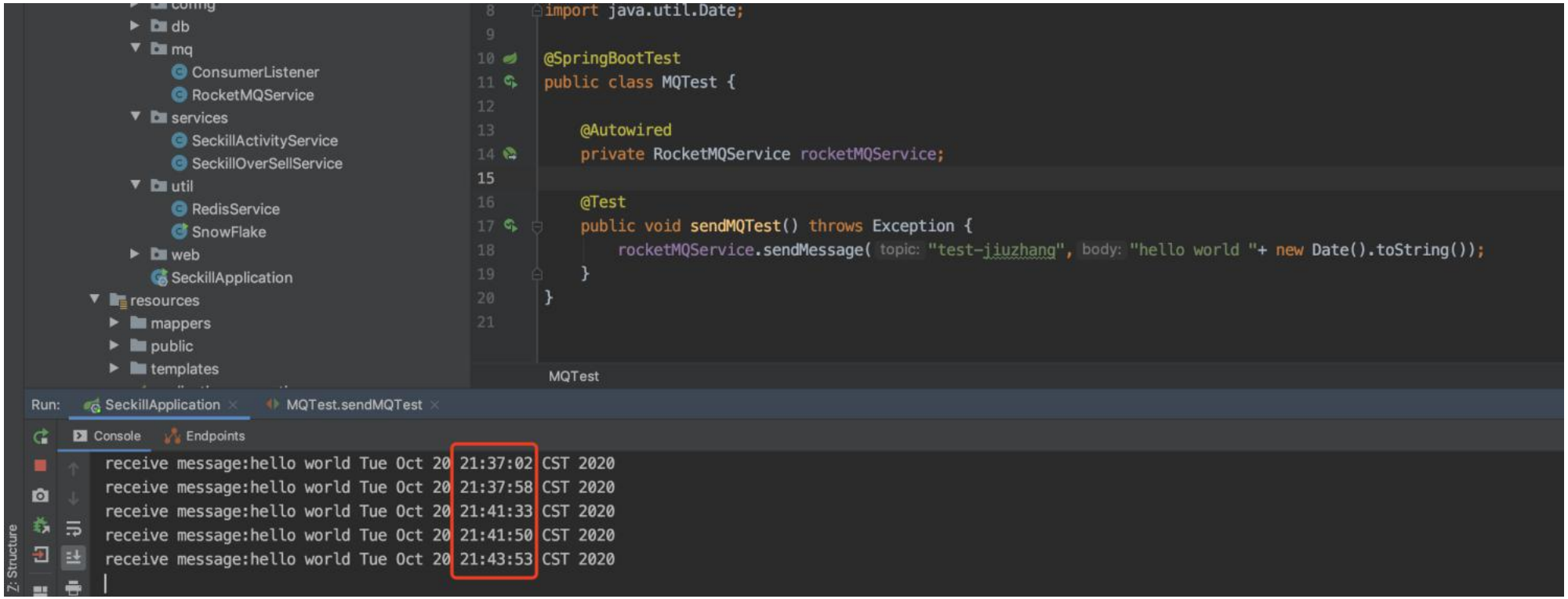
3. 消息消费者开发

从消息队列中取出消息

```
@Component
@RocketMQMessageListener(topic = "test-jiuzhang", consumerGroup = "conmuserGrop-jiuzhang")
public class ConsumerListener implements RocketMQListener<MessageExt> {
    @Override
    public void onMessage(MessageExt messageExt) {
        try {
            String body = new String(messageExt.getBody(), charsetName: "UTF-8");
            System.out.println("receive message:" + body);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
}
```

4. 测试消费消息

启动项目后，执行单元测试中发送消息的方法，能够正常收到发送的消息，证明rocketMQ在项目中发送消息和接收消息正常。



4．订单消息处理

1. 创建订单时发送订单消息

将订单信息发送到信息队列中

```java
/**
 * 创建订单
 */
public Order createOrder(long seckillActivityId,long userId) throws Exception {
    /*
     * 1.创建订单
     */
    SeckillActivity seckillActivity = seckillActivityDao.querySeckillActivityById(seckillActivityId);
    Order order = new Order();
    //采用雪花算法生成订单ID
    order.setOrderNo(String.valueOf(snowFlake.nextId()));
    order.setSeckillActivityId(seckillActivity.getId());
    order.setUserId(userId);
    order.setOrderAmount(seckillActivity.getSeckillPrice());
    /*
     *2.发送创建订单消息
     */
    rocketMQService.sendMessage( topic: "seckill_order", JSON.toJSONString(order));
    return order;
}
```

2. 消费者接收到订单消息后将订单信息插入到数据库中

将订单信息存入数据库

```java
@Slf4j
@Component
@RocketMQMessageListener(topic = "seckill_order", consumerGroup = "seckill_order_group")
public class OrderConsumer implements RocketMQListener<MessageExt> {

    @Autowired
    private OrderDao orderDao;

    @Autowired
    private SeckillActivityDao seckillActivityDao;

    @Override
    @Transactional
    public void onMessage(MessageExt messageExt) {
        //1.解析创建订单请求消息
        String message = new String(messageExt.getBody(), StandardCharsets.UTF_8);
        log.info("接收到创建订单请求: " + message);
        Order order = JSON.parseObject(message, Order.class);
        order.setCreateTime(new Date());
        //2.锁定库存
        boolean lockStockResult = seckillActivityDao.lockStock(order.getSeckillActivityId());
        if (lockStockResult) {
            //订单状态 0:没有可用库存，无效订单 1:已创建等待付款 ,2:支付完成
            order.setOrderStatus(1);
        } else {
            order.setOrderStatus(0);
        }
        //3.插入订单
        orderDao.insertOrder(order);
    }
}
```

库存扣减

扣减库存 SQL 使用数据库乐观锁来实现

```
<update id="lockStock" parameterType="java.lang.Long">
  update seckill_activity
  set available_stock = available_stock - 1,
      lock_stock = lock_stock + 1
  where id = #{id,jdbcType=BIGINT}
  and  available_stock > 0
</update>
```

4.4 测试订单创建流程

代码演示流程

回顾与总结

回顾并总结本节主要的知识点

本节课学习总结

2．订单生成

- [ ] 订单问题分析
- [ ] 设计用户表 & 订单表
- [ ] 实现创建订单功能
- [ ] 订单 ID 的生成方案
- [ ] 通过雪花算法生成订单 ID

3．整合消息中间件

- [ ] 整合 RocketMQ 到项目中
- [ ] 测试发送与接收

4．订单消息处理

- [ ] 发送创建订单信息
- [ ] 订单消息处理
- [ ] 扣减库存
- [ ] 测试订单创建流程

秒杀活动流程分析

九章算法
www.jiuzhang.com

整体流程