# JavaEE平台技术
# RocketMQ消息服务器

邱明 博士

厦门大学信息学院

mingqiu@xmu.edu.cn

# 提纲

- 为什么需要缓存
- 不同类型的缓存技术
- Redis简介

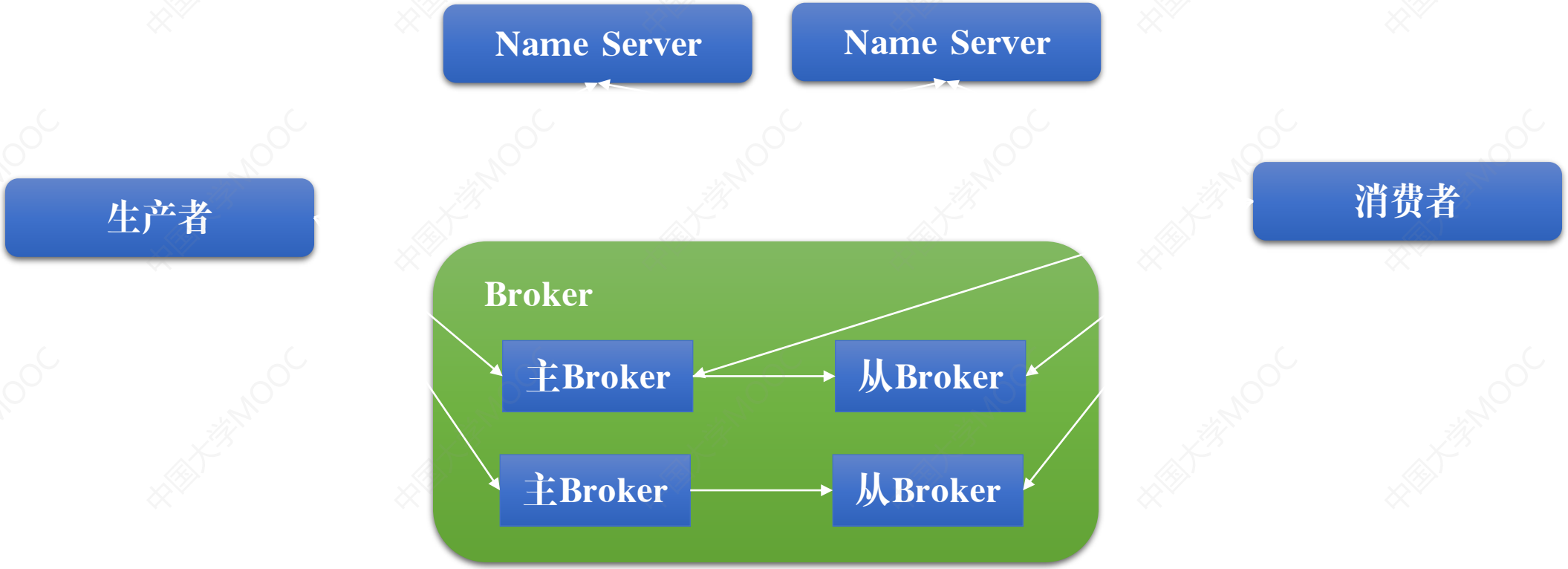
# 1 消息服务器

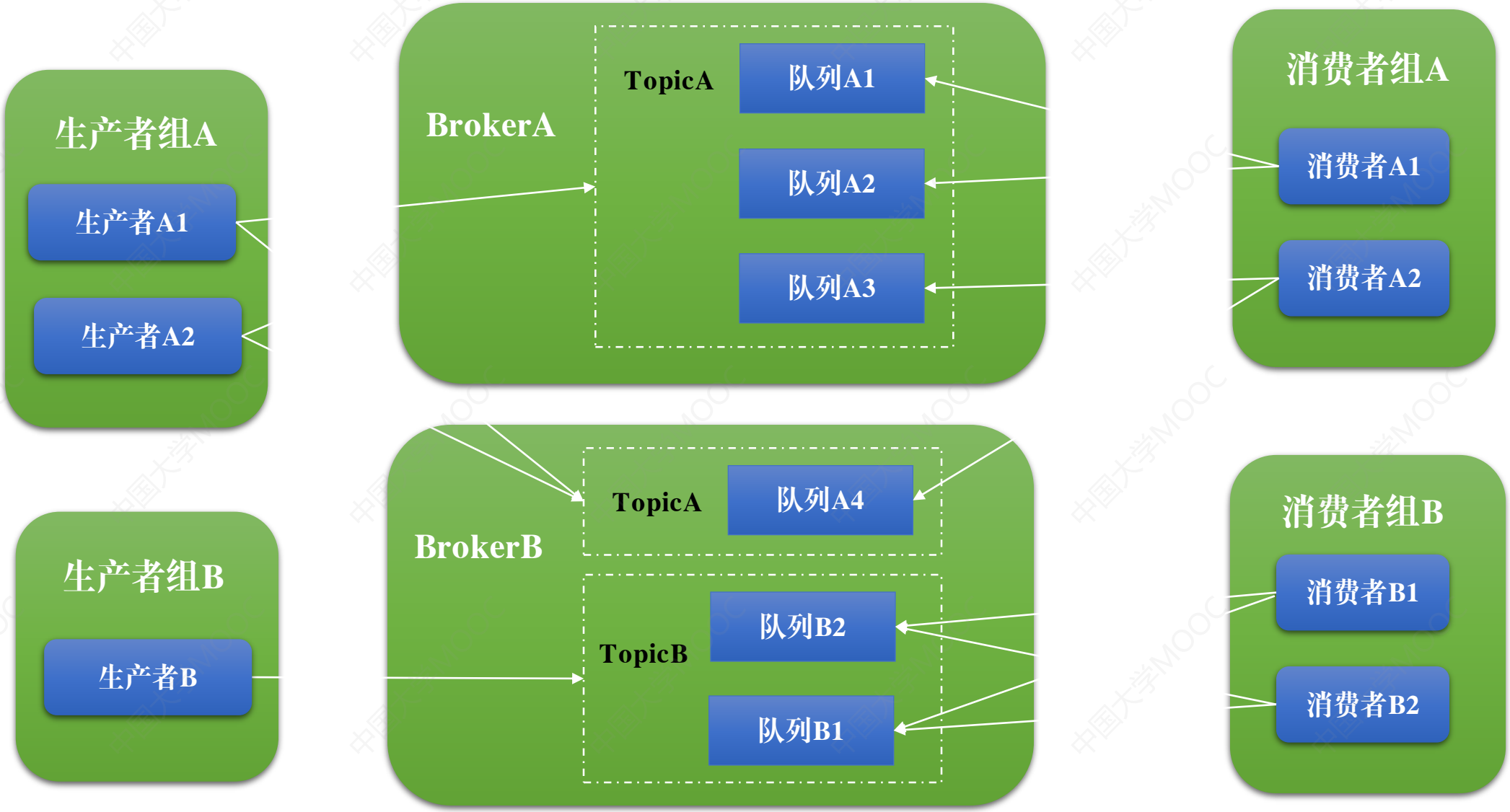
- 异步处理
- 系统解耦（解决不同重要程度、不同能力级别系统之间依赖导致一死全死）
- 削峰填谷（主要解决瞬时写压力大于应用服务能力导致消息丢失、系统奔溃等问题）日志处理
- 提升性能（当存在一对多调用时，可以发一条消息给消息系统，让消息系统通知相关系统）

# 1 消息服务器

# 1 消息服务器

# 1 消息服务器


Name Server
Name Server
生产者
消费者
Broker
主Broker
从Broker
主Broker
从Broker


生产者组A
生产者A1
生产者A2
生产者组B
生产者B
BrokerA
TopicA
队列A1
队列A2
队列A3
BrokerB
TopicA
队列A4
TopicB
队列B2
队列B1
消费者组A
消费者A1
消费者A2
消费者组B
消费者B1
消费者B2

# 1 消息服务器

- 可以提高应用的稳定性，当程序fail后，已经写入外部消息队列的数据依旧是保存的，
- 用线程的话，会占用主服务器资源， 消息队列的话， 可以放到其他机器上运行， 让主服务器尽量多的服务其他请求。
  - 如果用户不急着知道结果的操作， 用消息队列， 否则再考虑用不用线程。
- 解耦更充分，架构更合理多线程是在编程语言层面解决问题消息队列是在架构层面解决问题
  - 架构层面解决问题是“觉悟比较高的方式“，理想情况下应该限制语言层面滥用多线程，能不用就不用不关心执行结果的都可以放到消息队列，不需要及时到达，放到消息队列中慢慢消化

# 1 消息服务器

- Name Server：是一个几乎无状态节点，可集群部署，在消息队列 RocketMQ 版中提供命名服务，更新和发现 Broker 服务。
- Broker：消息中转角色，负责存储消息，转发消息。分为 Master Broker 和 Slave Broker，一个 Master Broker 可以对应多个 Slave Broker，但是一个 Slave Broker 只能对应一个 Master Broker。Broker 启动后需要完成一次将自己注册至 Name Server 的操作；随后每隔 30s 定期向 Name Server 上报 Topic 路由信息。
- 生产者：与 Name Server 集群中的其中一个节点（随机）建立长链接（Keep-alive），定期从 Name Server 读取 Topic 路由信息，并向提供 Topic 服务的 Master Broker 建立长链接，且定时向 Master Broker 发送心跳。
- 消费者：与 Name Server 集群中的其中一个节点（随机）建立长连接，定期从 Name Server 拉取 Topic 路由信息，并向提供 Topic 服务的 Master Broker、Slave Broker 建立长连接，且定时向 Master Broker、Slave Broker 发送心跳。Consumer 既可以从 Master Broker 订阅消息，也可以从 Slave Broker 订阅消息，订阅规则由 Broker 配置决定。

# 1 事务消息

- 使用消息队列来避免分布式事务
- 如果仔细观察生活的话，生活的很多场景已经给了我们提示。比如在北京很有名的姚记炒肝点了炒肝并付了钱后，他们并不会直接把你点的炒肝给你，而是给你一张小票，然后让你拿着小票到出货区排队去取。为什么他们要将付钱和取货两个动作分开呢？原因很多，其中一个很重要的原因是为了使他们接待能力增强（并发量更高）。还是回到我们的问题，只要这张小票在，你最终是能拿到炒肝的。
- 同理转账服务也是如此，当支付宝账户扣除1万后，我们只要生成一个凭证（消息）即可，这个凭证（消息）上写着“让余额宝账户增加1万”，只要这个凭证（消息）能可靠保存，我们最终是可以拿着这个凭证（消息）让余额宝账户增加1万的，即我们能依靠这个凭证（消息）完成最终一致性。

# 1 事务消息

- 业务与消息解耦方式
- 1）支付宝在扣款事务提交之前，向实时消息服务请求发送消息，实时消息服务只记录消息数据，而不真正发送，只有消息发送成功后才会提交事务；
- 2）当支付宝扣款事务被提交成功后，向实时消息服务确认发送。只有在得到确认发送指令后，实时消息服务才真正发送该消息；
- 3）当支付宝扣款事务提交失败回滚后，向实时消息服务取消发送。在得到取消发送指令后，该消息将不会被发送；
- 4）对于那些未确认的消息或者取消的消息，需要有一个消息状态确认系统定时去支付宝系统查询这个消息的状态并进行更新。
  - 假设在第2步支付宝扣款事务被成功提交后，系统挂了，此时消息状态并未被更新为“确认发送”，从而导致消息不能被发送。
- 优点：消息数据独立存储，降低业务系统与消息系统间的耦合；
- 缺点：一次消息发送需要两次请求；业务处理服务需要实现消息状态回查接口。

# 1 事务消息

- 消息重复投递的问题
- 比如余额宝处理完消息msg后，发送了处理成功的消息给支付宝，正常情况下支付宝应该要删除消息msg，但如果支付宝这时候悲剧的挂了，重启后一看消息msg还在，就会继续发送消息msg。解决方法很简单，在余额宝这边增加消息应用状态表（message_apply），通俗来说就是个账本，用于记录消息的消费情况，每次来一个消息，在真正执行之前，先去消息应用状态表中查询一遍，如果找到说明是重复消息，丢弃即可，如果没找到才执行，同时插入到消息应用状态表（同一事务）。