计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(9):185−189 [doi: 10.15888/j.cnki.csa.007063]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

# 采用消息队列实现数据一致性方法①

张　杰[1], 满曙光[2], 刘　凯[1], 周立军[1]

[1](海军航空大学 航空基础学院, 烟台 264001)
[2](烟台市公安局, 烟台 264001)
通讯作者: 张　杰, E-mail: 18106389886@163.com

**摘　要**: 针对构建分布式微服务中数据一致性问题, 本文总结了在处理分布式计算中数据一致性问题遵循原则, 分析实现微服务的幂等性设计重要性, 提出了一种采用事务型消息队列解决分布式微服务典型应用场景中数据一致性问题的方法, 并给出 RocketMQ 实现方式及原理, 实验表明事务型消息可以较好解决分布式数据一致性问题, 最后分析了上述方法的优缺点, 本文提出的在分布式微服务构建中数据一致性处理方法应用中具有一定的借鉴作用.

**关键词**: 微服务; 数据一致性; 幂等性; 事务型消息队列

引用格式: 张杰,满曙光,刘凯,周立军.采用消息队列实现数据一致性方法.计算机系统应用,2019,28(9):185–189. http://www.c-s-a.org.cn/1003-3254/7063.html

## Data Consistency Method Based on Message Queuing

ZHANG Jie[1], MAN Shu-Guang[2], LIU Kai[1], ZHOU Li-Jun[1]

[1](School of Aeronautical Basis, Naval Aeronautical University, Yantai 264001, China)
[2](Yantai Municipal Public Security Bureau, Yantai 264001, China)

**Abstract**: For the problem of data consistency in the construction of distributed micro-services, this paper summarizes the principles of data consistency in the process of distributed computing, analyzes the importance of idempotent design for realizing micro-services, and proposes a method of using transactional message queue to solve the problem of data consistency in typical application scenarios of distributed micro-services, and gives RocketMQ implementation method and principle. Experiments show that transactional messages can better solve the problem of distributed data consistency. Finally, the advantages and disadvantages of the above methods are analyzed. The data consistency processing method proposed in this study can be used for reference in the construction of distributed micro-services.

**Key words**: micro-service; data consistency; idempotent property; transactional message queue

随着企业级应用的业务复杂度和规模的不断扩大, 传统的单体应用系统在维护、部署、扩展以及稳定性、并发性等方面, 普遍存在难以逾越瓶颈, 这也导致各种相对独立的传统软件系统在集成时面临的困难, 如系统堆砌、问题定位难、扩展性差、可靠性不高、维护成本高等[1], 为适应移动互联网高速发展以及在项目开发敏捷、精益、持续交付等应用需求背景, 传统的单体应用系统面临功能重复开发、功能监控与评估(性能) 难以进行, 由于软件构件复用效率低, 当面对业务需求变更时, 使得应用系统臃肿、维护困难, 频繁部署, 甚至给软件测试带来更多不确定性, 导致系统无法持续工作[2], 此外高并发性是单体应用难以逾越的鸿沟, 为解决上述问题, 使用微服务 (micro-service) 实现组件化成为系统设计的新选择, 并得到飞速发展和应

① 基金项目: 国家自然科学基金 (61790550, 61790554)
Foundation item: National Natural Science Foundation of China (61790550, 61790554)
收稿时间: 2019-03-04; 修改时间: 2019-03-29; 采用时间: 2019-04-01; csa 在线出版时间: 2019-09-05

用, 微服务架构通过将系统按服务组件化分解, 服务之间通过 Http 等通信协议进行协作, 并且各个服务都可单独开发、部署, 最终通过服务之间组合与调用对外完成系统功能[3, 4].

微服务在解决上述问题同时, 也引入了诸多不确定因素, 如执行一项完整的业务, 需要调用多个微服务协同工作, 当依赖微服务调用出现故障 (操作失败), 已经完成的微服务如何处理[5,6], 此时主要涉及微服务的可用性与数据一致性等问题, 针对上述问题, 本文首先阐述了单体系统中事务与分布式系统事务的基本原理, 分析了微服务在数据一致性问题上遵循的原则, 提出了一种使用事务型消息队列实现微服务数据最终一致性方法, 通过典型应用场景分析, 给出使用 RocketMQ 消息队列实现了分布式数据一致性方法, 通过实验表明事务型消息在解决上述问题时具有易于实现、可靠性高、并发处理能力强等特点, 最后总结 RocketMQ 事务型消息队列实现难点及不足.

## 1 数据一致性基本原则

传统的单体应用系统中, 通常使用一个关系型数据库, 通过关系型数据库事务保证数据的一致性, 这种事务有四个基本要素 (ACID)[7]: 原子性、一致性、隔离性、持久性. 为了应对高并发的挑战, 应用系统需要多个数据库来支持, 可通过分布式事务来保证数据一致性, 根据 CAP 理论: 分布式系统不可能同时满足一致性、可用性和分区容错性这三个要求, 最多只能同时满足两个[8]. 鉴于网络硬件出现闪断、延迟丢包等问题不可避免, 分区容忍性必须需要实现, 同时可用性体现了分布式应用系统持续提供服务的能力, 若满足一致性则需付出在满足一致性之前阻塞其他并发访问的代价[9], 事实上可用性与分区容忍性优先级要高于数据一致性, 所以只能在数据一致性上做出取舍, 分布式数据一致性级别又可分为:

强一致性: 类似于单体事务数据一致性, 但实现起来往往对系统的并发性能影响大.

弱一致性: 约束了数据更新成功后, 不承诺立即可以读到写入的数据, 也不久承诺多久之后数据能够达到一致, 但会尽可能地保证到某个时间级别 (比如秒级), 数据能够达到一致状态.

最终一致性: 作为弱一致性的一个特例, 系统会保证在一定时间内, 能够达到数据一致的状态.

在微服务架构中, 数据访问与分布式架构相比更加复杂, 通常情况下, 数据都是每个微服务私有, 只能通过 API 的方式访问数据. 这种方式可以实现微服务间的松耦合, 使彼此独立的微服务更容易的进行扩展. 随之带来问题是: 数据不一致性既不能依靠底层数据库事务实现, 也无法通过统一的事务协调器来完成数据一致性, 传统的本地事务或分布式事务不适合微服务架构.

微服务架构作为分布式架构的一种, 数据一致性通常采用 BASE 理论, BASE 理论是对 CAP 理论的延伸, 核心思想是即使无法做到强一致性[10], 但应用可以采用适合的方式达到最终一致性 (Eventual Consitency), BASE 模型完全不同 ACID 模型, 该模型牺牲高一致性, 获得可用性和可靠性[11].

在微服务实现数据一致性时, 首先应保证调用微服务具有幂等性, 幂等性是指一个操作 (特定服务一次调用) 至多只会被处理一次, 后续调用都将返回第一次调用时的处理结果[12].

## 2 事务型消息一致性处理方法

### 2.1 一致性应用场景分析

在分布式架构中, 以学员选课应用场景为例, 基本业务逻辑如下:

(1) 选课服务 S1 负责学员执行选课操作, 完成选课信息保存;

(2) 统计服务 S2 负责统计选课信息, 执行汇总计算操作;

(3) 通知服务 S3 负责通知任课老师选课信息.

选课业务中体现的分布式数据一致性要求主要体现在:

(1) 选课服务 S1 完成选课操作成功, 统计服务 S2 成功接收到学员选课信息, 并进行汇总操作;

(2) 选课服务 S1 完成选课操作成功, 通知服务 S3 成功接收到学员选课信息;

在图 1 展示的业务中, 首先执行本地数据库事务方法, 其次发布消息, 当消息发布失败会导致消息发布者本地事务回滚, 现实中数据库事务回滚的成本相对于消息发布失败高很多, 这明显是不符合预期[13].

为了解决这个问题, 可以采用消息队列作为中间件 (如图 2), 消息队列普遍用于各微服务之间异步通讯[14], 为实现以上数据一致性要求, 采用事务型消息队列实

现微服务 S1、S2 和 S3 之间的提供异步通信服务, 首先将选课服务 S1 选课操作分解为 3 个步骤完成, 且封装在一个本地事务中:

(1) 将选课信息发布事务型消息到消息队列;

(2) 执行保存选课信息操作;

(3) 根据执行步骤 (2) 执行结果, 决定是否将消息投递给服务 S2 和 S3.

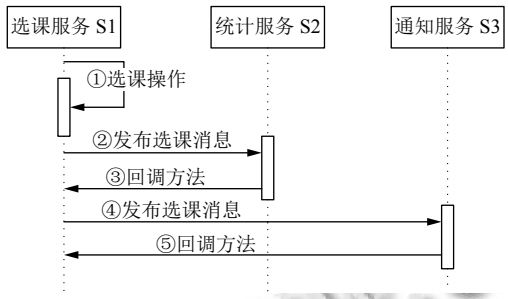


图 1 选课业务

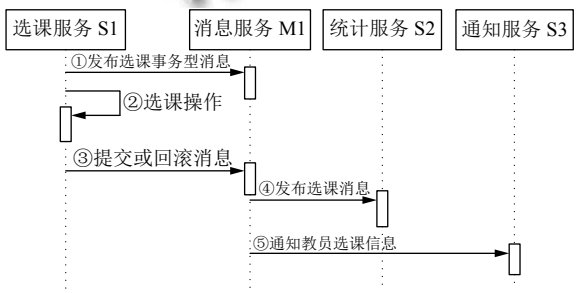


图 2 改造后选课业务

消息队列提供一种特殊类型的消息: 事务型消息, 这类消息的特点是: 消息队列收到消息后不会立刻投递消息到消息订阅者 (服务 S2 和 S3), 而是根据消息发布者应用的数据库事务状态决定消息是否投递. 如果选课服务 S1 数据库事务提交, 则消息投递到订阅者 (服务 S2 和 S3); 反之不投递.

### 2.2 RocketMQ 事务型消息队列

RocketMQ 是一个具有低延时、高并发、高可用、高可靠等特点的分布式消息中间件, 可作为各个微服务、平台、应用之间的通用服务, 还可完成异步解耦功能, 即挡住前端 (消息发送方) 的数据洪峰, 保证后端服务的稳定性[15], 而对于事务消息, 主要是通过消息的异步处理, 可以保证本地事务和消息发送同时成功执行或失败, 从而保证数据的最终一致性, RocketMQ 事务型消息队列主要流程如图 3 所示.

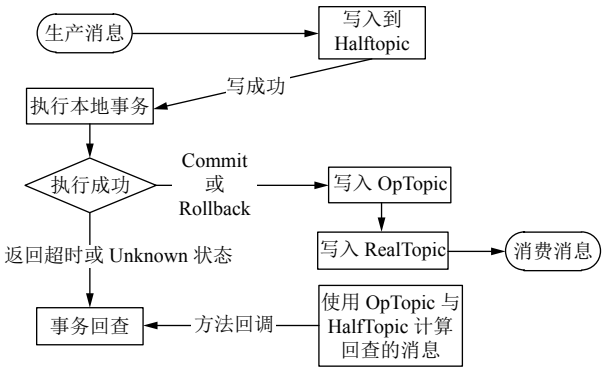


图 3 RocketMQ 事务型消息队列处理流程

(1) 生产者 (选课服务 S1) 同步发送 prepare 事务消息到 broker;

(2) broker 接收到消息后, 将该消息进行转换并写入 Half Topic, 写入成功后会给生产者返回成功状态;

(3) 生产者 (选课服务 S1) 获取到该消息的事务 Id, 进行本地事务处理;

(4) 本地事务执行成功提交 Commit, 若失败则提交 Rollback, 提交超时或 Unknow 状态则会触发 broker 的事务回查;

(5) 若提交 Commit 或 Rollback 状态, 则 Broker 将消息写入到 OpTopic, 该 Topic 的作用主要记录已经 Commit 或 Rollback 的 prepare 消息, Broker 利用 Half Topic 和 OpTopic 计算出需回查的事务消息. 如果是 Commit 消息, broker 还会将消息从 Half 取出来存储到 Topic 里, 从而消费者可正常进行消费, 如果是 Rollback 则不进行其他操作;

(6) 如果本地事务执行超时或返回 Unknow 状态, 则 broker 会进行事务回查. 若生产者执行本地事务超过 6 s 则进行第一次事务回查, 总共回查 15 次, 后续回查间隔时间是 60 s, broker 在每次回查时会将消息在 Half Topic 中再写 1 次.

(7) 执行事务回查时, 生产者可获取事务 Id, 检查该事务在本地执行情况, 返回状态同第 1 次执行本地事务一样.

RocketMQ 消息队列事务型消息一次成功投递需经 3 个 Topic, Half Topic 用于记录所有的 prepare 消息, Op Half Topic 记录已经提交了状态的 prepare 消息, Real Topic 事务消息真正 Topic, 在 Commit 后会将消息写入该 Topic, 进行消息的投递.

从上述流程可以看到事务消息保证了生产者发送

消息成功与本地执行事务的成功的一致性, 消费者在消费事务消息时, broker 处理事务消息的消费与普通消息是一样的, 若消费不成功, 则 broker 会重复投递该消息[16].

## 3 实现及应用分析

建立 RocketMQ 消息服务, 选课业务 (S1) 作为消息生产者, 消息服务 (M1) 为选课服务和通知服务、统计服务提供中间件, 通知服务 (S2) 和和通知服务 (S3) 作为消息消费者角色存在.

### 3.1 事务型消息生产者

选课业务作为消息生产者, 主要完成本地选课业务保存、选课消息产生、提供事务回查方法. 选课业务回查方法可以根据由 RocketMQ 回传的 key 去数据库查询, 判断这条数据到底是成功还是失败. 关键代码如下:

```
TransactionMQProducer p = new
TransactionMQProducer("CSP");
p.setNamesrvAddr("127.0.0.1:9876");
p.setTransactionCheckListener(new
TransactionCheckListener() {
public LocalTransactionState
checkLocalTransactionState(MessageExt
msg) {
return
getStateUtil(messageExt.getKeys());}});
ExecuterImpl tce =new
ExecuterImpl();
Message msg = new Message("CSTopic",
"CS","key",
course.toString().getBytes());
SendResult result =
p.sendMessageInTransaction(msg, tce,
"tq"); }
```

### 3.2 事务型消息消费

以通知服务 (S3) 为例说明, 设计事务型消息消费者关键代码:

```
public ConsumerTransaction() {
DefaultMQPushConsumer c = new
DefaultMQPushConsumer("CourseConsumer
");
c.setNamesrvAddr("127.0.0.1:9876");
c.subscribe("CourseSelect", "*");
c.registerMessageListener(new
Listener());
class Listener implements
MessageListenerConcurrently{
public ConsumeConcurrentlyStatus
consumeMessage(List<MessageExt> list,
ConsumeConcurrentlyContext ctx) {
for(MessageExt msg : list) {
String msgBody = new
String(msg.getBody(), "utf-8");
Opt(course.getJson(msgBody)); }
return
ConsumeConcurrentlyStatus.CONSU
ME_SUCCESS; }
}
```

### 3.3 幂等性设计

本文中选课业务作为微服务架构设计, 如果不支持幂等操作, 那将会出现相同的选课信息多次推送给后续的服务, 为避免上述情况出现, 可将通知服务和统计服务设计为幂等操作, 幂等的接口实际上就是可以重复调用, 每次接口调用的结果都是一样的.

幂等设计具体实现方法: 将选课的课程编号与学号作为组合主键, 建立一张去重表, 并且把上述主键标识作为唯一索引, 实现时, 把选课信息写入去去重表, 放在一个事务中, 如果重复创建, 数据库会抛出唯一约束异常, 操作就会回滚.

### 3.4 应用分析

在实现分布式数据一致性时, 尤其是微服务之间数据一致性, 为了提高并发性和可用性, 更多选择采用数据最终一致性方法, 在上述分析选课业务中, 选课微服务 S1 成功完成选课操作, 并不会直接在本地事务中完成对统计服务 S2 和通知服务 S3 的调用, 而是采用异步的方式通过消息队列完成, 达到最终数据的一致性, 即 S1 完成操作, S2 和 S3 收到消息.

使用事务型消息队列解决数据一致性问题时, 关键点在于: 当消息队列收不到事务型消息的 “提交 or 回滚” 消息时, 如何确保数据一致性. 在分布式网络架构中, 不可避免会出现网络闪短、消息队列服务短时间内不可用等情况, 会导致消息队列中消息长期处于

Half Topic 状态, 这也是消息队列提供“事务型消息” 特性必须解决的问题, 如果消息队列没有收到 “提交 or 回滚”消息, 则无法决定是否投递消息到消息订阅者, 此时消息队列会主动询问消息生产者 (选课服务 S1) 询问该消息的最终状态 (Commit 或是 Rollback), 该过程称为事务型消息状态回查, 具体设计方案如图 4.

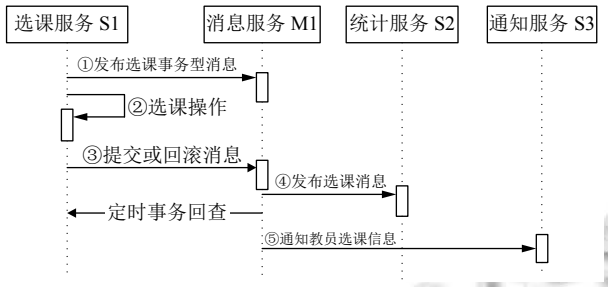


图 4 消息队列事务回查

消息队列事务型消息基于“二阶段”消息实现, 通过事务型消息状态回查方法, 即使当消息队列没有收到 “提交 or 回滚”消息, 也能保证事务型消息是否投递与消息发布者本地事务状态保持一致; 在使用消息队列解决数据一致性问题时, 还需要解决消息重复投递的问题, 通用方法是在消费消息的微服务 (S2 和 S3) 实现幂等性, 相同的选课消息至多只会被处理一次, 后续的调用都将返回第一次调用时的处理结果.

## 4 结束语

本文总结了在处理分布式计算 (微服务) 数据一致性问题遵循的原则, 分析实现微服务的幂等性设计的重要性, 提出了一种采用事务型消息队列解决分布式微服务典型应用场景中数据一致性问题的方法, 并给出 RocketMQ 消息队列工作模式, 分析了事务型消息队列实现数据一致性的原理与实现方式; 使用事务型消息队列在处理分布式微服务数据一致性, 事务消息消费端的消费方式和普通消息相同, RocketMQ 能保证消息能被消费端收到 (消息重试等机制), 采用事务型消息方法还需保证消息能够最终消费成功这个关键步骤, 同时微服务在消费消息时, 首先要保证后续业务 (S2 和 S3) 具有幂等性, 如果 consumer 消费失败时, RocketMQ 需要人工介入处理, 尽管这种情况出现概率极低.

## 参考文献


1 洪华军, 吴建波, 冷文浩. 一种基于微服务架构的业务系统设计与实现. 计算机与数字工程, 2018, 46(1): 149–154. [doi: 10.3969/j.issn.1672-9722.2018.01.032]

2 张晶, 黄小锋, 李春阳. 微服务框架的设计与实现. 计算机系统应用, 2017, 26(6): 259–262. [doi: 10.15888/j.cnki.csa.005796]

3 王方旭. 基于 Spring Cloud 和 Docker 的微服务架构设计. 中国信息化, 2018, (3): 53–55. [doi: 10.3969/j.issn.1672-5158.2018.03.024]

4 赵子晨, 朱志祥, 蒋来好. 构建基于 Dubbo 框架的 Spring Boot 微服务. 计算机与数字工程, 2018, 46(12): 2539–2543, 2551. [doi: 10.3969/j.issn.1672-9722.2018.12.030]

5 李磊, 李娟. Dubbo 服务框架技术在学习系统开发中的应用与实践. 计算机系统应用, 2017, 26(6): 244–248. [doi: 10.15888/j.cnki.csa.005810]

6 万年红. 面向服务的自适应云资源信息集成软件架构. 计算机应用, 2012, 32(1): 170–174.

7 李春阳, 刘迪, 崔蔚, 等. 基于微服务架构的统一应用开发平台. 计算机系统应用, 2017, 26(4): 43–48. [doi: 10.15888/j.cnki.csa.005757]

8 龙新征, 彭一明, 李若淼. 基于微服务框架的信息服务平台. 东南大学学报 (自然学报版), 2017, 47(S1): 48–52.

9 欧阳荣彬, 王倩宜, 龙新征. 基于微服务的数据服务框架设计. 华中科技大学学报 (自然科学版), 2016, 44(S1): 126–130.

10 周京晖. 集成消息服务和定时通知的分布式内存数据库. 软件, 2013, 34(1): 89–92. [doi: 10.3969/j.issn.1003-6970.2013.01.029]

11 李文逍, 杨小虎. 基于分布式缓存的消息中间件存储模型. 计算机工程, 2010, 36(13): 93–95. [doi: 10.3969/j.issn.1000-3428.2010.13.033]

12 徐进, 黄勃, 冯炯. 基于消息通信的分布式系统最终一致性平台. 计算机应用, 2017, 37(4): 1157–1163. [doi: 10.11772/j.issn.1001-9081.2017.04.1157]

13 朱涛, 郭进伟, 周欢, 等. 分布式数据库中一致性与可用性的关系. 软件学报, 2018, 29(1): 131–149. [doi: 10.13328/j.cnki.jos.005433]

14 李政, 武彤. 基于分布式消息队列的企业级全文检索模型研究. 计算机应用与软件, 2017, 34(6): 292–295. [doi: 10.3969/j.issn.1000-386x.2017.06.052]

15 欧志芳. 基于 RocketMQ 实现异构数据库同步. 网络安全技术与应用, 2016, (12): 99–100. [doi: 10.3969/j.issn.1009-6833.2016.12.066]

16 马跃, 颜睿陽, 孙建伟. 基于 RocketMQ 的 MQTT 消息推送服务器分布式部署方案. 计算机系统应用, 2018, 27(6): 83–86. [doi: 10.15888/j.cnki.csa.006381]