javaweb 面试总结（四、分布式事务、CAP 原理和 BASE 思想、JDBC 事务和 JTA 事务的区别、2PC 与 TCC 区别）

2018 年 04 月 09 日 21:34:43 寒山空明月 阅读数：2293

CAP 原理和 BASE 思想： http://www.jdon.com/37625

分布式事务如何处理**?**

> 解决方案有很多种!
>
> 比如**事务补偿**机制：即在事务链中的任何一个正向事务操作，都必须存在一个完全符合回滚规则的可逆事务。
>
> 或者**两阶段提交、三阶段提交**：分布式事务服务(DTS) 支付宝的 DTS 实现！最近也看见一个 tcc 方案 GitHub - changmingxie/tcc-transaction: tcc-transaction 是 TCC 型事务 java 实现 可以简单看一下;
>
> 或者利用**消息系统实现最终一致性;**

概念澄清

- **事务补偿机制**: 在事务链中的任何一个正向事务操作, 都必须存在一个完全符合回滚规则的可逆事务.
- **CAP 理论**: CAP(Consistency, Availability, Partition Tolerance), 阐述了一个分布式系统的三个主要方面, 只能同时择其二进行实现. 常见的有 CP 系统, AP 系统.
- **幂等性**: 简单的说, **业务操作支持重试,** 不会产生不利影响. 常见的实现方式: 为消息额外增加唯一 ID.
- **BASE**(Basically avaliable, soft state, eventually consistent): 是分布式事务实现的一种理论标准.
- **分布式缓存** ： redis 集群就算一种实现

> **CAP 理论**
>
> Consistency(一致性), 数据一致更新，所有数据变动都是同步的
>
> Availability(可用性), 好的响应性能

Partition tolerance(分区容忍性) 可靠性

定理：任何分布式系统只可同时满足二点，没法三者兼顾。

忠告：架构师不要将精力浪费在如何设计能满足三者的完美分布式系统，而是应该进行取舍。

ACID 模型

Atomicity 原子性：一个事务中所有操作都必须全部完成，要么全部不完成。

Consistency 一致性. 在事务开始或结束时，数据库应该在一致状态。

Isolation 隔离层. 事务将假定只有它自己在操作数据库，彼此不知晓。

Durability 持久性. 一旦事务完成，就不能返回。

关系数据库的 ACID 模型拥有 高一致性 + 可用性；

跨数据库事务：

2PC (two-phase commit)，

2PC is the anti-scalability pattern (Pat Helland) 是反可伸缩模式的，JavaEE 中的JTA 事务可以支持 2PC。因为 2PC 是反模式，尽量不要使用 2PC，使用 BASE 来回避。

TCC (Try-Confirm-Cancle)

是基于补偿型事务的 AP 系统的一种实现, 具有最终一致性：

BASE 思想

Base：一种 Acid 的替代方案

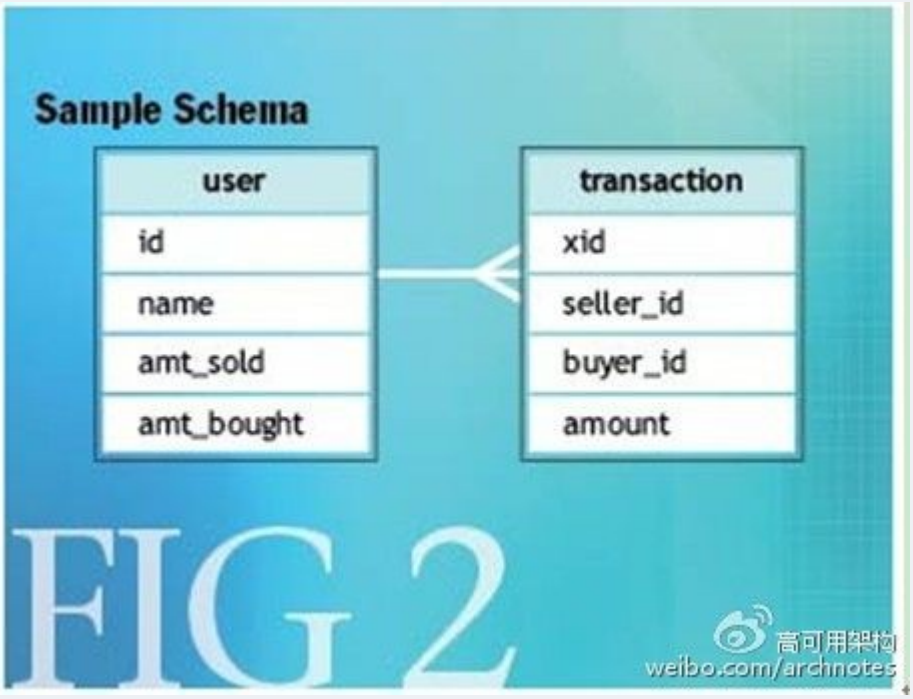
此方案是 eBay 的架构师 Dan Pritchett 在 2008 年发表给 ACM 的文章，是一篇解释 BASE 原则，或者说最终一致性的经典文章。文中讨论了 BASE 与 ACID 原则在保证数据一致性的基本差异。

如果 ACID 为分区的数据库提供一致性的选择，那么如何实现可用性呢？答案是

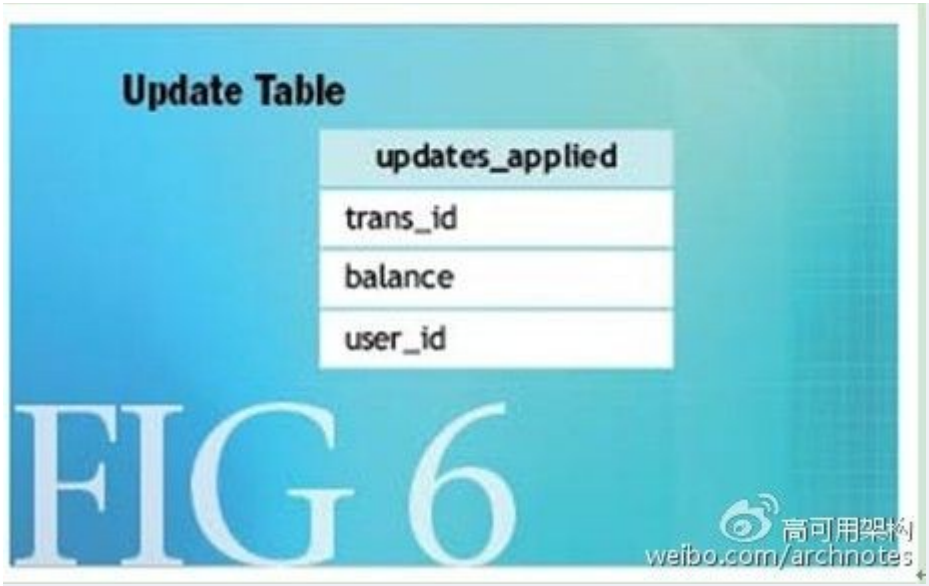
BASE (basically available, soft state, eventually consistent)

BASE 的可用性是通过**支持局部故障**而不是系统全局故障来实现的。下面是一个简单的例子：如果将用户分区在 5 个数据库服务器上，BASE 设计鼓励类似的处理方式，一个用户数据库的故障只影响这台特定主机那 20% 的用户。这里不涉及任何魔法，不过它确实可以带来更高的可感知的系统可用性。

文章中描述了一个最常见的场景，如果产生了一笔交易，需要在交易表增加记录，同时还要修改用户表的金额。这两个表属于不同的远程服务，所以就涉及到分布式事务一致性的问题。



文中提出了一个经典的解决方法，将主要修改操作以及更新用户表的消息**放在一个本地事务**来完成。同时为了避免重复消费用户表消息带来的问题，达到多次重试的幂等性，**增加一个更新记录表 updates_applied** 来记录已经处理过的消息。


Update Table
updates_applied
trans_id
balance
user_id
FIG 6
高可用架构
weibo.com/archnotes

系统的执行伪代码如下

```
Begin transaction
  Insert into transaction(id, seller_id, buyer_id, amount);
  Queue message "update user("seller", seller_id, amount)";
  Queue message "update user("buyer", buyer_id, amount)";
End transaction
For each message in queue
  Peek message
  Begin transaction
    Select count(*) as processed where trans_id=message.trans_id
      and balance=message.balance and user_id=message.user_id
    If processed == 0
      If message.balance == "seller"
        Update user set amt_sold=amt_sold + message.amount
            where id=message.id;
      Else
        Update user set amt_bought=amt_bought + message.amount
            where id=message.id;
      End if
      Insert into updates_applied
        (message.trans_id, message.balance, message.user_id);
    End if
  End transaction
  If transaction successful
    Remove message from queue
  End if
End for
```


基于以上方法，在第一阶段，通过本地的数据库的事务保障，增加了 transaction 表及消息队列 。

在第二阶段，分别读出消息队列（但不删除），通过判断更新记录表 updates_applied 来检测相关记录是否被执行，未被执行的记录会修改 user 表，然后增加一条操作记录到 updates_applied，事务执行成功之后再删除队列。

通过以上方法，达到了分布式系统的最终一致性。

参考：https://blog.csdn.net/liuquanyi/article/details/2065475

X/Open 组织 与 X/Open DTP 模型 与 XA 关系

> X/Open **组织**（即现在的 Open Group）定义了分布式事务处理模型。
>
> X/Open DTP 模型（1994）包括应用程序（AP）、事务管理器（TM）、资源管理器（RM）、通信资源管理器（CRM）四部分。
>
> XA 就是 X/Open DTP 定义的交易中间件与数据库之间的接口规范（即接口函数），交易中间件用它来通知数据库事务的开始、结束以及提交、回滚等。XA 接口函数由数据库厂商提供。
>
> 通常情况下，交易中间件与数据库通过 XA 接口规范，使用两阶段提交来完成一个全局事务，XA 规范的基础是两阶段提交协议。
>
> JTA(Java Transaction API)是符合 X/Open DTP 模型的，事务管理器和资源管理器之间也使用了 XA 协议。 本质上也是借助两阶段提交协议来实现分布式事务的

ACID、BASE 和 CAP 原理

参考地址：https://blog.csdn.net/sinat_27186785/article/details/52032510

分布式事务框架--代码：https://github.com/QNJR-GROUP/EasyTransaction

分布式事务场景：

- 无需分布式事务
  - 最常用
  - 最优先使用
- 使用消息队列完成的最终一致性事务
  - 适用于业务主逻辑无需外部数据变更协助来完成的最终一致性事务
  - 常见
  - 若一定要与其他服务写接口发生交互，则优先使用

    - 依据是否保证投递到订阅者，分为可靠消息及最大努力交付消息
    - 有时业务要求一些本质是异步的操作同步返回结果，若同步返回失败则后台异步补单。这种业务本质也归属于无需外部数据变更以协助完成的最终一致性，但介于其同步时要返回结果，其有区别于可靠消息。
- 使用传统补偿完成的最终一致性事务
    - 适用于需要获取远程执行结果来决定逻辑事务走向 且 可以进行补偿的业务
    - 次常见
    - 若使用消息队列不能解决的事务问题优先考虑使用基于补偿的最终一致性事务
- 使用 TCC 完成最终一致性事务
    - 适用于需要获取远程执行结果来决定逻辑事务走向 且 不可以进行补偿的业务
    - 最不常见
    - 最终解决办法，囊括所有必须使用 2PC 实现的场景。编码量最大，性能消耗最大，应尽量避免使用本类型的事务

参考： https://www.cnblogs.com/luoyunfei99/articles/6803682.html

分布式事务原理：分段式提交 -- 2PC (two-phase commit)

分布式事务通常采用 2PC 协议，全称 Two Phase Commitment Protocol。该协议主要为了解决在分布式数据库场景下，所有节点间数据一致性的问题。分布式事务通过 2PC 协议将提交分成两个阶段：

- prepare；
- commit/rollback

阶段一为准备（prepare）阶段。即所有的参与者准备执行事务并锁住需要的资源。参与者 ready 时，向 transaction manager 报告已准备就绪。
阶段二为提交阶段（commit）。当 transaction manager 确认所有参与者都 ready 后，向所有参与者发送 commit 命令。

缺点

- 两阶段提交中的第二阶段, 协调者需要等待所有参与者发出 yes 请求, 或者一个参与者发出 no 请求后, 才能执行提交或者中断操作. 这会造成**长时间同时锁住多个资源, 造成性能瓶颈**, 如果参与者有一个耗时长的操作, 性能损耗会更明显.

- 实现复杂, 不利于系统的扩展, 不推荐.

参考： https://blog.csdn.net/congyihao/article/details/70195154

TCC (Try-Confirm-Cancle)

是基于补偿型事务的 AP 系统的一种实现, 具有最终一致性:

优点

- TCC 能够对分布式事务中的各个资源进行分别锁定, 分别提交与释放, 例如, 假设有 AB 两个操作, 假设 A 操作耗时短, 那么 A 就能较快的完成自身的 try-confirm-cancel 流程, 释放资源. 无需等待 B 操作. 如果事后出现问题, 追加执行补偿性事务即可.
- TCC 是绑定在各个子业务上的(除了 cancle 中的全局回滚操作), 也就是各服务之间可以在一定程度上"异步并行"执行.

注意事项

- 事务管理器(协调器)这个节点必须以带同步复制语义的高可用集群(HAC)方式部署.
- 事务管理器(协调器)还需要使用多数派算法来避免集群发生脑裂问题.

适用场景

- 严格一致性
- 执行时间短
- 实时性要求高

举例: 红包, 收付款业务.

参考： https://blog.csdn.net/congyihao/article/details/70195154

异步确保型

通过将一系列同步的事务操作变为基于消息执行的异步操作, 避免了分布式事务中的同步阻塞操作的影响.

这个方案真正实现了两个服务的解耦, 解耦的关键就是**异步消息**和补偿性事务.

这里以一个例子作为讲解:

需要额外说明的一点, 就是事务消息投递到 MQ 订阅方后, 并不一定能够成功执行. 需要 MQ 订阅方主动给予消费反馈(ack)

- 如果 MQ 订阅方执行远程事务成功, 则给予消费成功的 ack, 那么 MQ Server 可以安全将事务消息移除;
- 如果执行失败, MQ Server 需要对消息重新投递, 直至消费成功.

注意事项

- 消息中间件在系统中扮演一个重要的角色, 所有的事务消息都需要通过它来传达, 所以消息中间件也需要支持 HAC 来确保事务消息不丢失.
- 根据业务逻辑的具体实现不同，还可能需要对消息中间件增加消息不重复, 不乱序等其它要求.

适用场景

- 执行周期较长
- 实时性要求不高

例如:

- 跨行转账/汇款业务(两个服务分别在不同的银行中)
- 退货/退款业务
- 财务, 账单统计业务(先发送到消息中间件, 然后进行批量记账)

JDBC 事务和 JTA 事务的区别：
https://www.cnblogs.com/drizzlewithwind/p/5711653.html
Java 中的事务——JDBC 事务和 JTA 事务:
https://www.cnblogs.com/chengpeng15/p/5802930.html

一、事务概述

事务表示一个由一系列的数据库操作组成的不可分割的逻辑单位，其中的操作要么全做要么全都不做。
与事务相关的操作主要有:

BEGIN TRANSACTION;　开始一个事务，方法是：begin()
COMMIT;　提交一个事务，方法是：commit()
ROLLBACK;　回滚一个事务，方法是：rollback()
PREPARE;　准备提交一个事务，方法是：prepare()

二、事务的特性（ACID）

1、原子性：同一个事务的操作要么全部成功执行，要么全部撤消
2、隔离性：事务的所有操作不会被其它事务干扰
3、一致性：在操作过程中不会破坏数据的完整性
4、时效性 ：事务的结果必须持久保存于介质上

三、JDBC 和 JTA 事务区别

简单的说 jta 是多库的事务 jdbc 是单库的事务;
JDBC 事务由 Connnection 对象控制管理，也就是说，事务管理实际上是在 JDBC Connection 中实现。事务周期限于 Connection 的生命周期。
JTA(Java Transaction API)提供了跨数据库连接（或其他 JTA 资源）的事务管理能力。JTA 事务管理则由 JTA 容器实现，J2ee 框架中事务管理器与应用程序，资源管理器，以及应用服务器之间的事务通讯。
Java 事务 API（Java Transaction API，简称 JTA ） 是一个 Java 企业版 的应用程序接口，在 Java 环境中，允许完成跨越多个 XA 资源的分布式事务。

四、JTA 的优缺点

介绍:

JTA 的优点很明显，就是提供了分布式事务的解决方案，严格的 ACID。但是，标准的 JTA 方式的事务管理在日常开发中并不常用，因为他有很多缺点:

缺点:

实现复杂

通常情况下，JTA UserTransaction 需要从 JNDI 获取。这意味着，如果我们使用 JTA，就需要同时使用 JTA 和 JNDI。

JTA 本身就是个笨重的 API

通常 JTA 只能在应用服务器环境下使用，因此使用 JTA 会限制代码的复用性。

五、事物处理方式总结

Java 事务的类型有三种：JDBC 事务、JTA(Java Transaction API)事务、容器事务，其中JDBC 的事务操作用法比较简单，适合于处理同一个数据源的操作。JTA 事务相对复杂，可以用于处理跨多个数据库的事务，是分布式事务的一种解决方案。

这里还要简单说一下，虽然JTA 事务是Java 提供的可用于分布式事务的一套API，但是不同的J2EE 平台的实现都不一样，并且都不是很方便使用，所以，一般在项目中不太使用这种较为负责的API。现在业内比较常用的分布式事务解决方案主要有异步消息确保型、TCC、最大努力通知等。关于这几种分布式事务解决方案，我会在后面的文章中介绍。欢迎关注与交流。