对分布式事务及两阶段提交、三阶段提交的理解

一、分布式数据一致性

在分布式系统中，为了保证数据的高可用，通常会将数据保留多个副本(replica)，这些副本会放置在不同的物理的机器上。

1.什么是数据一致性

在数据有多份副本的情况下，如果网络、服务器或者软件出现故障，会导致部分副本写入成功，部分副本写入失败。这就造成各个副本之间的数据不一致，数据内容冲突，造成事实上的数据不一致。

2.CAP定理

CAP理论认为在分布式的环境下设计和部署系统时，有3个核心的需求：

> **Consistency，Availability和Partition Tolerance，即CAP**

Consistency：一致性，这个和数据库ACID的一致性类似，但这里关注的所有数据节点上的数据一致性和正确性，而数据库的ACID关注的是在在一个事务内，对数据的一些约束。系统在执行过某项操作后仍然处于一致的状态。在分布式系统中，更新操作执行成功后所有的用户都应该读取到最新值。

Availability：可用性，每一个操作总是能够在一定时间内返回结果。需要注意“一定时间”和“返回结果”。“一定时间”是指，系统结果必须在给定时间内返回。“返回结果”是指系统返回操作成功或失败的结果。

Partition Tolerance：分区容忍性，是否可以对数据进行分区。这是考虑到性能和可伸缩性。

3.数据一致性模型

一些分布式系统通过复制数据来提高系统的可靠性和容错性，并且将数据的不同的副本存放在不同的机器。

强一致性： 当更新操作完成之后，任何多个后续进程或者线程的访问都会返回最新的更新过的值。这种是对用户最友好的，就是用户上一次写什么，下一次就保证能读到什么。根据 CAP理论，这种实现需要牺牲可用性。

弱一致性： 系统并不保证续进程或者线程的访问都会返回最新的更新过的值。用户读到某一操作对系统特定数据的更新需要一段时间，我们称这段时间为“不一致性窗口”。系统在数据写入成功之后，不承诺立即可以读到最新写入的值，也不会具体的承诺多久之后可以读到。

最终一致性： 是弱一致性的一种特例。系统保证在没有后续更新的前提下，系统最终返回上一次更新操作的值。在没有故障发生的前提下，不一致窗口的时间主要受通信延迟，系统负载和复制副本的个数影响。DNS是一个典型的最终一致性系统。

二、典型的分布式事务实例

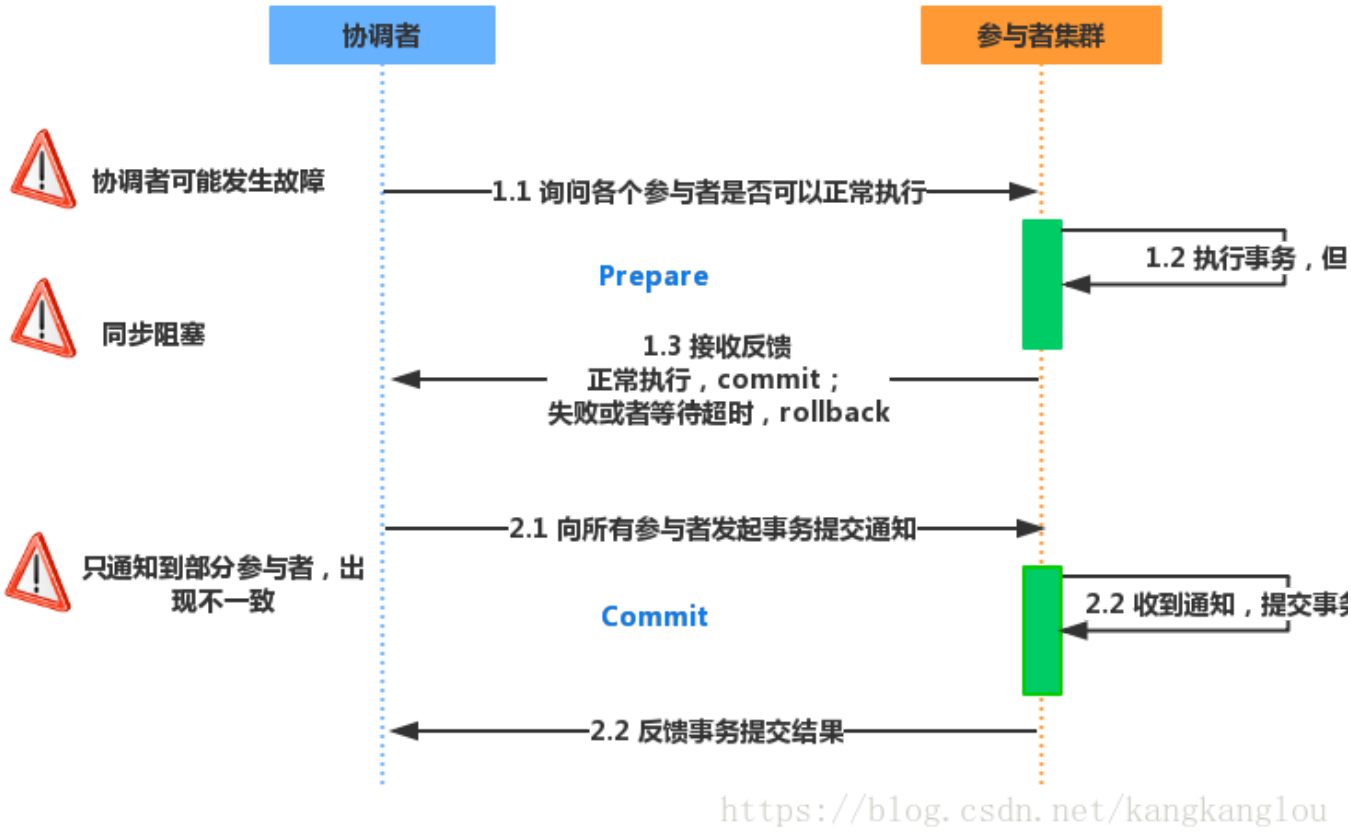
跨行转账问题是一个典型的分布式事务，用户A向B的一个转账1000，要进行A的余额-1000，B的余额+1000，显然必须保证这两个操作的事务性。 类似的还有，电商系统中，当有用户下单后，除了在订单表插入记，还要在商品表更新库存等，特别是随着微服务架构的流行，分布式事务的场景更变得更普遍。

三、两阶段提交协议

两阶段提交协议是协调所有分布式原子事务参与者，并决定提交或取消（回滚）的分布式算法。

1.协议参与者

在两阶段提交协议中，系统一般包含两类机器（或节点）：一类为协调者（coordinator），通常一个系统中只有一个；另一类为事务参与者（participants，cohorts或workers），一般包含多个，在数据存储系统中可以理解为数据副本的个数。协议中假设每个节点都会记录写前日志（write-ahead log）并持久性存储，即使节点发生故障日志也不会丢失。协议中同时假设节点不会发生永久性故障而且任意两个节点都可以互相通信。



2.两个阶段的执行

- **请求阶段（commit-request phase，或称表决阶段，voting phase）**：在请求阶段，协调者将通知事务参与者准备提交或取消事务，然后进入表决过程。在表决过程中，参与者将告知协调者自己的决策：同意（事务参与者本地作业执行成功）或取消（本地作业执行故障）。
- **提交阶段（commit phase）**：在该阶段，协调者将基于第一个阶段的投票结果进行决策：提交或取消。 当且仅当所有的参与者同意提交事务协调者才通知所有的参与者提交事务，否则协调者将通知所有的参与者取消事务。参与者在接收到协调者发来的消息后将执行响应的操作。

（3）两阶段提交的缺点

- **同步阻塞问题**：执行过程中，所有参与节点都是事务阻塞型的。当参与者占有公共资源时，其他第三方节点访问公共资源不得不处于阻塞状态。
- **单点故障**：由于协调者的重要性，一旦协调者发生故障。参与者会一直阻塞下去。尤其在第二阶段，协调者发生故障，那么所有的参与者还都处于锁定事务资源的状态中，而无法继续完成事务操作。（如果是协调者挂掉，可以重新选举一个协调者，但是无法解决因为协调者宕机导致的参与者处于阻塞状态的问题）

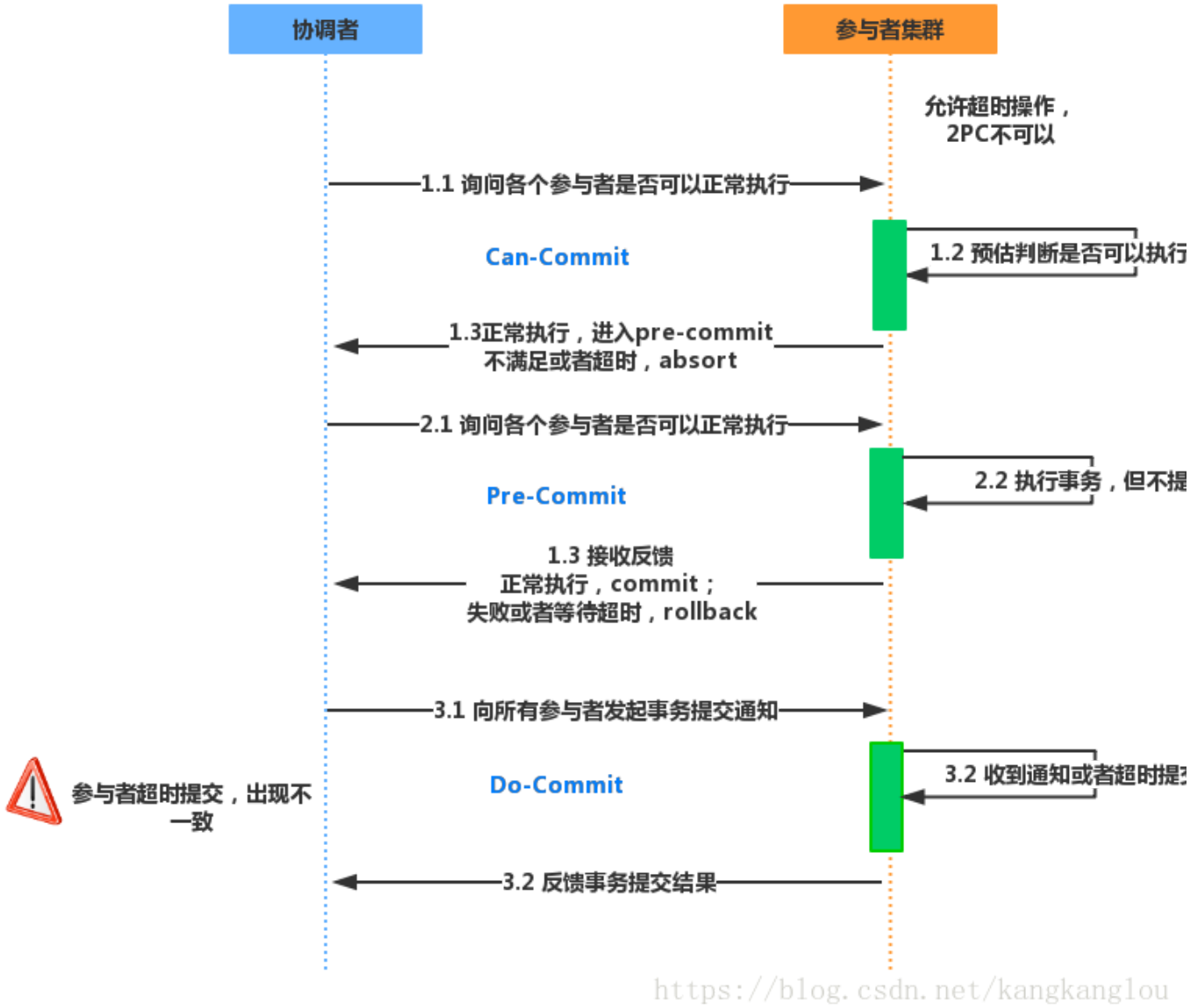
- **数据不一致：**在二阶段提交的阶段二中，当协调者向参与者发送commit请求之后，发生了局部网络异常或者在发送commit请求过程中协调者发生了故障，这回导致只有一部分参与者接受到了commit请求。而在这部分参与者接到commit请求之后就会执行commit操作。但是其他部分未接到commit请求的机器则无法执行事务提交。于是整个分布式系统便出现了数据部一致性的现象。

（4）两阶段提交无法解决的问题

当协调者出错，同时参与者也出错时，两阶段无法保证事务执行的完整性。考虑协调者再发出commit消息之后宕机，而唯一接收到这条消息的参与者同时也宕机了。那么即使协调者通过选举协议产生了新的协调者，这条事务的状态也是不确定的，没人知道事务是否被已经提交。

四、三阶段提交协议

三阶段提交协议在协调者和参与者中都引入超时机制，并且把两阶段提交协议的第一个阶段拆分成了两步：询问，然后再锁资源，最后真正提交。



（1）三个阶段的执行

- **CanCommit阶段：** 3PC的CanCommit阶段其实和2PC的准备阶段很像。 协调者向参与者发送commit请求，参与者如果可以提交就返回Yes响应，否则返回No响应。
- **PreCommit阶段：** Coordinator根据Cohort的反应情况来决定是否可以继续事务的PreCommit操作。 根据响应情况，有以下两种可能：

A.假如Coordinator从所有的Cohort获得的反馈都是Yes响应，那么就会进行事务的预执行：

> 1. 发送预提交请求。Coordinator向Cohort发送PreCommit请求，并进入Prepared阶段。
> 2. 事务预提交。Cohort接收到PreCommit请求后，会执行事务操作，并将undo和redo信息记录到事务日志中。
> 3. 响应反馈。如果Cohort成功的执行了事务操作，则返回ACK响应，同时开始等待最终指令。

B.假如有任何一个Cohort向Coordinator发送了No响应，或者等待超时之后，Coordinator都没有接到Cohort的响应，那么就中断事务：

> 1. 发送中断请求。Coordinator向所有Cohort发送abort请求。
> 2. 中断事务。Cohort收到来自Coordinator的abort请求之后（或超时之后，仍未收到Cohort的请求），执行事务的中断。

- **DoCommit阶段：**该阶段进行真正的事务提交，也可以分为以下两种情况:

执行提交

> 1. 发送提交请求。Coordinator接收到Cohort发送的ACK响应，那么他将从预提交状态进入到提交状态。并向所有Cohort发送doCommit请求。
> 2. 事务提交。Cohort接收到doCommit请求之后，执行正式的事务提交。并在完成事务提交之后释放所有事务资源。
> 3. 响应反馈。事务提交完之后，向Coordinator发送ACK响应。
> 4. 完成事务。Coordinator接收到所有Cohort的ACK响应之后，完成事务。

中断事务

> Coordinator没有接收到Cohort发送的ACK响应（可能是接受者发送的不是ACK响应，也可能响应超时），那么就会执行中断事务。

（2）三阶段提交协议和两阶段提交协议的不同

对于协调者(Coordinator)和参与者(Cohort)都设置了超时机制（在2PC中，只有协调者拥有超时机制，即如果在一定时间内没有收到cohort的消息则默认失败）。在2PC的准备阶段和提交阶段之间，插入预提交阶段，使3PC拥有CanCommit、PreCommit、DoCommit三个阶段。PreCommit是一个缓冲，保证了在最后提交阶段之前各参与节点的状态是一致的。

（3）三阶段提交协议的缺点

如果进入PreCommit后，Coordinator发出的是abort请求，假设只有一个Cohort收到并进行了abort操作，而其他对于系统状态未知的Cohort会根据3PC选择继续Commit，此时系统状态发生不一致性。