Zookeeper1 前序

一致性

> 首先提出几个问题：
>
> - 什么是分布式一致性?
> - 分布式一致性分为哪些类型?
> - 分布式系统达到一致性后将会是一个什么样的状态?

背景

> 背景：
>
> **火车站的售票系统有些系统**：既要快速地相应用户，同时还要保证系统的数据对任意客户端都是真实可靠的。
>
> **银行的转账系统**：需要为用户保证绝对可靠的数据安全，虽然在数据一致上存在延时，但最终务必保证严格的一致。
>
> **网购系统**：虽然向用户展示了一些可以说是"错误"的数据，但是在整个系统使用过程中，一定会在某个流程上对系统数据进行准确无误的检查，从而避免用户发生不必要的损失，就像网购系统。

一致性级别

强一致性：

> 这种一致性级别是最符合用户直觉的，它要求系统写入什么，读出来也会是什么，用户体验好，但实现起来往往对系统的性能影响比较大。（因为时间要足够短）

弱一致性：

> 这种一致性级别约束了系统在写入成功后，不承诺立即可以读到写入的值，也不具体承诺多久之后数据能够达到一致，但是会尽可能保证某个时间级别后，数据能够达到一致状态。弱一致性还可以再细分：
>
> - 会话一致性： 该一致性级别只保证对于写入的值，在同一个用户端会话中可以读到一致的值，但其他的会话不能保证
> - 用户一致性： 该一致性级别只保证对写入的值，在同一个用户中可以读到一致的值，但其他用户不能保证

最终一致性：

> 最终一致性是弱一致性的一个特例，系统会保证在一定时间内，能够达到一个数据一致的状态。这里之所以将最终一致性单独提出来，因为它是弱一致性中非常重要的一种一致性模型，也是业界在大型分布式系统的数据一致上比较推崇的模型。

分布式架构

集中式和分布式的特点

集中式特点:

- 部署结构简单，不需要考虑多个节点之间的分布式协作问题

分布式特点:

- 分布性
- 对等性
- 并发性
- 缺乏全局时钟

ACID

> 事务（Transaction）是由一系列对系统中数据进行访问与更新的操作所组成的一个程序执行逻辑单元。事务具有四个特性，这个一般在数据库相关的书籍有提到。

事务的四个特性:

- 原子性

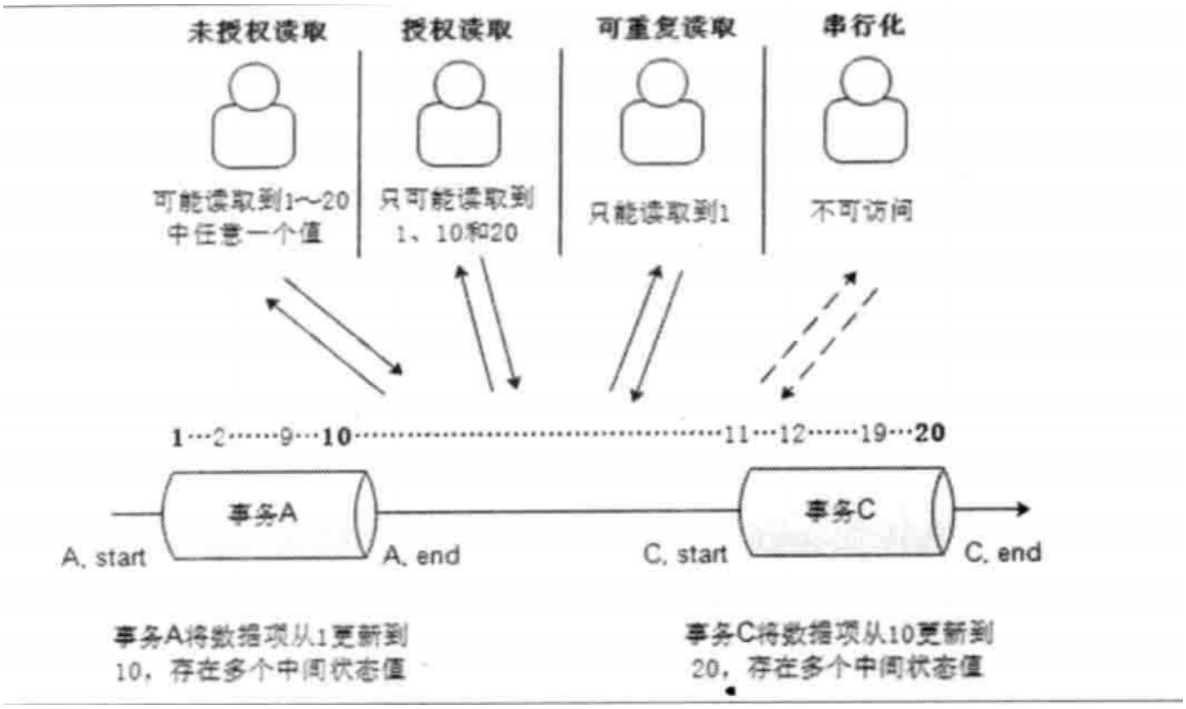
  ```
  事务的执行只有两种状态：
  1. 全部执行
  2. 全部不执行
  ```

- 一致性

  ```
  数据库只能从一个一致性状态转变到另一个一致性状态。
  ```

- 隔离性

  ```
  事务的隔离性是指在并发环境中，并发事务是相互隔离的。在sql中定义了四种事务隔离级别：
  1. 未授权读取 （Read Uncommitted)
  2. 授权读取 （Read Committed）
  3. 可重复读取 （Repeatable Read）
  4. 串行化 （Serializable）
  ```



隔离等级	脏读	可重复读	幻读
未授权读取	存在	不可以	存在
授权读取	不存在	不可以	存在
可重复读取	不存在	可以	存在
串行化	不存在	可以	不存在

- 持久性

  > 也被称为永久性，一个事务一旦提交，对数据库中的变更就应该是永久性。

如果想要让分布式实现一个能够保证ACID分布式事务处理系统就显得格外的复杂，所以并不适用。

CAP

> CAP理论告诉我们：一个分布式系统不可能同时满足一致性 C（consistency） 、可用性（A: Availability）和分区容错率（P: Partition tolerance） 这三个基本需求，最多满足两个。

- 一致性

  > 在分布式环境中，一致性是指数据在多个副本之间能否保持一致的特性。在分布式系统中，如果能做到针对一个数据项的更新操作执行成功后，所有用户都可以读取到最新的值，那么这样的系统就被认为具有强一致性

- 可用性

  > 可用性是指系统提供的服务必须一直处于可用的状态，对于用户的每一个操作请求总是能够在有限的时间内返回结果

- 分区容错性

> 分布容错性：分布式系统在遇到任何网络分区故障的时候，仍然需要能够保证对外提供满足一致性和可用性的服务，除非是整个网络环境都发生了故障。

BASE 理论（重点）

> BASE 是 Basically Available （基本可用）、Soft state （软状态）、 Eventually consistent （最终一致性） 其核心思想是：即使无法做到强一致性，但每个应用都可以根据自身的业务特点，采用适当的方式来使系统达到最终一致性。

- 基本可用性

> 基本可用是指分布式系统出现不可预支故障的时候，允许损失部分可用性：
> 1．响应上的损失
> 2．功能上的损失

- 弱状态

> 弱状态也称软状态，是允许系统中的数据存在的中间状态，并认为该中间状态的存在不会影响系统的整体可用性，即允许系统在不同节点的数据副本之间进行数据同步的过程存在延时

- 最终一致性

> 最终一致性强调的是系统中所有的数据副本，在经过一段时间的同步后，最终能够达到一个一致的状态，而不用实时保证系统数据的强一致性。

最终一致性一般存在以下五个类型

- 因果一致性 （Causal consistency)

> 如果进程A在更新完某个数据项后通知了进程B，那么进程B之后对该数据项的访问都应该能够获取到进程A更新后的最新值。

- 读己之所写（Read your write）

> 进程A更新一个数据项之后，它自己总是能够访问到更新过的最新值，而不会看到旧值

- 会话一致性（Session consistency）

> 会话一致性将系统数据的访问过程框定了一个会话中：系统能保证在同一个有效的会话中实现“读己之所写”的一致性。执行更新操作后，客户端能够在同一个会话中始终找到该数据项的最新值。

- 单调读一致性（Monotonic read consistency）

> 如果一个进程从系统中读取一个数据项的某个值之后，那么系统对于该进程后续的任何数据访问都不应该返回更旧的值。

- 单调写一致性（Monotonic write consistency)

  > 写的操作被顺序执行

具体的分布式系统架构设计中，CID特性与BASE理论往往会结合在一起使用。