

文从ACID到CAP到BASE (/a/1190000004468442)

xixicat (https://segmentfault.com/u/xixicat) 2月21日发布

分布式理论系列

- 从ACID到CAP到BASE (https://segmentfault.com/a/1190000004468442)
- 2PC到3PC到Paxos到Raft到ISR (https://segmentfault.com/a/1190000004474543)
- 复制、分片和路由 (https://segmentfault.com/a/1190000004485355)
- 副本更新策略 (https://segmentfault.com/a/1190000004480546)
- 负载均衡算法及手段 (https://segmentfault.com/a/1190000004492447)

序

本文主要讲述分布式系统开发的一些相关理论基础。

一、ACID

事务的四个特征：

1、Atomic原子性

事务必须是一个原子的操作序列单元，事务中包含的各项操作在一次执行过程中，要么全部执行成功，要么全部不执行，任何一项失败，整个事务回滚，只有全部都执行成功，整个事务才算成功。

2、Consistency一致性

事务的执行不能破坏数据库数据的完整性和一致性，事务在执行之前和之后，数据库都必须处于一致性状态。

3、Isolation隔离性

在并发环境中，并发的任务是相互隔离的，一个任务的执行不能被其他任务干扰。即不同的任务并发操纵相同的数据时，每个任务都有各自完整的数据空间，即一个任务内部的操作及使用的数据对其他并发任务是隔离的，并发执行的各个任务之间不能相互干扰。

SQL中的4个事务隔离级别：

（1）读未提交

允许脏读。如果一个任务正在处理某一数据，并对其进行了更新，但同时尚未完成任务，因此任务没有提交，与此同时，允许另一个任务也能够访问该数据。例如A将变量n从0累加到10才提交任务，此时B可能读到n变量从0到10之间的所有中间值。

（2）读已提交

允许不可重复读。只允许读到已经提交的数据。即任务A在将n从0累加到10的过程中，B无法看到n的中间值，之中只能看到10。同时有任务C进行从10到20的累加，此时B在同一个任务内再次读时，读到的是20。

（3）可重复读

允许幻读。保证在事务处理过程中，多次读取同一个数据时，其值都和任务开始时刻时是一致的。禁止脏读、不可重复读。幻读即同样的任务操作，在前后两个时间段内执行对同一个数据项的读取，可能出现不一致的结果。保证B在同一个任务内，多次读取n的值，读到的都是初始值0。幻读，就是不同任务，读到的n的数据可能是0，可能10，可能是20

（4）串行化

最严格的事务，要求所有事务被串行执行，不能并发执行。

如果不对事务进行并发控制，我们看看数据库并发操作是会有那些异常情形

- （1）一类丢失更新：两个事物读同一数据，一个修改字段1，一个修改字段2，后提交的恢复了先提交修改的字段。
- （2）二类丢失更新：两个事物读同一数据，都修改同一字段，后提交的覆盖了先提交的修改。
- （3）脏读：读到了未提交的值，万一该事物回滚，则产生脏读。
- （4）不可重复读：两个查询之间，被另外一个任务修改了数据的内容，产生内容的不一致。
- （5）幻读：两个查询之间，被另外一个任务插入或删除了记录，产生结果集的不一致。

4、Durability持久性

一个事务一旦提交，它对数据库中对应数据的状态变更就应该是永久性的，即使发生系统崩溃或机器宕机，只要数据库能够重新启动，那么一定能够将其恢复到事务成功结束时的状态。

二、CAP定理

一个分布式系统不可能同时满足一致性Consistency、可用性Availability、分区容错性Partition tolerance这三个基本需求，最多只能同时满足其中的两项。

1、一致性

分布式环境中，一致性是指多个副本之间能否保持一致的特性。在一致性的需求下，当一个系统在数据一致的状态下执行更新操作后，应该保证系统的数据仍然处理一致的状态。

2、可用性

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。

- （1）有限时间内
对于用户的一个操作请求，系统必须能够在指定的时间（响应时间）内返回对应的处理结果，如果超过了这个时间范围，那么系统就被认为是不可用的。即这个响应时间必须在一个合理的值内，不让用户感到失望。
- （2）返回正常结果
要求系统在完成对用户请求的处理后，返回一个正常的响应结果。正常的响应结果通常能够明确地反映出对请求的处理结果，即成功或失败，而不是一个让用户感到困惑的返回结果。比如返回一个系统错误如OutOfMemory，则认为系统是不可用的。

3、分区容错性

即分布式系统在遇到任何网络分区故障时，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。

网络分区，是指分布式系统中，不同的节点分布在不同的子网络（机房/异地网络）中，由于一些特殊的原因导致这些子网络之间出现网络不连通的状态，但各个子网络的内部网络是正常的，从而导致整个系统的网络环境被切分成了若干孤立的区域。组成一个分布式系统的每个节点的加入与退出都可以看做是一个特殊的网络分区。

三、CAP的应用

1、放弃P

放弃分区容错性的话，则放弃了分布式，放弃了系统的可扩展性

2、放弃A

放弃可用性的话，则在遇到网络分区或其他故障时，受影响的服务需要等待一定的时间，再此期间无法对外提供政策的服务，即不可用

3、放弃C

放弃一致性的话（这里指强一致），则系统无法保证数据保持实时的一致性，在数据达到最终一致性时，有个时间窗口，在时间窗口内，数据是不一致的。

对于分布式系统来说，P是不能放弃的，因此架构师通常是在可用性和一致性之间权衡。

四、BASE定理

Basically Available（基本可用）、Soft state（软状态）、Eventually consistent（最终一致性），基于CAP定理演化而来，核心思想是即时无法做到强一致性，但每个应用都可以根据自身业务特点，采用适当的方式来使系统达到最终一致性。

1、Basically Available（基本可用）

基本可用是指分布式系统在出现不可预知的故障的时候，允许损失部分可用性，但不等于系统不可用。

（1）响应时间上的损失

当出现故障时，响应时间增加

（2）功能上的损失

当流量高峰期时，屏蔽一些功能的使用以保证系统稳定性（服务降级）

2、Soft state（软状态）

与硬状态相对，即是指允许系统中的数据存在中间状态，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。

3、Eventually consistent（最终一致性）

强调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。其本质是需要系统保证最终数据能够达到一致，而不需要实时保证系统数据的强一致性。

最终一致性可分为如下几种：

- （1）因果一致性（Causal consistency）
即进程A在更新完数据后通知进程B，那么之后进程B对该项数据的范围都是进程A更新后的最新值。
- （2）读己之所写（Read your writes）
进程A更新一项数据后，它自己总是能访问到自己更新过的最新值。
- （3）会话一致性（Session consistency）
将数据一致性框定在会话当中，在一个会话当中实现读己之所写的一致性。即执行更新后，客户端在同一个会话中始终能读到该项数据的最新值
- （4）单调读一致性（Monotonic read consistency）
如果一个进程从系统中读取出一个数据项的某个值后，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。
- （5）单调写一致性（Monotoic write consistency）
一个系统需要保证来自同一个进程的写操作被顺序执行。

BASE定理是提出通过牺牲一致性来获得可用性，并允许数据在一段时间内是不一致的，但最终达到一致状态。

其他

待补充

参考

- 从Paxos到Zookeeper：分布式一致性原理与实践 (<http://book.douban.com/subject/26292004/>)

2月21日发布 (/a/1190000004468442)

1 推荐

收藏

你可能感兴趣的文章

- 说说分布式事务(一) (<https://segmentfault.com/a/1190000004474144>) 74 浏览
- kbengine开源分布式游戏服务端引擎 (<https://segmentfault.com/a/1190000000663501>) 2 收藏, 971 浏览
- Riak的分布式数据库模型 - 分布式数据库相关理论 Part3 (<https://segmentfault.com/a/1190000002802548>) 5 收藏, 639 浏览
- 本文采用 知识共享署名 3.0 中国大陆许可协议 (<http://creativecommons.org/licenses/by/3.0/cn>)，可自由转载、引用，但需署名作者且注明文章出处。


讨论区

请先 登录 () 后评论

本文隶属于专栏

xixicat (<https://segmentfault.com/blog/xixicat>)

spring boot , docker and so on

 xixicat (<https://segmentfault.com/u/xixicat>)
作者

关注专栏

分享扩散：

...