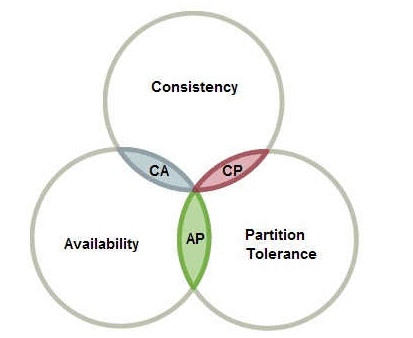
# CAP原则

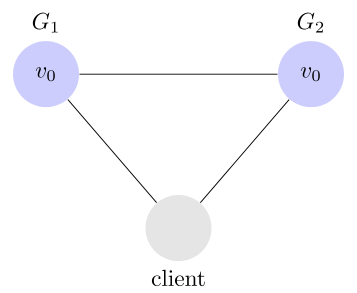
CAP原则又称CAP定理， Consistency（一致性）、 Availability（可用性）、Partition tolerance（分区容错性），三者的缩写，且三者不能共存。



## 分区容忍性（P）

以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择

大多数分布式系统都分布在多个子网络。每个子网络就叫做一个区（partition）。分区容错的意思是，区间通信可能失败。比如，一台服务器放在中国，另一台服务器放在美国，这就是两个区，它们之间可能无法通信。



上图中，G1 和 G2 是两台跨区的服务器。G1 向 G2 发送一条消息，G2 可能无法收到。系统设计的时候，必须考虑到这种情况。

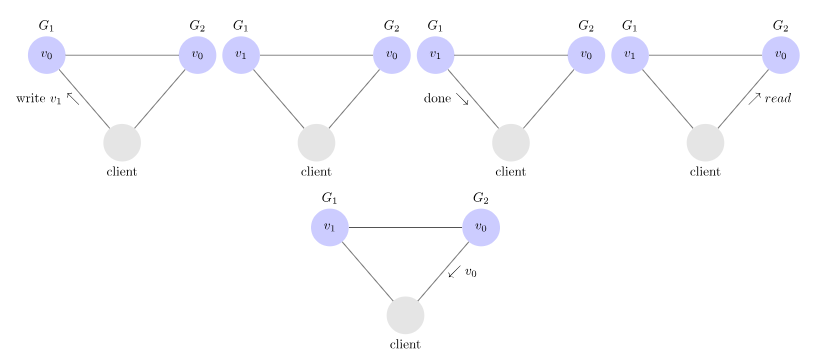
一般来说，分区容错无法避免，因此可以认为 CAP 的 P 总是成立。CAP 定理告诉我们，剩下的 C 和 A 无法同时做到。

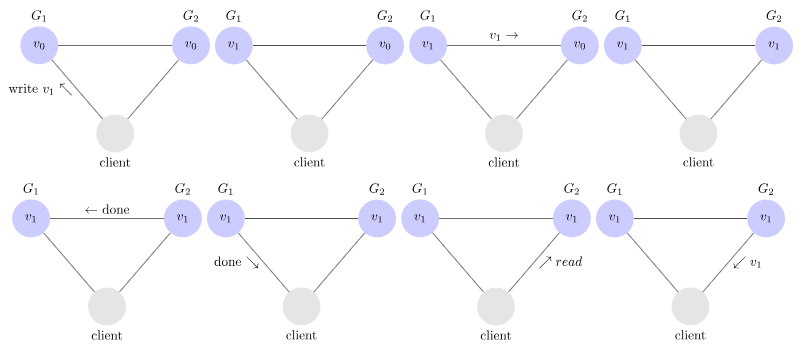
## 一致性（C）

Consistency 中文叫做"一致性"，访问分布式系统中的所有的节点，在同一时刻获取同样的值。（等同于所有节点访问同一份最新的数据副本）

举例来说，某条记录是 v0，用户向 G1 发起一个写操作，将其改为 v1。

非一致性系统，如果client从G2获取则得到v0；一致性系统从G2获取则获取v1。





## 可用性（A）

Availability 中文叫做"可用性"，意思是只要收到用户的请求，服务器就必须给出回应。用户可以选择向 G1 或 G2 发起读操作。不管是哪台服务器，只要收到请求，就必须告诉用户，到底是 v0 还是 v1，否则就不满足可用性。

## Consistency 和 Availability 的矛盾

一致性和可用性，为什么不可能同时成立？答案很简单，因为可能通信失败（即出现分区容错）。

如果保证 G2 的一致性，那么 G1 必须在写操作时，锁定 G2 的读操作和写操作。只有数据同步后，才能重新开放读写。锁定期间，G2 不能读写，没有可用性不。

如果保证 G2 的可用性，那么势必不能锁定 G2，所以一致性不成立。

综上所述，G2 无法同时做到一致性和可用性。系统设计时只能选择一个目标。如果追求一致性，那么无法保证所有节点的可用性；如果追求所有节点的可用性，那就没法做到一致性。

# BASE理论

BASE理论是Basically Available(基本可用)，Soft State（软状态）和Eventually Consistent（最终一致性）三个短语的缩写。

其核心思想是：既是无法做到强一致性（Strong consistency），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（Eventual consistency）

## 基本可用（Basically Available）

什么是基本可用呢？假设系统，出现了不可预知的故障，但还是能用，相比较正常的系统而言：

* 响应时间上的损失：正常情况下的搜索引擎0.5秒即返回给用户结果，而基本可用的搜索引擎可以在2秒作用返回结果。
* 功能上的损失：在一个电商网站上，正常情况下，用户可以顺利完成每一笔订单。但是到了大促期间，为了保护购物系统的稳定性，部分消费者可能会被引导到一个降级页面。

## 软状态（Soft State）

什么是软状态呢？相对于原子性而言，要求多个节点的数据副本都是一致的，这是一种“硬状态”。软状态指的是：允许系统中的数据存在中间状态，并认为该状态不影响系统的整体可用性，即允许系统在多个不同节点的数据副本存在数据延时。

## 最终一致性（Eventually Consistent）

上面说软状态，然后不可能一直是软状态，必须有个时间期限。在期限过后，应当保证所有副本保持数据一致性，从而达到数据的最终一致性。这个时间期限取决于网络延时、系统负载、数据复制方案设计等等因素。

而在实际工程实践中，最终一致性分为5种：

* 因果一致性（Causal consistency）

因果一致性指的是：如果节点A在更新完某个数据后通知了节点B，那么节点B之后对该数据的访问和修改都是基于A更新后的值。于此同时，和节点A无因果关系的节点C的数据访问则没有这样的限制。

* 读己之所写（Read your writes）

读己之所写指的是：节点A更新一个数据后，它自身总是能访问到自身更新过的最新值，而不会看到旧值。其实也算一种因果一致性。

* 会话一致性（Session consistency）

会话一致性将对系统数据的访问过程框定在了一个会话当中：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，也就是说，执行更新操作之后，客户端能够在同一个会话中始终读取到该数据项的最新值。

* 单调读一致性（Monotonic read consistency）

单调读一致性指的是：如果一个节点从系统中读取出一个数据项的某个值后，那么系统对于该节点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）

单调写一致性指的是：一个系统要能够保证来自同一个节点的写操作被顺序的执行。

在实际的实践中，这5种系统往往会结合使用，以构建一个具有最终一致性的分布式系统。实际上，不只是分布式系统使用最终一致性，关系型数据库在某个功能上，也是使用最终一致性的。比如备份，数据库的复制过程是需要时间的，这个复制过程中，业务读取到的值就是旧的。当然，最终还是达成了数据一致性。这也算是一个最终一致性的经典案例。

# ACID

ACID，是指[数据库管理系统](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F)（[DBMS](https://zh.wikipedia.org/wiki/DBMS)）在写入或更新资料的过程中，为保证[事务](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93%E4%BA%8B%E5%8A%A1)（transaction）是正确可靠的，所必须具备的四个特性：[原子性](https://zh.wikipedia.org/w/index.php?title=%E5%8E%9F%E5%AD%90%E6%80%A7&action=edit&redlink=1)（atomicity，或称不可分割性）、[一致性](https://zh.wikipedia.org/wiki/%E4%B8%80%E8%87%B4%E6%80%A7_(%E6%95%B0%E6%8D%AE%E5%BA%93))（consistency）、[隔离性](https://zh.wikipedia.org/wiki/%E9%9A%94%E9%9B%A2%E6%80%A7" \o "隔离性)（isolation，又称独立性）、[持久性](https://zh.wikipedia.org/w/index.php?title=%E6%8C%81%E4%B9%85%E6%80%A7&action=edit&redlink=1)（durability）。

## Atomicity（原子性）

一个事务（transaction）中的所有操作，或者全部完成，或者全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。即，事务不可分割、不可约简。

## Consistency（一致性）

在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设约束、触发器、级联回滚等。[1]

## Isolation（隔离性）

数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括未提交读（Read uncommitted）、提交读（read committed）、可重复读（repeatable read）和串行化（Serializable）。

| **隔离级别** | **脏读（Dirty Read）** | **不可重复读（NonRepeatable Read）** | **幻读（Phantom Read）** |
| --- | --- | --- | --- |
| **未提交读（Read uncommitted）** | 可能 | 可能 | 可能 |
| **已提交读（Read committed）** | 不可能 | 可能 | 可能 |
| **可重复读（Repeatable read）** | 不可能 | 不可能 | 可能 |
| **可串行化（Serializable ）** | 不可能 | 不可能 | 不可能 |

* 未提交读(Read Uncommitted)：允许脏读，也就是可能读取到其他会话中未提交事务修改的数据
* 提交读(Read Committed)：只能读取到已经提交的数据。Oracle等多数数据库默认都是该级别 (不重复读)
* 可重复读(Repeated Read)：可重复读。在同一个事务内的查询都是事务开始时刻一致的，InnoDB默认级别。在SQL标准中，该隔离级别消除了不可重复读，但是还存在幻象读
* 串行读(Serializable)：完全串行化的读，每次读都需要获得表级共享锁，读写相互都会阻塞

1. **脏读:** 脏读就是指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。
2. **不可重复读:**是指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。
3. **幻读:**第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。

## Durability（持久性）

事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。