# CAP

Consistency、Availability和Partition-tolerance。

**强一致性**：等同于所有节点访问同一份最新的数据副本。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的值，这样的系统被认为具有强一致性。

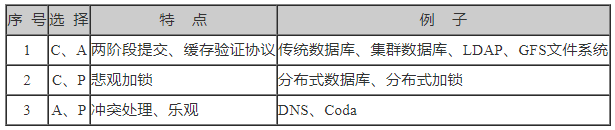
**可用性**：对数据更新具备高可用性。每一个操作总是能够在一定的时间内返回结果，这里需要注意的是“一定时间内”和“返回结果”。

**分区容错性**：能容忍网络分区。分区容错性可以理解为系统在存在网络分区的情况下仍然可以接受请求(满足一致性和可用性)。这里网络分区是指由于某种原因网络被分成若干个孤立的区域，而区域之间互不相通。还有一些人将分区容错性理解为系统对节点动态加入和离开的处理能力，因为节点的加入和离开可以认为是集群内部的网络分区。

如果Partition-tolerance的需求为0，只需要满足CA，可以延伸出ACID协议，比如RDBMS的事务一致性。然而对于互联网的分布式系统，鉴于高压力、大数据，那么必须对分区容忍性要求高，且高可用性，则采用BASE 弱一致性或者最终一致性。而对于很多互联网应用来说完全牺牲一致性是不可取的，否则数据是混乱的，那么系统可用性再高分布式再好也没有了价值。牺牲一致性，只是不再要求关系型数据库中的强一致性。从客户体验出发，最终一致性的关键是时间窗口，尽量让用户“用户感知到的一致性”。

## CAP的组合

根据CAP理论，这三个条件最多只能实现两点，不能三者兼顾，因此在架构设计的时候必须进行取舍。然而三个要素的取舍不是绝对的0%，100%的关系。 而是某个度的衡量，譬如C(30%), P(80%), A(99%)等。而且大系统内套小系统，大小子系统对CAP三个参数的要求也不一样。（http://www.infoq.com/cn/articles/cap-twelve-years-later-how-the-rules-have-changed/）



放弃P：如果想避免分区容错性问题的发生，一种做法是将所有的数据(与事务相关的)都放到一台机器上。虽然无法100%地保证系统不会出错，但不会碰到由分区带来的负面效果。当然，这个选择会严重影响系统的扩展性。

放弃A：相对于放弃“分区容错性”来说，其反面就是放弃可用性。一旦遇到分区容错故障，那么受到影响的服务需要等待数据一致，因此在等待期间系统就无法对外提供服务。

放弃C：这里所说的放弃一致性，并不是完全放弃数据的一致性，而是**放弃数据的强一致性**，而保留数据的最终一致性。以网络购物为例，对只剩最后一件库存的商品，如果同时接收到了两份订单，那么较晚的订单将被告知商品售罄。

除了这些组合，还有比如BASE，BASE支持最终一致性，其实是放弃C的一个特例。

**一致性与可用性**：在网络隔离的情况下，数据库将数据集中，就要接受数据丢失的风险。

**一致性与扩展性**：看得出即使读写一致性保证降低了副本集的扩展性，只有在原子写模型中才可以以一种相对可扩展的方式处理写冲突。原子读改写模型通过给数据加上临时性的全局锁来避免冲突。这表明，数据或操作之间的依赖，即使是很小范围内或很短时间的，也会损害扩展性。所以精心设计数据模型，将数据分片分开存放对于扩展性非常重要。

**一致性与延迟**：当数据库需要提供强一致性或者持久性的时候应该偏向于读写所有副本技术。但是很明显一致性与请求延迟成反比，所以使用若干副本技术会是比较中允的办法。

**故障转移与一致性、扩展性、延迟**：有趣的是容错性与一致性、扩展性、延迟的取舍冲突并不剧烈。通过合理的放弃一些性能与一致性，集群可以容忍多达 up to 的节点失效。这种折中在两阶段提交与 PAXOS 协议的区别里体现得很明显。这种折中的另一个例子是增加特定的一致性保障，比如使用严格会话进程的“读己所写”，但这又增加了故障转移的复杂性。

# ACID

ACID是强一致性协议，Atomicity、Consistency、Isolation和Durability

原子性：事务中的操作要么都做，要么都不做；

一致性：系统必须始终处在强一致状态下；

隔离性：一个事务的执行不能被其他事务所干扰；

持续性：一个已提交的事务对数据库中数据的改变是永久性的。

事务是用户定义的一个数据库操作序列，这些操作要么全不做，要么全做，是一个不可分割的工作单位。调当程序员定义的事务完成时，数据库处于一致的状态，比如对于转帐来说，事务完成时必须是A少了多少钱B就多了多少钱。ACID是事务所具有的特性。

# BASE

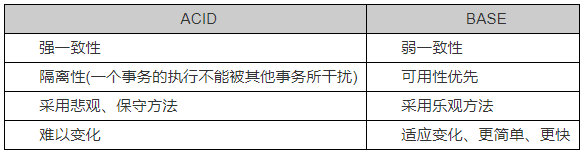
BASE是弱一致性协议，Basically Available、Soft-state、Eventually consistency。

基本可用：系统能够基本运行，一直提供服务；

软状态：系统不要求一直保持强一致状态；

最终一致性：系统需要在某一时刻后达到一致性要求。

# ACID和BASE的比较



# 一致性理解

在分布式系统中，强一致性从client和server两个角度分析。

从client角度理解：更新操作执行成功后，所有的用户都能够读取到最新的值；

从server角度理解：在任意时刻，所有节点中的数据是一样的，要求更新过的数据能被后续的访问都能看到。要求无论更新操作是在哪个数据副本上执行，之后所有的读操作都要能获得最新的数据。

对于单副本数据来说，读写操作是在同一数据上执行的，容易保证一致性，但是对于多副本的数据来说，则需要使用分布式事务协议（2PC或者Paxos等）。

如果数据更新后，能容忍后续部分访问不到，但是不是全部都访问不到，则是弱一致性。在这种一致性下，用户读到某一操作对系统特定数据的更新需要一段时间，我们将这段时间称为"不一致性窗口"。**弱一致性有弱到什么程度的问题，因此包含很多种不同的实现，目前分布式系统中广泛实现的是最终一致性。**

正如CAP理论所指出的，一致性、可用性和分区容错性不能同时满足。对于数据不断增长的系统（如社会计算、网络服务的系统），它们**对可用性及分区容错性的要求高于强一致性**，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上，由于维护数据副本的一致性代价很高，因此许多系统采用弱一致性来提高性能，一些不同的一致性模型也相继被提出。

**最终一致性是弱一致性的一种**，不保证在任意时刻任意节点上的同一份数据都是相同的，但是随着时间的迁移，不同节点上的同一份数据总是在向趋同的方向变化，在这种一致性下系统保证用户最终能够读取到某操作对系统特定数据的更新。也可以简单理解为在一段时间后，节点间的数据会最终达到一致性状态，这其实就是BASE。此种情况下，如果没有发生失败，"不一致性窗口"的大小依赖于交互延迟、系统的负载，以及复制技术中副本的个数（这个可以理解为master/slave模式中，slave的个数）。DNS系统可以说是在最终一致性方面最出名的系统，当更新一个域名的IP以后，根据配置策略以及缓存控制策略的不同，最终所有的客户都会看到最新的值。

最终一致性模型也根据其提供的不同保证可以划分为更多的模型：

因果一致性：假如有相互独立的A、B、C三个进程对数据进行操作。进程A对某数据进行更新后并将该操作通知给B，那么B接下来的读操作能够读取到A更新的数据值。但是由于A没有将该操作通知给C，那么系统将不保证C一定能够读取到A更新的数据值。

读自写一致性：这个一致性是指用户更新某个数据后读取该数据时能够获取其更新后的值，而其他的用户读取该数据时则不能保证读取到最新值。

会话一致性：是指读取自写更新一致性被限制在一个会话的范围内，也就是说提交更新操作的用户在同一个会话里读取该数据时能够保证数据是最新的。

单调一致性：是指用户读取某个数据值，后续操作不会读取到该数据更早版本的值。

时间轴一致性：要求数据的所有副本以相同的顺序执行所有的更新操作。

# 参考

<http://blog.csdn.net/duxingxia356/article/details/43992015>

<http://www.infoq.com/cn/articles/cap-twelve-years-later-how-the-rules-have-changed/>