亿级流量系统多级缓存架构4

日志

异常控制/错误编码

细分异常，多下功夫，功夫不是白费的，有可能一天、一个月、一年一个错误编码也没抛出来，但是一旦出了问题能够急速定位，根本不用看代码。

我们永远不能保证系统没有bug，bug可以藏的 很深埋的很久，但我们不怕，因为我们的伏兵也一 直在，你一跳我们立马抓，毫不犹豫。

异步

异步本身不是什么高深的技术，关键是哪些 业务可以走异步，这更体现架构师的业务理解能力 和综合能力

6个9

99.9999%

可用性、数据一致性

ACID特性

以传统数据库为代表，依然坚守：原子性、一致性、隔离性、持久性

CAP原则

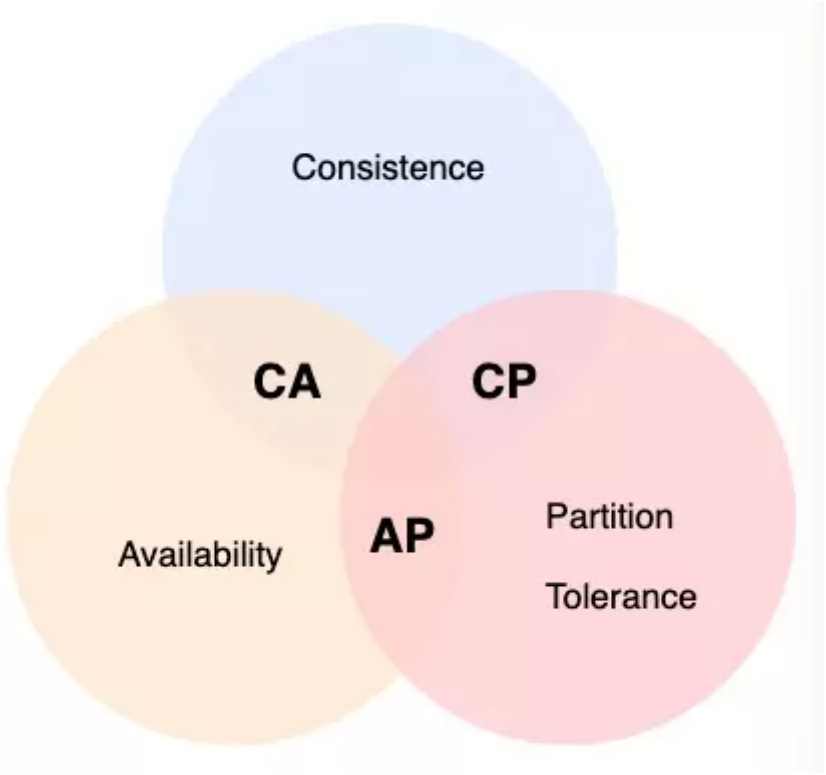
EricBrewer

https://mwhittaker.github.io/blog/an_illustrated_proof_of_the_cap_theorem/

对于开发或设计分布式系统的架构师工程师来说，CAP是必须要掌握的理论。

CAP定理又被成为布鲁尔定理，是加州大学计算机科学家埃里克·布鲁尔提出来的猜想，后来被证明成为分布式计算领域公认的定理

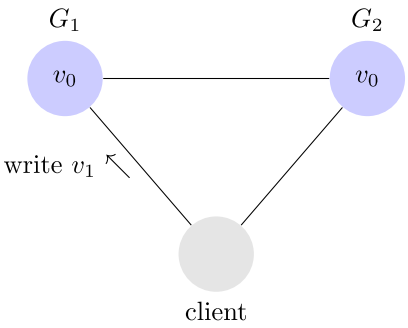
CAP定义，在高并发的场景下要做取舍，在大型集群中分区容错很难保证，一旦要确保容错性，那么就会损失数据一致性和高可用特性。



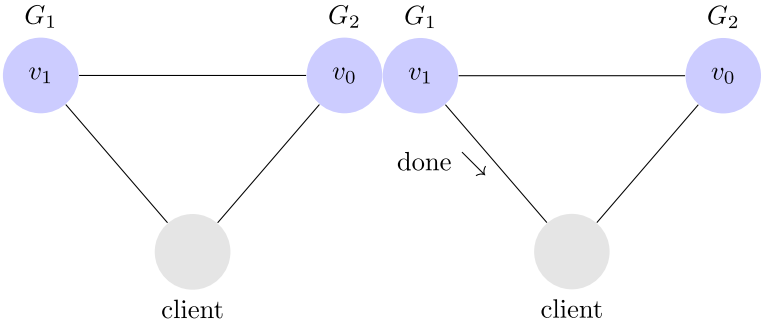
C

Consistency -> 一致性

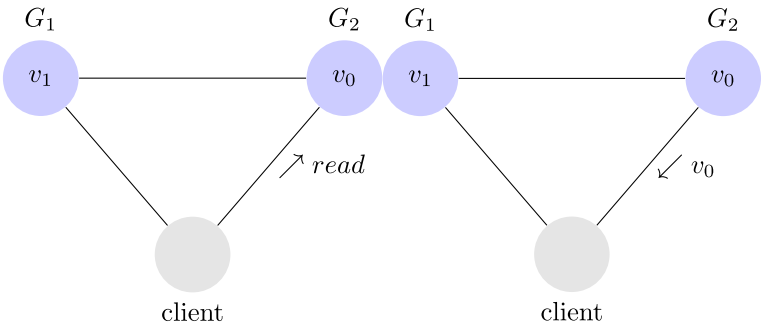
G1，G2为数据节点，同时存储了键值对 key=v:value=0



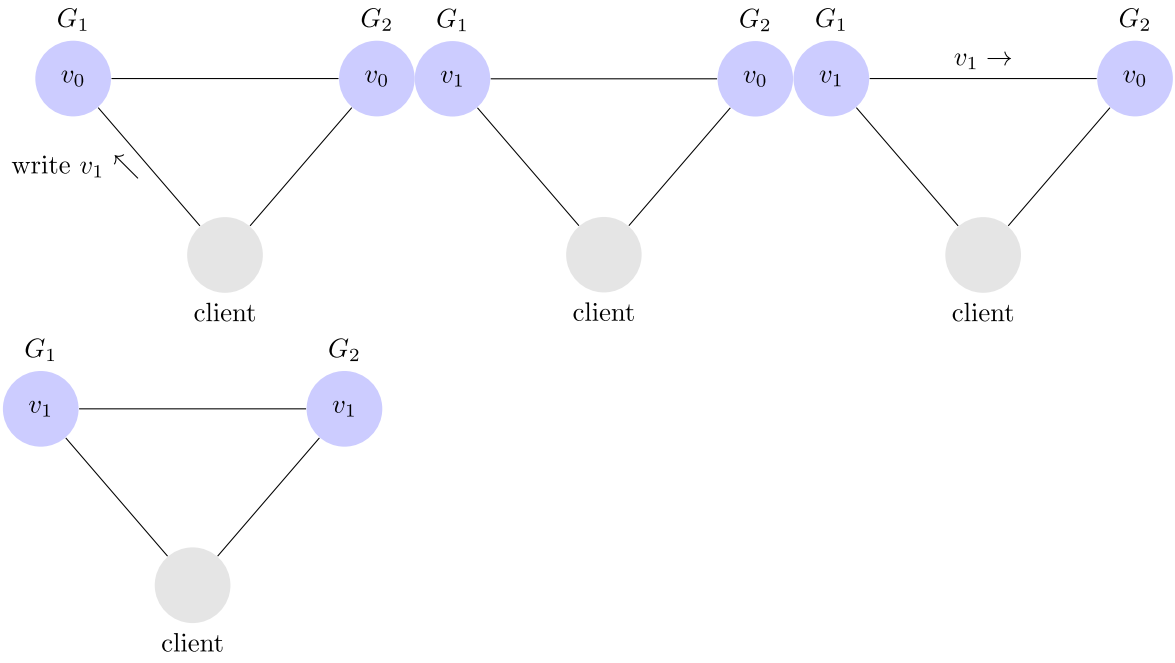
⭕ 上图：**向G1 写入数据**

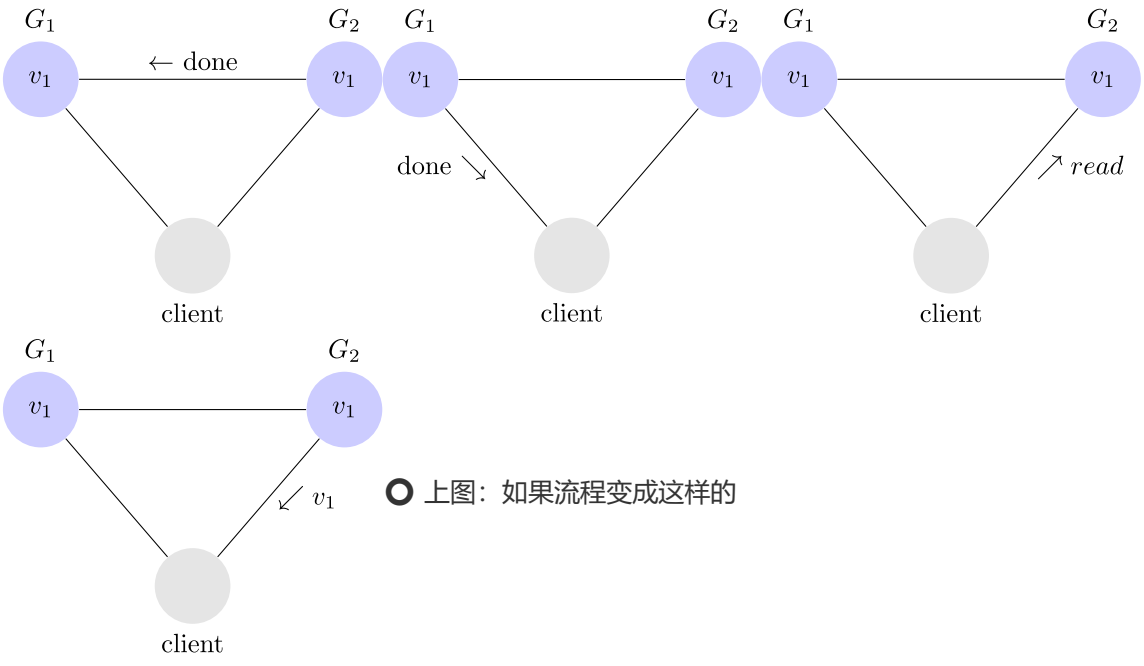


⭕ 上图：数据写入完成，G1->v:1，写入完成后，在向G1读取数据的时候就会得到v:1，此时是一致性



⭕ 上图：那么此时如果向G2发起读请求的话，因为数据没有同步，就会得到v:0,此时数据不一致





⭕ 上图：如果流程变成这样的

- 写入G1

- G1向G2同步数据
- **等待**同步完成
- 通知写完成
- 读取数据

似乎得到了一致性

一致性是指分布式系统中，数据在多节点存在副本，那么数据如果**一直不修改**，在读的时候是不存在问题的，你访问哪个节点的数据都一样

可一旦要是发生了**修改**，那么数据同步无法在修改的**瞬间**广播到所有副本节点

那么在读的时候就可能发生**数据脏读**

脑裂

数据库

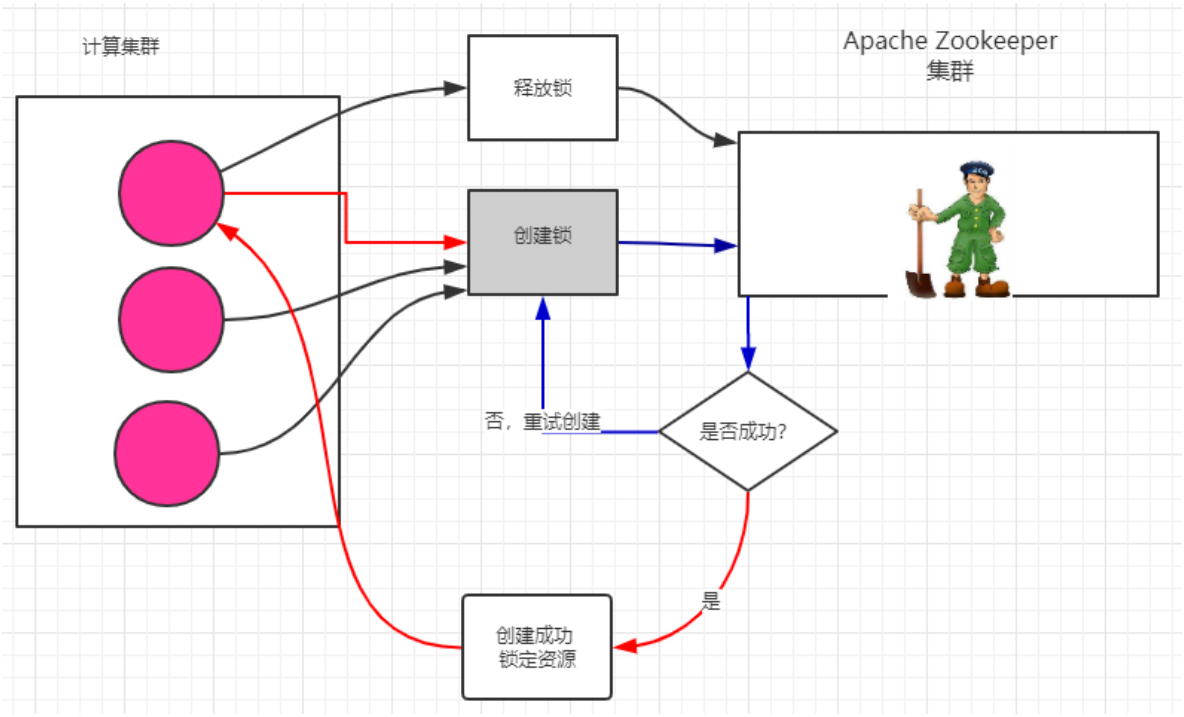
单点/AP

Redis

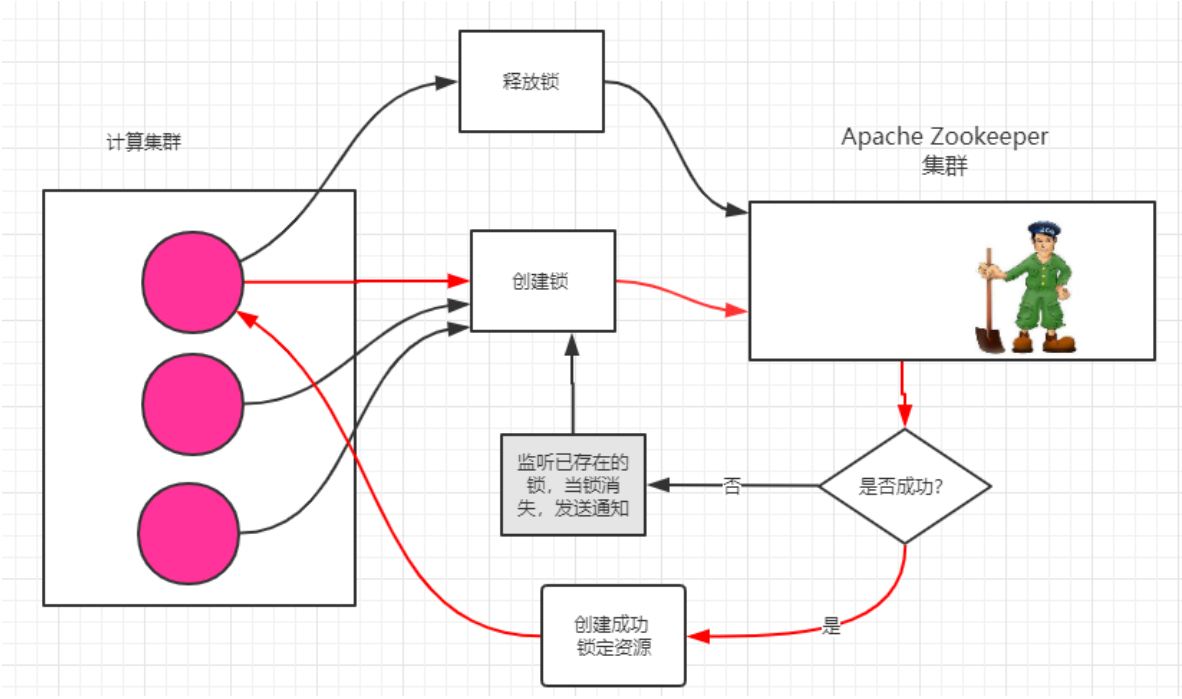
A/P

Zookeeper

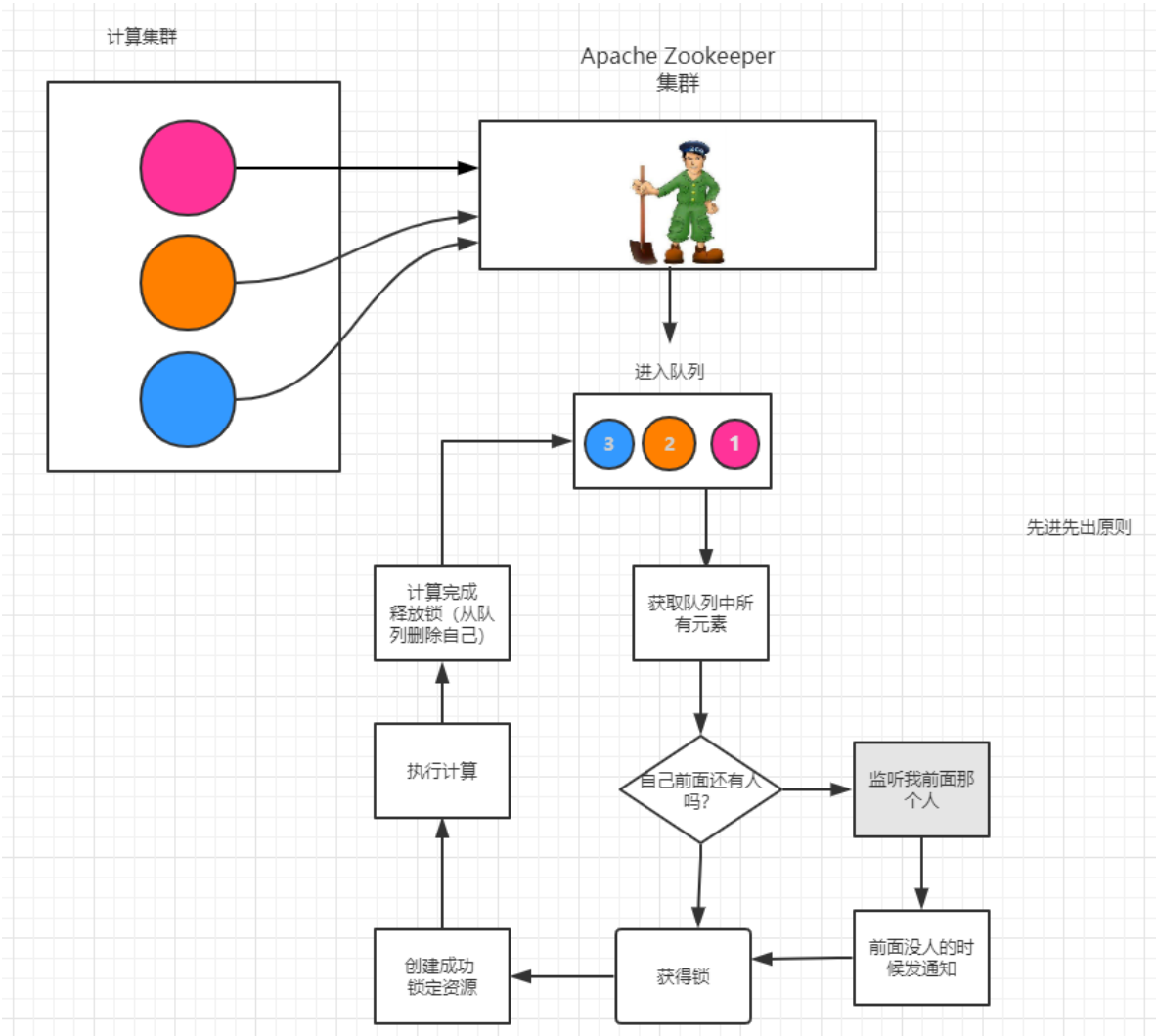
饥渴式



懒汉式



洗浴中心式



数据篡改

拜占庭将军问题

分布式系统 一致性问题

- 强一致性
- 弱一致性
- 最终一致性

CAP中的一致性

A

Availability -> 可用性

指的是服务是否可用，范围涵盖终端客户访问我们的系统或者是集群内部相互通讯交换数据

也就是说在Client向Server发起请求时，服务器返回了正确的响应，称之为可用，反之为不可用

这里有一个问题，如果发送请求在300年后给我返回数据了，那么算不算可用?

所以要提出访问延迟的概念，在某个时间范围内响应才算可用。

1s法则

1S法则"是面向Web侧，H5链路上加载性能 和体验方向上的一个指标，具体指:

- "强网" (4G/WIFI)下，1秒完全完成页面加载，包括首屏 资源，可看亦可用;
- 3G下1秒完成首包的返回 ;
- 2G下1秒完成建连。

高可用架构在实施的时候可以分两个方向

客户端容错

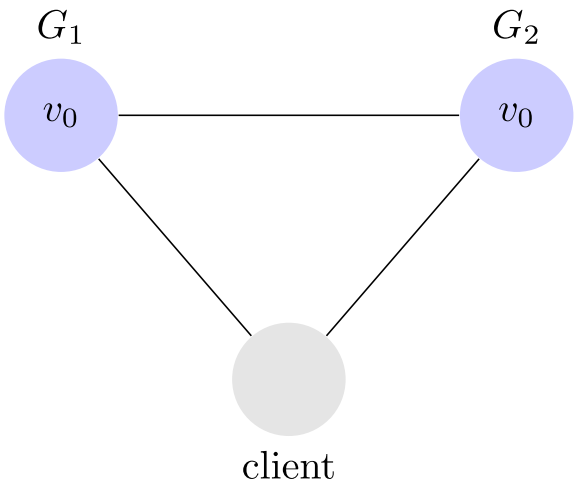
例如：游戏服务器

服务器端容错

例如：Nginx负载均衡

P

Partition tolerance -> 分区容错性



啥叫**分区容错**?

发生在分布式系统**内部**互访通信，

是指分布式网络中部分**网络**不可用，

但系统依然正常对外提供**服务**。

上图中 G1,G2是两台服务器，G1 向 G2 发送一条消息，G2 可能无法收到

比如：北京的订单系统，访问上海的库存系统

分区容错性是指分区具有容错性，我们可以尽可能的提高容错性，但是无法避免，

如果发生失败，就要在A和C之间做出选择。
要么停止系统进行错误恢复，要么继续服务但是降低一致性，所以我们说只能保证AP或CP。

在互相隔离的空间中，提供数据服务的系统。
CAP抽象：不同空间的数据，在同一时间，状态一致。

C：代表状态一致
A：代表同一时间
P：代表不同空间

CP

不同空间中的数据，如果要求他们所有状态一致，则必然不在同一时间。

AP

不同空间中，如果要求同一时间都可以从任意的空间拿到数据，则必然数据的状态不一致。

CA

不同空间的数据，如果要求任意时间都可以从任意空间拿到状态一致的数据，则空间数必然为1.

要强一致性的地方:

- 唯一ID生成, 这种性能很差. 并发不高.
- 对一致性要求比较高的系统，例如银行转账

要高可用: 最终一致, 即有可能读到脏数据. 但是一段时间之后总是能够读到新数据.

Zookeeper和Eureka

zookeepr

保证**CP**，即任何时刻对zookeeper的访问请求能得到一致性的数据结果，同时系统对网络分割具备容错性，但是它不能保证每次服务的可用性。从实际情况来分析，在使用zookeeper获取服务列表时，如果zk正在选举或者zk集群中半数以上的机器不可用，那么将无法获取数据。所以说，zk不能保证服务可用性。

eureka

保证**AP**，eureka在设计时优先保证可用性，每一个节点都是平等的，一部分节点挂掉不会影响到正常节点的工作，不会出现类似zk的选举leader的过程，客户端发现向某个节点注册或连接失败，会自动切换到其他的节点，只要有一台eureka存在，就可以保证整个服务处在可用状态，只不过有可能这个服务上的信息并不是最新的信息。

BASE 理论

eBay的架构师Dan Pritchett**源于对大规模分布式系统的实践总结**，在ACM上发表文章提出BASE理论，BASE理论是对CAP理论的延伸，核心思想是即使无法做到强一致性（StrongConsistency，CAP的一致性就是强一致性），但应用可以采用适合的方式达到最终一致性（Eventual Consitency）。

```
ACM国际大学生程序设计竞赛（英文全称：ACM International Collegiate Programming Contest（简称ACM-ICPC或ICPC））
```

Basically Available

基本可用

在分布式系统出现故障的时候，允许损失部分可用性,支持分区失败，即保证核心可用。

Soft State

软状态

接受一段时间的状态不同步，及中间状态，而改中间状态不影响系统整体可用性。这里的中间状态就是CAP理论中的数据不一致性。

Eventually Consistent

最终一致性

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。在期限过后系统能够保证在没有其他新的更新操作的情况下，数据最终一定能够达到一致的状态，因此所有客户端对系统的数据访问最终都能够获取到最新的值。

强一致性