# 理论部分

## 拜占庭将军问题

拜占庭将军问题是一个共识问题，问题的核心是军中可能有叛徒却要保证进攻的一致。引申到计算机邻域发展成一种容错和一致性理论，在这个问题中传递消息的信道是绝得可信的，也就是说不存在消息丢失或者消息无法传达等问题，**两军问题**是信道有问题的情况。如果使用口头信息(3M+1)，当且仅当超过2/3的将军是忠诚的时候该问题才可解，也就是说1个将军可以扰乱2个将军；如果使用不可伪造的书面信息，对于任何数目的将军和叛徒，该问题都是可解的。

**一个可靠的计算机系统必须能够处理一个或多个的组件的失败。一个失败的组件可能会表现出一种经常被忽略的行为：向系统的其他部分发送相矛盾的信息。**可以将处理这种失败的情况的问题抽象出来，就是这里的拜占庭将军问题。因为有叛徒将军，叛徒会向不同的将军发出不同的信息。

## CAP-ACID-BASE

Consistency、Availability、Partition-tolerance

Atomicity、Consistency、Isolation、Durability

Basically-Available、Soft-state、Eventually-consistency

理解CAP理论的最简单方式是想象两个节点分处分区两侧。

允许至少一个节点更新状态会导致数据不一致，即丧失了C性质;

如果为了保证数据一致性，将分区一侧的节点设置为不可用，那么又丧失了A性质;

除非两个节点可以互相通信，才能既保证C又保证A，这又会导致丧失P性质;

不确切地说，NoSQL运动的主题其实是创造**各种可用性优先、数据一致性其次的方案**；而传统数据库坚守ACID特性，做的是相反的事情。

关于数据一致性和可用性的争论，首先表现为ACID和BASE之争。当时BASE还不怎么被接受，主要是大家看重ACID的优点而不愿意放弃。提出CAP理论目的是证明有必要开拓更广阔的设计空间，因此才有了“三选二”公式。

“三选二”的观点在几个方面起了误导作用，**首先**，由于分区很少发生，那么在系统不存在分区的情况下没什么理由牺牲C或A。**其次**，C与A之间的取舍可以在同一系统内以非常细小的粒度反复发生，而每一次的决策可能因为具体的操作，乃至因为牵涉到特定的数据或用户而有所不同。**最后**，这三种性质都可以在程度上衡量，并不是非黑即白的有或无。可用性显然是在0%到100%之间连续变化的，一致性分很多级别，连分区也可以细分为不同含义，如系统内的不同部分对于是否存在分区可以有不一样的认知。

有时候在跨区域的系统，放弃强一致性来避免保持数据一致所带来的高延迟是非常有意义的。Yahoo的PNUTS系统因为以异步的方式维护远程副本而带来数据一致性的问题。但好处是主副本就放在本地，减小操作的等待时间。这个策略在实际中很实用，因为一般来讲，用户数据大都会根据用户的（日常）地理位置做分区。最理想的状况是每一位用户都在他的数据主副本附近。

Facebook使用了相反的策略：主副本被固定在一个地方，因此远程用户一般访问到的是离他较近，但可能已经过时的数据副本。不过当用户更新其页面的时候是直接对主副本进行更新，而且该用户的所有读操作也被短暂转向从主副本读取，尽管这样延迟会比较高。20秒后，该用户的流量被重新切换回离他较近的副本，此时副本应该已经同步好了刚才的更新。

## paxos

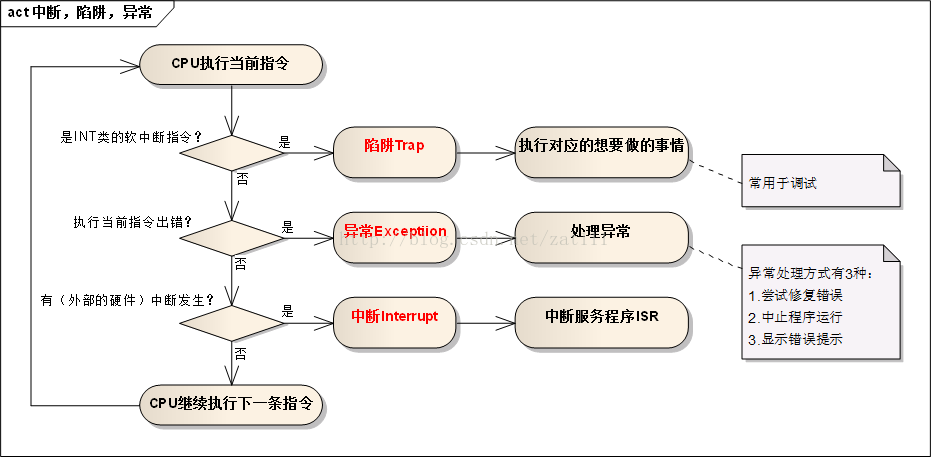
实际分布式系统一般是基于消息传递的异步分布式系统，进程可能会慢、被杀死或者重启，消息可能会延迟、丢失、重复、乱序等。在一个可能发生上述异常的分布式系统中如何就某个值达成一致，形成一致的决议，保证不论发生以上任何异常，都不会破坏决议的一致性，这些正是一致性算法要解决的问题。

存储系统多副本，主从同步不管是强一致同方式步还是异步同步方式，都无法同时保证数据的C和A，而**Paxos、Raft等分布式一致性算法则可在一致性和可用性之间取得很好的平衡，在保证一定的可用性的同时，能够对外提供强一致性，因此Paxos、Raft等分布式一致性算法被广泛的用于管理副本的一致性，提供高可用性。**

Paxos算法需要解决的问题就是如何在一个可能发生上述异常的分布式系统中，**快速且正确地在集群内部对某个数据的值达成一致，并且保证不论发生以上任何异常，都不会破坏整个系统的一致性。**

# 系统知识

## 中断、异常和陷阱



中断、异常和陷阱所对应的处理函数一般都可以称之为中断服务程序ISR，比如在陷阱中决定要做的事情，异常中决定如何响应出现的异常，中断中处理对应的事情。

中断是为了设备与CPU之间的通信，比如时钟中断、硬盘读写服务请求中断等。中断是异步的，因为从逻辑上来说，中断的产生与当前正在执行的进程无关。事实上，中断是如此有用，Linux用它来统计时钟，进行硬盘读写等。上面的中断我们可以称之为“硬件中断”，还有一种叫“软件中断”或者叫“可编程中断”，这类中断由编程者用int指令来触发，在Linux中唯一一个可编程中断就是int 0x80系统调用。

异常是由正在执行的进程产生的，包括出错（fault）、陷入（trap）、可编程异常。当从异常返回时，出错会重新执行那条指令，比如缺页异常；而陷入不会重新执行，比如调试程序时加的断点。