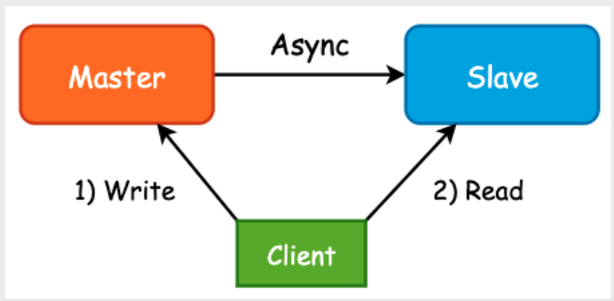
CAP

前言

当我们的业务出现爆发式的增长，单机无法提供稳定且高效的服务时，我们就需要将单机应用扩展至分布式架构。在进行水平扩展时，有状态的服务，例如 Redis、MongoDB 或者是 MySQL 的水平扩展将会成为一个难点

CAP 理论更像是一个标尺，用来衡量一个系统在一致性、可用性和分区容错性之间的取舍，在我们设计分布式系统组件时，可以使用 CAP 理论来检验最终的结果是否符合业务要求

异步复制模型



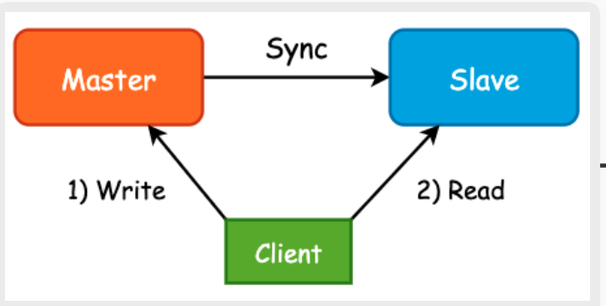
如左图所示，在异步复制模型中，Master 通过另一个线程将变更数据发送给 Slave，并不会阻塞 Master 的处理流程

异步复制模型可以说是进行读写分离、实现高可用的第一选择，因为它配置简单，并且 Master 节点并不会阻塞到 "日志复制" 这一动作上

1. 一致性（Consistency）
   - 显而易见地，由于 Master 是异步向 Slave 发送变更日志的，那么在 Slave 未接收到这些日志之前，Master 和 Slave 之间的数据是不一致性的
   - 在读写分离架构中，假设用户向 Master 写入了一条评论，并且立马从 Slave 节点尝试读取这条评论，就有可能会出现返回结果为空的情况。这种情况称之为读已之写，此时只能从业务层面去处理读写一致性
2. 可用性（Availability）
   - 可用性是指任何来自客户端的请求，不管访问哪个非故障节点，不管是否出现了网络分区，都能得到正常的响应数据，但不保证是同一份最新数据
3. 分区容错性（Partition Tolerance）
   - 这里的分区指的是网络分区，也就是集群中本应连通的多个节点，因为网络的高延迟或者其它原因被分割成了多个区域
   - 分区容错性是指当网络分区发生时，系统仍然可以正常工作。对于 Master 节点来说，当 Slave 出现网络分区导致不可达时，最多是异步复制通道断开，并不会影响 Master 的正常运行

在异步复制模型时，假如出现了网络分区，那么 Slave 的数据一定是滞后于 Master 节点的，但是这并不影响 Slave 对外提供服务，只是说数据不一定是最新的而已。因此，异步复制模型在出现分区时，只能满足可用性，无法满足一致性

同步复制模型



左图为同步复制模型，当 Master 接收数据写入时，在真正写入数据之前，需要同步地将数据变更发送给 Slave，只有 Slave 返回 ACK 之后，Master 才可以提交此数据

以 MySQL 无损复制为例，Master 在 write && fsync binlog 之后，将 binlog 发送给 Slave。Slave 接收并应用此条 binlog 之后，回送 ACK 给 Master。Master 只有在收到 ACK 以后才可以提交此事务，也就是二阶段提交的第二阶段

1. 一致性（Consistency）
   - 由于 Master 是同步向 Slave 发送变更日志的，那么只要 Slave 提交了此条变更日志，Master 和 Slave 就是一致的
2. 可用性（Availability）
   - 假设集群中出现了网络分区，那么 Master 将无法同步地将数据复制给 Slave，也就是说，Master 将返回错误信息
   - 或者说，Master 和 Slave 之间出现网络延迟，那么 Client 需要等待较长时间才能将数据写入，此时可用性会降低
3. 分区容错性（Partition Tolerance）
   - 如上所示，当发生网络分区时，系统将不可用

在分布式系统中，网络分区是必须要考虑的。因此，在 CAP 理论中，P 总是需要被满足。而对于同步复制模型来说，只能满足一致性和分区容错性（CP）

CAP 的不可能三角

通过上述两个基本复制模型我们可以看到，只要涉及到网络通信，那么就会发生网络波动、高延迟、数据丢失等情况，即分区故障必须考虑

那么此时，我们要么选择一致性（Consistency），要么选择可用性（Availability）

- 选择一致性
  - 当我们选择了一致性时，例如同步复制模型，那么不管如何读写数据，都能保证数据在各个节点间是一致的。但是，一旦出现节点宕机或者是出现分区时，系统将不可用（放弃了可用性）
- 选择可用性
  - 当我们选择了可用性的，当节点出现宕机或者是出现分区时，并不会影响系统的运行，只会出现数据不一致的情况，也就是说，放弃了一致性

🤔 简单来说，CAP 理论就是这样的一个矛盾体

- 为了增强集群的分区容错性，将数据复制到更多的节点上，但是此时数据的复制就会变多，强一致性很难保证
- 如果要保持强一致性，那么更新所有节点数据的时间也就会变长，此时可用性又会降低

对于 CP 模型来说，放弃了可用性，选择了一致性，适用于对数据一致性要求非常高的场景，例如金融业务、Kubernetes 集群中的资源信息等。使用 Raft 协议实现的金融级 MySQL 集群、etcd 或者 Consul 等，都可以认为是 CP 模型

对于 AP 模型来说，放弃了强一致性，使用最终一致性来保证数据的最终一致，适用于一致性要求不高的场景，例如 CDN 缓存、DNS 缓存等