20180326_分布式系统应对单点故障策略选择

```
1- hdfs 中关于 NameNode 和 DataNode 的 主备机制
2- zookeper 维护的 主从 机制
```

一，引入

在现在的网络大环境下，越来越大的信息量导致了web应用系统一步步的进行变革。应用架构经历了四次大的改变：从 **单一应用架构** 到 **垂直应用架构** 再到 **分布式服务框架** 最后到现在的 **流动计算框架**。相应的，服务器的处理数据量也有了一个大的飞跃，随之而来的就是过大的请求量给服务器带来的挑战。而单点服务器再不能适用于现在的线上环境，所以下面对单点故障的应对策略简单总结下。

二，两种策略

目前博主接触到的应对单点故障有两种。第一种，hadoop2.x中hdfs集群架构采用的高可用架构（HA）中采用的主备模式。第二种，zookeeper集群中采用的主从模式。下面就这两种模式结合典型应用场景进行讲解，欢迎各位看官指正，一起学习，共同进步。

三，主备方式详解

主备方式通常是一台主机、一台或多台备机，在正常情况下主机对外提供服务，并把数据同步到备机，当主机宕机后，备机立刻开始服务。优点是对客户端毫无影响，缺点也很明显，在绝大多数时间内备机是一直没使用，被浪费着的。

主备模式的应用场景博主举两个例子：

（1）hdfs文件系统中HA架构：

在hadoop中，通常可以大致分为两个大的版本hadoop1.x和hadoop2.x。在1.x版本中，分布式系统为基本的一个NN(namenode)节点加上多台DN（datanode）节点，当然你也可以通过给NN配置SNN（secondary namenode）节点来加快启动速度。

但是SNN并不能为NN节点提供备份，因为SNN的主要工作只是将NN中的元数据（fsimage）和操作日（edits）拷贝过来进行文件合并。虽然元数据会在初始化时加载到内存中并对它持久化，但是其中块位置信息是不提供持久化的，hadoop为了数据的准确性，块位置只能通过各DN节点上报块报告获取，所以SNN中的备份不完整。这样的系统因为单点故障的问题不能够胜任线上环境。

而到了hadoop2.x，为了应对单点故障，提出了HA（高可用）架构：

（a）简述

典型的HA集群，两个单独的NN，在任何时候，一个NN处于活动状态，另一个处于待机状态。主NN负责处 理来自客户端的读写请求，备NN进行同步，在必要时提供故障转移。两个NN同时访问一个共享存储设 备，主NN将元数据写入其中，备NN从中读取数据进行同步。

（b）HA架构

在关于共享存储设备的选择上，因为NFS(网络共享文件)也会有单点故障问题，所以使用了

Zookeeper（ZK）+JournalNode(JN)来实现。

从下往上说：主备NN共用DN集群，各DN向两者发送块报告和心跳包，但只响应主NN的指令。主NN将元数 据存储在JN集群中，备NN进行数据同步。主备NN由对应的两个故障切换控制器进行管理，主备故障切换 控制器属于ZK集群。主备NN向控制器发送心跳包，由ZK进行监控并且在单点故障时进行主备切换。

以上就是HA的主备方式实现，有意思的是该系统中，客户端会先访问ZK，由它返回主NN位置，客户端再去访问主NN，由它返回DN块位置，客户端在第三次才访问对应的DN进行读写操作。

（2）分布式redis中的HA：

Redis HA中使用比较多的是keepalived，它使主机备机对外提供同一个虚拟IP，客户端通过虚拟IP进行操作，正常期间主机一直对外提供服务，宕机后VIP自动漂移到备机上。

这边说明下，redis比较特殊。redis支持主从的模式，但是它是用来进行读写数据分离。redis中Master会将数据同步到slave，而slave不会将数据同步到master。Slave启动时会连接master来同步数据。这是一个典型的分布式读写分离模型。我们可以利用master来插入数据，slave提供检索服务，这样可以有效减少单个机器的并发访问数量。通过增加Slave DB的数量，读的性能可以线性增长。为了避免Master DB的单点故障，集群一般都会采用两台Master DB做双机热备，所以整个集群的读和写的可用性都非常高。

四，主从方式详解

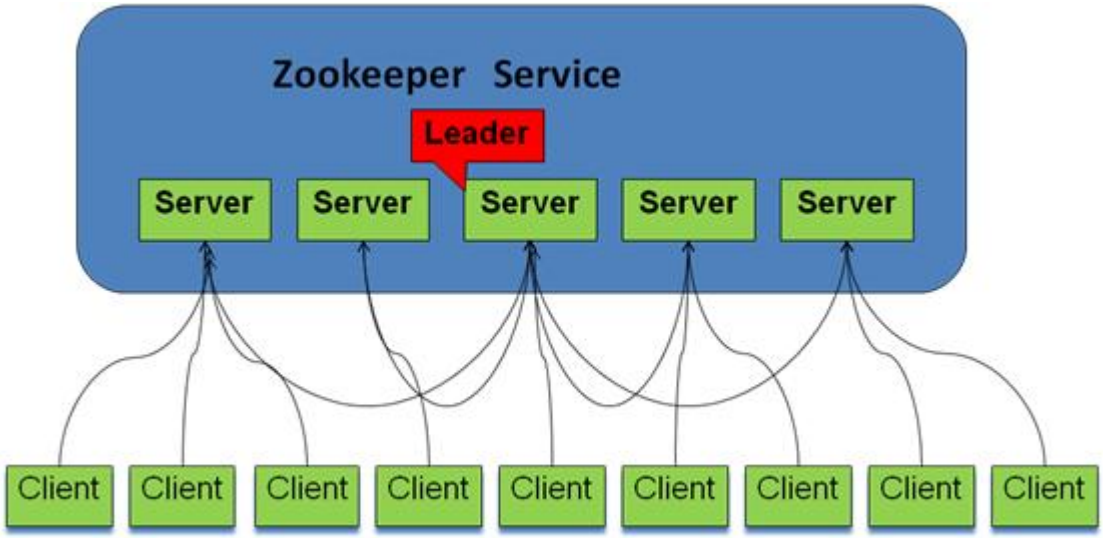
这种采取一主多从的办法，主从之间进行数据同步。 当Master宕机后，通过选举算法(Paxos、Raft)从slave中选举出新Master继续对外提供服务，主机恢复后以slave的身份重新加入。主从另一个目的是进行读写分离，这是当单机读写压力过高的一种通用型解决方案。 其主机的角色只提供写操作或少量的读，把多余读请求通过负载均衡算法分流到单个或多个slave服务器上。 缺点是主机宕机后，Slave虽然被选举成新Master了，但对外提供的IP服务地址却发生变化了，意味着会影响到客户端。 解决这种情况需要一些额外的工作，在当主机地址发生变化后及时通知到客户端，客户端收到新地址后，使用新地址继续发送新请求。

主从的方式的典型例子是zookeeper：

Zookeeper中的角色主要有以下三类，如下表所示：

角色		描述
领导者（Leader）		领导者负责进行投票的发起和决议，更新系统状态
学习者（Learner）	跟随者（Follower）	Follower 用于接收客户请求并向客户端返回结果，在选主过程中参与投票
	观察者（ObServer）	ObServer 可以接收客户端连接，将写请求转发给 leader 节点。但 ObServer 不参加投票过程，只同步 leader 的状态。ObServer 的目的是为了扩展系统，提高读取速度
客户端（Client）		请求发起方

系统模型如图所示：



组成Zookeeper的各个服务器必须要能相互通信。他们在内存中保存了服务器状态，也保存了操作的日志，并且持久化快照。只要大多数的服务器是可用的，那么Zookeeper就是可用的。

我们已经知道了一个zookeeper集群中，有一个处于leader身份的节点，其他的节点都是flower状态。

那么一个leader是怎么产生的呢？这就是zookeeper中的选举规则，默认的选举规则称为：FastLeaderELection（是PAXOS算法的实现，网上的资料还有提到另外的选举算法，实际上它们的核心思想都是一样的）