https://www.cnblogs.com/mushroom/p/4752962.html

[Redis高可用分布式内部交流(九)](https://www.cnblogs.com/mushroom/p/4752962.html)

这是上月在公司内部的一次分享，现把PPT及交流内容整理成博客。

**阅读目录：**

1. [高可用](https://www.cnblogs.com/mushroom/p/4752962.html#one)
2. [数据同步](https://www.cnblogs.com/mushroom/p/4752962.html#two)
3. [分布式](https://www.cnblogs.com/mushroom/p/4752962.html#three)
4. [分布式集群时代](https://www.cnblogs.com/mushroom/p/4752962.html#four)
5. [总结](https://www.cnblogs.com/mushroom/p/4752962.html#five)

**高可用**

高可用（High Availability），是当一台服务器停止服务后，对于业务及用户毫无影响。 停止服务的原因可能由于网卡、路由器、机房、CPU负载过高、内存溢出、自然灾害等不可预期的原因导致，在很多时候也称单点问题。  
解决单点问题主要有2种方式：

**主备方式**

这种通常是一台主机、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。   
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进行数据操作，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。

优点是对客户端毫无影响，仍然通过VIP操作。  
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。

**主从方式**

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。

**数据同步**

无论是主备还是主从都牵扯到数据同步的问题，这也分2种情况：

* 同步方式：当主机收到客户端写操作后，以同步方式把数据同步到从机上，当从机也成功写入后，主机才返回给客户端成功，也称数据强一致性。 很显然这种方式性能会降低不少，当从机很多时，可以不用每台都同步，主机同步某一台从机后，从机再把数据分发同步到其他从机上，这样提高主机性能分担同步压力。 在Redis中是支持这杨配置的，一台master，一台slave，同时这台salve又作为其他slave的master。
* 异步方式：主机接收到写操作后，直接返回成功，然后在后台用异步方式把数据同步到从机上。 这种同步性能比较好，但无法保证数据的完整性，比如在异步同步过程中主机突然宕机了，也称这种方式为数据弱一致性。

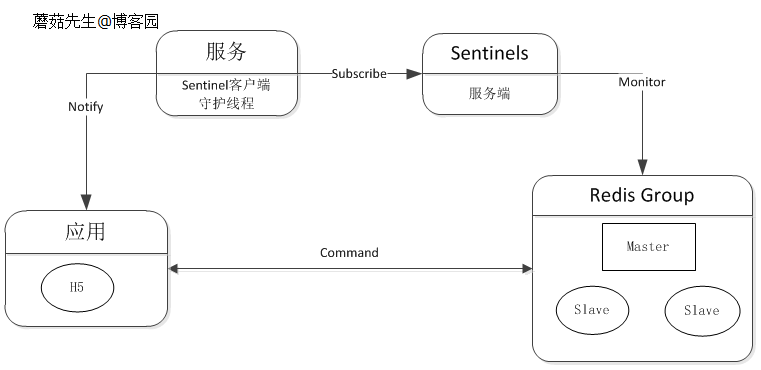
Redis主从同步采用的是异步方式，因此会有少量丢数据的危险。还有种弱一致性的特例叫最终一致性，这块详细内容可参见CAP原理及一致性模型。

**方案选择**

keepalived方案配置简单、人力成本小，在数据量少、压力小的情况下推荐使用。 如果数据量比较大，不希望过多浪费机器，还希望在宕机后，做一些自定义的措施，比如报警、记日志、数据迁移等操作，推荐使用主从方式，因为和主从搭配的一般还有个管理监控中心。

宕机通知这块，可以集成到客户端组件上，也可单独抽离出来。 Redis官方Sentinel支持故障自动转移、通知等，详情见[低成本高可用方案设计(四)](http://www.cnblogs.com/mushroom/p/4526912.html)。

逻辑图：



**分布式**

分布式(distributed), 是当业务量、数据量增加时，可以通过任意增加减少服务器数量来解决问题。

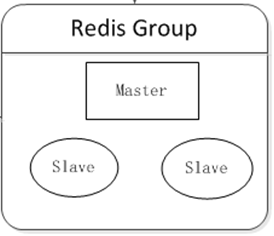
**集群时代**

至少部署两台Redis服务器构成一个小的集群，主要有2个目的：

* 高可用性：在主机挂掉后，自动故障转移，使前端服务对用户无影响。
* 读写分离：将主机读压力分流到从机上。

可在客户端组件上实现负载均衡，根据不同服务器的运行情况，分担不同比例的读请求压力。

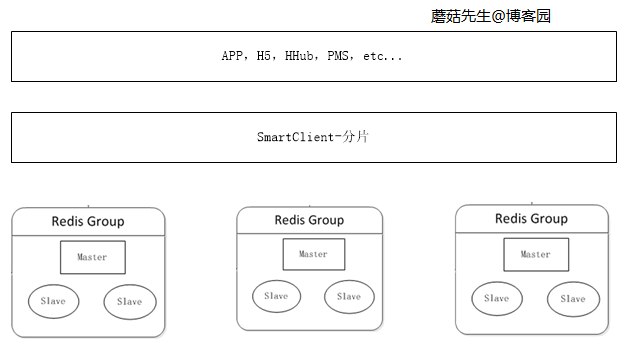
逻辑图：



**分布式集群时代**

当缓存数据量不断增加时，单机内存不够使用，需要把数据切分不同部分，分布到多台服务器上。   
可在客户端对数据进行分片，数据分片算法详见[C#一致性Hash详解](http://www.cnblogs.com/mushroom/p/4375469.html)、[C#之虚拟桶分片](http://www.cnblogs.com/mushroom/p/4542772.html)。

逻辑图：



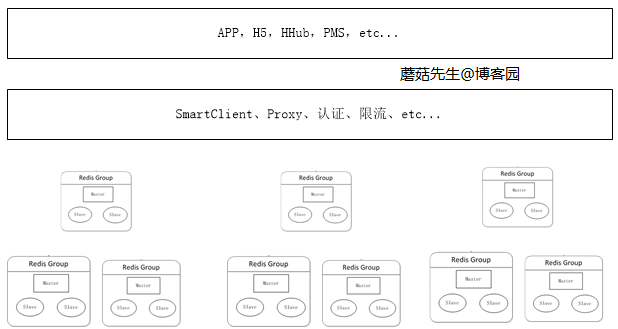
**大规模分布式集群时代**

当数据量持续增加时，应用可根据不同场景下的业务申请对应的分布式集群。 这块最关键的是缓存治理这块，其中最重要的部分是加入了代理服务。 应用通过代理访问真实的Redis服务器进行读写，这样做的好处是：

* 避免越来越多的客户端直接访问Redis服务器难以管理，而造成风险。
* 在代理这一层可以做对应的安全措施，比如限流、授权、分片。
* 避免客户端越来越多的逻辑代码，不但臃肿升级还比较麻烦。
* 代理这层无状态的，可任意扩展节点，对于客户端来说，访问代理跟访问单机Redis一样。

目前楼主公司使用的是客户端组件和代理两种方案并存，因为通过代理会影响一定的性能。 代理这块对应的方案实现有Twitter的Twemproxy和豌豆荚的codis。

逻辑图：



**总结**

分布式缓存再向后是云服务缓存，对使用端完全屏蔽细节，各应用自行申请大小、流量方案即可，如淘宝OCS云服务缓存。  
分布式缓存对应需要的实现组件有：

* 一个缓存监控、迁移、管理中心。
* 一个自定义的客户端组件，上图中的SmartClient。
* 一个无状态的代理服务。
* N台服务器。

谢谢大家