高可用Redis服务架构分析与搭建

基于内存的Redis应该是目前各种web开发业务中最为常用的key-value数据库了，我们经常在业务中用其存储用户登陆态（Session存储），加速一些热数据的查询（相比较mysql而言，速度有数量级的提升），做简单的消息队列（LPUSH和BRPOP）、订阅发布（PUB/SUB）系统等等。规模比较大的互联网公司，一般都会有专门的团队，将Redis存储以基础服务的形式提供给各个业务调用。

不过任何一个基础服务的提供方，都会被调用方问起的一个问题是：你的服务是否具有高可用性？最好不要因为你的服务经常出问题，导致我这边的业务跟着遭殃。最近我所在的项目中也自己搭了一套小型的“高可用”Redis服务，在此做一下自己的总结和思考。

首先我们要定义一下对于Redis服务来说怎样才算是高可用，即在各种出现异常的情况下，依然可以正常提供服务。或者宽松一些，出现异常的情况下，只经过很短暂的时间即可恢复正常服务。所谓异常，应该至少包含了以下几种可能性：

【异常1】某个节点服务器的某个进程突然down掉（例如某开发手残，把一台服务器的redis-server进程kill了）

【异常2】某台节点服务器down掉，相当于这个节点上所有进程都停了（例如某运维手残，把一个服务器的电源拔了；例如一些老旧机器出现硬件故障）

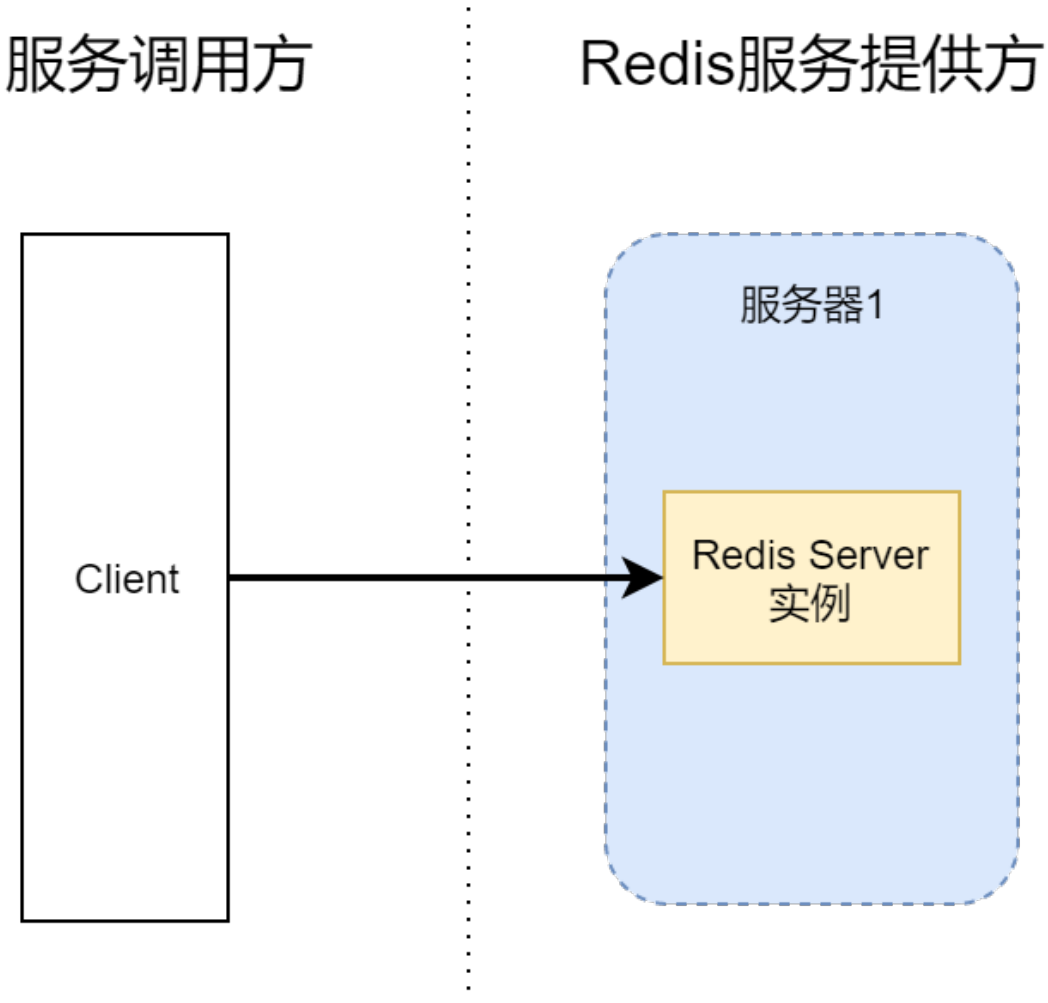
【异常3】任意两个节点服务器之间的通信中断了（例如某临时工手残，把用于两个机房通信的光缆挖断了）

其实以上任意一种异常都是小概率事件，而做到高可用性的基本指导思想就是：多个小概率事件同时发生的概率可以忽略不计。只要我们设计的系统可以容忍短时间内的单点故障，即可实现高可用性。

对于搭建高可用Redis服务，网上已有了很多方案，例如Keepalived，Codis，Twemproxy，Redis Sentinel。其中Codis和Twemproxy主要是用于大规模的Redis集群中，也是在Redis官方发布Redis Sentinel之前twitter和豌豆荚提供的开源解决方案。我的业务中数据量并不大，所以搞集群服务反而是浪费机器了。最终在Keepalived和Redis Sentinel之间做了个选择，选择了官方的解决方案Redis Sentinel。

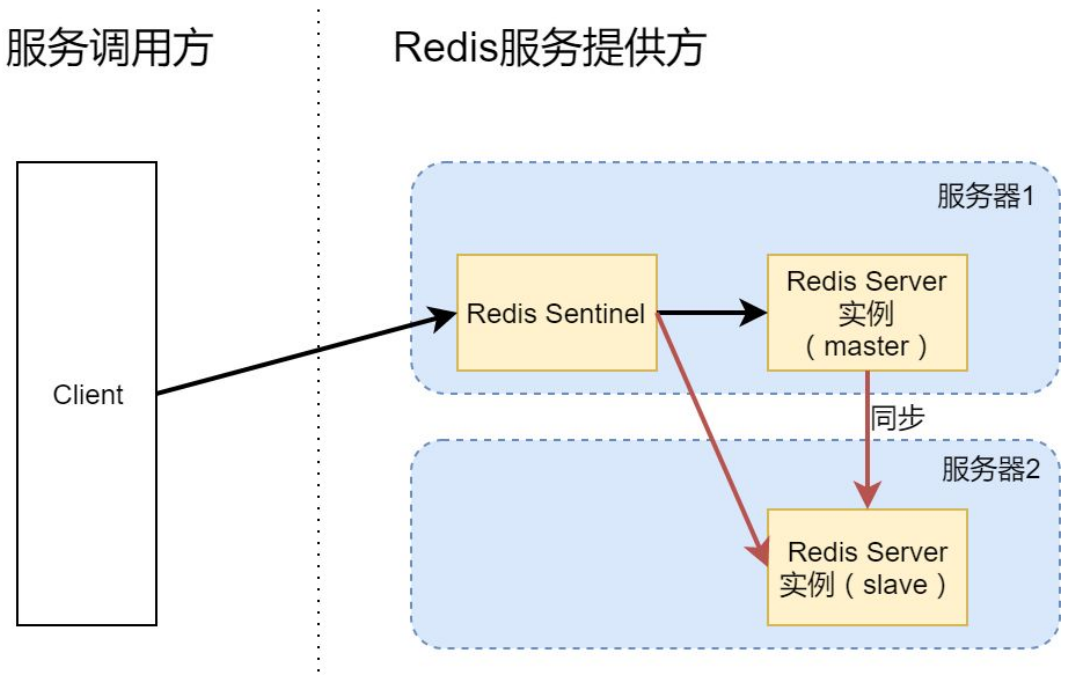
Redis Sentinel可以理解为一个监控Redis Server服务是否正常的进程，并且一旦检测到不正常，可以自动地将备份（slave）Redis Server启用，使得外部用户对Redis服务内部出现的异常无感知。我们按照由简至繁的步骤，搭建一个最小型的高可用的Redis服务。

方案1：单机版Redis Server，无Sentinel



一般情况下，我们搭的个人网站，或者平时做开发时，会起一个单实例的Redis Server。调用方直接连接Redis服务即可，甚至Client和Redis本身就处于同一台服务器上。这种搭配仅适合个人学习娱乐，毕竟这种配置总会有单点故障的问题无法解决。一旦Redis服务进程挂了，或者服务器1停机了，那么服务就不可用了。并且如果没有配置Redis数据持久化的话，Redis内部已经存储的数据也会丢失。

方案2：主从同步Redis Server，单实例Sentinel

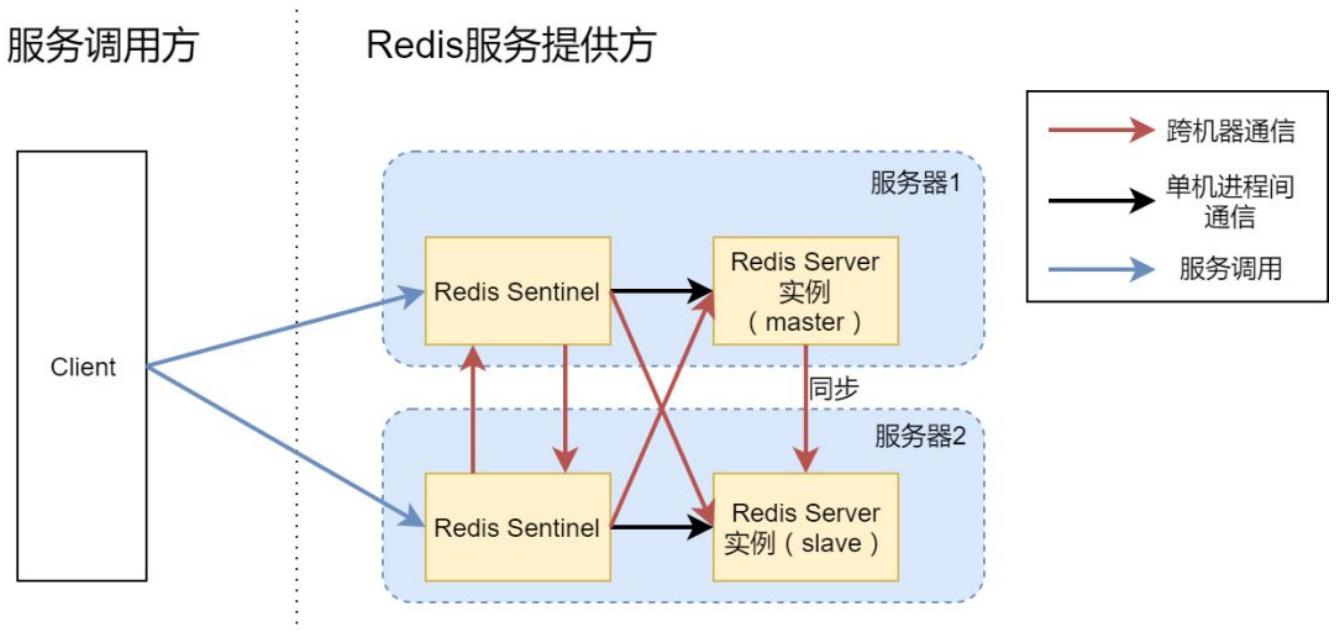


为了实现高可用，解决方案1中所述的单点故障问题，我们必须增加一个备份服务，即在两台服务器上分别各启动一个Redis Server进程，一般情况下由master提供服务，slave只负责同步和备份。与此同时，在额外启动一个Sentinel进程，监控两个Redis Server实例的可用性，以便在master挂掉的时候，及时把slave提升到master的角色继续提供服务，这样就实现了Redis Server的高可用。这基于一个高可用服务设计的依据，即单点故障本身就是个小概率事件，而多个单点同时故障（即master和slave同时挂掉），可以认为是（基本）不可能发生的事件。

对于Redis服务的调用方来说，现在要连接的是Redis Sentinel服务，而不是Redis Server了。常见的调用过程是，client先连接Redis Sentinel并询问目前Redis Server中哪个服务是master，哪些是slave，然后再去连接相应的Redis Server进行操作。当然目前的第三方库一般都已经实现了这一调用过程，不再需要我们手动去实现（例如Nodejs的ioredis，PHP的predis，Golang的go-redis/redis，JAVA的jedis等）。

然而，我们实现了Redis Server服务的主从切换之后，又引入了一个新的问题，即Redis Sentinel本身也是个单点服务，一旦Sentinel进程挂了，那么客户端就没办法链接Sentinel了。所以说，方案2的配置并无法实现高可用性。

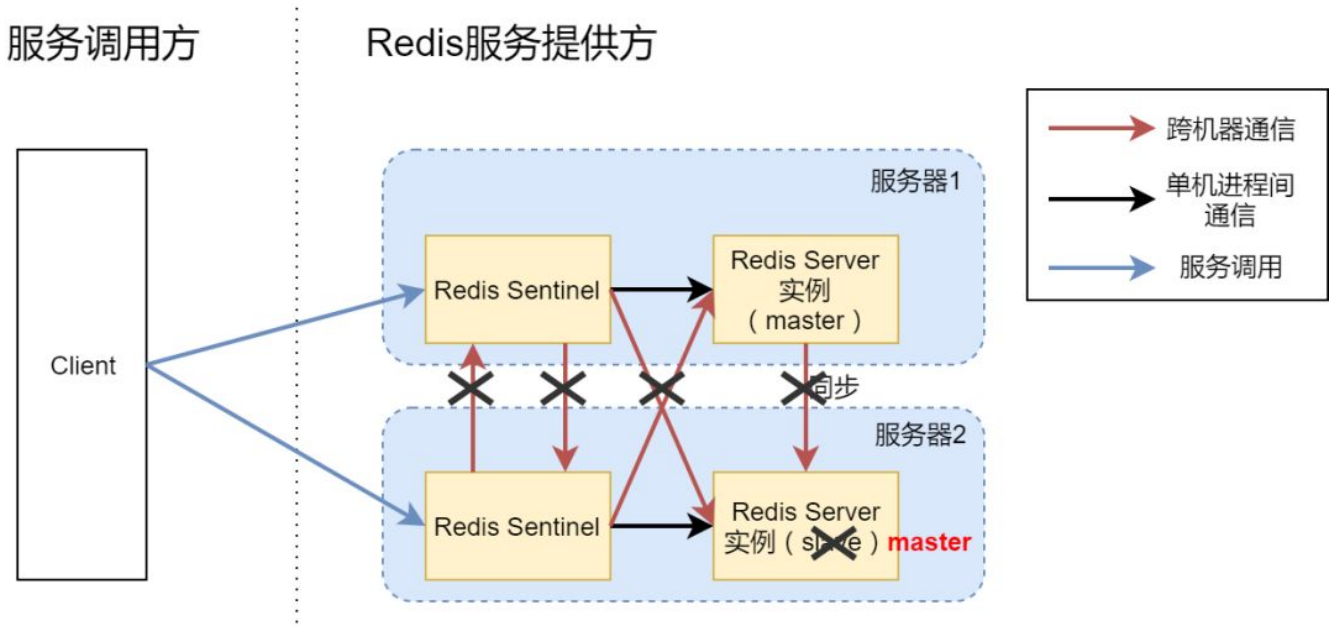
方案3：主从同步Redis Server，双实例Sentinel



为了解决方案2的问题，我们把Redis Sentinel进程也额外启动一份，两个Sentinel进程同时为客户端提供服务发现的功能。对于客户端来说，它可以连接任何一个Redis Sentinel服务，来获取当前Redis Server实例的基本信息。通常情况下，我们会在Client端配置多个Redis Sentinel的链接地址，Client一旦发现某个地址连接不上，会去试图连接其他的Sentinel实例，这当然也不需要我们手动实现，各个开发语言中比较热门的redis连接库都帮我们实现了这个功能。我们预期是：即使其中一个Redis Sentinel挂掉了，还有另外一个Sentinel可以提供服务。

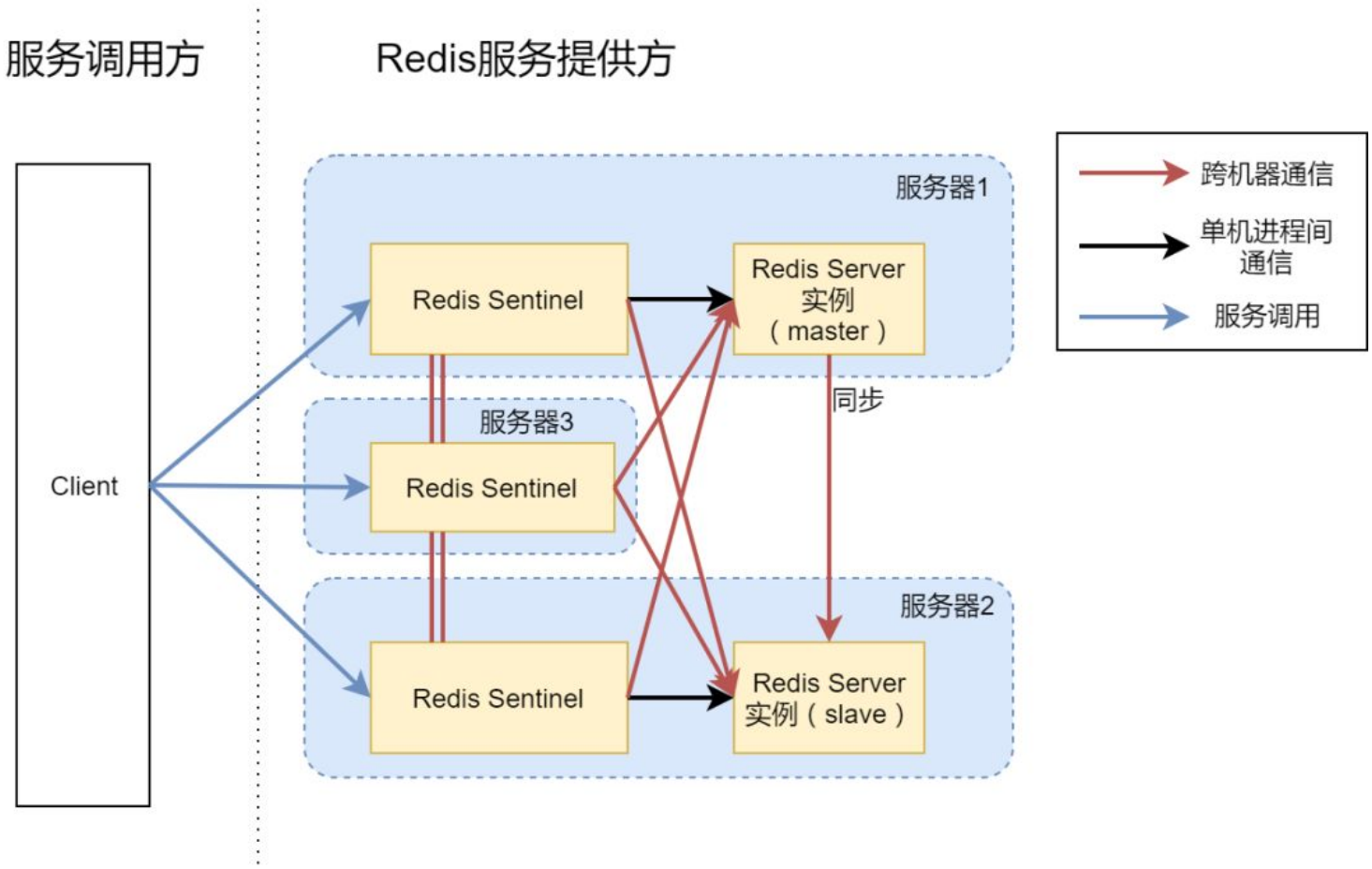
然而，愿景是美好的，现实却是很残酷的。如此架构下，依然无法实现Redis服务的高可用。方案3示意图中，红线部分是两台服务器之间的通信，而我们所设想的异常场景（【异常2】）是，某台服务器整体down机，不妨假设服务器1停机，此时，只剩下服务器2上面的Redis Sentinel和slave Redis Server进程。这时，Sentinel其实是不会将仅剩的slave切换成master继续服务的，也就导致Redis服务不可用，因为Redis的设定是只有当超过50%的Sentinel进程可以连通并投票选取新的master时，才会真正发生主从切换。本例中两个Sentinel只有一个可以连通，等于50%并不在可以主从切换的场景中。

你可能会问，为什么Redis要有这个50%的设定？假设我们允许小于等于50%的Sentinel连通的场景下也可以进行主从切换。试想一下【异常3】，即服务器1和服务器2之间的网络中断，但是服务器本身是可以运行的。如下图所示：



实际上对于服务器2来说，服务器1直接down掉和服务器1网络连不通是一样的效果，反正都是突然就无法进行任何通信了。假设网络中断时我们允许服务器2的Sentinel把slave切换为master，结果就是你现在拥有了两个可以对外提供服务的Redis Server。Client做任何的增删改操作，有可能落在服务器1的Redis上，也有可能落在服务器2的Redis上（取决于Client到底连通的是哪个Sentinel），造成数据混乱。即使后面服务器1和服务器2之间的网络又恢复了，那我们也无法把数据统一了（两份不一样的数据，到底该信任谁呢？），数据一致性完全被破坏。

方案4：主从同步Redis Server，三实例Sentinel



鉴于方案3并没有办法做到高可用，我们最终的版本就是上图所示的方案4了。实际上这就是我们最终搭建的架构。我们引入了服务器3，并且在3上面又搭建起一个Redis Sentinel进程，现在由三个Sentinel进程来管理两个Redis Server实例。这种场景下，不管是单一进程故障、还是单个机器故障、还是某两个机器网络通信故障，都可以继续对外提供Redis服务。

实际上，如果你的机器比较空闲，当然也可以把服务器3上面也开启一个Redis Server，形成1 master + 2 slave的架构，每个数据都有两个备份，可用性会提升一些。当然也并不是slave越多越好，毕竟主从同步也是需要时间成本的。

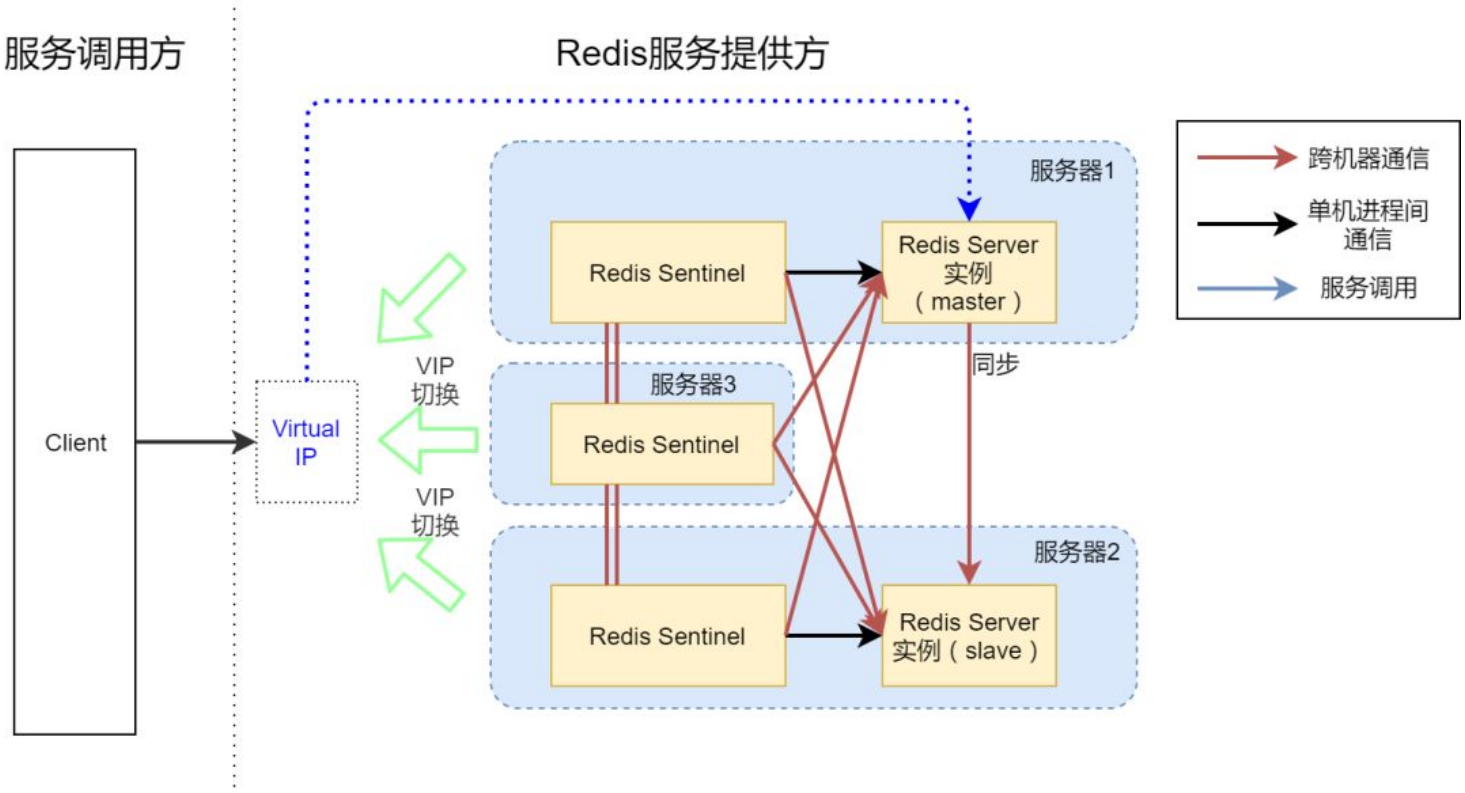
在方案4中，一旦服务器1和其他服务器的通信完全中断，那么服务器2和3会将slave切换为master。对于客户端来说，在这么一瞬间会有2个master提供服务，并且一旦网络恢复了，那么所有在中断期间落在服务器1上的新数据都会丢失。如果想要部分解决这个问题，可以配置Redis Server进程，让其在检测到自己网络有问题的时候，立即停止服务，避免在网络故障期间还有新数据进来（可以参考Redis的min-slaves-to-write和min-slaves-max-lag这两个配置

项）。

至此，我们就用3台机器搭建了一个高可用的Redis服务。其实网上还有更加节省机器的办法，就是把一个Sentinel进程放在Client机器上，而不是服务提供方的机器上。只不过在公司里面，一般服务的提供方和调用方并不来自同一个团队。两个团队共同操作同一个机器，很容易因为沟通问题导致一些误操作，所以出于这种人为因素的考虑，我们还是采用了方案4的架构。并且由于服务器3上面只跑了一个Sentinel进程，对服务器资源消耗并不多，还可以用服务器3来跑一些其他的服务。

易用性：像使用单机版Redis一样使用Redis Sentinel

作为服务的提供方，我们总是会讲到用户体验问题。在上述方案当中始终有一个让Client端用的不是那么舒服的地方。对于单机版Redis，Client端直接连接Redis Server，我们只需要给一个ip和port，Client就可以使用我们的服务了。而改造成Sentinel模式之后，Client不得不采用一些支持Sentinel模式的外部依赖包，并且还要修改自己的Redis连接配置，这对于“矫情”的用户来讲显然是不能接收的。有没有办法还是像在使用单机版的Redis那样，只给Client一个固定的ip和port就可以提供服务呢？



答案当然是肯定的。这可能就要引入虚拟IP（Virtual IP，VIP），如上图所示。我们可以把虚拟IP指向Redis Server master所在的服务器，在发生Redis主从切换的时候，会触发一个回调脚本，回调脚本中将VIP切换至slave所在的服务器。这样对于Client端来说，他仿佛在使用的依然是一个单机版的高可用Redis服务。

结语

搭建任何一个服务，做到“能用”其实是非常简单的，就像我们运行一个单机版的Redis。不过一旦要做到“高可用”，事情就会变得复杂起来。业务中使用了额外的两台服务器，3个Sentinel进程+1个Slave进程，只是为了保证在那小概率的事故中依然做到服务可用。在实际业务中我们还启用了supervisor做进程监控，一旦进程意外退出，会自动尝试重新启动。

> 作者：HorstXu
>
> 来源：http://www.cnblogs.com/xuning/p/8464625.html