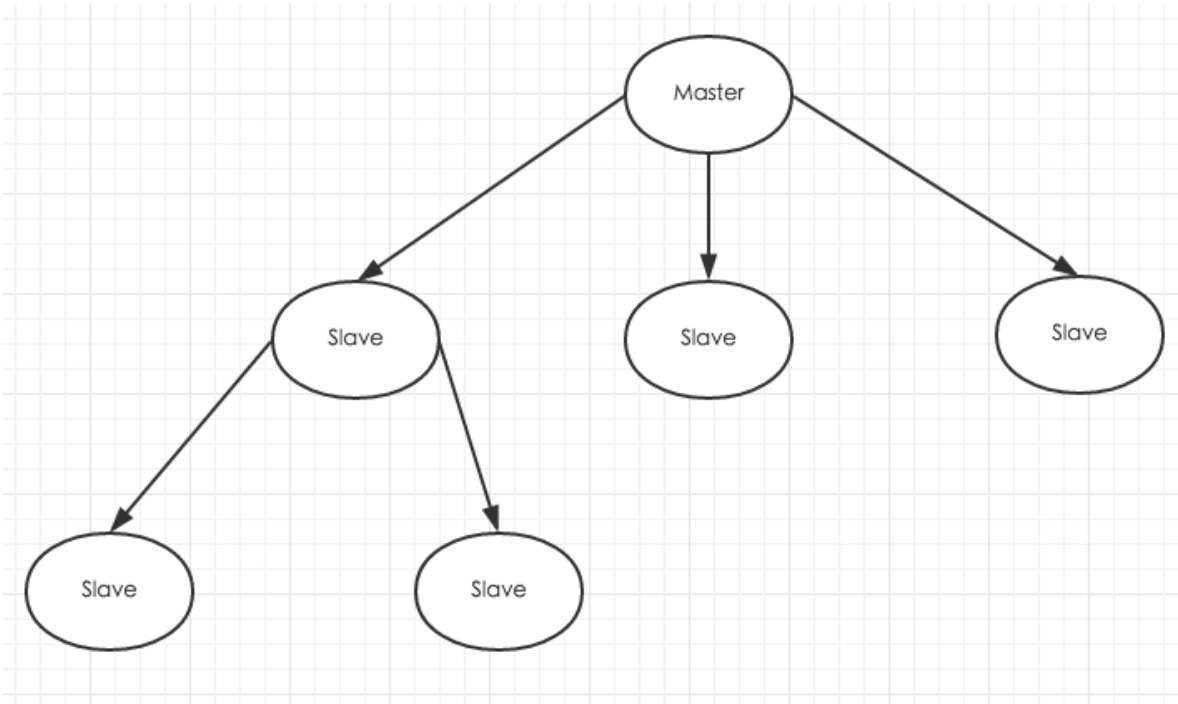
和Mysql主从复制的原因一样，Redis虽然读取写入的速度都特别快，但是也会产生读压力特别大的情况。为了分担读压力，Redis支持主从复制，Redis的主从结构可以采用一主多从或者级联结构，Redis主从复制可以根据是否是全量分为全量同步和增量同步。下图为级联结构。



全量同步 Redis全量复制一般发生在Slave初始化阶段，这时Slave需要将Master上的所有数据都复制一份。具体步骤如下：

- 从服务器连接主服务器，发送SYNC命令；
- 主服务器接收到SYNC命名后，开始执行BGSAVE命令生成RDB文件并使用缓冲区记录此后执行的所有写命令；
- 主服务器BGSAVE执行完后，向所有从服务器发送快照文件，并在发送期间继续记录被执行的写命令；
- 从服务器收到快照文件后丢弃所有旧数据，载入收到的快照；
- 主服务器快照发送完毕后开始向从服务器发送缓冲区中的写命令；
- 从服务器完成对快照的载入，开始接收命令请求，并执行来自主服务器缓冲区的写命令；

完成上面几个步骤后就完成了从服务器数据初始化的所有操作，从服务器此时可以接收来自用户的读请求。

增量同步 Redis增量复制是指Slave初始化后开始正常工作时主服务器发生的写操作同步到从服务器的过程。 增量复制的过程主要是主服务器每执行一个写命令就会向从服务器发送相同的写命令，从服务器接收并执行收到的写命令。

Redis主从同步策略 主从刚刚连接的时候，进行全量同步；全同步结束后，进行增量同步。当然，如果有需要，slave 在任何时候都可以发起全量同步。redis 策略是，无论如何，首先会尝试进行增量同步，如不成功，要求从机进行全量同步。

注意点 如果多个Slave断线了，需要重启的时候，因为只要Slave启动，就会发送sync请求和主机全量同步，当多个同时出现的时候，可能会导致Master IO剧增宕机。

Redis主从复制的配置十分简单，它可以使从服务器是主服务器的完全拷贝。需要清除Redis主从复制的几点重要内容：

1）Redis使用异步复制。但从Redis 2.8开始，从服务器会周期性的应答从复制流中处理的数据量。``2）一个主服务器可以有多个从服务器。``3）从服务器也可以接受其他从服务器的连接。除了多个从服务器连接到一个主服务器之外，多个从服务器也可以连接到一个从服务器上，形成一个`` ``图状结构。``4）Redis主从复制不阻塞主服务器端。也就是说当若干个从服务器在进行初始同步时，主服务器仍然可以处理请求。``5）主从复制也不阻塞从服务器端。当从服务器进行初始同步时，它使用旧版本的数据来应对查询请求，假设你在redis.conf配置文件是这么配置的。`` ``否则的话，你可以配置当复制流关闭时让从服务器给客户端返回一个错误。但是，当初始同步完成后，需要删除旧的数据集和加载新的数据集，在`` ``这个短暂的时间内，从服务器会阻塞连接进来的请求。``6）主从复制可以用来增强扩展性，使用多个从服务器来处理只读的请求（比如，繁重的排序操作可以放到从服务器去做），也可以简单的用来做数据冗余。``7）使用主从复制可以为主服务器免除把数据写入磁盘的消耗：在主服务器的redis.conf文件中配置“避免保存”（注释掉所有“保存“命令），然后连接一个配`` ``置为“进行保存”的从服务器即可。但是这个配置要确保主服务器不会自动重启（要获得更多信息请阅读下一段）

1）Redis使用异步复制。但从Redis 2.8开始，从服务器会周期性的应答从复制流中处理的数据量。
2）一个主服务器可以有多个从服务器。
3）从服务器也可以接受其他从服务器的连接。除了多个从服务器连接到一个主服务器之外，多个从服务器也可以连接到一　个从服务器上，形成一个图状结构。
4）Redis主从复制不阻塞主服务器端。也就是说当若干个从服务器在进行初始同步时，主服务器仍然可以处理请求。
5）主从复制也不阻塞从服务器端。当从服务器进行初始同步时，它使用旧版本的数据来应对查询请求，假设你在redis.conf配置文件是这么配置的。

否则的话，你可以配置当复制流关闭时让从服务器给客户端返回一个错误。但是，当初始同步完成后，需要删除旧的数据集和加载新的数据集，在这个短暂的时间内，从服务器会阻塞连接进来的请求。
6）主从复制可以用来增强扩展性，使用多个从服务器来处理只读的请求（比如，繁重的排序操作可以放到从服务器去做），也可以简单的用来做数据冗余。
7）使用主从复制可以为主服务器免除把数据写入磁盘的消耗：在主服务器的redis.conf文件中配置“避免保存”（注释掉所有“保存“命令），然后连接一个配置为“进行保存”的从服务器即可。但是这个配置要确保主服务器不会自动重启（要获得更多信息请阅读下一段）

主从复制的一些特点：

1）采用异步复制；``2）一个主redis可以含有多个从redis；``3）每个从redis可以接收来自其他从redis服务器的连接；``4）主从复制对于主redis服务器来说是非阻塞的，这意味着当从服务器在进行主从复制同步过程中，主redis仍然可以处理外界的访问请求；``5）主从复制对于从redis服务器来说也是非阻塞的，这意味着，即使从redis在进行主从复制过程中也可以接受外界的查询请求，只不过这时候从redis返回的是以前老的数据，`` ``如果你不想这样，那么在启动redis时，可以在配置文件中进行设置，那么从redis在复制同步过程中来自外界的查询请求都会返回错误给客户端；（虽然说主从复制过程中`` ``对于从redis是非阻塞的，但是当从redis从主redis同步过来最新的数据后还需要将新数据加载到内存中，在加载到内存的过程中是阻塞的，在这段时间内的请求将会被阻，`` ``但是即使对于大数据集，加载到内存的时间也是比较多的）；``6）主从复制提高了redis服务的扩展性，避免单个redis服务器的读写访问压力过大的问题，同时也可以给为数据备份及冗余提供一种解决方案；``7）为了编码主redis服务器写磁盘压力带来的开销，可以配置让主redis不在将数据持久化到磁盘，而是通过连接让一个配置的从redis服务器及时的将相关数据持久化到磁盘，`` ``不过这样会存在一个问题，就是主redis服务器一旦重启，因为主redis服务器数据为空，这时候通过主从同步可能导致从redis服务器上的数据也被清空；

1）采用异步复制；
2）一个主redis可以含有多个从redis；
3）每个从redis可以接收来自其他从redis服务器的连接；
4）主从复制对于主redis服务器来说是非阻塞的，这意味着当从服务器在进行主从复制同步过程中，主redis仍然可以处理外界的访问请求；
5）主从复制对于从redis服务器来说也是非阻塞的，这意味着，即使从redis在进行主从复制过程中也可以接受外界的查询请求，只不过这时候从redis返回的是以前老的数据，
如果你不想这样，那么在启动redis时，可以在配置文件中进行设置，那么从redis在复制同步过程中来自外界的查询请求都会返回错误给客户端；（虽然说主从复制过程中
对于从redis是非阻塞的，但是当从redis从主redis同步过来最新的数据后还需要将新数据加载到内存中，在加载到内存的过程中是阻塞的，在这段时间内的请求将会被阻，
但是即使对于大数据集，加载到内存的时间也是比较多的）；
6）主从复制提高了redis服务的扩展性，避免单个redis服务器的读写访问压力过大的问题，同时也可以给为数据备份及冗余提供一种解决方案；
7）为了编码主redis服务器写磁盘压力带来的开销，可以配置让主redis不在将数据持久化到磁盘，而是通过连接让一个配置的从redis服务器及时的将相关数据持久化到磁盘，
不过这样会存在一个问题，就是主redis服务器一旦重启，因为主redis服务器数据为空，这时候通过主从同步可能导致从redis服务器上的数据也被清空；

Redis大概主从同步是怎么实现的?

全量同步：``master服务器会开启一个后台进程用于将redis中的数据生成一个rdb文件，与此同时，服务器会缓存所有接收到的来自客户端的写命令（包含增、删、改），当后台保存进程``处理完毕后，会将该rdb文件传递给slave服务器，而slave服务器会将rdb文件保存在磁盘并通过读取该文件将数据加载到内存，在此之后master服务器会将在此期间缓存的``命令通过redis传输协议发送给slave服务器，然后slave服务器将这些命令依次作用于自己本地的数据集上最终达到数据的一致性。 部分同步：``从redis 2.8版本以前，并不支持部分同步，当主从服务器之间的连接断掉之后，master服务器和slave服务器之间都是进行全量数据同步，但是从redis 2.8开``始，即使主从连接中途断掉，也不需要进行全量同步，因为从这个版本开始融入了部分同步的概念。部分同步的实现依赖于在master服务器内存中给每个slave服务器维护了``一份同步日志和同步标识，每个slave服务器在跟master服务器进行同步时都会携带自己的同步标识和上次同步的最后位置。当主从连接断掉之后，slave服务器隔断时间``（默认1s）主动尝试和master服务器进行连接，如果从服务器携带的偏移量标识还在master服务器上的同步备份日志中，那么就从slave发送的偏移量开始继续上次的同步``操作，如果slave发送的偏移量已经不再master的同步备份日志中（可能由于主从之间断掉的时间比较长或者在断掉的短暂时间内master服务器接收到大量的写操作），则``必须进行一次全量更新。在部分同步过程中，master会将本地记录的同步备份日志中记录的指令依次发送给slave服务器从而达到数据一致。

主从同步中需要注意几个问题

1）在上面的全量同步过程中，master会将数据保存在rdb文件中然后发送给slave服务器，但是如果master上的磁盘空间有效怎么办呢？那么此时全部同步对于master来说``将是一份十分有压力的操作了。此时可以通过无盘复制来达到目的，由master直接开启一个socket将rdb文件发送给slave服务器。（无盘复制一般应用在磁盘空间有限但是网``络状态良好的情况下） 2）主从复制结构，一般slave服务器不能进行写操作，但是这不是死的，之所以这样是为了更容易的保证主和各个从之间数据的一致性，如果slave服务器上数据进行了修改，``那么要保证所有主从服务器都能一致，可能在结构上和处理逻辑上更为负责。不过你也可以通过配置文件让从服务器支持写操作。（不过所带来的影响还得自己承担哦。。。） 3）主从服务器之间会定期进行通话，但是如果master上设置了密码，那么如果不给slave设置密码就会导致slave不能跟master进行任何操作，所以如果你的master服务器``上有密码，那么也给slave相应的设置一下密码吧（通过设置配置文件中的masterauth）； 4）关于slave服务器上过期键的处理，由master服务器负责键的过期删除处理，然后将相关删除命令已数据同步的方式同步给slave服务器，slave服务器根据删除命令删除``本地的key。

当主服务器不进行持久化时复制的安全性

在进行主从复制设置时，强烈建议在主服务器上开启持久化，当不能这么做时，比如考虑到延迟的问题，应该将实例配置为避免自动重启。 为什么不持久化的主服务器自动重启非常危险呢？``为了更好的理解这个问题，看下面这个失败的例子，其中主服务器和从服务器中数据库都被删除了。 设置节点A为主服务器，关闭持久化，节点B和C从节点A复制数据。``这时出现了一个崩溃，但Redis具有自动重启系统，重启了进程，因为关闭了持久化，节点重启后只有一个空的数据集。``节点B和C从节点A进行复制，现在节点A是空的，所以节点B和C上的复制数据也会被删除。``当在高可用系统中使用Redis Sentinel，关闭了主服务器的持久化，并且允许自动重启，这种情况是很危险的。``比如主服务器可能在很短的时间就完成了重启，以至于Sentinel都无法检测到这次失败，那么上面说的这种失败的情况就发生了。 如果数据比较重要，并且在使用主从复制时关闭了主服务器持久化功能的场景中，都应该禁止实例自动重启。

Redis主从复制是如何工作的

如果设置了一个从服务器，在连接时它发送了一个SYNC命令，不管它是第一次连接还是再次连接都没有关系。然后主服务器开始后台存储，并且开始缓存新连接进来的修改数据的命令。当后台存储完成后，主服务器把数据文件发送到从服务器，``从服务器将其保存在磁盘上，然后加载到内存中。然后主服务器把刚才缓存的命令发送到从服务器。这是作为命令流来完成的，并且``和Redis协议本身格式相同。你可以通过telnet自己尝试一下。在Redis服务器工作时连接到Redis端口，发送SYNC命令，会看到一个批量的传输，并且主服务器接收``的每一个命令都会通过telnet会话重新发送一遍。当主从服务器之间的连接由于某些原因断开时，从服务器可以自动进行重连接。当有多个从服务器同时请求同步时，主服务器只进行一个后台存储。当连接断开又重新连上之后，一般都会进行一个完整的重新同步，但是从Redis2.8开始，只重新同步一部分也可以。

部分重新同步

从Redis 2.8开始，如果遭遇连接断开，重新连接之后可以从中断处继续进行复制，而不必重新同步。它的工作原理是这样：``主服务器端为复制流维护一个内存缓冲区（``in``-memory backlog）。主从服务器都维护一个复制偏移量（replication offset）和master run ``id，``当连接断开时，从服务器会重新连接上主服务器，然后请求继续复制，假如主从服务器的两个master run ``id``相同，并且指定的偏移量在内存缓冲``区中还有效，复制就会从上次中断的点开始继续。如果其中一个条件不满足，就会进行完全重新同步（在2.8版本之前就是直接进行完全重新同步）。``因为主运行``id``不保存在磁盘中，如果从服务器重启了的话就只能进行完全同步了。部分重新同步这个新特性内部使用PSYNC命令，旧的实现中使用SYNC命令。Redis2.8版本可以检测出它所连接的服务器是否支持PSYNC命令，不支持的``话使用SYNC命令。

无磁盘复制

通常来讲，一个完全重新同步需要在磁盘上创建一个RDB文件，然后加载这个文件以便为从服务器发送数据。如果使用比较低速的磁盘，这种操作会给主服务器带来较大的压力。Redis从2.8.18版本开始尝试支持无磁盘的复制。``使用这种设置时，子进程直接将RDB通过网络发送给从服务器，不使用磁盘作为中间存储。

配置

主从复制的配置十分简单：把下面这行加入到从服务器的配置文件中即可。``slaveof 192.168.1.1 6379``当然你需要把其中的192.168.1.1 6379替换为你自己的主服务器IP（或者主机名``hostname``）和端口。另外你可以调用SLAVEOF命令，``主服务器就会开始与从服务器同步。关于部分重新同步，还有一些针对复制内存缓冲区的优化参数。查看Redis介质中的Redis.conf示例获得更多信息。使用repl-diskless-``sync``配置参数来启动无磁盘复制。使用repl-diskless-``sync``-delay 参数来配置传输开始的延迟时间，以便等待``更多的从服务器连接上来。查看Redis介质中的Redis.conf示例获得更多信息。

只读从服务器

从Redis 2.6开始，从服务器支持只读模式，并且是默认模式。这个行为是由Redis.conf文件中的slave-``read``-only 参数控制的，``可以在运行中通过CONFIG SET来启用或者禁用。只读的从服务器会拒绝所有写命令，所以对从服务器不会有误写操作。但这不表示可以把从服务器实例暴露在危险的网络环境下，``因为像DEBUG或者CONFIG这样的管理命令还是可以运行的。不过你可以通过使用rename-``command``命令来为这些命令改名来增加安全性。你可能想知道为什么只读限制还可以被还原，使得从服务器还可以进行写操作。虽然当主从服务器进行重新同步或者从服务器重启后，``这些写操作都会失效，还是有一些使用场景会想从服务器中写入临时数据的，但将来这个特性可能会被去掉。

限制有N个以上从服务器才允许写入

从Redis 2.8版本开始，可以配置主服务器连接N个以上从服务器才允许对主服务器进行写操作。但是，因为Redis使用的是异步主从复制，``没办法确保从服务器确实收到了要写入的数据，所以还是有一定的数据丢失的可能性。这一特性的工作原理如下：``1）从服务器每秒钟``ping``一次主服务器，确认处理的复制流数量。``2）主服务器记住每个从服务器最近一次``ping``的时间。``3）用户可以配置最少要有N个服务器有小于M秒的确认延迟。``4）如果有N个以上从服务器，并且确认延迟小于M秒，主服务器接受写操作。还可以把这看做是CAP原则（一致性，可用性，分区容错性）不严格的一致性实现，虽然不能百分百确保一致性，但至少保证了丢失的数据不会超过M秒内的数据量。如果条件不满足，主服务器会拒绝写操作并返回一个错误。``1）min-slaves-to-write（最小从服务器数）``2）min-slaves-max-lag（从服务器最大确认延迟）

====== 通过redis实现服务器崩溃等数据恢复 ====== 由于redis存储在内存中且提供一般编程语言常用的数据结构存储类型，所以经常被用于做服务器崩溃宕机的数据恢复处理。服务器可以在某些指定过程中将需要保存的数据以json对象等方式存储到redis中，也就是我们常说的快照，当服务器运行时读取redis来判断是否有待需要恢复数据继续处理的业务。当一次业务处理结束后再删除redis的数据即可。**redis提供两种将内存数据导出到硬盘实现数据备份的方法**：

1）RDB方式(默认) RDB方式的持久化是通过快照（snapshotting）完成的，当符合一定条件时Redis会自动将内存中的所有数据进行快照并存储在硬盘上。进行快照的条件可以由用户在配置文件中自定义，由两个参数构成：时间和改动的键的个数。当在指定的时间内被更改的键的个数大于指定的数值时就会进行快照。RDB是redis默认采用的持久化方式，在配置文件中已经预置了3个条件： save 900 1 #900秒内有至少1个键被更改则进行快照 save 300 10 #300秒内有至少10个键被更改则进行快照 save 60 10000 #60秒内有至少10000个键被更改则进行快照

可以存在多个条件，条件之间是"或"的关系，只要满足其中一个条件，就会进行快照。 如果想要禁用自动快照，只需要将所有的save参数删除即可。 Redis默认会将快照文件存储在当前目录(可CONFIG GET dir来查看)的dump.rdb文件中，可以通过配置dir和dbfilename两个参数分别指定快照文件的存储路径和文件名。

Redis实现快照的过程

- Redis使用fork函数复制一份当前进程（父进程）的副本（子进程）；
- 父进程继续接收并处理客户端发来的命令，而子进程开始将内存中的数据写入硬盘中的临时文件；
- 当子进程写入完所有数据后会用该临时文件替换旧的RDB文件，至此一次快照操作完成。
- 在执行fork的时候操作系统（类Unix操作系统）会使用写时复制（copy-on-write）策略，即fork函数发生的一刻父子进程共享同一内存数据，当父进程要更改其中某片数据时（如执行一个写命令），操作系统会将该片数据复制一份以保证子进程的数据不受影响，所以新的RDB文件存储的是执行fork一刻的内存数据。

Redis在进行快照的过程中不会修改RDB文件，只有快照结束后才会将旧的文件替换成新的，也就是说任何时候RDB文件都是完整的。这使得我们可以通过定时备份RDB文件来实 现Redis数据库备份。RDB文件是经过压缩（可以配置rdbcompression参数以禁用压缩节省CPU占用）的二进制格式，所以占用的空间会小于内存中的数据大小，更加利于传输。

除了自动快照，还可以手动发送SAVE或BGSAVE命令让Redis执行快照，两个命令的区别在于，前者是由主进程进行快照操作，会阻塞住其他请求，后者会通过fork子进程进行快照操作。 Redis启动后会读取RDB快照文件，将数据从硬盘载入到内存。根据数据量大小与结构和服务器性能不同，这个时间也不同。通常将一个记录一千万个字符串类型键、大小为1GB的快照文件载入到内 存中需要花费20～30秒钟。 通过RDB方式实现持久化，一旦Redis异常退出，就会丢失最后一次快照以后更改的所有数据。这就需要开发者根据具体的应用场合，通过组合设置自动快照条件的方式来将可能发生的数据损失控制在能够接受的范围。如果数据很重要以至于无法承受任何损失，则可以考虑使用AOF方式进行持久化。

2）AOF方式 默认情况下Redis没有开启AOF(append only file)方式的持久化，可以在redis.conf中通过appendonly参数开启：

appendonly yes 在启动时Redis会逐个执行AOF文件中的命令来将硬盘中的数据载入到内存中，载入的速度相较RDB会慢一些

开启AOF持久化后每执行一条会更改Redis中的数据的命令，Redis就会将该命令写入硬盘中的AOF文件。AOF文件的保存位置和RDB文件的位置相同，都是通过dir参数设置的，默认的文件名是appendonly.aof，可以通过appendfilename参数修改：

appendfilename appendonly.aof 配置redis自动重写AOF文件的条件

auto-aof-rewrite-percentage 100 # 当目前的AOF文件大小超过上一次重写时的AOF文件大小的百分之多少时会再次进行重写，如果之前没有重写过，则以启动时的AOF文件大小为依据

auto-aof-rewrite-min-size 64mb # 允许重写的最小AOF文件大小 配置写入AOF文件后，要求系统刷新硬盘缓存的机制

appendfsync always # 每次执行写入都会执行同步，最安全也最慢 appendfsync everysec # 每秒执行一次同步操作

appendfsync no # 不主动进行同步操作，而是完全交由操作系统来做（即每30秒一次），最快也最不安全

Redis允许同时开启AOF和RDB，既保证了数据安全又使得进行备份等操作十分容易。此时重新启动Redis后Redis会使用AOF文件来恢复数据，因为AOF方式的持久化可能丢失的数据更少