# Redis主从复制原理

一、什么是redis主从复制？

主从复制，当用户往Master端写入数据时，通过Redis Sync机制将数据文件发送至Slave，Slave也会执行相同的操作确保数据一致；且实现Redis的主从复制非常简单。

二、redis主从复制特点

1、同一个Master可以拥有多个Slaves。

2、Master下的Slave还可以接受同一架构中其它slave的链接与同步请求，实现数据的级联复制，即Master->Slave->Slave模式；

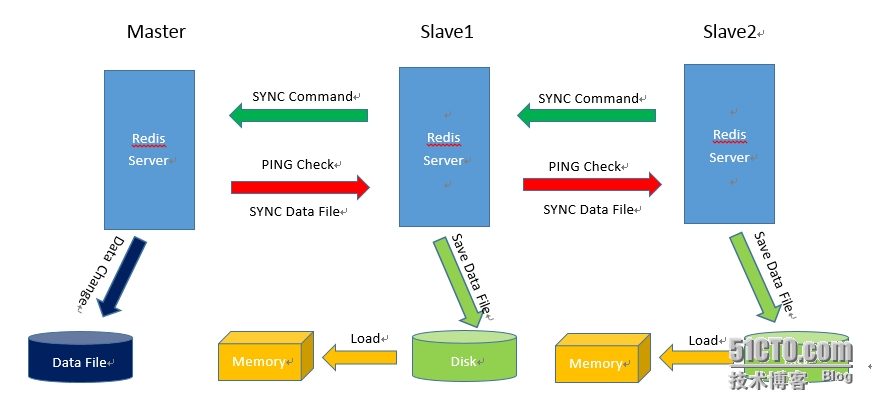
3、Master以非阻塞的方式同步数据至slave，这将意味着Master会继续处理一个或多个slave的读写请求；

4、Slave端同步数据也可以修改为非阻塞的方式，当slave在执行新的同步时，它仍可以用旧的数据信息来提供查询；否则，当slave与master失去联系时，slave会返回一个错误给客户端；

5、主从复制具有可扩展性，即多个slave专门提供只读查询与数据的冗余，Master端专门提供写操作；

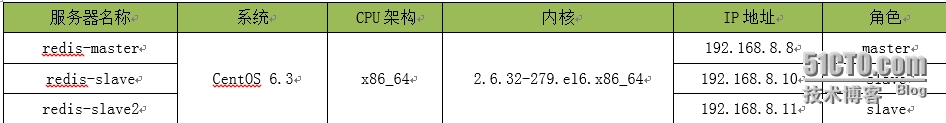
6、通过配置禁用Master数据持久化机制，将其数据持久化操作交给Slaves完成，避免在Master中要有独立的进程来完成此操作。

三、redis主从复制原理

[](http://s3.51cto.com/wyfs02/M02/38/6A/wKiom1Ozz5OThc6NAAGUIzDDlQs366.jpg)

当启动一个Slave进程后，它会向Master发送一个SYNC Command，请求同步连接。无论是第一次连接还是重新连接，Master都会启动一个后台进程，将数据快照保存到数据文件中，同时Master会记录所有修改数据的命令并缓存在数据文件中。后台进程完成缓存操作后，Master就发送数据文件给Slave，Slave端将数据文件保存到硬盘上，然后将其在加载到内存中，接着Master就会将所有修改数据的操作，将其发送给Slave端。若Slave出现故障导致宕机，恢复正常后会自动重新连接，Master收到Slave的连接后，将其完整的数据文件发送给Slave，如果Mater同时收到多个Slave发来的同步请求，Master只会在后台启动一个进程保存数据文件，然后将其发送给所有的Slave，确保Slave正常。

四、服务器资源列表

[](http://s3.51cto.com/wyfs02/M01/38/6A/wKiom1Ozz7PilaRGAAD2EbEmDy4850.jpg)  
五、redis主从复制实现细节

1. 初次同步步骤简略
2. Slave服务器连接到Master服务器
3. Slave服务器发送SYNC指令/请求
4. Master服务器备份数据库到.rdb文件
5. Master服务器吧.rdb文件传输给Slave服务器
6. Slave服务器把.rdb文件导入导数据库
7. 增量同步

当Master服务器有数据更新的时候，会立刻同步到所有的Slave服务器。

当我们在Master服务器增减数据的时候，就会触发replicationFeedSlaves()。

该函数的主要作用就是把用户执行的命令发送到所有的Slave服务器，让Slave服务器执行，这样就实现了同步功能。

六、配置过程

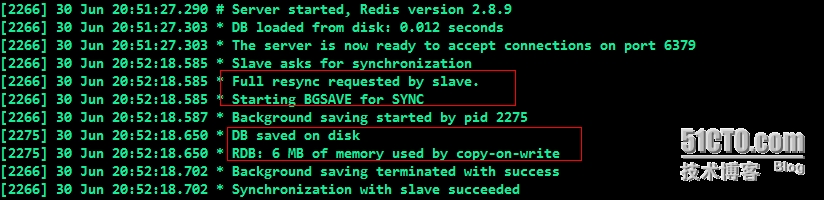
关于Redis的安装与配置这里便不操作，想了解的朋友请阅读：高性能数据库缓存之redis（一）[http://cfwlxf.blog.51cto.com/3966339/1423106](http://cfwlxf.blog.51cto.com/3966339/1423106" \t "_blank)

### 3、1 Master端操作如下：

#### 运行redis服务

1. [root@redis\_master sh]# redis-server/etc/redis/redis.conf

**查询redis运行日志**

[](http://s3.51cto.com/wyfs02/M01/38/6A/wKioL1Ozz6bw05VBAAJdwyDX2eg504.jpg)

##通过阅读日志文件输出的一些信息，可以看出Master与Slave建立连接时，需要执行的会话机制：加载数据文件至硬盘，用时0.012秒，可想而知速度是多么的快，当然得依据数据的大小去评测；服务连接至6379端口，收到Slave同步连接请求，开启“BGSAVE”同步等；

清除Master端数据库中所有Key

1. [root@redis\_master sh]# redis-cli
2. 127.0.0.1:6379> FLUSHALL
3. OK
4. 127.0.0.1:6379> keys \*
5. (empty list or **set**)

### 3、2 Slave端操作如下：

[root@redis\_slave ~]# vim/etc/redis/redis.conf

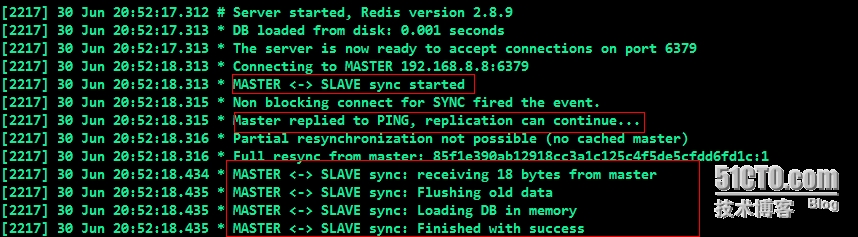
#添加Master端的IP与端口

1. # slaveof <masterip><masterport>
2. slaveof 192.168.8.8 6379

#### 运行redis

1. [root@redis\_slave ~]# redis-server/etc/redis/redis.conf

#### 查询Slave运行日志

[](http://s3.51cto.com/wyfs02/M02/38/6A/wKioL1Ozz7vRolYwAAOCbyhb-MU029.jpg)

##分析redis日志，可以看出Slave与Master建立连接，数据同步的过程；如：发送SYNC命令，与Master端192.168.8.8:6379建立连接，然后Slave sync started；随后Master发送PING命令检查Slave的存活状态，复制被继续….

#### 查询数据库中的所有key

1. [root@redis\_slave ~]# redis-cli
2. 127.0.0.1:6379> keys \*
3. (empty list or **set**)

### 3、3 slave2端操作如下：

[root@redis\_slave2 ~]# vim/etc/redis/redis.conf

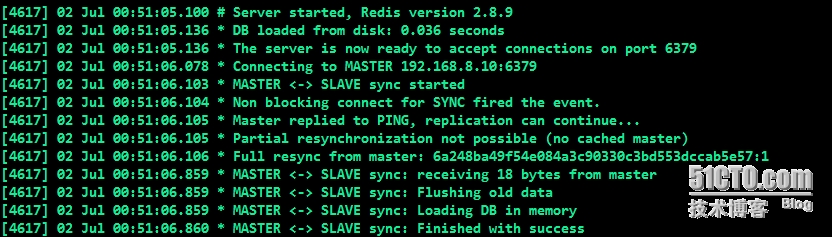
#添加Slave端的IP与端口，实现级联复制；

1. # slaveof <masterip><masterport>
2. slaveof 192.168.8.10 6379

**#运行redis服务**

1. [root@redis\_slave2 ~]# redis-server/etc/redis/redis.conf

#### 查询redis运行日志

[](http://s3.51cto.com/wyfs02/M00/38/6B/wKiom1Oz0AGRPpzyAAOAB3zsl1Q226.jpg)

##结果与Slave1类似，只不过Slave2与Slave1（192.168.8.10:6379）建立连接，同步数据；Master的级联复制便是这样，Master->Slave1->Slave2;

#### #查询数据库的所有key

[root@redis\_slave2 ~]# redis-cli

127.0.0.1:6379> keys \*

(empty list or set)

### 3、4 master端操作如下：

1. [root@redis\_master sh]# redis-cli
2. 127.0.0.1:6379> MSET ID 1005 NAMEMariaDB City BeiJing
3. OK
4. 127.0.0.1:6379> MGET ID NAME City
5. 1) "1005"
6. 2) "MariaDB"
7. 3) "BeiJing"
8. 127.0.0.1:6379> keys \*
9. 1) "NAME"
10. 2) "ID"
11. 3) "City"

### 3、5 客户端验证同步结果

#### slave1端验证

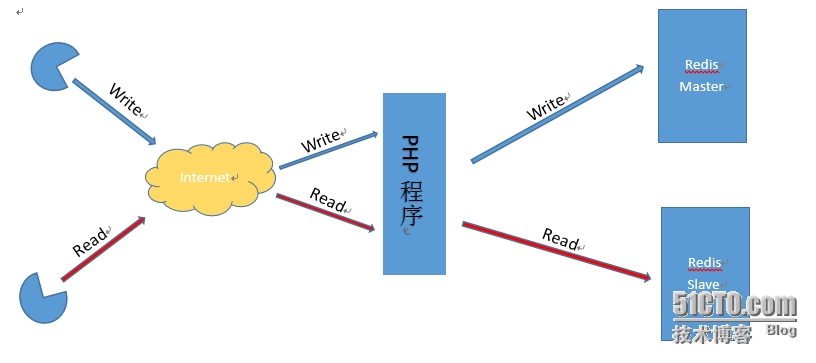
1. [root@redis\_slave ~]# redis-cli
2. 127.0.0.1:6379> auth !@#aedf
3. 127.0.0.1:6379> keys \*
4. 1) "City"
5. 2) "NAME"
6. 3) "ID"
7. 127.0.0.1:6379> MGET ID NAME City
8. 1) "1005"
9. 2) "MariaDB"
10. 3) "BeiJing"

**slave2端验证**

1. [root@redis\_slave2 ~]# redis-cli
2. 127.0.0.1:6379> keys \*
3. 1) "ID"
4. 2) "NAME"
5. 3) "City"
6. 127.0.0.1:6379> MGET ID NAME City
7. 1) "1005"
8. 2) "MariaDB"
9. 3) "BeiJing"

### 四、Master write，Slave read机制

Redis的主从复制，通过程序实现数据的读写分离，让Master负责处理写请求，Slave负责处理读请求；通过扩展Slave处理更多的并发请求，减轻Master端的负载，如下图：

[](http://s3.51cto.com/wyfs02/M02/38/6B/wKioL1Ozz-3yEjbBAAEZvvETcp8136.jpg)

此图画得比较简易，展示了实现Redis读写分离的过程，通过判断用户读写请求，将write请求发送给Redis Master处理，Read请求发送给Redis Slave处理，文章中的不足之处，欢迎大家指点。

## 五．Redis持久化出现的问题

Redis的持久化方式有两种：RDB和AOF

RDB就是快照存储，如设置save 3600 1000表示当之前一次快照操作3600秒即1小时后，如果有超过1000个key改动就进行快照操作产生dump.rdb文件，每次到快照时间就更新文件。

AOF不同于RDB，它只存储所有的写入操作，该方式有分为两个步骤：1、fsync是将内存中的写入操作写入aof文件 2、rewrite则是将写入操作合并，如：set aa 1；set aa 2 两个写操作在rewrite后就变成一个操作set aa 2。

问题发现：

1、只是用RDB方式存储数据时

在大数据量的时候，只采用RDB存储数据，当进行RDB操作的同时，会导致redis的set操作非常的耗时，这就是大数据量时redis的持久化服务暂停现象。

而一般我们为了保证数据的容错性，要求频繁的快照（即频繁的执行RDB操作），而这时若出现问题则是不能容忍的。

2、AOF和RDB方式结合

AOF在重写的时候会占用大量的CPU和内存资源，导致服务器出现短暂的停止现象。问题是配置项no-appendfsync-on-rewrite默认在rewrite操作时进行fsync操作，当设置为yes—表示不进行同步，则set和get都没有延迟

那不同步是否会造成数据的丢失呢？不会，redis会将新的写入操作放在内存，当rewrite操作完成后将新的写入操作直接挂在aof文件中。

至此，这个持久化问题可以解决了，只需要采用aof和rdb两种方式结合，并在配置aof使得在rewrite时不进行fsync即可。