Redis 3.2安装及主从复制详细配置
一、编译安装

1、下载安装包：`http://download.redis.io/releases/redis-3.2.3.tar.gz`
目前最新的版本，你也可以打开redis官方的下载页下载合适版本`http://redis.io/download`
2、开始安装

```
 wget http://download.redis.io/releases/redis-3.2.3.tar.gz
$ tar xzf redis-3.2.3.tar.gz
$ cd redis-3.2.3
$ make
$ make test  #可以先执行make test，查看是否缺少依赖包
$ yum install tcl -y  #执行完发现需要安装tcl组件
$ make install
```

无报错则安装完成，程序文件被安装在了`/usr/local/bin`目录下
3、复制配置文件`redis.conf`

```
$ mkdir /etc/redis
$ cp ../redis-3.2.3/redis.conf /etc/redis/6379.conf  #复制安装目录下生成的redis配置文件到etc目录
```

4、启动报错处理我们先不做任何配置启动测试一下

```
$ /usr/local/bin/redis-server /etc/redis/6379.conf
                _.-``__ ''-._
           _.-``    `.  `_.  ''-._           Redis 3.2.3 (00000000/0) 64 bit
       .-`` .-```.  ```\/    _.,_ ''-._
      (    '      ,       .-`  | `,    )     Running in standalone mode
      |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
      |    `-._   `._    /     _.-'    |     PID: 1106
       `-._    `-._  `-./  _.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |           http://redis.io
       `-._    `-._`-.__.-'_.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |
       `-._    `-._`-.__.-'_.-'    _.-'
           `-._    `-.__.-'    _.-'
               `-._        _.-'
                   `-.__.-'

8989:S 17 Aug 15:33:05.472 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is
8989:S 17 Aug 15:33:05.472 # Server started, Redis version 3.2.3
8989:S 17 Aug 15:33:05.472 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix
8989:S 17 Aug 15:33:05.472 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create late
```

`redis`默认是前台运行，我们后边再说怎么配置后台运行，这时发现有三个错误，请不要忽略这些信息，接下来我们处理一下这三个错误：
1）第一个错误大概是说`somaxconn`的值`128`设置过小，从`/proc/sys/net/core/somaxconn`这个路径也可大概知道这个值的设置是关于网络连接中某个最大值的限定设置，此值表示网络连接的队列大小，在配置文件`redis.conf`中的“`tcp-backlog 511`”就配置在高并发环境下的最大队列大小，此值受限于系统的`somaxconn`与`tcp_max_syn_backlog`这两个值，所以应该把这两个内核参数值调大，具体解决方法如下：

```
$ vim /etc/sysctl.conf
$ net.core.somaxconn = 20480  #最大队列长度，应付突发的大并发连接请求，默认为128
$ net.ipv4.tcp_max_syn_backlog = 20480  #半连接队列长度，此值受限于内存大小，默认为1024
$ sysctl -p  #使参数生效
```

2）报错里已经说明了解决方法

```
警告：过量使用内存设置为0！在低内存环境下，后台保存可能失败。为了修正这个问题，
请在/etc/sysctl.conf 添加一项 'vm.overcommit_memory = 1' ，
然后重启（或者运行命令'sysctl vm.overcommit_memory=1' ）使其生效。


vm.overcommit_memory不同的值说明：

0 表示检查是否有足够的内存可用，如果是，允许分配；如果内存不够，拒绝该请求，并返回一个错误给应用程序。
1 允许分配超出物理内存加上交换内存的请求
2 内核总是返回true
redis的数据回写机制分为两种
```

```
同步回写即SAVE命令。redis主进程直接写数据到磁盘。当数据量大时，这个命令将阻塞，响应时间长
异步回写即BGSAVE命令。redis 主进程fork一个子进程，复制主进程的内存并通过子进程回写数据到磁盘。
由于RDB文件写的时候fork一个子进程。相当于复制了一个内存镜像。当时系统的内存是4G，而redis占用了
近3G的内存，因此肯定会报内存无法分配。如果 「vm.overcommit_memory」设置为0，在可用内存不足的情况
下，就无法分配新的内存。如果 「vm.overcommit_memory」设置为1。 那么redis将使用交换内存。
```

解决方法

```
$ vim /etc/sysctl
$ vm.overcommit_memory = 1  #末尾追加
$ sysctl -p  #参数生效
注：使用交换内存并不是一个完美的方案。最好的办法是扩大物理内存。
```

3) 关闭THP透明内存

```
`Transparent Huge Pages (THP)`告警，这是一个关于透明内存巨页的话题。简单来说内存可管理的最小
单位是page，一个page通常是4kb，那1M内存就会有256个page，CPU通过内置的内存管理单元管理page表
记录。Huge Pages就是表示page的大小已超过4kb了，一般是2M到1G，它的出现主要是为了管理超大内存。
个人理解上TB的内存。而THP就是管理Huge Pages的一个抽象层次，根据一些资料显示THP会导致内存锁
影响性能，所以一般建议关闭此功能。
   “/sys/kernel/mm/transparent_hugepage/enabled”有三个值，如下：
  $ cat /sys/kernel/mm/transparent_hugepage/enabled

    always [madvise] never
    ####
    # always 尽量使用透明内存，扫描内存，有512个 4k页面可以整合，就整合成一个2M的页面
    # never 关闭，不使用透明内存
    # madvise 避免改变内存占用
```

关于THP的内容就介绍到这里，现在根据警告是做些修改：

```
$ vim /etc/rc.local
$ echo never > /sys/kernel/mm/transparent_hugepage/enabled  #在开机脚本里追加此命令
```

至此这三个错误就处理好了，再启动就没有错误产生了！

二、配置文件说明

```
$ vim /etc/redis/6379.conf
```

1、后台运行

```
daemonize yes
```

2、bind监听地址

```
bind 192.168.1.10
默认bind的填写的127.0.0.1这样配置是只允许本地访问，如果想远程访问就改为本机网卡绑定的ip地址。
我这边有个问题，就是填写为网卡ip后，本地就不能登录了，bind可以填写多个ip，格式为
bind 192.168.1.10 127.0.0.1 中间用空格分隔，但是没有效果，还是无法本地登录，这个问题待后边再测试。
```

3、配置密码

```
# requirepass foobared
去掉前边#注释，修改foobared为你想配置的任意密码
```

4、日志文件

```
logfile "/var/log/redis.log"
```

5、其他参数

除了上诉几个设置外，还有改端口等一些参数，有需要可以按照配置文件的说明修改。

三、开机启动脚本

`redis`安装目录自带了启动脚本，但是不支持`restart`，还有在配置了密码验证之后就就不能用了，我贴出一个修改后的启动脚本，可以解决这两个问题。

```
$ vim /etc/init.d/redis

# chkconfig: 2345 90 10
# description: service of redis for start and stop add by tomener

PATH=/usr/local/bin:/sbin:/usr/bin:/bin
REDISPORT=6379
EXEC=/usr/local/bin/redis-server
REDIS_CLI=/usr/local/bin/redis-cli

PIDFILE=/var/run/redis_6379.pid
CONF="/etc/redis/6379.conf"
AUTH="Passwd"
BIND_IP='127.0.0.1'

case "$1" in
        start)
                if [ -f $PIDFILE ]
                then
                        echo "$PIDFILE exists, process is already running or crashed."
                else
                        echo "Starting Redis server..."
                        $EXEC $CONF
                fi
                if [ "$?"="0" ]
                then
                        echo "Redis is running..."
                fi
                ;;
        stop)
                if [ ! -f $PIDFILE ]
                then
                        echo "$PIDFILE exists, process is not running."
                else
                        PID=$(cat $PIDFILE)
                        echo "Stopping..."
                        $REDIS_CLI -h $BIND_IP -a $AUTH -p $REDISPORT  SHUTDOWN
                        sleep 2
                        while [ -x $PIDFILE ]
                        do
                                echo "Waiting for Redis to shutdown..."
                                sleep 1
                        done
                        echo "Redis stopped"
                fi
                ;;
        restart|force-reload)
                ${0} stop
                ${0} start
                ;;
        *)
                echo "Usage: /etc/init.d/redis {start|stop|restart|force-reload}" >&2
                exit 1
esac
```

修改其中的配置文件路径、你前一步配置的密码等后保存退出。

```
$ chkconfig redis on
$ service redis restart
```

设置开机启动（根据自己需要来，redis是内存数据库，性能和持久化需要权衡。在配置了主从复制后，有一种方案是停掉master的持久化功能，这时master服务器上redis需要慎重配置开机启动,后边会详细说明。）

四、Redis持久化

主要是两种方式，可以都用，也可以只开一种，根据自己业务需要配置。

rdb方式

`rdb`是`redis`对数据进行持久化而保存到硬盘的数据文件。

工作原理

```
当redis生成dump.rdb文件时，工作过程如下：
redis主进程fork一个子进程
fork出来的子进程将内存的数据集dump到临时的RDB中
当子进程对临时的RDB文件写入完毕，redis用新的RDB文件代替旧的RDB文件
```

默认情况下相关配置如下：

```
save 900 1
save 300 10
save 60 10000
```

其意义为：

```
当1个key更新值时每900秒保存一次数据到硬盘
当10个key更新值时每300秒保存一次到硬盘
当10000个key更新值时每60秒保存一次到硬盘
```

换句话说，当你重启服务器时数据是可能会丢失的，如果数据量小的时候，你会丢失5分钟以内的数据；如果数据量大的时候，你会丢失一分钟以内的数据。

所以`set name 1`后重启服务器，`get name`时的结果会是`nil`，如果你想避免这种情况发生，那么可以`sav`后重启服务器。

然而大多数情况下我们需要防止的是服务器突发情况下的重启，这时候可能没有机会save，所以还是会造成数据的丢失。所以你可以设置`save ""`这样的话会每次写操作到保存到硬盘，但是`redis`作为缓存却需要每次到保存到硬盘已丧失了其作为缓存的意义，因此这种方法是不可取的。

还有一种方式，在每次写操作时使用`bgsave`命令，可以直接返回操作结果并异步将其保存到硬盘。

`rdb`方式的优点是保存的数据存储量小（只有数据）`重启速度很快`；缺点`是可能会造成数据的丢失`。

aof方式

`aof`本质是`redis`操作（写操作）日志文件。aof默认是未开启的，需要在配置文件中进行设置，在配置文件中将这一行改为`appendonly yes`就可以了。

工作原理

```
AOF : append only file。
每当Redis执行一个改变数据集的命令时，这个命令都会被追加到AOF文件的末尾。
当redis重新启动时，程序可以通过AOF文件恢复数据。
```

那么会在什么时候`append`到文件末尾呢？有三种方式：

```
appendfsync always
appendfsync everysec
appendfsync no


appendfsync always每次有新命令追加到 AOF 文件时就执行一次 fsync ：非常慢，也非常安全
appendfsync everysec每秒 fsync 一次：足够快（和使用 RDB 持久化差不多），并且在故障时只会丢失 1 秒钟的数据。
appendfsync no从不 fsync ：将数据交给操作系统来处理。更快，也更不安全的选择。

推荐（并且也是默认）的措施为每秒 fsync 一次， 这种 fsync 策略可以兼顾速度和安全性。
```

aof能够保证数据的安全，但是在重启时比较耗时，而且aof文件的体积比rdb文件大。

使用

在同时开启rdb和aof模式时，会采用aof模式来读取数据。在正常的使用中，如果不是十分在乎短时间内的数据丢失的时候，使用rdb方式会使服务器的效率更高，更节省cpu和硬盘；如果担心数据丢失的话，aof方式无疑会是更好的选择。

五、Redis主从复制

1、概述

`Redis`的`replication`机制允许`slave`从`master`那里通过网络传输拷贝到完整的数据备份。具有以下特点：

- 1.异步复制，从2.8版本开始，slave能不时地从master那里获取到数据。
- 2.允许单个master配置多个slave
- 3.slave允许其它slave连接到自己。一个slave除了可以连接master外，它还可以连接其它的slave。形成一个图状的架构。
- 4.master在进行replication时是非阻塞的，这意味着在replication期间，master依然能够处理客户端的请求。
- 5.slave在replication期间也是非阻塞的，也可以接受来自客户端的请求，但是它用的是之前的旧数据。可以通过配置来决定slave是否在进行replication时用旧数据响应客户端的请求，如果配置为否，那么slave将会返回一个错误消息给客户端。不过当新的数据接收完全后，必须将新数据与旧数据替换，即删除旧数据，在替换数据的这个时间窗口内，slave将会拒绝客户端的请求和连接。
- 6.一般使用replication来可以实现扩展性，例如说，可以将多个slave配置为“只读”，或者是纯粹的数据冗余备份。
- 7.能够通过replication来避免master每次持久化时都将整个数据集持久化到硬盘中。只需把master配置为不进行持久化操作(把配置文件中持久化相关的配置项注释掉即可)，然后连接上一个slave，这个slave则被配置持久化选项。不过需要注意的是，在这个方案中，必须确保master不会自动启动。

2、Master持久化功能关闭时Replication的安全性

当有需要使用到`replication`机制时，一般都会强烈建议把`master`的持久化开关打开。即使为了避免持久化带来的延迟影响，不把持久化开关打开，那么也应该把`master`配置为不会自动启动的。

为了更好地理解当一个不进行持久化的master如果允许自动启动所带来的危险性。可以看看下面这种失败情形：

假设我们有一个`redis`节点`A`，设置为`master`，并且关闭持久化功能，另外两个节点`B`和`C`是它的`slave`，并从`A`复制数据。
如果`A`节点崩溃了导致所有的数据都丢失了，它会有重启系统来重启进程。但是由于持久化功能被关闭了，所以即使它重启了，它的数据集是空的。
而`B`和`C`依然会通过`replication`机制从`A`复制数据，所以`B`和`C`会从`A`那里复制到一份空的数据集，并用这份空的数据集将自己本身的非空的数据集替换掉。于是就相当于丢失了所有的数据。

即使使用一些HA工具，比如说`sentinel`来监控`master-slaves`集群，也会发生上述的情形，因为`master`可能崩溃后迅速恢复。速度太快而导致`sentinel`无法察觉到一个`failure`的发生。

当数据的安全很重要、持久化开关被关闭并且有`replication`发生的时候，那么应该`**禁止实例的自启动**`。

3、replication工作原理
如果你为`master`配置了一个`slave`，不管这个`slave`是否是第一次连接上`Master`，它都会发送一个`SYNC`命令给`master`请求复制数据。

`master`收到`SYNC`命令后，会在后台进行数据持久化，持久化期间，`master`会继续接收客户端的请求，它会把这些可能修改数据集的请求缓存在内存中。当持久化进行完毕以后，`master`会把这份数据集发送给`slave`，`slave`会把接收到的数据进行持久化，然后再加载到内存中。然后，`master`再将之前缓存在内存中的命令发送给`slave`。

当`master`与`slave`之间的连接由于某些原因而断开时，`slave`能够自动重连`master`，如果`master`收到了多个`slave`并发连接请求，它只会进行一次持久化，而不是一个连接一次，然后再把这一份持久化的数据发送给多个并发连接的`slave`。

当`master`和`slave`断开重连后，一般都会对整份数据进行复制。但从`redis2.8`版本开始，支持部分复制。

4、数据部分复制
从`2.8`版本开始，`slave`与`master`能够在网络连接断开重连后只进行部分数据复制。

`master`会在其内存中创建一个复制流的等待队列，`master`和它所有的`slave`都维护了复制的数据下标和`master`的进程`id`，因此，当网络连接断开后，`slave`会请求`master`继续进行未完成的复制，从所记录的数据下标开始。如果进程`id`变化了，或者数据下标不可用，那么将会进行一次全部数据的复制。

5、支持部分数据复制的命令是PSYNC

不需硬盘参与的`Replication`
一般情况下，一次复制需要将内存的数据写到硬盘中，再将数据从硬盘读进内存，再发送给`slave`。

对于速度比较慢的硬盘，这个操作会给`master`带来性能上的损失。`Redis2.8`版本开始，实验性地加上了无硬盘复制的功能。这个功能能将数据从内存中直接发送到`slave`，而不用经过硬盘的存储。

不过这个功能目前处于实验阶段，还未正式发布。

6、主从配置

与`replication`相关的配置比较简单，只需要把下面一行加到`slave`的配置文件中：

```
slaveof 192.168.1.1 6379
```

你只需要把`ip`地址和端口号改一下。当然，你也可以通过客户端发送`SLAVEOF`命令给`slave`。

部分数据复制有一些可调的配置参数，请参考`redis.conf`文件。

无硬盘复制功能可以通过`repl-diskless-sync`来配置，另外一个配置项`repl-diskless-sync-delay`用来配置当收到第一个请求时，等待多个`slave`一起来请求之间的间隔时间。

7、只读的slave

从`redis2.6`版本开始，`slave`支持只读模式，而且是`默认`的。可以通过配置项`slave-read-only`来进行配置，并且支持客户端使用`CONFIG SET`命令来动态修改配置。

只读的`slave`会拒绝所有的写请求，只读的`slave`并不是为了防范不可信的客户端，毕竟一些管理命令例如`DEBUG`和`CONFIG`在只读模式下还是可以使用的。如果确实要确保安全性，那么可以在配置文件中将一些命令重新命名。

也许你会感到很奇怪，为什么能够将一个只读模式的`slave`恢复为可写的呢，尽管可写，但是只要`slave`一同步`master`的数据，就会丢失那些写在`slave`的数据。不过还是有一些合法的应用场景需要存储瞬时数据会用到这个特性。`不过，之后可能会考虑废除掉这个特性。`

```
Setting a slave to authenticate to a master
```

如果`master`通过`requirepass`配置项设置了密码，`slave`每次同步操作都需要验证密码，可以通过在`slave`的配置文件中添加以下配置项：

```
masterauth <password>
```

也可以通过客户端在运行时发送以下命令：

```
config set masterauth <password>
```

8、至少N个slave才允许向master写数据

从`redis2.8`版本开始，`master`可以被配置为，只有当`master`当前有至少`N`个`slave`连接着的时候才接受写数据的请求。

然而，由于`redis`是异步复制的，所以它并不能保证`slave`会受到一个写请求，所以总有一个数据丢失的时间窗口存在。

这个机制的工作原理如下所示：

- slave每秒发送ping心跳给master，询问当前复制了多少数据。
- master会记录下它上次收到某个slave的ping心跳是什么时候。
- 使用者可以配置一个时间，来指定ping心跳的发送不应超过的一个超时时间
- 如果master有至少N个slave，并且ping心跳的超时不超过M秒，那么它就会接收写请求。

也许你会认为这情形好似`CAP`理论中弱化版的`C(consistency)`，因为写请求并不能保证数据的一致性，但这样做，至少数据丢失被限制在了限定的时间内。即`M`秒。

如果`N`和`M`的条件都无法达到，那么`master`会回复一个错误信息。写请求也不会被处理。

有两个配置项用来配置上文中提到的`N`和`M`：

```
min-slaves-to-write <number of slaves>
min-slaves-max-lag <number of seconds>
```

如果需要了解更多，请查阅`redis.conf`配置文件。

六、常用命令

```
redis-cli
客户端命令工具
$ /usr/local/bin/redis-cli -h 192.168.1.1
192.168.1.1:6379> auth Passwd
OK
登录,密码验证
192.168.1.1:6379>info
查看数据库状态
192.168.1.1:6379>info replication
查看slave的复制状态
192.168.1.1:6379>set key 123
插入数据
192.168.1.1:6379>keys *
列出数据
flushdb
清空当前数据
flushall
清除所有数据库
```

可视化客户端工具

`Redis Desktop Manager`（`RedisDesktopManager，RDM`）是一个快速、简单、支持跨平台的`Redis`桌面管理工具，基于`Qt 5`开发，支持通过`SSH Tunnel`连接。

```
下载地址：https://redisdesktop.com/download
```