一个非常详尽的redis入门介绍:http://www.jb51.net/article/56448.htm

**Redis在Linux下的启动**

1、进入redis/src目录下 ./redis-server ../redis.conf 启动服务区端，并装在上一层目录的配置文件。

注：在redis.conf中 requirepass 配置密码 daemonize yes (yes时在后台运行,no在前台运行)可自行配置。

2、在redis/src目录下 ./redis.cli 运行redis客户端。配置了密码的话，进入客户端后需要输入 auth 你的密码 然后才能进入客户端，接下来就可以执行下一节的命令了。

3、通过客户端来关闭redis服务端

127.0.0.1:6379> shutdown

外网怎么测试连接redis服务器呢，首先需要防火墙允许redis端口6379开放出来。

a) iptables -I INPUT 4 -p tcp -m state --state NEW -m tcp --dport 6379 -j ACCEPT

#允许6379端口

b) service iptables save #保存iptables规则

通过命令行传递参数

由于Redis 2.6可以直接通过命令行传递Redis配置参数。这对于测试目的非常有用。

./redis-server --port 6380 --slaveof 127.0.0.1 6379

通过命令行传递的参数的格式与redis.conf文件中使用的参数格式完全相同，除了该关键字是前缀--。

在服务器运行时更​​改Redis配置

请注意，即时修改配置对redis.conf文件没有任何影响，因此在下次重新启动Redis时，将使用旧配置。

[CONFIG REWRITE](https://redis.io/commands/config-rewrite)，它会自动扫描您的redis.conf文件并更新与当前配置值不匹配的字段。不存在但设置为默认值的字段未添加。您的配置文件中的注释被保留。

可以在不停止和重新启动服务的情况下即时重新配置Redis，或者使用特殊命令CONFIG SET和 CONFIG GET以编程方式查询当前配置

**Redis中的数据类型**

1. String 类型 不多介绍

两行代码 set name lyf

get name

输出: lyf

1. List 类型

llist myList 1 //在链表的左侧新增元素 如果不存在myList 则新建myList

rlist myList2 //在链表的右侧新增元素

lrange myList 0 1 //获取从编号0到编号1的元素

lrange myList 0 -1 //获取从编号0到-1的元素

1. set 类型 是一种无序的集合，集合中的元素没有先后顺序(有点类似数学中的集合！)。

sadd myset one //在集合中添加元素

sadd myset two

smembers myset //显示集合中的所有元素

sismember myset three //判断three 是不是myset中的元素

sunion myset1 myset2 //求两个集合的并集

4、sorted sets (有序集合) 上面的第三种是无序集合。

很多时候，我们都将redis中的有序集合叫做zsets，这是因为在redis中，有序集合相关的操作指令都是以z开头的，比如zrange、zadd、zrevrange、zrangebyscore等等。

zadd myzset 1 hello1 //如不存在，创建新的zsets 带序号

zadd myzset 2 helllo2

zrange myzset 0 -1 //带序号的输出 myzset全部数据

5、哈希

hashes存的是字符串和字符串值之间的映射，比如一个用户要存储其全名、姓氏、年龄等等，就很适合使用哈希。

hmset user:001 username lyf age 16 //创建HashMap 并赋值

//红色的为hashmap的名字 后面的为键值对 奇数为键 偶数为值 一一对应 不能为空

hgetall user:001 //获取对应的hashmap

hset user:001 username lyf2 //更改user:001 中username的值

Redis的持久化方式

Redis提供了两种持久化的方式，分别是RDB（Redis DataBase）和AOF（Append Only File）。

1. RDB，简而言之，就是在不同的时间点，将redis存储的数据生成快照并存储到磁盘等介质上；
2. AOF，则是换了一个角度来实现持久化，那就是将redis执行过的所有写指令记录下来，在下次redis重新启动时，只要把这些写指令从前到后再重复执行一遍，就可以实现数据恢复了。

其实RDB和AOF两种方式也可以同时使用，在这种情况下，如果redis重启的话，则会优先采用AOF方式来进行数据恢复，这是因为AOF方式的数据恢复完整度更高。

如果你没有数据持久化的需求，也完全可以关闭RDB和AOF方式，这样的话，redis将变成一个纯内存数据库，就像memcache一样。

RDB：

RDB方式，是将redis某一时刻的数据持久化到磁盘中，是一种快照式的持久化方法。

redis在进行数据持久化的过程中，会先将数据写入到一个临时文件中，待持久化过程都结束了，才会用这个临时文件替换上次持久化好的文件。正是这种特性，让我们可以随时来进行备份，因为快照文件总是完整可用的。

对于RDB方式，redis会单独创建（fork）一个子进程来进行持久化，而主进程是不会进行任何IO操作的，这样就确保了redis极高的性能。

如果需要进行大规模数据的恢复，且对于数据恢复的完整性不是非常敏感，那RDB方式要比AOF方式更加的高效。

虽然RDB有不少优点，但它的缺点也是不容忽视的。如果你对数据的完整性非常敏感，那么RDB方式就不太适合你，因为即使你每5分钟都持久化一次，当redis故障时，仍然会有近5分钟的数据丢失。所以，redis还提供了另一种持久化方式，那就是AOF。

AOF：

AOF，英文是Append Only File，即只允许追加不允许改写的文件。

如前面介绍的，AOF方式是将执行过的写指令记录下来，在数据恢复时按照从前到后的顺序再将指令都执行一遍，就这么简单。

我们通过配置redis.conf中的appendonly yes就可以打开AOF功能。如果有写操作（如SET等），redis就会被追加到AOF文件的末尾。

默认的AOF持久化策略是每秒钟fsync一次（fsync是指把缓存中的写指令记录到磁盘中），因为在这种情况下，redis仍然可以保持很好的处理性能，即使redis故障，也只会丢失最近1秒钟的数据。

如果在追加日志时，恰好遇到磁盘空间满、inode满或断电等情况导致日志写入不完整，也没有关系，redis提供了redis-check-aof工具，可以用来进行日志修复。

因为采用了追加方式，如果不做任何处理的话，AOF文件会变得越来越大，为此，redis提供了AOF文件重写（rewrite）机制，即当AOF文件的大小超过所设定的阈值时，redis就会启动AOF文件的内容压缩，只保留可以恢复数据的最小指令集。举个例子或许更形象，假如我们调用了100次INCR指令，在AOF文件中就要存储100条指令，但这明显是很低效的，完全可以把这100条指令合并成一条SET指令，这就是重写机制的原理。

在进行AOF重写时，仍然是采用先写临时文件，全部完成后再替换的流程，所以断电、磁盘满等问题都不会影响AOF文件的可用性，这点大家可以放心。

AOF方式的另一个好处，我们通过一个“场景再现”来说明。某同学在操作redis时，不小心执行了FLUSHALL，导致redis内存中的数据全部被清空了，这是很悲剧的事情。不过这也不是世界末日，只要redis配置了AOF持久化方式，且AOF文件还没有被重写（rewrite），我们就可以用最快的速度暂停redis并编辑AOF文件，将最后一行的FLUSHALL命令删除，然后重启redis，就可以恢复redis的所有数据到FLUSHALL之前的状态了。是不是很神奇，这就是AOF持久化方式的好处之一。但是如果AOF文件已经被重写了，那就无法通过这种方法来恢复数据了。

虽然优点多多，但AOF方式也同样存在缺陷，比如在同样数据规模的情况下，AOF文件要比RDB文件的体积大。而且，AOF方式的恢复速度也要慢于RDB方式。

如果你直接执行BGREWRITEAOF命令，那么redis会生成一个全新的AOF文件，其中便包括了可以恢复现有数据的最少的命令集。

如果运气比较差，AOF文件出现了被写坏的情况，也不必过分担忧，redis并不会贸然加载这个有问题的AOF文件，而是报错退出。这时可以通过以下步骤来修复出错的文件：

1.备份被写坏的AOF文件

2.运行redis-check-aof –fix进行修复

3.用diff -u来看下两个文件的差异，确认问题点

4.重启redis，加载修复后的AOF文件

【聊聊redis持久化 – AOF重写】

AOF重写的内部运行原理，我们有必要了解一下。

在重写即将开始之际，redis会创建（fork）一个“重写子进程”，这个子进程会首先读取现有的AOF文件，并将其包含的指令进行分析压缩并写入到一个临时文件中。

与此同时，主工作进程会将新接收到的写指令一边累积到内存缓冲区中，一边继续写入到原有的AOF文件中，这样做是保证原有的AOF文件的可用性，避免在重写过程中出现意外。

当“重写子进程”完成重写工作后，它会给父进程发一个信号，父进程收到信号后就会将内存中缓存的写指令追加到新AOF文件中。

当追加结束后，redis就会用新AOF文件来代替旧AOF文件，之后再有新的写指令，就都会追加到新的AOF文件中了。

如何选择RDB和AOF

对于我们应该选择RDB还是AOF，官方的建议是两个同时使用。这样可以提供更可靠的持久化方案。

**Redis主从 – 用法**

我们如果需要设置读写分离，只需要在主服务器中设置：

slave-read-only yes

**Sentinel 哨兵**

在sentinel.conf 配置文件中， 我们可以找到port 属性，这里是用来设置sentinel 的端口，一般情况下，至少会需要三个哨兵对redis 进行监控，我们可以通过修改端口启动多个sentinel 服务。

# The port that this sentinel instance will run on

port 26379

**sentinel monitor mymaster 127.0.0.1 6379** 1

mymaster ：master服务的名称，随便定义

192.168.137.101 6379：master的ip和端口

1：在至少1个哨兵实例都认为master down后把master标记为down（objective down

sentinel.conf具体配置参考http://blog.csdn.net/jinshuaiwang/article/details/44679797

**Redis事务**

四条语句

1. MULTI 开启一个事务
2. EXEC 开始执行这个事务
3. DISCARD 取消一个事务
4. WATCH 用来监视一些key，一旦这些key在事务执行完成前被改变，则取消事务的执行。

调用EXEC之前的错误”，有可能是由于语法有误导致的，也可能时由于内存不足导致的。只要出现某个命令无法成功写入缓冲队列的情况，redis都会进行记录，在客户端调用EXEC时，redis会**拒绝执行这一事务**。比如说

multi

----ok

set age 11

------queued

haha //语法错误 没有进入事务队列 报错了

------ (error) ERR unknown command 'haha'

set age 10

------queued

exec

------- (error) EXECABORT Transaction discarded because of previous errors. //整个事务都不执行，被服务端给无情的拒绝了整个事务

输入multi开启事务后，redis只会检测你的语法错误，输入 haha 这种不符合语法的 属于exec之前的错误。redis会直接拒绝执行整个事务。但是例如以下的:

multi

-----ok

set age 15 // 设置字符串 queued

-----queued

sadd age 16 //语法正确， 但是age是字符串不是集合 依旧queued

-----queued

set age 13

-----queued

exec

-----1) OK

-----2) (error) WRONGTYPE Operation against a key holding the wrong kind of value

-----3) OK //其他两条依旧执行了，中间的错误语句，并没有中断事务，redis不管这些中间出错的语句，依旧执行完整个事务。！！！！

最后一个指令“WATCH”，这是一个很好用的指令，它可以帮我们实现类似于“乐观锁”的效果，即CAS（check and set）。

WATCH本身的作用是“监视key是否被改动过”，而且支持同时监视多个key，只要还没真正触发事务，WATCH都会尽职尽责的监视，一旦发现某个key被修改了，在执行EXEC时就会返回nil，表示事务无法触发。

Redis 配置文件

可以使用引号将包含空格的字符串提供为参数，如以下示例所示：

requirepass "hello world"

redis配置文件被分成了几大块区域，它们分别是：

1.通用（general）

2.快照（snapshotting）

3.复制（replication）

4.安全（security）

5.限制（limits)

6.追加模式（append only mode)

7.LUA脚本（lua scripting)

8.慢日志（slow log)

9.事件通知（event notification）

Redis广播与订阅

几个客户端常用命令:

1. Subscribe channel\_name 订阅某个频道
2. Publish channel\_name message 向某个频道发送信息
3. Psubscribe pattern [pattern] 模糊订阅频道

订阅一个或多个符合给定模式的频道。

每个模式以 \* 作为匹配符，比如 it\* 匹配所有以 it 开头的频道( it.news 、 it.blog 、 it.tweets 等等)， news.\* 匹配所有以 news. 开头的频道( news.it 、 news.global.today 等等)，诸如此类。

注意：subscribe是一个阻塞的方法，在取消订阅该频道前，会一直阻塞在这，只有当取消了订阅才会执行下面的other code，参考上面代码，我在onMessage里面收到消息后，调用了this.unsubscribe(); 来取消订阅，这样才会执行后面的other code

Subsribe 是订阅的意思。

Jedis中:

Jedis jedis=new Jedis("192.168.85.2");

jedis.auth("123456");

jedis.publish(“mychannel”,”hello from idea”); --------java客户端中publish

jedis.subscribe(new jedisPubSub () {

public void onMessage(java.lang.String channel, java.lang.String message) {

super.onMessage(channel, message);

System.out.println("subsribe"+message);

}

},"mychannel"); ------jedis订阅channel 注意subscribe是一个阻塞的方法

Set hh 10

Incr hh ----hh++

10 incr hh -- 做十次hh自加

Redus Cluster

注意 主从 与 节点的区别 :主机与从机其实都是节点.(分为主节点,从节点)

Redis 集群的主从复制模型

为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型,每个节点都会有几个slvae机.

在我们例子中具有A，B，C三个节点的集群,在没有复制模型的情况下,如果节点B失败了，那么整个集群就会以为缺少5501-11000这个范围的槽而不可用.

然而如果在集群创建的时候（或者过一段时间）我们为每个节点添加一个从节点A1，B1，C1,那么整个集群便有三个master节点和三个slave节点组成，这样在节点B失败后，集群便会选举B1为新的主节点继续服务，整个集群便不会因为槽找不到而不可用了

不过当B和B1 都失败后，集群是不可用的.

配置集群;

集群其实就是一群redis实例,但是他们启动时使用的是集群配置文件.

要让集群正常运作至少需要三个主节点，不过在刚开始试用集群功能时， 强烈建议使用六个节点： 其中三个为主节点， 而其余三个则是各个主节点的从节点

下面是一个最少选项的集群的配置文件:

**port 7000**

**cluster-enabled yes //用于开redis实例的集群模式**

**cluster-config-file nodes.conf**

**//设定了保存节点配置文件的路径， 默认值为 nodes.conf,** **节点配置文件无须人为修改， 它由 Redis 集群在启动时创建， 并在有需要时自动进行更新。**

**cluster-node-timeout 5000**

**appendonly yes**

复制6份,端口号设置为 7001-7006.然后分别启动 以此配置文件启动,就可以跑起来6个Redis cluster实例了.cluster –config-file 文件可能也需要分别设置一下.接下来需要将6个cluster实例串起来成为一个整体的集群. 官方在src/目录下提供了一个 redis-trib.rb 程序

第一次使用,如果没有装ruby,需要安装ruby.

yum -y install ruby ruby-devel rubygems rpm-build

再用 gem 这个命令来安装 redis接口 gem是ruby的一个工具包.

gem install redis //等一会儿就好了

调用程序启动集群:

./redis-trib.rb create --replicas 0 127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004

--replicas 0 表示我们希望为集群中的每个主节点创建0个从节点。

其他参数则是这个集群实例的地址列表,3个master redis-trib 会打印出一份预想中的配置给你看， 如果你觉得没问题的话， 就可以输入 yes ， redis-trib 就会将这份配置应用到集群当中,让各个节点开始互相通讯,最后可以得到如下信息：

[OK] All 16384 slots covered

这表示集群中的 16384 个槽都有至少一个主节点在处理， 集群运作正常。

客户端连接集群

./redis-cli -c -p 7001

当然,如果你不想按上面一步步操作来.redis在redis-3.2.10/utils/create-cluster文件夹下提供了一个create-cluster的shell 脚本.(CTM放在最后给老子看,cnm)

In order to start a 6 nodes cluster with 3 masters and 3 slaves just type the following commands:

create-cluster start

create-cluster create

Reply to yes in step 2 when the redis-trib utility wants you to accept the cluster layout.

You can now interact with the cluster, the first node will start at port 30001 by default. When you are done, stop the cluster with:

create-cluster stop.

Please read the README inside this directory for more information on how to run the script.

集群设置密码;

1.密码设置

方式一：修改所有Redis集群中的redis.conf文件加入：

masterauth passwd123

requirepass passwd123

说明：这种方式需要重新启动各节点

方式二：进入各个实例进行设置：

./redis-cli -c -p 7000

config set masterauth passwd123

config set requirepass passwd123

config rewrite

之后分别使用./redis-cli -c -p 7001，./redis-cli -c -p 7002…..命令给各节点设置上密码

注意：各个节点密码都必须一致，否则Redirected就会失败，这种方式会把密码写入到redis.conf里面去，且不用重启

2.设置密码之后如果需要使用redis-trib.rb的各种命令

如：redis-trib.rb check 127.0.0.1，则会报错ERR] Sorry, can’t connect to node 127.0.0.1:7000

解决办法：

vim /usr/lib/ruby/gems/1.8/gems/redis-3.3.0/lib/redis/client.rb,然后修改passord

class Client

DEFAULTS = {

:url => lambda { ENV["REDIS\_URL"] },

:scheme => "redis",

:host => "127.0.0.1",

:port => 6379,

:path => nil,

:timeout => 5.0,

:password => "passwd123",

:db => 0,

:driver => nil,

:id => nil,

:tcp\_keepalive => 0,

:reconnect\_attempts => 1,

:inherit\_socket => false

}

2.带密码访问集群

./redis-cli -c -p 7000 -a passwd123

3.Java客户端使用

在java程序里面使用最新的 redis.clients架包，老版本的包里可能没有提供密码访问的方式

例如maven程序：

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.9.0</version>

<type>jar</type>

</dependency>

然后代码里面使用带密码的构造函数

redisCluster = new JedisCluster(nodeList, TIME\_OUT, 3000, 5, "passwd123", poolConfig);

搭建redis集群时报错“create ERR Slot 9838 is already busy (Redis::CommandError)”，看打印好像分配slots时就重叠了一部分，请问这要怎么解决呢。

用redis-cli登录到每个节点执行flushall和cluster reset就可以了