**Redis实战**

# 课程目标

NoSQL介绍

Redis介绍

Redis的安装

Redis的基本数据类型

SpringBoot整合Redis

Redis主从复制、安全、哨兵

Redis事物、持久化机制、发布与订阅模式

Redis集群

# NoSQL介绍

NoSQL 是 Not Only SQL 的缩写，意即"不仅仅是SQL"的意思，泛指非关系型的数据库。强调Key-Value Stores和文档数据库的优点，而不是单纯的反对RDBMS。

NoSQL产品是传统关系型数据库的功能阉割版本，通过减少用不到或很少用的功能，来大幅度提高产品性能

NoSQL产品 redis、mongodb [Membase](http://www.couchbase.org/membase)、[HBase](http://hbase.apache.org/)

# Redis 简介

## 3.1什么是Redis?

Redis 是完全开源免费的，遵守BSD协议，是一个高性能的key-value数据库。

Redis 与其他 key - value 缓存产品有以下三个特点：

Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。

Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。

Redis支持数据的备份，即master-slave模式的数据备份。

## 3.2 Redis应用场景

主要能够体现 解决数据库的访问压力。

例如:短信验证码时间有效期、session共享解决方案

## 3.3 Redis优势

性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。

丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

原子 – Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行。

丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

## 3.4 Redis与其他key-value存储有什么不同？

Redis有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。

Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

# Redis安装

## 4.1 windows 安装redis

## 4.2 linux 安装redis

Redis的官方下载网址是：[http://redis.io/download](http://redis.io/download" \t "_blank)  (这里下载的是Linux版的Redis源码包）

Redis服务器端的默认端口是6379。

**这里以虚拟机中的Linux系统如何安装Redis进行讲解。**

 在windows系统中下载好Redis的源码包。

1. 通过WinSCP工具，将Redis的源码包由windows上传到Linux系统的这个目录/opt/redis (即根目录下的lamp文件夹）。

2. 解压缩。

tar -zxf redis-2.6.17.tar.gz

3. 切换到解压后的目录。

cd redis-2.6.17            ( 一般来说，解压目录里的INSTALL文件或README文件里写有安装说明，可参考之）

4. 编译。

make

（注意，编译需要C语言编译器gcc的支持，如果没有，需要先安装gcc。可以使用rpm -q gcc查看gcc是否安装）

（利用yum在线安装gcc的命令    yum -y install gcc )

（如果编译出错，请使用make clean清除临时文件。之后，找到出错的原因，解决问题后再来重新安装。 ）

5. 进入到src目录。

cd src

6. 执行安装。

make install

到此就安装完成。但是，由于安装redis的时候，我们没有选择安装路径，故是默认位置安装。在此，我们可以将可执行文件和配置文件移动到习惯的目录。

cd /usr/local

mkdir -p /usr/local/redis/bin

mkdir -p /usr/local/redis/etc

cd /lamp/redis-2.6.17

cp ./redis.conf /usr/local/redis/etc

cd src

cp mkreleasehdr.sh redis-benchmark redis-check-aof redis-check-dump redis-cli redis-server redis-sentinel /usr/local/redis/bin

7．开放linux 6379 端口

1.编辑 /etc/sysconfig/iptables 文件：vi /etc/sysconfig/iptables  
加入内容并保存：-A RH-Firewall-1-INPUT -m state –state NEW -m tcp -p tcp –dport 6379 -j ACCEPT  
2.重启服务：/etc/init.d/iptables restart  
3.查看端口是否开放：/sbin/iptables -L -n

**比较重要的3个可执行文件：**

redis-server：Redis服务器程序

redis-cli：Redis客户端程序，它是一个命令行操作工具。也可以使用telnet根据其纯文本协议操作。

redis-benchmark：Redis性能测试工具，测试Redis在你的系统及配置下的读写性能。

### 4.2.1 Redis的启动命令：

/usr/local/redis/bin/redis-server

或

cd /usr/local/redis/bin

./redis-server /usr/local/redis/etc/redis.conf    为redis-server指定配置文

### 4.2.2修改 redis.conf文件

daemonize yes --- 修改为yes 后台启动

requirepass 123456  ----注释取消掉设置账号密码

ps aux | grep '6379' --- 查询端口

kill -15 9886 --- 杀死重置

kill -9 9886 --- 强制杀死

service iptables stop 停止防火强

### 4.2.3 redis命令连接方式

./redis-cli -h 127.0.0.1 -p 6379 -a "123456" --- redis 使用账号密码连接

PING 结果表示成功

### 4.2.4 停止redis

redis-cli shutdown  或者 kill redis进程的pid

### 4.2.5 关闭防火墙

# Redis客户端连接方式

使用redisclient-win32.x86.1.5

# Redis的基本数据类型

## 6.1 字符串类型(String)

|  |
| --- |
| redis 127.0.0.1:6379> SET mykey "redis"  OK  redis 127.0.0.1:6379> GET mykey  "redis" |

在上面的例子中，SET和GET是redis中的命令，而mykey是键的名称。

Redis字符串命令用于管理Redis中的字符串值。以下是使用Redis字符串命令的语法。

redis 127.0.0.1:6379> COMMAND KEY\_NAME

Shell

**示例**

redis 127.0.0.1:6379> SET mykey "redis"

OK

redis 127.0.0.1:6379> GET mykey

"redis"

Shell

在上面的例子中，SET和GET是redis中的命令，而mykey是键的名称。

Redis字符串命令

下表列出了一些用于在**Redis**中管理字符串的基本命令。

| **编号** | **命令** | **描述说明** |
| --- | --- | --- |
| 1 | [SET key value](http://www.yiibai.com/redis/strings_set.html) | 此命令设置指定键的值。 |
| 2 | [GET key](http://www.yiibai.com/redis/strings_get.html) | 获取指定键的值。 |
| 3 | [GETRANGE key start end](http://www.yiibai.com/redis/strings_getrange.html) | 获取存储在键上的字符串的子字符串。 |
| 4 | [GETSET key value](http://www.yiibai.com/redis/strings_getset.html) | 设置键的字符串值并返回其旧值。 |
| 5 | [GETBIT key offset](http://www.yiibai.com/redis/strings_getbit.html) | 返回在键处存储的字符串值中偏移处的位值。 |
| 6 | [MGET key1 [key2..]](http://www.yiibai.com/redis/strings_mget.html) | 获取所有给定键的值 |
| 7 | [SETBIT key offset value](http://www.yiibai.com/redis/strings_setbit.html) | 存储在键上的字符串值中设置或清除偏移处的位 |
| 8 | [SETEX key seconds value](http://www.yiibai.com/redis/strings_setex.html) | 使用键和到期时间来设置值 |
| 9 | [SETNX key value](http://www.yiibai.com/redis/strings_setnx.html) | 设置键的值，仅当键不存在时 |
| 10 | [SETRANGE key offset value](http://www.yiibai.com/redis/strings_setrange.html) | 在指定偏移处开始的键处覆盖字符串的一部分 |
| 11 | [STRLEN key](http://www.yiibai.com/redis/strings_strlen.html) | 获取存储在键中的值的长度 |
| 12 | [MSET key value [key value …]](http://www.yiibai.com/redis/strings_mset.html) | 为多个键分别设置它们的值 |
| 13 | [MSETNX key value [key value …]](http://www.yiibai.com/redis/strings_msetnx.html) | 为多个键分别设置它们的值，仅当键不存在时 |
| 14 | [PSETEX key milliseconds value](http://www.yiibai.com/redis/strings_psetex.html) | 设置键的值和到期时间(以毫秒为单位) |
| 15 | [INCR key](http://www.yiibai.com/redis/strings_incr.html) | 将键的整数值增加1 |
| 16 | [INCRBY key increment](http://www.yiibai.com/redis/strings_incrby.html) | 将键的整数值按给定的数值增加 |
| 17 | [INCRBYFLOAT key increment](http://www.yiibai.com/redis/strings_incrbyfloat.html) | 将键的浮点值按给定的数值增加 |
| 18 | [DECR key](http://www.yiibai.com/redis/strings_decr.html) | 将键的整数值减1 |
| 19 | [DECRBY key decrement](http://www.yiibai.com/redis/strings_decrby.html) | 按给定数值减少键的整数值 |
| 20 | [APPEND key value](http://www.yiibai.com/redis/strings_append.html) | 将指定值附加到键 |

## 6.2 列表类型(List)

Redis列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）

一个列表最多可以包含 232 - 1 个元素 (4294967295, 每个列表超过40亿个元素)。

redis 127.0.0.1:6379> LPUSH runoobkey redis

(integer) 1

redis 127.0.0.1:6379> LPUSH runoobkey mongodb

(integer) 2

redis 127.0.0.1:6379> LPUSH runoobkey mysql

(integer) 3

redis 127.0.0.1:6379> LRANGE runoobkey 0 10

1) "mysql"

2) "mongodb"

3) "redis"

Redis 列表命令

下表列出了列表相关的基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [BLPOP key1 [key2 ] timeout](http://www.runoob.com/redis/lists-blpop.html)  移出并获取列表的第一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。 |
| 2 | [BRPOP key1 [key2 ] timeout](http://www.runoob.com/redis/lists-brpop.html)  移出并获取列表的最后一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。 |
| 3 | [BRPOPLPUSH source destination timeout](http://www.runoob.com/redis/lists-brpoplpush.html)  从列表中弹出一个值，将弹出的元素插入到另外一个列表中并返回它； 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。 |
| 4 | [LINDEX key index](http://www.runoob.com/redis/lists-lindex.html)  通过索引获取列表中的元素 |
| 5 | [LINSERT key BEFORE|AFTER pivot value](http://www.runoob.com/redis/lists-linsert.html)  在列表的元素前或者后插入元素 |
| 6 | [LLEN key](http://www.runoob.com/redis/lists-llen.html)  获取列表长度 |
| 7 | [LPOP key](http://www.runoob.com/redis/lists-lpop.html)  移出并获取列表的第一个元素 |
| 8 | [LPUSH key value1 [value2]](http://www.runoob.com/redis/lists-lpush.html)  将一个或多个值插入到列表头部 |
| 9 | [LPUSHX key value](http://www.runoob.com/redis/lists-lpushx.html)  将一个值插入到已存在的列表头部 |
| 10 | [LRANGE key start stop](http://www.runoob.com/redis/lists-lrange.html)  获取列表指定范围内的元素 |
| 11 | [LREM key count value](http://www.runoob.com/redis/lists-lrem.html)  移除列表元素 |
| 12 | [LSET key index value](http://www.runoob.com/redis/lists-lset.html)  通过索引设置列表元素的值 |
| 13 | [LTRIM key start stop](http://www.runoob.com/redis/lists-ltrim.html)  对一个列表进行修剪(trim)，就是说，让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。 |
| 14 | [RPOP key](http://www.runoob.com/redis/lists-rpop.html)  移除并获取列表最后一个元素 |
| 15 | [RPOPLPUSH source destination](http://www.runoob.com/redis/lists-rpoplpush.html)  移除列表的最后一个元素，并将该元素添加到另一个列表并返回 |
| 16 | [RPUSH key value1 [value2]](http://www.runoob.com/redis/lists-rpush.html)  在列表中添加一个或多个值 |
| 17 | [RPUSHX key value](http://www.runoob.com/redis/lists-rpushx.html)  为已存在的列表添加值 |

## 6.3 Redis 集合(Set)

Redis的Set是string类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。

Redis 中 集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

集合中最大的成员数为 232 - 1 (4294967295, 每个集合可存储40多亿个成员)。

**实例**

redis 127.0.0.1:6379> SADD runoobkey redis

(integer) 1

redis 127.0.0.1:6379> SADD runoobkey mongodb

(integer) 1

redis 127.0.0.1:6379> SADD runoobkey mysql

(integer) 1

redis 127.0.0.1:6379> SADD runoobkey mysql

(integer) 0

redis 127.0.0.1:6379> SMEMBERS runoobkey

1) "mysql"

2) "mongodb"

3) "redis"

在以上实例中我们通过 **SADD** 命令向名为 **runoobkey** 的集合插入的三个元素。

**Redis 集合命令**

下表列出了 Redis 集合基本命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [SADD key member1 [member2]](http://www.runoob.com/redis/sets-sadd.html)  向集合添加一个或多个成员 |
| 2 | [SCARD key](http://www.runoob.com/redis/sets-scard.html)  获取集合的成员数 |
| 3 | [SDIFF key1 [key2]](http://www.runoob.com/redis/sets-sdiff.html)  返回给定所有集合的差集 |
| 4 | [SDIFFSTORE destination key1 [key2]](http://www.runoob.com/redis/sets-sdiffstore.html)  返回给定所有集合的差集并存储在 destination 中 |
| 5 | [SINTER key1 [key2]](http://www.runoob.com/redis/sets-sinter.html)  返回给定所有集合的交集 |
| 6 | [SINTERSTORE destination key1 [key2]](http://www.runoob.com/redis/sets-sinterstore.html)  返回给定所有集合的交集并存储在 destination 中 |
| 7 | [SISMEMBER key member](http://www.runoob.com/redis/sets-sismember.html)  判断 member 元素是否是集合 key 的成员 |
| 8 | [SMEMBERS key](http://www.runoob.com/redis/sets-smembers.html)  返回集合中的所有成员 |
| 9 | [SMOVE source destination member](http://www.runoob.com/redis/sets-smove.html)  将 member 元素从 source 集合移动到 destination 集合 |
| 10 | [SPOP key](http://www.runoob.com/redis/sets-spop.html)  移除并返回集合中的一个随机元素 |
| 11 | [SRANDMEMBER key [count]](http://www.runoob.com/redis/sets-srandmember.html)  返回集合中一个或多个随机数 |
| 12 | [SREM key member1 [member2]](http://www.runoob.com/redis/sets-srem.html)  移除集合中一个或多个成员 |
| 13 | [SUNION key1 [key2]](http://www.runoob.com/redis/sets-sunion.html)  返回所有给定集合的并集 |
| 14 | [SUNIONSTORE destination key1 [key2]](http://www.runoob.com/redis/sets-sunionstore.html)  所有给定集合的并集存储在 destination 集合中 |
| 15 | [SSCAN key cursor [MATCH pattern] [COUNT count]](http://www.runoob.com/redis/sets-sscan.html)  迭代集合中的元素 |

## 6.4 Redis 有序集合(sorted set)

Redis 有序集合和集合一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

有序集合的成员是唯一的,但分数(score)却可以重复。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。 集合中最大的成员数为 232 - 1 (4294967295, 每个集合可存储40多亿个成员)。

**实例**

redis 127.0.0.1:6379> ZADD runoobkey 1 redis

(integer) 1

redis 127.0.0.1:6379> ZADD runoobkey 2 mongodb

(integer) 1

redis 127.0.0.1:6379> ZADD runoobkey 3 mysql

(integer) 1

redis 127.0.0.1:6379> ZADD runoobkey 3 mysql

(integer) 0

redis 127.0.0.1:6379> ZADD runoobkey 4 mysql

(integer) 0

redis 127.0.0.1:6379> ZRANGE runoobkey 0 10 WITHSCORES

1) "redis"

2) "1"

3) "mongodb"

4) "2"

5) "mysql"

6) "4"

在以上实例中我们通过命令 **ZADD** 向 redis 的有序集合中添加了三个值并关联上分数。

**Redis 有序集合命令**

下表列出了 redis 有序集合的基本命令:

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [ZADD key score1 member1 [score2 member2]](http://www.runoob.com/redis/sorted-sets-zadd.html)  向有序集合添加一个或多个成员，或者更新已存在成员的分数 |
| 2 | [ZCARD key](http://www.runoob.com/redis/sorted-sets-zcard.html)  获取有序集合的成员数 |
| 3 | [ZCOUNT key min max](http://www.runoob.com/redis/sorted-sets-zcount.html)  计算在有序集合中指定区间分数的成员数 |
| 4 | [ZINCRBY key increment member](http://www.runoob.com/redis/sorted-sets-zincrby.html)  有序集合中对指定成员的分数加上增量 increment |
| 5 | [ZINTERSTORE destination numkeys key [key ...]](http://www.runoob.com/redis/sorted-sets-zinterstore.html)  计算给定的一个或多个有序集的交集并将结果集存储在新的有序集合 key 中 |
| 6 | [ZLEXCOUNT key min max](http://www.runoob.com/redis/sorted-sets-zlexcount.html)  在有序集合中计算指定字典区间内成员数量 |
| 7 | [ZRANGE key start stop [WITHSCORES]](http://www.runoob.com/redis/sorted-sets-zrange.html)  通过索引区间返回有序集合成指定区间内的成员 |
| 8 | [ZRANGEBYLEX key min max [LIMIT offset count]](http://www.runoob.com/redis/sorted-sets-zrangebylex.html)  通过字典区间返回有序集合的成员 |
| 9 | [ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT]](http://www.runoob.com/redis/sorted-sets-zrangebyscore.html)  通过分数返回有序集合指定区间内的成员 |
| 10 | [ZRANK key member](http://www.runoob.com/redis/sorted-sets-zrank.html)  返回有序集合中指定成员的索引 |
| 11 | [ZREM key member [member ...]](http://www.runoob.com/redis/sorted-sets-zrem.html)  移除有序集合中的一个或多个成员 |
| 12 | [ZREMRANGEBYLEX key min max](http://www.runoob.com/redis/sorted-sets-zremrangebylex.html)  移除有序集合中给定的字典区间的所有成员 |
| 13 | [ZREMRANGEBYRANK key start stop](http://www.runoob.com/redis/sorted-sets-zremrangebyrank.html)  移除有序集合中给定的排名区间的所有成员 |
| 14 | [ZREMRANGEBYSCORE key min max](http://www.runoob.com/redis/sorted-sets-zremrangebyscore.html)  移除有序集合中给定的分数区间的所有成员 |
| 15 | [ZREVRANGE key start stop [WITHSCORES]](http://www.runoob.com/redis/sorted-sets-zrevrange.html)  返回有序集中指定区间内的成员，通过索引，分数从高到底 |
| 16 | [ZREVRANGEBYSCORE key max min [WITHSCORES]](http://www.runoob.com/redis/sorted-sets-zrevrangebyscore.html)  返回有序集中指定分数区间内的成员，分数从高到低排序 |
| 17 | [ZREVRANK key member](http://www.runoob.com/redis/sorted-sets-zrevrank.html)  返回有序集合中指定成员的排名，有序集成员按分数值递减(从大到小)排序 |
| 18 | [ZSCORE key member](http://www.runoob.com/redis/sorted-sets-zscore.html)  返回有序集中，成员的分数值 |
| 19 | [ZUNIONSTORE destination numkeys key [key ...]](http://www.runoob.com/redis/sorted-sets-zunionstore.html)  计算给定的一个或多个有序集的并集，并存储在新的 key 中 |
| 20 | [ZSCAN key cursor [MATCH pattern] [COUNT count]](http://www.runoob.com/redis/sorted-sets-zscan.html)  迭代有序集合中的元素（包括元素成员和元素分值） |

## 6.5 Redis 哈希(Hash)

Redis hash 是一个string类型的field和value的映射表，hash特别适合用于存储对象。

Redis 中每个 hash 可以存储 232 - 1 键值对（40多亿）。

**实例**

127.0.0.1:6379> HMSET runoobkey name "redis tutorial" description "redis basic commands for caching" likes 20 visitors 23000

OK

127.0.0.1:6379> HGETALL runoobkey

1) "name"

2) "redis tutorial"

3) "description"

4) "redis basic commands for caching"

5) "likes"

6) "20"

7) "visitors"

8) "23000"

**hset key mapHey MapValue**

在以上实例中，我们设置了 redis 的一些描述信息(name, description, likes, visitors) 到哈希表的 **runoobkey** 中。

**Redis hash 命令**

下表列出了 redis hash 基本的相关命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [HDEL key field2 [field2]](http://www.runoob.com/redis/hashes-hdel.html)  删除一个或多个哈希表字段 |
| 2 | [HEXISTS key field](http://www.runoob.com/redis/hashes-hexists.html)  查看哈希表 key 中，指定的字段是否存在。 |
| 3 | [HGET key field](http://www.runoob.com/redis/hashes-hget.html)  获取存储在哈希表中指定字段的值。 |
| 4 | [HGETALL key](http://www.runoob.com/redis/hashes-hgetall.html)  获取在哈希表中指定 key 的所有字段和值 |
| 5 | [HINCRBY key field increment](http://www.runoob.com/redis/hashes-hincrby.html)  为哈希表 key 中的指定字段的整数值加上增量 increment 。 |
| 6 | [HINCRBYFLOAT key field increment](http://www.runoob.com/redis/hashes-hincrbyfloat.html)  为哈希表 key 中的指定字段的浮点数值加上增量 increment 。 |
| 7 | [HKEYS key](http://www.runoob.com/redis/hashes-hkeys.html)  获取所有哈希表中的字段 |
| 8 | [HLEN key](http://www.runoob.com/redis/hashes-hlen.html)  获取哈希表中字段的数量 |
| 9 | [HMGET key field1 [field2]](http://www.runoob.com/redis/hashes-hmget.html)  获取所有给定字段的值 |
| 10 | [HMSET key field1 value1 [field2 value2 ]](http://www.runoob.com/redis/hashes-hmset.html)  同时将多个 field-value (域-值)对设置到哈希表 key 中。 |
| 11 | [HSET key field value](http://www.runoob.com/redis/hashes-hset.html)  将哈希表 key 中的字段 field 的值设为 value 。 |
| 12 | [HSETNX key field value](http://www.runoob.com/redis/hashes-hsetnx.html)  只有在字段 field 不存在时，设置哈希表字段的值。 |
| 13 | [HVALS key](http://www.runoob.com/redis/hashes-hvals.html)  获取哈希表中所有值 |
| 14 | HSCAN key cursor [MATCH pattern] [COUNT count]  迭代哈希表中的键值对。 |

# Redis jedis

## 7.1 引入Maven依赖

|  |
| --- |
| **<!-- https://mvnrepository.com/artifact/redis.clients/jedis -->**  **<dependency>**  **<groupId>redis.clients</groupId>**  **<artifactId>jedis</artifactId>**  **<version>2.9.0</version>**  **</dependency>** |

## 7.2 Java代码

|  |
| --- |
| public class TestRedis {  private Jedis jedis;    @Before  public void setup() {  //连接redis服务器，192.168.0.100:6379  jedis = new Jedis("192.168.0.100", 6379);  //权限认证  jedis.auth("admin");  }    /\*\*  \* redis存储字符串  \*/  @Test  public void testString() {  //-----添加数据----------  jedis.set("name","xinxin");//向key-->name中放入了value-->xinxin  System.*out*.println(jedis.get("name"));//执行结果：xinxin    jedis.append("name", " is my lover"); //拼接  System.*out*.println(jedis.get("name"));    jedis.del("name"); //删除某个键  System.*out*.println(jedis.get("name"));  //设置多个键值对  jedis.mset("name","liuling","age","23","qq","476777XXX");  jedis.incr("age"); //进行加1操作  System.*out*.println(jedis.get("name") + "-" + jedis.get("age") + "-" + jedis.get("qq"));    }    /\*\*  \* redis操作Map  \*/  @Test  public void testMap() {  //-----添加数据----------  Map<String, String> map = new HashMap<String, String>();  map.put("name", "xinxin");  map.put("age", "22");  map.put("qq", "123456");  jedis.hmset("user",map);  //取出user中的name，执行结果:[minxr]-->注意结果是一个泛型的List  //第一个参数是存入redis中map对象的key，后面跟的是放入map中的对象的key，后面的key可以跟多个，是可变参数  List<String> rsmap = jedis.hmget("user", "name", "age", "qq");  System.*out*.println(rsmap);    //删除map中的某个键值  jedis.hdel("user","age");  System.*out*.println(jedis.hmget("user", "age")); //因为删除了，所以返回的是null  System.*out*.println(jedis.hlen("user")); //返回key为user的键中存放的值的个数2  System.*out*.println(jedis.exists("user"));//是否存在key为user的记录 返回true  System.*out*.println(jedis.hkeys("user"));//返回map对象中的所有key  System.*out*.println(jedis.hvals("user"));//返回map对象中的所有value    Iterator<String> iter=jedis.hkeys("user").iterator();  while (iter.hasNext()){  String key = iter.next();  System.*out*.println(key+":"+jedis.hmget("user",key));  }  }    /\*\*  \* jedis操作List  \*/  @Test  public void testList(){  //开始前，先移除所有的内容  jedis.del("java framework");  System.*out*.println(jedis.lrange("java framework",0,-1));  //先向key java framework中存放三条数据  jedis.lpush("java framework","spring");  jedis.lpush("java framework","struts");  jedis.lpush("java framework","hibernate");  //再取出所有数据jedis.lrange是按范围取出，  // 第一个是key，第二个是起始位置，第三个是结束位置，jedis.llen获取长度 -1表示取得所有  System.*out*.println(jedis.lrange("java framework",0,-1));    jedis.del("java framework");  jedis.rpush("java framework","spring");  jedis.rpush("java framework","struts");  jedis.rpush("java framework","hibernate");  System.*out*.println(jedis.lrange("java framework",0,-1));  }    /\*\*  \* jedis操作Set  \*/  @Test  public void testSet(){  //添加  jedis.sadd("user","liuling");  jedis.sadd("user","xinxin");  jedis.sadd("user","ling");  jedis.sadd("user","zhangxinxin");  jedis.sadd("user","who");  //移除noname  jedis.srem("user","who");  System.*out*.println(jedis.smembers("user"));//获取所有加入的value  System.*out*.println(jedis.sismember("user", "who"));//判断 who 是否是user集合的元素  System.*out*.println(jedis.srandmember("user"));  System.*out*.println(jedis.scard("user"));//返回集合的元素个数  }    @Test  public void test() throws InterruptedException {  //jedis 排序  //注意，此处的rpush和lpush是List的操作。是一个双向链表（但从表现来看的）  jedis.del("a");//先清除数据，再加入数据进行测试  jedis.rpush("a", "1");  jedis.lpush("a","6");  jedis.lpush("a","3");  jedis.lpush("a","9");  System.*out*.println(jedis.lrange("a",0,-1));// [9, 3, 6, 1]  System.*out*.println(jedis.sort("a")); //[1, 3, 6, 9] //输入排序后结果  System.*out*.println(jedis.lrange("a",0,-1));  }    @Test  public void testRedisPool() {  RedisUtil.getJedis().set("newname", "中文测试");  System.*out*.println(RedisUtil.getJedis().get("newname"));  } |

# SpringBoot集成Redis

## 8.1、引入maven依赖

|  |
| --- |
| **<parent>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-parent</artifactId>**  **<version>1.5.3.RELEASE</version>**  **</parent>**  **<dependencies>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-data-redis</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-data-redis</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-web</artifactId>**  **</dependency>**  **</dependencies>** |

## 8.2、新增配置文件信息

|  |
| --- |
| ########################################################  ###Redis (RedisConfiguration)  ########################################################  spring.redis.database=0  spring.redis.host=127.0.0.1  spring.redis.port=6379  spring.redis.password=123456  spring.redis.pool.max-idle=8  spring.redis.pool.min-idle=0  spring.redis.pool.max-active=8  spring.redis.pool.max-wait=-1  spring.redis.timeout=5000 |

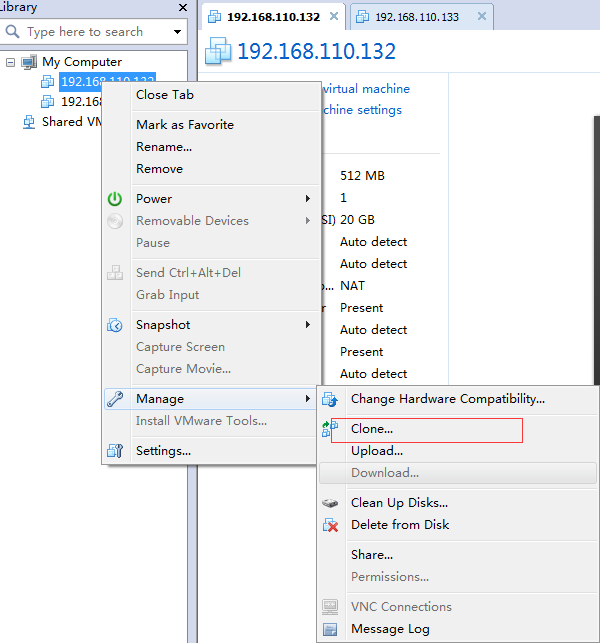
## 8.3、Java代码

|  |
| --- |
| **@Service**  **public class RedisService {**  **@Autowired**  **private StringRedisTemplate stringRedisTemplate;**  **public void setStr(String key, String value) {**  **setStr(key, value, null);**  **}**  **public void setStr(String key, String value, Long time) {**  **stringRedisTemplate.opsForValue().set(key, value);**  **if (time != null)**  **stringRedisTemplate.expire(key, time, TimeUnit.*SECONDS*);**  **}**  **public Object getKey(String key) {**  **return stringRedisTemplate.opsForValue().get(key);**  **}**  **public void delKey(String key) {**  **stringRedisTemplate.delete(key);**  **}**  **}** |

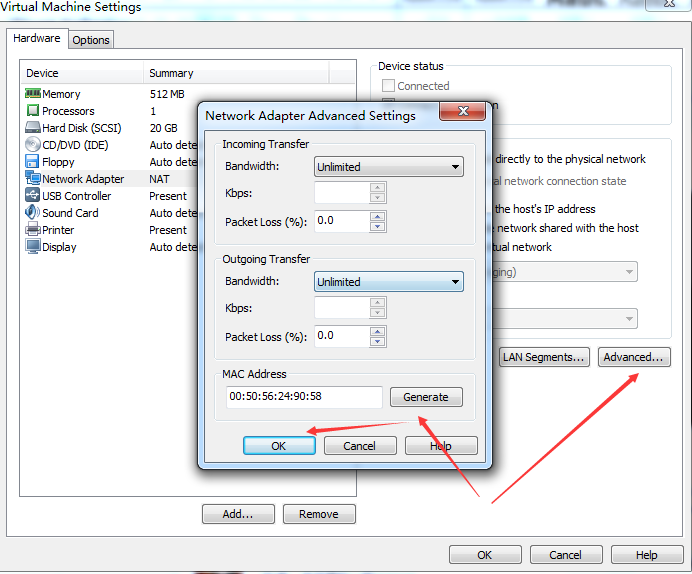
# Redis主从复制

克隆三台linux虚拟机

## 9.1.1、克隆虚拟机



## 9.1.2、生成新的mack地址



## 9.1.3、主从复制配置

redis主从复制

概述

1、redis的复制功能是支持多个数据库之间的数据同步。一类是主数据库（master）一类是从数据库（slave），主数据库可以进行读写操作，当发生写操作的时候自动将数据同步到从数据库，而从数据库一般是只读的，并接收主数据库同步过来的数据，一个主数据库可以有多个从数据库，而一个从数据库只能有一个主数据库。

2、通过redis的复制功能可以很好的实现数据库的读写分离，提高服务器的负载能力。主数据库主要进行写操作，而从数据库负责读操作。

主从复制过程

主从复制过程：见下图



**过程：**

1：当一个从数据库启动时，会向主数据库发送sync命令，

2：主数据库接收到sync命令后会开始在后台保存快照（执行rdb操作），并将保存期间接收到的命令缓存起来

3：当快照完成后，redis会将快照文件和所有缓存的命令发送给从数据库。

4：从数据库收到后，会载入快照文件并执行收到的缓存的命令。

#### 9.1.3.1 修改redis.conf

修改从redis中的 redis.conf文件

slaveof 192.168.33.130 6379

masterauth 123456--- 主redis服务器配置了密码,则需要配置

# Redis哨兵机制

## 10.1 什么是哨兵机制

Redis的哨兵(sentinel) 系统用于管理多个 Redis 服务器,该系统执行以下三个任务:

·        **监控(Monitoring)**: 哨兵(sentinel) 会不断地检查你的Master和Slave是否运作正常。

·        **提醒(Notification)**:当被监控的某个 Redis出现问题时, 哨兵(sentinel) 可以通过 API 向管理员或者其他应用程序发送通知。

·        **自动故障迁移(Automatic failover)**:当一个Master不能正常工作时，哨兵(sentinel) 会开始一次自动故障迁移操作,它会将失效Master的其中一个Slave升级为新的Master, 并让失效Master的其他Slave改为复制新的Master; 当客户端试图连接失效的Master时,集群也会向客户端返回新Master的地址,使得集群可以使用Master代替失效Master。

哨兵(sentinel) 是一个分布式系统,你可以在一个架构中运行多个哨兵(sentinel) 进程,这些进程使用流言协议(gossipprotocols)来接收关于Master是否下线的信息,并使用投票协议(agreement protocols)来决定是否执行自动故障迁移,以及选择哪个Slave作为新的Master.

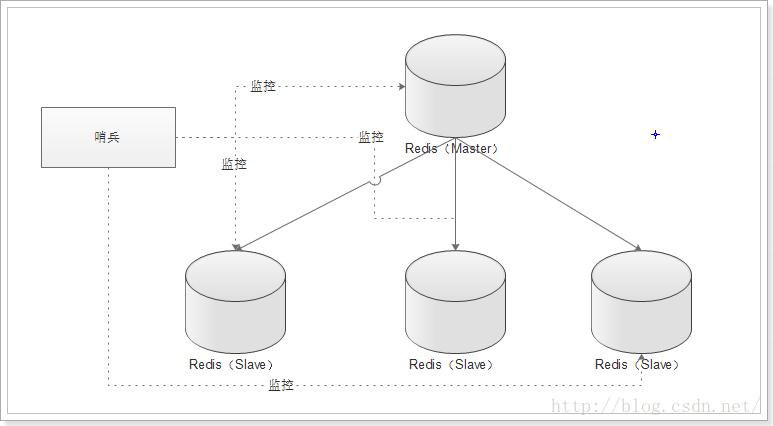
每个哨兵(sentinel) 会向其它哨兵(sentinel)、master、slave定时发送消息,以确认对方是否”活”着,如果发现对方在指定时间(可配置)内未回应,则暂时认为对方已挂(**所谓的”主观认为宕机” Subjective Down,简称sdown**).

若“哨兵群”中的多数sentinel,都报告某一master没响应,系统才认为该master"彻底死亡"(**即:客观上的真正down机,Objective Down,简称odown**),通过一定的vote算法,从剩下的slave节点中,选一台提升为master,然后自动修改相关配置.

虽然哨兵(sentinel) 释出为一个单独的可执行文件 redis-sentinel ,但实际上它只是一个运行在特殊模式下的 Redis 服务器，你可以在启动一个普通 Redis 服务器时通过给定 --sentinel 选项来启动哨兵(sentinel).

哨兵(sentinel) 的一些设计思路和zookeeper非常类似

单个哨兵(sentinel)



## 10.2 哨兵模式修改配置

实现步骤:

1.拷贝到etc目录

cp sentinel.conf /usr/local/redis/etc

2.修改sentinel.conf配置文件

sentinel monitor mymast 192.168.110.133 6379 1 #主节点 名称 IP 端口号 选举次数

3. 修改心跳检测 5000毫秒

sentinel down-after-milliseconds mymaster 5000

4.sentinel parallel-syncs mymaster 2 --- 做多多少合格节点

5. 启动哨兵模式

./redis-server /usr/local/redis/etc/sentinel.conf --sentinel &

6. 停止哨兵模式

# Redis事物

## 11.1、Redis事物

Redis 事务可以一次执行多个命令， 并且带有以下两个重要的保证：

事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

一个事务从开始到执行会经历以下三个阶段：

开始事务。

命令入队。

执行事务。

## 11.2、实例

以下是一个事务的例子， 它先以 MULTI 开始一个事务， 然后将多个命令入队到事务中， 最后由 EXEC 命令触发事务， 一并执行事务中的所有命令：

|  |
| --- |
| redis 127.0.0.1:6379> MULTI  OK  redis 127.0.0.1:6379> SET book-name "Mastering C++ in 21 days"  QUEUED  redis 127.0.0.1:6379> GET book-name  QUEUED  redis 127.0.0.1:6379> SADD tag "C++" "Programming" "Mastering Series"  QUEUED  redis 127.0.0.1:6379> SMEMBERS tag  QUEUED  redis 127.0.0.1:6379> EXEC  1) OK  2) "Mastering C++ in 21 days"  3) (integer) 3  4) 1) "Mastering Series"  2) "C++"  3) "Programming" |

## 11.3、Redis 事务命令

下表列出了 redis 事务的相关命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [DISCARD](http://www.runoob.com/redis/transactions-discard.html)  取消事务，放弃执行事务块内的所有命令。 |
| 2 | [EXEC](http://www.runoob.com/redis/transactions-exec.html)  执行所有事务块内的命令。 |
| 3 | [MULTI](http://www.runoob.com/redis/transactions-multi.html)  标记一个事务块的开始。 |
| 4 | [UNWATCH](http://www.runoob.com/redis/transactions-unwatch.html)  取消 WATCH 命令对所有 key 的监视。 |
| 5 | [WATCH key [key ...]](http://www.runoob.com/redis/transactions-watch.html)  监视一个(或多个) key ，如果在事务执行之前这个(或这些) key 被其他命令所改动，那么事务将被打断。 |

# Redis持久化

### 12.1、什么是Redis持久化

什么是Redis持久化,就是将内存数据保存到硬盘。

Redis 持久化存储 (AOF 与 RDB 两种模式)

### 12.2、RDB持久化

RDB 是在某个时间点将数据写入一个临时文件，持久化结束后，用这个临时文件替换上次持久化的文件，达到数据恢复。  
优点：**使用单独子进程来进行持久化，主进程不会进行任何 IO 操作，保证了 redis 的高性能**  
缺点：**RDB 是间隔一段时间进行持久化，如果持久化之间 redis 发生故障，会发生数据丢失。所以这种方式更适合数据要求不严谨的时候**

这里说的这个执行数据写入到临时文件的时间点是可以通过配置来自己确定的，通过配置**redis 在 n 秒内如果超过 m 个 key 被修改**这执行一次 RDB 操作。这个操作就类似于在这个时间点来保存一次 Redis 的所有数据，一次快照数据。所有这个持久化方法也通常叫做 snapshots。

RDB 默认开启，redis.conf 中的具体配置参数如下；

|  |
| --- |
| *#dbfilename：持久化数据存储在本地的文件*  dbfilename dump.rdb  *#dir：持久化数据存储在本地的路径，如果是在/redis/redis-3.0.6/src下启动的redis-cli，则数据会存储在当前src目录下*  dir ./  *##snapshot触发的时机，save*  *##如下为900秒后，至少有一个变更操作，才会snapshot*  *##对于此值的设置，需要谨慎，评估系统的变更操作密集程度*  *##可以通过“save “””来关闭snapshot功能*  *#save时间，以下分别表示更改了1个key时间隔900s进行持久化存储；更改了10个key300s进行存储；更改10000个key60s进行存储。*  save 900 1  save 300 10  save 60 10000  *##当snapshot时出现错误无法继续时，是否阻塞客户端“变更操作”，“错误”可能因为磁盘已满/磁盘故障/OS级别异常等*  stop-writes-on-bgsave-error yes  *##是否启用rdb文件压缩，默认为“yes”，压缩往往意味着“额外的cpu消耗”，同时也意味这较小的文件尺寸以及较短的网络传输时间*  rdbcompression yes |

### 12.3、AOF持久化

Append-only file，将“操作 + 数据”以格式化指令的方式追加到操作日志文件的尾部，在 append 操作返回后(已经写入到文件或者即将写入)，才进行实际的数据变更，“日志文件”保存了历史所有的操作过程；当 server 需要数据恢复时，可以直接 replay 此日志文件，即可还原所有的操作过程。AOF 相对可靠，它和 mysql 中 bin.log、apache.log、zookeeper 中 txn-log 简直异曲同工。AOF 文件内容是字符串，非常容易阅读和解析。  
优点：可以保持更高的数据完整性，如果设置追加 file 的时间是 1s，如果 redis 发生故障，最多会丢失 1s 的数据；且如果日志写入不完整支持 redis-check-aof 来进行日志修复；AOF 文件没被 rewrite 之前（文件过大时会对命令进行合并重写），可以删除其中的某些命令（比如误操作的 flushall）。  
缺点：AOF 文件比 RDB 文件大，且恢复速度慢。

我们可以简单的认为 AOF 就是日志文件，此文件只会记录“变更操作”(例如：set/del 等)，如果 server 中持续的大量变更操作，将会导致 AOF 文件非常的庞大，意味着 server 失效后，数据恢复的过程将会很长；事实上，一条数据经过多次变更，将会产生多条 AOF 记录，其实只要保存当前的状态，历史的操作记录是可以抛弃的；因为 AOF 持久化模式还伴生了“AOF rewrite”。  
AOF 的特性决定了它相对比较安全，如果你期望数据更少的丢失，那么可以采用 AOF 模式。如果 AOF 文件正在被写入时突然 server 失效，有可能导致文件的最后一次记录是不完整，你可以通过手工或者程序的方式去检测并修正不完整的记录，以便通过 aof 文件恢复能够正常；同时需要提醒，如果你的 redis 持久化手段中有 aof，那么在 server 故障失效后再次启动前，需要检测 aof 文件的完整性。

AOF 默认关闭，开启方法，修改配置文件 reds.conf：appendonly yes

|  |
| --- |
| *##此选项为aof功能的开关，默认为“no”，可以通过“yes”来开启aof功能*  *##只有在“yes”下，aof重写/文件同步等特性才会生效*  appendonly yes  *##指定aof文件名称*  appendfilename appendonly.aof  *##指定aof操作中文件同步策略，有三个合法值：always everysec no,默认为everysec*  appendfsync everysec  *##在aof-rewrite期间，appendfsync是否暂缓文件同步，"no"表示“不暂缓”，“yes”表示“暂缓”，默认为“no”*  no-appendfsync-on-rewrite no  *##aof文件rewrite触发的最小文件尺寸(mb,gb),只有大于此aof文件大于此尺寸是才会触发rewrite，默认“64mb”，建议“512mb”*  auto-aof-rewrite-min-size 64mb  *##相对于“上一次”rewrite，本次rewrite触发时aof文件应该增长的百分比。*  *##每一次rewrite之后，redis都会记录下此时“新aof”文件的大小(例如A)，那么当aof文件增长到A\*(1 + p)之后*  *##触发下一次rewrite，每一次aof记录的添加，都会检测当前aof文件的尺寸。*  auto-aof-rewrite-percentage 100 |

AOF 是文件操作，对于变更操作比较密集的 server，那么必将造成磁盘 IO 的负荷加重；此外 linux 对文件操作采取了“延迟写入”手段，即并非每次 write 操作都会触发实际磁盘操作，而是进入了 buffer 中，当 buffer 数据达到阀值时触发实际写入(也有其他时机)，这是 linux 对文件系统的优化，但是这却有可能带来隐患，如果 buffer 没有刷新到磁盘，此时物理机器失效(比如断电)，那么有可能导致最后一条或者多条 aof 记录的丢失。通过上述配置文件，可以得知 redis 提供了 3 中 aof 记录同步选项：

always：每一条 aof 记录都立即同步到文件，这是最安全的方式，也以为更多的磁盘操作和阻塞延迟，是 IO 开支较大。

everysec：每秒同步一次，性能和安全都比较中庸的方式，也是 redis 推荐的方式。如果遇到物理服务器故障，有可能导致最近一秒内 aof 记录丢失(可能为部分丢失)。

no：redis 并不直接调用文件同步，而是交给操作系统来处理，操作系统可以根据 buffer 填充情况 / 通道空闲时间等择机触发同步；这是一种普通的文件操作方式。性能较好，在物理服务器故障时，数据丢失量会因 OS 配置有关。

其实，我们可以选择的太少，everysec 是最佳的选择。如果你非常在意每个数据都极其可靠，建议你选择一款“关系性数据库”吧。  
AOF 文件会不断增大，它的大小直接影响“故障恢复”的时间, 而且 AOF 文件中历史操作是可以丢弃的。AOF rewrite 操作就是“压缩”AOF 文件的过程，当然 redis 并没有采用“基于原 aof 文件”来重写的方式，而是采取了类似 snapshot 的方式：基于 copy-on-write，全量遍历内存中数据，然后逐个序列到 aof 文件中。因此 AOF rewrite 能够正确反应当前内存数据的状态，这正是我们所需要的；\*rewrite 过程中，对于新的变更操作将仍然被写入到原 AOF 文件中，同时这些新的变更操作也会被 redis 收集起来(buffer，copy-on-write 方式下，最极端的可能是所有的 key 都在此期间被修改，将会耗费 2 倍内存)，当内存数据被全部写入到新的 aof 文件之后，收集的新的变更操作也将会一并追加到新的 aof 文件中，此后将会重命名新的 aof 文件为 appendonly.aof, 此后所有的操作都将被写入新的 aof 文件。如果在 rewrite 过程中，出现故障，将不会影响原 AOF 文件的正常工作，只有当 rewrite 完成之后才会切换文件，因为 rewrite 过程是比较可靠的。\*

触发 rewrite 的时机可以通过配置文件来声明，同时 redis 中可以通过 bgrewriteaof 指令人工干预。

redis-cli -h ip -p port bgrewriteaof

因为 rewrite 操作 /aof 记录同步 /snapshot 都消耗磁盘 IO，redis 采取了“schedule”策略：无论是“人工干预”还是系统触发，snapshot 和 rewrite 需要逐个被执行。

AOF rewrite 过程并不阻塞客户端请求。系统会开启一个子进程来完成。

### 12.4、AOF与RDB区别

*OF 和 RDB 各有优缺点，这是有它们各自的特点所决定：*

1) AOF 更加安全，可以将数据更加及时的同步到文件中，但是 AOF 需要较多的磁盘 IO 开支，AOF 文件尺寸较大，文件内容恢复数度相对较慢。  
\*2) snapshot，安全性较差，它是“正常时期”数据备份以及 master-slave 数据同步的最佳手段，文件尺寸较小，恢复数度较快。

可以通过配置文件来指定它们中的一种，或者同时使用它们(不建议同时使用)，或者全部禁用，在架构良好的环境中，master 通常使用 AOF，slave 使用 snapshot，主要原因是 master 需要首先确保数据完整性，它作为数据备份的第一选择；slave 提供只读服务(目前 slave 只能提供读取服务)，它的主要目的就是快速响应客户端 read 请求；但是如果你的 redis 运行在网络稳定性差 / 物理环境糟糕情况下，建议你 master 和 slave 均采取 AOF，这个在 master 和 slave 角色切换时，可以减少“人工数据备份”/“人工引导数据恢复”的时间成本；如果你的环境一切非常良好，且服务需要接收密集性的 write 操作，那么建议 master 采取 snapshot，而 slave 采用 AOF。

# Redis发布订阅

Redis 发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

Redis 客户端可以订阅任意数量的频道。

下图展示了频道 channel1 ， 以及订阅这个频道的三个客户端 —— client2 、 client5 和 client1 之间的关系：



当有新消息通过 PUBLISH 命令发送给频道 channel1 时， 这个消息就会被发送给订阅它的三个客户端：



### 13.1实例

以下实例演示了发布订阅是如何工作的。在我们实例中我们创建了订阅频道名为 **redisChat**:

redis 127.0.0.1:6379> SUBSCRIBE redisChat

Reading messages... (press Ctrl-C to quit)

1) "subscribe"

2) "redisChat"

3) (integer) 1

现在，我们先重新开启个 redis 客户端，然后在同一个频道 redisChat 发布两次消息，订阅者就能接收到消息。

redis 127.0.0.1:6379> PUBLISH redisChat "Redis is a great caching technique"

(integer) 1

redis 127.0.0.1:6379> PUBLISH redisChat "Learn redis by runoob.com"

(integer) 1

# 订阅者的客户端会显示如下消息

1) "message"

2) "redisChat"

3) "Redis is a great caching technique"

1) "message"

2) "redisChat"

3) "Learn redis by runoob.com"

## Redis 发布订阅命令

下表列出了 redis 发布订阅常用命令：

|  |  |
| --- | --- |
| **序号** | **命令及描述** |
| 1 | [PSUBSCRIBE pattern [pattern ...]](http://www.runoob.com/redis/pub-sub-psubscribe.html)  订阅一个或多个符合给定模式的频道。 |
| 2 | [PUBSUB subcommand [argument [argument ...]]](http://www.runoob.com/redis/pub-sub-pubsub.html)  查看订阅与发布系统状态。 |
| 3 | [PUBLISH channel message](http://www.runoob.com/redis/pub-sub-publish.html)  将信息发送到指定的频道。 |
| 4 | [PUNSUBSCRIBE [pattern [pattern ...]]](http://www.runoob.com/redis/pub-sub-punsubscribe.html)  退订所有给定模式的频道。 |
| 5 | [SUBSCRIBE channel [channel ...]](http://www.runoob.com/redis/pub-sub-subscribe.html)  订阅给定的一个或多个频道的信息。 |
| 6 | [UNSUBSCRIBE [channel [channel ...]]](http://www.runoob.com/redis/pub-sub-unsubscribe.html)  指退订给定的频道。 |

# Redis集群