Redis缓存数据库

# 认识Redis数据库

## NoSQL数据库简介

在世界上最热门的话题其实就是NoSQL数据库，可是对于数据库的认识，从传统的做法上来讲，最初只有关系型数据库，但是关系型数据库并不是数据库一直以来的持续发展。

关系型数据库的操作核心语法：SQL，但是SQL语法要求太严格了，而且在进行SQL处理的时候一旦处理不当就会产生大量的笛卡尔积，所以有很多的人不认可SQL操作，而这一部分人就主张使用命令来完成数据库的操作，例如：微软在90年代的时候有一个数据库：FoxPro，这个数据库最初的时候是只能够依靠自己的命令来进行处理，而后由于SQL的大力推广，后来FoxPro数据库不得已又支持了SQL语法。

### 关系型数据库的“问题”

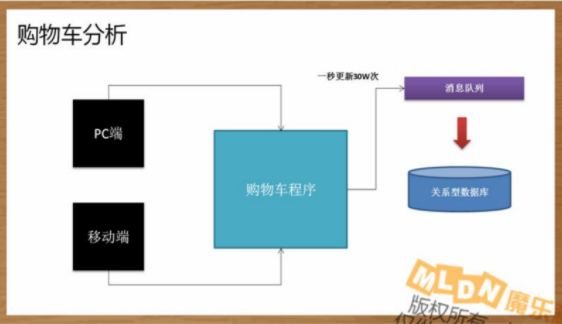
* 利用ACID原则（原子性，一致性，隔离性，持久性）保证数据完整性；
* 行列的规范化存储；
* 预定义结构；
* 存储数据量“小”；
* 结构化查询；

传统关系型中的事务控制是一个很好的机制，但是其本身有一个最大的缺点：速度慢（处理速度慢）。单节点的数据库你即使再厉害，你所在的服务器的硬件性能再好，你也无法承受几亿条数据。

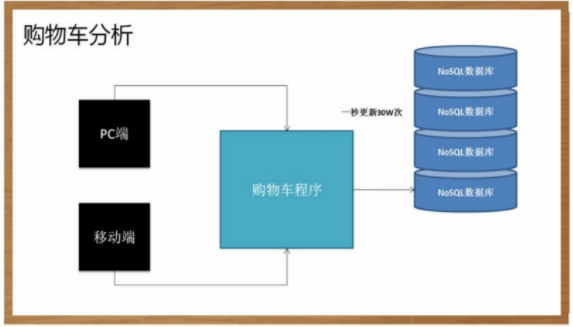
### NoSQL数据库

* NoSQL（NoSQL = Not Only SQL），即“不仅仅是SQL”；
* NoSQL特点：
  + 采用数据集存储；
  + 动态结构定义；
  + 存储精简；
  + 可扩展性强；
  + 适合云计算；

首先在实际的开发之中，关系型数据库不能够被替代，因为它存储的数据都是结构化的数据，但是一些操作频率较高的数据呢？例如：在所有的电商平台上都会存在有一个购物车功能，那么现在的技术开发要求考虑PC端、移动端数据通用。



但是这样的解决方案太麻烦了，因为你现在既然要操作的是数据库，数据库最好可以有着大并发数据访问的处理支持。而传统关系型数据库因为有了事务这一概念，才让整个的执行变得异常缓慢。



传统关系型数据库依然要进行保存，依然要保留原始的结构化数据库，而NoSQL要负责处理那些高并发的用户数据操作。

### NoSQL数据库分类

* 键值（Key-Value）存储数据库：采用Hash表结构存储（简单、易部署）
  + Tokyo Cabinet/Tyrant、Redis、Voldemort、Oracle BDB、MemacacheDB
* 列式存储数据库：应对分布式存储的海量数据，采用列族的形式保存数据
  + Cassandra、Hbase、Riak
* 文档型数据库：灵感是来自于Lotus Notes，采用类似JSON的形式存储。
  + CouchDB、MongoDB、SequoiaDB
* 图形（Graph）数据库：Neo4j、FlockDB
* 对象存储：通过类似面向对象语言的语法操作数据库
  + db4o、Versant
* xml数据库：搞笑的存储XML数据，并支持XML的内部查询语法，比如Xquery、Xpath。
  + Berkeley DB XML、BaseX

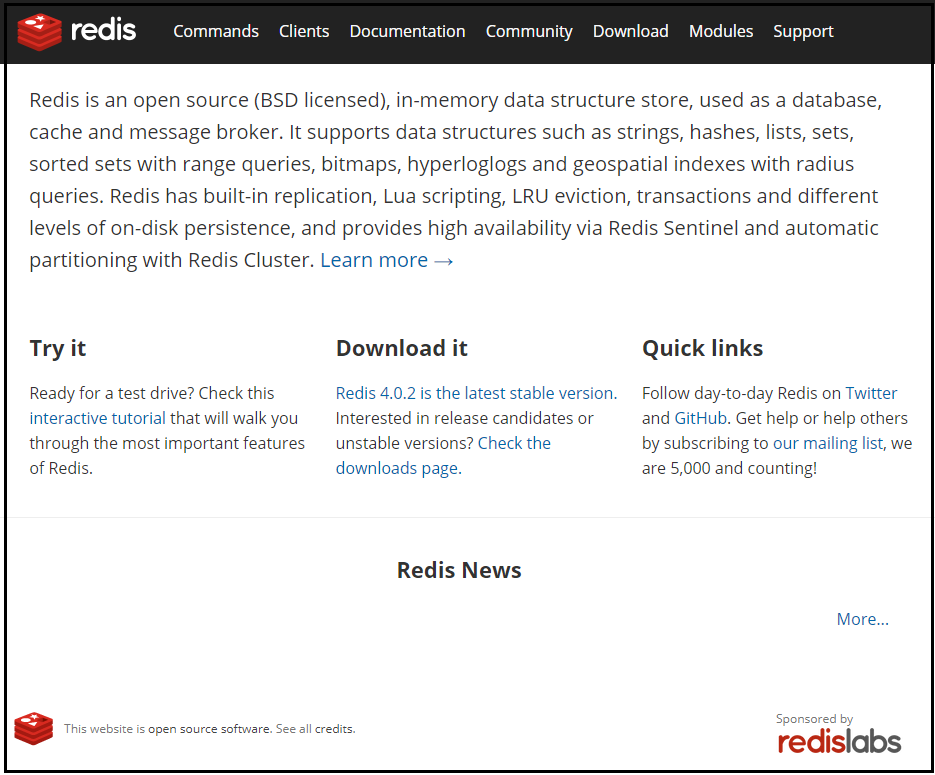
在现在的开发之中，国内使用率最高的NoSQL数据库：Redis、Memacached、MongoDB。

### 总结

如果你现在的项目之中没有NoSQL数据库，基本上就可以定性：属于传统项目，不需要考虑所谓的高并发处理机制，你项目的使用人群会比较小众。

## Redis数据库简介

首先Redis提供给用户的只是一个源代码的开发程序包。



如果要想学习Redis最好的做法还是看官方文档。

### Redis简介

* Redis是由意大利人Salvatore Sanfilippo（网名：antirez）开发的一款内存高速缓存数据库；
* Redis全称为：Remote Dictionary Server（远程数据服务），该软件使用C语言编写；
* Redis是一个开源的，先进的key-value存储可用于构建高性能，可扩展的Web应用程序的解决方案；
* 支持丰富的数据类型，如：string、list、set、zset(sorted set)、hash；
* Redis官方网站：<http://www.redis.io>

以后使用Redis的时候尽量选择3.0以上的版本，因为新版本才存在有一个官方提供的redis集群支持（RedisCluster）。

### Redis特点

* 异常快速：Redis是非常快的，每秒可以执行大约110000个设置操作，81000个/每秒的读取操作；
* 支持丰富的数据类型：Redis支持大多数开发人员已经知道的数据类型，例如列表、集合、可排序集合、哈希等数据类型；
* 这使得在应用中很容易解决的各种问题，因为我们知道哪些问题处理使用哪种数据类型更好解决；
* 原子性操作：所有Redis的操作都是源自，从而确保当两个客户同事访问Redis服务器得到的是更新后的值（最新值）；
* MultiUtility工具：Redis是一个多功能实用工具，可以在很多如：缓存，消息传递队列中使用（Redis原生支持发布/订阅），在应用程序中，如：web应用程序回话，网站页面点击数等任何短暂数据；

Redis整体的特点就是速度快，支持的数据类型多，但是并不只有Redis具备有这样的特点，Redis是属于缓存数据库的第二代产品，二第一代产品就是memcached数据库。

### Redis与Memcached

* Redis支持比memcached更多的数据类型；
* Redis支持主从结构（Master-Slave）可以实现数据备份；
* Redis支持数据持久化，可以将数据保存到磁盘之中，重启时依然可用。

Redis与Memacache有哪些不同？

Redis支持更多的数据类型，而且Redis数据库中的数据允许被持久化（保存到磁盘）。

# Redis安装与配置

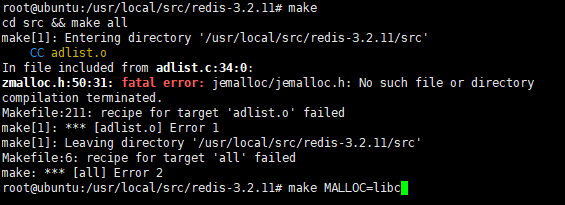
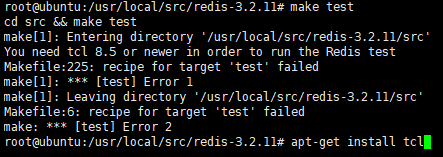
Redis给出的开发包只是源代码的开发包（C程序的源文件）、既然是C语言的文件，那么如果要想使用Redis，就必须进行代码的编译处理。理论上这些编译的环境可以在任意的操作系统之中进行，但是考虑到实际的应用环境，更多的是在Linux下完成。

## Redis编译与安装

如果非要在Windows中执行，那么可以下载一个cygwin的Unix模拟环境。

1. 在Linux环境下解压Redis开发包，建议解压缩到“/usr/local/src”目录之中；

|  |
| --- |
| tar xzvf redis-3.x.x.tar.gz –C /usr/local/src/ |

1. 进入到Redis源代码所在的安装目录：cd /usr/local/src/redis-3.x.x/；
2. 进行程序的编译处理，输入：make；
   1. 如果make没下的话就先下载make：apt-get install make；
   2. 安装gcc：apt-get install gcc；
   3. 还是报错的话：make MALLOC=libc；
   4.   
      apt-get install tcl；
   5. 经过以上4步就可以进行make编译了；
3. 编译完成之后要进行Redis安装，输入：make install；
4. Redis属于内存缓存数据库没那么如果你现在是一台单独的Redis服务器，则应该考虑将所有的可用内存都叫哥Redis来进行支配，所以理论上还需要执行如下的一行代码；

|  |
| --- |
| echo “vm.overcommit\_memory=1” >> /etc/sysctl.conf |

本次的操作指的是进行一次内存的分配策略，在进行设置的时候“vm.overconmmit\_memory”属性有如下三个取值：

* “0”：表示在进行处理的时候首先要检查是否有足够的内存供应，如果现在没有足够的内存供应，则无法分配，内存申请失败，如果有可用内存则直接进行申请开辟；
* “1”：表示将所有的内存都交给应用使用，而不关心当前的内存状态如何；
* “2”：表示允许分配超过所有物理内存和交换空间的内存的综合；

1. 将以上的配置写入到内核参数之中：/sbin/sysctl –p
2. 为了方便使用Redis数据库，那么建立一个Redis支持的命令工具目录：

|  |
| --- |
| mkdir –p /usr/local/redis/{bin,conf} |

1. 通过源代码目录拷贝出Redis所需要的程序运行文件：

|  |  |
| --- | --- |
| 1、拷贝Redis服务启动程序： | cp /usr/local/src/redis-3.x.x/src/redis-server /usr/local/redis/bin/ |
| 2、拷贝Redis命令行客户端： | cp /usr/local/src/redis-3.x.x/src/redis-cli /usr/local/redis/bin/ |
| 3、拷贝Redis性能测试工具： | cp /usr/local/src/redis-3.x.x/src/redis-benchmark /usr/local/redis/bin/ |

1. 拷贝出一个配置文件：

|  |
| --- |
| cp /usr/local/src/redis-3.x.x/redis.conf /usr/local/redis/conf/ |

这样就表示当前的Redis数据库已经将所有的程序文件安装成功。

## Redis数据库配置

如果要想配置Redis数据库，主要的配置文件就是“redis.conf”，所有的配置项一定要在此处完成。

1. Redis作为一个具备有持久化功能的缓存数据库，所以其一定会有一个用于数据库存储的目录，那么一般在Redis处理的时候会有三类文件需要做保存：Redis运行时的pid、Redis相关处理日志、Redis的数据文件，所以建立一个目录用于存放这些数据：

|  |
| --- |
| mkdir –p /usr/data/redis/{run,logs,dbcache} |

1. 修改redis.conf的配置文件：vim /usr/local/redis/conf/redis.conf;

|  |  |
| --- | --- |
| 1、配置Redis运行端口： | port 6379 |
| 2、配置Redis是否为后台运行： | daemonize yes |
| 3、设置进程保存路径： | pidfile /usr/data/redis/run/redis\_6379.pid |
| 4、设置日志保存目录： | logfile “/usr/data/redis/logs/redis.log” |
| 5、该Redis支持的数据库个数： | database 16 |
| 6、保存数据文件目录： | dir /usr/data/redis/dbcache |
| 7、开启保护模式： | protected-mode yes |

开启保护模式时，如果不设置密码，是无法远程访问的。

1. 启动Redis服务：/usr/local/redis/bin/redis-server /usr/local/redis/conf/redis.conf
   1. 如果要启动Redis-Server就必须明确的指明要使用的redis.conf配置文件；
2. Redis启动会占用6379的端口，所以查看端口：netstat –nptl；
3. 启动Redis客户端：

|  |  |
| --- | --- |
| 方法一：连接本机6379端口的Redis | /usr/local/redis/bin/redis-cli |
| 方法二：连接远程的格式 | /usr/local/redis/bin/redis-cli –h 127.0.0.1 –p 6379 |

1. 设置一个数据：set key-1 value-1
2. 取得数据：get key-1
3. 关闭Redis服务：
   1. 取得要关闭的Redis服务的进程，而后使用kill直接杀死；
   2. 直接使用killall redis-server干掉所有的Redis服务。

## 总结

apt-get install make

apt-get install ruby ruby-dev rubygems

# Redis数据操作

## redis-benchmark测试与操作命令

redis支持各种的数据类型，而且Redis操作的速度很快的，在Redis数据库里面有一个“redis-benchmark”性能测试工具，可以直接使用这个工具来观察Redis使用，该命令所需要的参数如下：

|  |
| --- |
| root@ubuntu:/usr/local/redis/bin# redis-benchmark --help  Usage: redis-benchmark [-h <host>] [-p <port>] [-c <clients>] [-n <requests]> [-k <boolean>]  -h <hostname> Server hostname (default 127.0.0.1)  -p <port> Server port (default 6379)  -s <socket> Server socket (overrides host and port)  -a <password> Password for Redis Auth  -c <clients> Number of parallel connections (default 50)  -n <requests> Total number of requests (default 100000)  -d <size> Data size of SET/GET value in bytes (default 2)  --dbnum <db> SELECT the specified db number (default 0)  -k <boolean> 1=keep alive 0=reconnect (default 1)  -r <keyspacelen> Use random keys for SET/GET/INCR, random values for SADD  Using this option the benchmark will expand the string \_\_rand\_int\_\_  inside an argument with a 12 digits number in the specified range  from 0 to keyspacelen-1. The substitution changes every time a command  is executed. Default tests use this to hit random keys in the  specified range.  -P <numreq> Pipeline <numreq> requests. Default 1 (no pipeline).  -e If server replies with errors, show them on stdout.  (no more than 1 error per second is displayed)  -q Quiet. Just show query/sec values  --csv Output in CSV format  -l Loop. Run the tests forever  -t <tests> Only run the comma separated list of tests. The test  names are the same as the ones produced as output.  -I Idle mode. Just open N idle connections and wait. |

范例：下面模拟1000个客户端进行数据的处理

|  |
| --- |
| /usr/local/redis/bin/redis-benchmark -h 127.0.0.1 -p 6379 -c 1000 -d 10 -n 10000 |

该命令的具体参数含义如下：

* “-h”：要连接的Redis服务器的地址；
* “-p”：要连接Redis的运行端口；
* “-c”：指的是模拟的客户端数量；
* “-d”：每一次操作的数据长度；
* “-n”：每一个用户发出的请求数量。

随后观察到的测试结果，会包含有如下的信息内容：

* “PING\_INLINE”： 53763.44 requests per second；
* “PING\_BULK”： 44843.05 requests per second；
* “SET”： 52083.33 requests per second；
* “GET”： 64102.56 requests per second；
* “INCR”： 52631.58 requests per second；
* “LPUSH”： 59171.60 requests per second；
* “RPUSH”： 44052.86 requests per second；
* “LPOP”： 50000.00 requests per second；
* “RPOP”： 59523.81 requests per second；
* “SADD”： 56179.77 requests per second；
* “HSET”： 58823.53 requests per second；
* “SPOP”： 51020.41 requests per second；
* “LPUSH”： 52631.58 requests per second；

会清楚的发现，Redis在整体的执行速度上的确是很快，毕竟现在的虚拟机的配置水平是很低的。Redis最大的特征在于可以进行各种数据类型的操作，包括：字符串、数字、集合（List、Set、Hash）。

## 字符串类型

在之前使用的set、get命令操作的都是字符串，而字符串支持的操作命令有如下几种：

* 设置操作数据：set key value；
* 查询数据：get key；
* 不覆盖设置：setnx key value；
* 设置数据有效期：setex key value；
* 设置多个key：mset key1 value1 key2 value2 ...；
* 不覆盖设置多个key：msetnx key1 value1 key2 value2 ...；
* 追加内容：append key追加内容；
* 取得数据长度：strlen key；
* 删除指定数据：del key1 key2 key3；

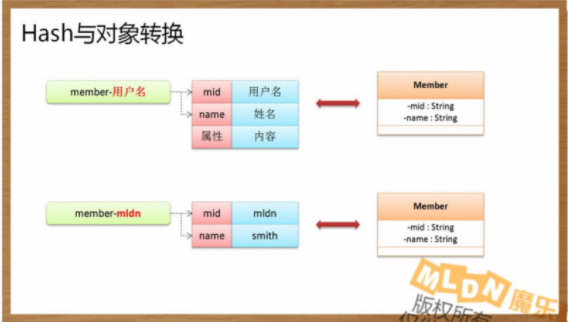
在Redis里面有一个特别重要的命令“keys \*”，可以进行全部数据的列出。其中后面的“\*”表示数据的匹配。

1. 设置新的数据：set username-kuhnwei kuhn;
2. 取得对应的数据：get username-kuhnwei;
   1. 如果在进行数据查询的时候没有对应的key的内容，则返回的是“(nil)”数据；
   2. 在数据取得的时候没有“\*”的统配，只有“keys \*”命令提供有这一操作。
3. 清空仓库：flushdb / flushall；
4. 不覆盖设置内容：setnx key value；
   1. 对于setnx的返回结果有如下两类：0（表示没有设置成功）、1（表示成功）；
   2. 如果你现在设置的两个数据的key的内容是相同的，默认情况下是会发生有覆盖的问题的；
5. 设置数据保存的有效时间：setex code 10 7x9d
   1. 现在就表示该设置的内容在10秒之后就会自动销毁；
   2. 在实际的使用之中验证码的信息保存就使用了此类数据类型，因为该类型的数据到点后自动消失；
   3. 在某一个key具备有指定的有效设置之后实际上可以使用“ttl key”查看当前剩余有效时间，如果该内容已经消失则返回“-2”，如果没有消失，则返回剩余的时间，或者在有效内使用“persist key”，让该key取消有效时间，这个时候如果使用ttl命令查看则返回的结果是“-1”；
6. 设置多个key的内容：mset key1 value1 key2 value2 key3 value3；
   1. 如果此时设置有相同的key的信息，那么默认会出现覆盖的问题；
7. 不覆盖的设置多个key内容：mest key1 value1 key2 value2；
   1. 当包含有一个重复的内容以及两个不重复的内容，因为有一个存在所以返回的是“0”，并且所有的内容都不允许设置；
8. 内容追加：append key value，返回的是当前的内容的长度；
9. 取得指定key的内容长度：strlen key；
10. 删除指定的数据内容：del key1 key2 key3；
    1. 如果现在指定的key不存在，也不会影响到程序的执行，只会删除掉存在的key的信息。

## Hash数据类型

Hash是一种最为常见的数据类型，其报酬的数据结构为“key = value”，但是需要提醒的是这个时候的key与value并不是redis中的key和value，hash是作为一种类型，可以这样理解关系：“RedisKey = HashValue”，利用hash类型可以报酬更多的数据。

* 存放hash数据：hset key filed value；
* 取得hash数据：hget key filed；
* 不覆盖设置：hsetnx key filed value；
* 批量设置：hmset key filed1 value1 filed2 value2 ...；
* 批量获取：hmget key filed1 filed2 ...；
* 判断某个数据是否存在：hexists key filed；
* 取得全部内容数量：hlen key；
* 取得所有key：hkeys key；
* 取得hash中所有内容：hgetall key；
* 数据加法：hincrby key 数字；



如果要想在某一个数据的key上保存更多的内容，那么就应该使用Hash类型完成。

## 数字操作

现在已经解除过了基本类型和Hash类型，可以发现，在整个的数据保存的时候都可以进行数字的保存，所以现在也可以针对于数字进行各种处理，处理的命令如下：

~ 普通数据类型：

* 自增处理：incr key;
* 自增指定处理：incrby key 数字；
* 自减处理：decr key；
* 自减指定数据：decrby key；

~ Hash数据类型：

* 数据加法：hincrby key filed 数字；

## List数据类型

List数据类型是一种比较麻烦的处理操作。在实际的项目开发里面，List就是一个链表结构，那么链表结构的时间复杂度是n，而且在进行链表数据处理的时候主要是进行内容的保存、节点的设置、递归遍历。

* 从左边入栈：lpush key value1 value2 value3 ...；
* 取得指定索引位置的内容：lrange key start stop；
* 从右边入栈：rpush key value1 value2 value3 ...；
* 在指定元素前/后追加内容：linsert key before|after pivot value；
* 修改指定索引的内容：lset key index value；
* 删除数据：lrem key count value；
* 保留指定范围的数据：ltrim key start stop；不在指定范围的数据将被删除；
* 栈顶元素出栈：lpop key；返回出栈元素；
* 栈底元素出栈：rpop key；返回出栈元素；
* 将移除的数据保存到另一个集合：rpoplpush source destination；
* 取得指定索引的内容：lindex key index；
* 取得集合长度（元素个数）：llen key；

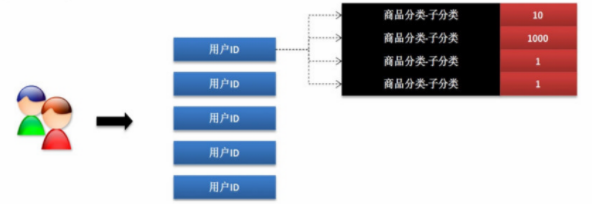
## Set数据类型

Set也是一种常见的数据类型，其最大的特征是基于数据的集合比对处理，例如：可以实现数据的交集、并集、差集。

* 向集合添加元素：sadd key value1 value2 ...；
* 查询set集合：smembers key；
* 删除集合元素：srem key value；
* 从集合中弹出元素（该元素自动进行删除处理）：spop key ...；
* 返回两个集合的差集合：sdiff key1 key2 ...；
* 返回的差集保存到新的集合之中：sdiffstore destination(存储集合) key1 key2 ...；
* 交集计算：sinter key1 key2；
* 将交集保存到新的集合之中：sinterstore destination(存储集合) key1 key2 ...；
* 并集计算，将两个集合合并在一起：sunion key1 key2 ...；
* 将并集保存到新的集合之中：sunionstore destination(存储集合) key1 key2 ...；
* 从key1中移除的元素添加到key2之中：smove key1 key2；
* 取得集合长度：scard key；
* 测试该指定内容是否是该集合的内容：sismember key member；
* 随机返回集合的一个内容，但不删除该内容：srandmember key；

## Redis数据类型总结

Redis严格来讲由于其处理的速度很快，某种程度上已经不完全属于数据库的支持了，所以对于数据类型我个人可以给出一些简单的总结：

1. 基本类型（String、int）：基本类型的操作更加适合于用户进行短期的数据存储，因为在实际的开发之中，字符串可以进行各种复杂操作，而且字符串也可以描述出各种数据的含义。那么在实际开发之中，可以利用此类型使用Nginx的集群数据保存、Shiro的集群数据保存、SpringData数据保存（序列化）、JSON数据的保存；
2. Hash类型：Hash类型更多的情况下描述的是一个结构化的信息，但是这种结构化的信息个人认为不如对象序列化好用，但是至少需要知道Hash这样的类型可以进行内容的详细分类；
3. List（栈、队列）：在实际的开发之中可以利用此类型实现消息队列的功能，或者进行缓冲的功能，很多公司有可能不直接使用消息队列中间件，而直接李立勇Redis替代。
4. Set数据类型：最大的支持功能在于集合的运算上，例如：相似度检测、好友推荐等等，都可以通过这种集合处理方式来完成。
5. SortedSet：主要进行数据的流式分析：
   1. 就一用户浏览商品操作为例。  
      

# Redis深入操作

## Redis事务控制

Redis本身支持事务处理，但是这种支持的事务处理本身是存在设计缺陷的，而且与传统的关系型数据库的事务处理有些不同，首先先来看一下Redis中的事务支持命令：

* 打开事务：multi；
* 取消事务：discard；
* 提交事务：exec；

范例：观察Redis中的事务：

|  |  |  |
| --- | --- | --- |
| 设置一个数据： | set age 30 | 返回OK |
| 打开事务支持： | multi | 返回OK |
| 进行数据操作： | set age 300 | 返回QUEUED |
| 进行数据操作： | set age 3000 | 返回QUEUED |
| 关闭事务： | discard | 返回OK |

一旦开启了Redis的事务控制，则表示所有的更新操作都要追加到一个更新队列之中。执行完数据操作之后，如果没有提交事务而是直接关闭事务，则执行的操作追加到的更新队列将会回滚到最初的状态，数据没有被改变。

|  |  |  |
| --- | --- | --- |
| 设置一个数据： | set age 30 | 返回OK |
| 打开事务支持： | multi | 返回OK |
| 进行数据操作： | set age 50 | 返回QUEUED |
| 提交事务： | exec | 返回1）OK |

如果在事务开启状态下进行了更新处理，随后只有执行了exec指令后才表示事务真正提交，才会真正影响到原始数据。

但是需要提醒的是，Redis设计之初就是不考虑事务的，所以以上的事务只能够说是Redis的一个玩笑，因为这种事务本身设计的并不完善。

|  |  |
| --- | --- |
| 设置一个数据： | set name kuhnwei |
| 打开事务支持： | multi |
| 进行数据操作： | incr name |
| 进行数据操作： | set age 70 |
| 提交事务： | exec |

这个时候一旦提交事务一定会出现错误“ERR value is not an integer or out of range”，因为name的数据不是数字，所以无法进行数字的增长。而且一旦出现了错误之后，其它的更新操作依然可以正常完成。

## Redis乐观锁

在数据库执行操作的时候，为了保证数据的一致性，即：A用户更新数据的时候B用户不能够更新。所谓的锁在数据库设计上分为两种：

* 悲观锁：基于数据库的操作实现；
  + select \* from table\_name where id=1 for update;
* 乐观锁：基于算法的实现，在数据表上追加一个锁的处理列；

在Redis里面是直接支持乐观锁的，如果要想观察乐观锁的处理，则可以打开两个不同的session来进行处理。

1、【第一个session】执行如下操作：

|  |  |  |
| --- | --- | --- |
| 设置一个数据： | set age 30 | 返回OK |
| 进行该数据的监听： | watch age | 返回OK |
| 启动事务： | multi | 返回OK |

这个时候并没有对age数据做任何的修改；

2、【第二个session】执行如下的操作：

|  |  |  |
| --- | --- | --- |
| 覆盖age数据： | set age 50 | 返回OK |
| 取得age数据： | get age | 返回“50” |

3、【第一个session】执行如下的操作：

|  |  |  |
| --- | --- | --- |
| 修改age数据： | set age 40 | 返回QUEUED |
| 提交事务： | exec | 返回(nil) |

此时由于第二个session已经更新了原始的数据，那么久表示该原始数据上的一个标记列更新，这样当第一个session再进行更新的时候会发现返回了“(nil)”，意味着本次更新失败。

## Redis密码配置

如果说你现在的Redis没有进行认证的处理操作，那么所有的程序都可以进行连接。直接使用redis-cli命令只需呀设置上主机名和端口号就可以进行连接处理。那么为了安全必须要设置密码。

在Redis数据库设计的时候并没有像其它数据库那样准备了一堆复杂的密码或者权限的概念，在Redis里面只需要设置一个认证密码就可以解决问题。

1. 修改redis.conf配置文件：vim /usr/local/redis/conf/redis.conf，追加一下内容：
   1. requirepass 650901
2. 随后关闭掉redis服务并且重新启动：
   1. 关闭redis服务：killall redis-server；
   2. 重启redis服务：/usr/local/redis/bin/redis-server /usr/local/redis/conf/redis.conf；
3. 登录redis服务器：redis-cli –h 127.0.0.1 –p 6379;
   1. 这个时候发现可以正常登录Redis数据库，但是一旦执行数据操作“keys \*”，那么就会出现如下错误信息：(error) NOAUTH Authentication required.
   2. 此时执行如下命令进行认证处理：auth 650901；
4. 直接在登录客户端的时候进行认证编写，使用一个“-a”的参数即可：

|  |
| --- |
| redis-cli –h 127.0.0.1 –p 6379 –a 650901 |

所有的Redis服务一定要有认证密码，而且作为Redis集群设计的时候一定要将所有密码统一。

## Redis性能监控

Redis是作为缓存数据库使用，那么一旦在开发之中使用了Redis，就表示有可能会大面积的去向Redis里面保存数据，那么现在如果要想知道当前的运行状态，那么久需要对其进行监控处理，如果要进行Redis监控，必须通过其它组件完成，本次使用一个“redis-stat”工具实现Redis监控操作，这个工具可以直接通过GitHub找到。

redis-stat下载地址：<https://github.com/junegunn/redis-stat>

1. 为了更加清楚的发现redis-stat特点，下面建立三个Redis运行进程，模拟方式很简单，配置不同的Redis端口即可，也就是说需要准备不同的redis.conf配置文件；
2. 建立redis数据的保存目录，要求可以同时保存三个Redis进程；

|  |
| --- |
| mkdir –p /usr/local/data/redis/{redis-6379,redis-6380,redis-6381}/{run,logs,dbcache} |

1. 将之前的redis.conf配置文件拷贝一份；

|  |
| --- |
| cp /usr/local/redis/conf/redis.conf /usr/local/redis/conf/redis-6379.conf |

1. 编辑每一个配置文件，修改各自的内容，以redis-6379为例：  
   vim /usr/local/redis/conf/redis-6379.conf

|  |  |
| --- | --- |
| 取消外网访问限制： | # bing 127.0.0.1 |
| 设置端口： | port 6379 |
| 设置pid保存路径： | pidfile /usr/local/redis/redis-6379/run/redis\_6379.pid |
| 设置日志文件路径： | logfile “/usr/local/redis/redis-6379/logs/redis.log” |
| 数据文件目录： | dir /usr/local/redis/redis-6379/dbcache |

1. 将redis-6379.conf复制为reidis-6380.conf、redis-6381.conf

|  |
| --- |
| cp /usr/local/redis/conf/redis-6379.conf /usr/local/redis/conf/redis-6380.conf  cp /usr/local/redis/conf/redis-6379.conf /usr/local/redis/conf/redis-6381.conf |

随后进入到每一个配置文件进行6379内容的更新：

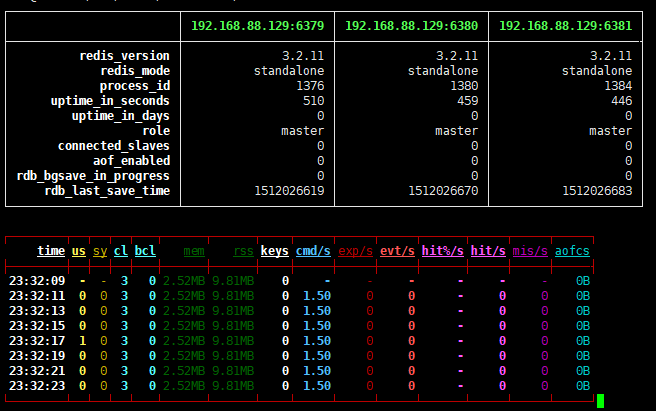
* 1,$s/6379/6380/g
* 1,$s/6379/6381/g

1. 启动所有的Redis服务：

|  |
| --- |
| /usr/local/redis/bin/redis-server /usr/local/redis/conf/redis-6379.conf  /usr/local/redis/bin/redis-server /usr/local/redis/conf/redis-6380.conf  /usr/local/redis/bin/redis-server /usr/local/redis/conf/redis-6381.conf |

1. 通过github下载redis-stat开发包：
   1. 如果要想使用redis-stat检查包必须下载ruby相关环境：apt-get install ruby ruby-dev rubygems;
   2. 将该工具下载到“/urs/local”目录之中：cd /usr/local/；
2. 下载下来的redis-stat里面实际上只有一个执行命令：/usr/local/redis-stat/bin/redis-stat;
   1. 进入到redis-stat所在目录：cd /usr/local/redis-stat/bin;
   2. 如果要想使用这个命令则必须使用ruby进行该命令的处理：gem install redis-stat；
3. 启动redis-stat工具进行监听控制；

|  |
| --- |
| redis-stat 192.168.88.129:6379 192.168.88.129:6380 192.168.88.129:6381 –a 650901 |

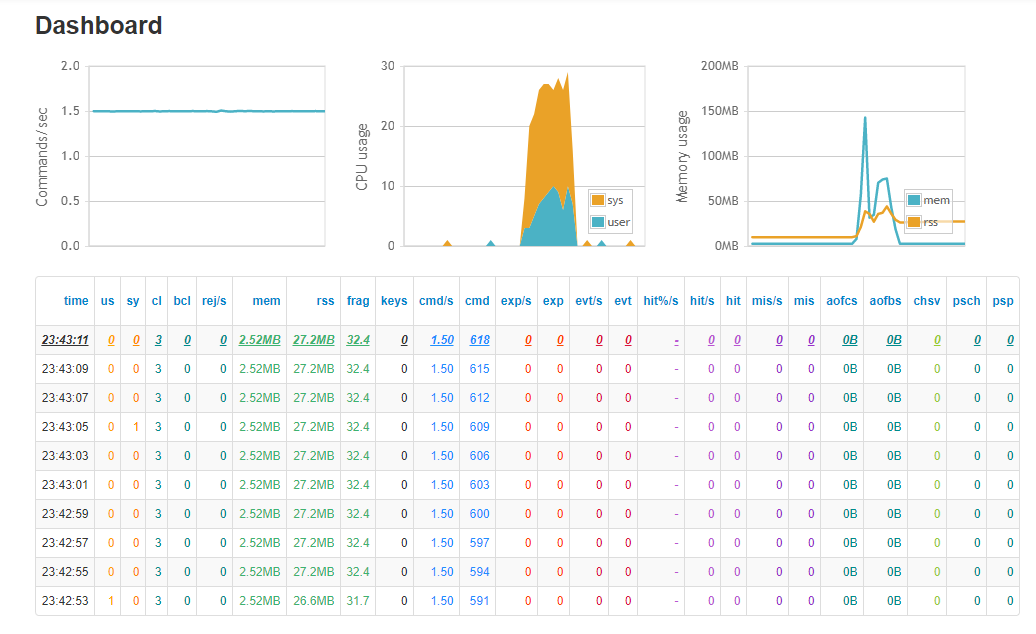


1. 该工具还支持web启动查看，也就是说它内部自动提供有一个HttpServer：

|  |
| --- |
| /usr/local/redis-stat/bin/redis-stat 192.168.88.129:6379 192.168.88.129:6380 192.168.88.129:6381 –a 650901 --server=80 --daemon --verbose |

1. 使用redis提供的工具来做测试：

|  |
| --- |
| /usr/local/redis/bin/redis-benchmark -h 192.168.88.129 -p 6379 -c 1000 -d 10 -n 10000 |
| /usr/local/redis/bin/redis-benchmark -h 192.168.88.129 -p 6380 -c 1000 -d 10 -n 10000 |
| /usr/local/redis/bin/redis-benchmark -h 192.168.88.129 -p 6381 -c 1000 -d 10 -n 10000 |



# 使用Java操作Redis

Redis本身提供有一个Java语言支持（不仅仅支持Java，各个语言都支持），这个组件为jedis，也就是说如果你要进行Redis编程开发，就需要通过Maven获取该组件。

|  |
| --- |
| <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.9.0</version>  </dependency> |

## 连接Redis数据库

1. 建立一个jedis的maven项目；
2. 修改pom.xml配置文件，追加相应的开发包以及插件；

|  |
| --- |
| <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>examples</artifactId>  <groupId>com.kuhnwei.examples</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>redis-in-java</artifactId>  <packaging>jar</packaging>  <name>redis-in-java</name>  <url>http://kuhnwei.com</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>  <dependencies>  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.9.0</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  </dependencies> </project> |

1. 如果要进行Redis数据库的连接，千万不要做地址的绑定处理。

|  |
| --- |
| package com.kuhnwei.examples.jedis; import redis.clients.jedis.Jedis; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/30 16:07  \*/* public class ConnectRedisServer {  public static final String *HOST* = "192.168.88.129";  public static final int *PORT* = 6379;  public static final String *PASSWORD* = "650901";  public static void main(String[] args) {  Jedis jedis = new Jedis(*HOST*, *PORT*);  jedis.auth(*PASSWORD*);  System.*out*.println(jedis.ping());  jedis.close();  } } |
| PONG |

如果此时服务器返回的内容为“PONG”，就表示连接成功。

## Jedis数据操作

Jedis如果要进行数据操作实际上与Redis使用的命令是相同的，也就是说整体的jedis处理的时候所使用的方法名称就是命令名称。

1. 设置字符串数据

|  |
| --- |
| package com.kuhnwei.examples.jedis; import redis.clients.jedis.Jedis; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/30 16:07  \*/* public class BaseDataDemo {  public static final String *HOST* = "192.168.88.129";  public static final int *PORT* = 6379;  public static final String *PASSWORD* = "650901";  public static void main(String[] args) {  Jedis jedis = new Jedis(*HOST*, *PORT*);  jedis.auth(*PASSWORD*);  jedis.set("keyStr", "Hello World.");  System.*out*.println(jedis.get("keyStr"));  jedis.close();  } } |

1. 设置有效时间的数据：

|  |
| --- |
| package com.kuhnwei.examples.jedis; import redis.clients.jedis.Jedis; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/30 16:07  \*/* public class BaseDataDemo {  public static final String *HOST* = "192.168.88.129";  public static final int *PORT* = 6379;  public static final String *PASSWORD* = "650901";  public static void main(String[] args) throws Exception{  Jedis jedis = new Jedis(*HOST*, *PORT*);  jedis.auth(*PASSWORD*);  jedis.setex("code", 3, "a3d4");  System.*out*.println("立即获取数据：" + jedis.get("code"));  Thread.*sleep*(3000);  System.*out*.println("3秒后获取数据：" + jedis.get("code"));  jedis.close();  } } |

1. 操作Hash操作：

|  |
| --- |
| package com.kuhnwei.examples.jedis; import redis.clients.jedis.Jedis; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/30 16:07  \*/* public class HashDataDemo {  public static final String *HOST* = "192.168.88.129";  public static final int *PORT* = 6379;  public static final String *PASSWORD* = "650901";  public static void main(String[] args) {  Jedis jedis = new Jedis(*HOST*, *PORT*);  jedis.auth(*PASSWORD*);  jedis.hset("keyMap", "id", "1");  jedis.hset("keyMap", "name", "kuhn");  System.*out*.println(jedis.hget("keyMap", "id"));  System.*out*.println(jedis.hget("keyMap", "name"));  jedis.close();  } } |

1. 操作List数据：

|  |
| --- |
| package com.kuhnwei.examples.jedis; import redis.clients.jedis.Jedis; import java.util.List; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/30 16:07  \*/* public class ListDataDemo {  public static final String *HOST* = "192.168.88.129";  public static final int *PORT* = 6379;  public static final String *PASSWORD* = "650901";  public static void main(String[] args) {  Jedis jedis = new Jedis(*HOST*, *PORT*);  jedis.auth(*PASSWORD*);  jedis.lpush("keyList", "1", "2", "3", "4", "5");  jedis.rpush("keyList", "1", "2", "3", "4", "5");  System.*out*.println(jedis.lpop("keyList"));  System.*out*.println(jedis.lpop("keyList"));  List<String> all = jedis.lrange("keyList", 0, -1);  System.*out*.println(all);  jedis.close();  } } |

1. 操作Set操作：

|  |
| --- |
| package com.kuhnwei.examples.jedis; import redis.clients.jedis.Jedis; import java.util.Set; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/30 16:07  \*/* public class SetDataDemo {  public static final String *HOST* = "192.168.88.129";  public static final int *PORT* = 6379;  public static final String *PASSWORD* = "650901";  public static void main(String[] args) {  Jedis jedis = new Jedis(*HOST*, *PORT*);  jedis.auth(*PASSWORD*);  jedis.sadd("keySet1", "1", "2", "3", "4", "5");  jedis.sadd("keySet2", "0", "9", "3", "4", "5");  Set<String> all = jedis.sinter("keySet1", "keySet2");  System.*out*.println(all);  System.*out*.println(jedis.smembers("keySet1"));  System.*out*.println(jedis.smembers("keySet2"));  jedis.close();  } } |

1. 操作SorteSet集合：

|  |
| --- |
| package com.kuhnwei.examples.jedis; import redis.clients.jedis.Jedis; import java.util.HashMap; import java.util.Map; import java.util.Set; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/30 16:07  \*/* public class SortedSetDataDemo {  public static final String *HOST* = "192.168.88.129";  public static final int *PORT* = 6379;  public static final String *PASSWORD* = "650901";  public static void main(String[] args) {  Jedis jedis = new Jedis(*HOST*, *PORT*);  jedis.auth(*PASSWORD*);  Map<String, Double> map = new HashMap<String, Double>(5);  map.put("pid-1-1", 2.0);  map.put("pid-1-2", 1.0);  map.put("pid-2-1", 5.0);  jedis.zadd("keySortedSet", map);  Set<String> zrangeByScore = jedis.zrangeByScore("keySortedSet", 1.0, 5.0);  System.*out*.println(zrangeByScore);  jedis.close();  } } |

1. keys处理操作，使用keys可以获取全部的key信息：

|  |
| --- |
| package com.kuhnwei.examples.jedis; import redis.clients.jedis.Jedis; import java.util.Set; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/30 16:07  \*/* public class KeysDemo {  public static final String *HOST* = "192.168.88.129";  public static final int *PORT* = 6379;  public static final String *PASSWORD* = "650901";  public static void main(String[] args) throws Exception{  Jedis jedis = new Jedis(*HOST*, *PORT*);  jedis.auth(*PASSWORD*);  Set<String> keys = jedis.keys("\*");  System.*out*.println(keys);  jedis.close();  } } |

从Jedis整体的设计风格，不会让使用者觉得其特别复杂，相反会觉得可以轻松上手。

## Jedis连接池

如果直接使用Jedis可以直接进行数据库的操作，但是每一次的打开和关闭实在是太浪费性能了，所以为了提升操作性能，一定要使用数据库连接池的模式来进行处理，幸运的是Jedis开发包直接提供了数据库连接池的操作支持。

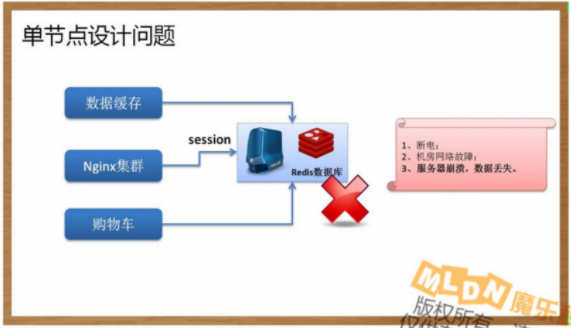
范例：实现数据库连接池

|  |
| --- |
| package com.kuhnwei.examples.jedis; import redis.clients.jedis.Jedis; import redis.clients.jedis.JedisPool; import redis.clients.jedis.JedisPoolConfig; */\*\*  \** ***@author*** *Kuhn Wei  \** ***@email*** *email@kuhnwei.com  \** ***@created*** *2017/11/30 16:07  \*/* public class JedisConnectionPool {  public static final String *HOST* = "192.168.88.129";  public static final int *PORT* = 6379;  public static final String *PASSWORD* = "650901";  public static final int *TIMEOUT* = 2000;  public static final int *MAX\_TOTAL* = 1000;  public static final int *MAX\_IDLE* = 200;  public static final int *MAX\_WAIT\_MILLIS* = 1000;  public static final boolean *TEST\_ON\_BORROW* = true;  public static void main(String[] args) throws Exception{  JedisPoolConfig poolConfig = new JedisPoolConfig();  // 设置最大连接数  poolConfig.setMaxTotal(*MAX\_TOTAL*);  // 设置空闲的连接上  poolConfig.setMaxIdle(*MAX\_IDLE*);  // 设置最大等待时间  poolConfig.setMaxWaitMillis(*MAX\_WAIT\_MILLIS*);  // 设置是否要进行连接测试，以保证返回的连接为可用连接  poolConfig.setTestOnBorrow(*TEST\_ON\_BORROW*);  JedisPool pool = new JedisPool(poolConfig, *HOST*, *PORT*, *TIMEOUT*, *PASSWORD*);  Jedis jedis = pool.getResource();  jedis.set("email", "email@kuhnwei.com");  jedis.close();  pool.close();  } } |

# Redis主从模式

## 主从模式简介

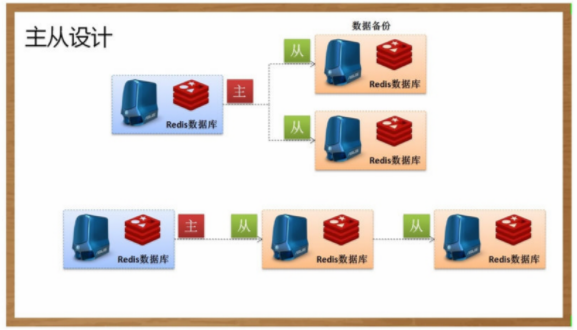
Redis数据库与传统数据库属于并行关系，也就是说传统的关系型数据库保存的是结构化数据，而Redis保存的是一些所谓的“临时”数据，因为Redis具备一项很强的功能“持久化数据”，发现Redis好像也可以做一些传统数据库的开发。但是现在Redis除了可以进行数据的存储之外，实际上也可以在一些系统的架构设计之中作为数据的缓冲点。



那么一旦出现了问题之后必须保证所有的数据可以进行快速恢复处理，所以这个时候数据备份就成为了一个核心主题，对于备份实际上有两种做法：

* 离线备份：停掉服务，拿硬盘拷贝；
* 在线备份：当数据做出任何处理操作的时候都进行及时的备份，所以这样的设计在Redis里面成为主从设计。

## 主从模式配置



如果要想实现主从模式的配置，首先一定要准备出三台Redis实例，本次为了方便将在一台主机上进行模拟，也就是说在这一台主机上将准备三个Redis实例，分别对应的端口为：6379、6380、6381，其中6379运行的Redis服务为主服务，二其它两个端口运行的服务为从服务。

1. 如果要想进行主从模式匹配，主服务器上不需要做出任何的变化，也就是说主服务器根本就不关心是否有从服务器；
2. 所有的从服务的配置文件（redis-6380.conf、redis-6381.conf）必须要求明确的设置出它对应的主服务器。
   1. 编辑redis-6380.conf配置文件：vim /usr/local/redis/conf/redis-6380.conf;

|  |  |
| --- | --- |
| 配置主服务器的IP地址 | slaveof 192.168.88.129 6379 |
| 设置主服务器的密码： | masterauth 650901 |

* 1. 随后redis-6381.conf配置文件采用与之一样的方式完成处理；

1. 启动所有的redis数据服务；

|  |
| --- |
| /usr/local/redis/bin/redis-server /usr/local/redis/conf/redis-6379.conf  /usr/local/redis/bin/redis-server /usr/local/redis/conf/redis-6380.conf  /usr/local/redis/bin/redis-server /usr/local/redis/conf/redis-6381.conf |

1. 登录6379端口的Redis服务（主服务）查看所有的副本信息；

|  |
| --- |
| /usr/local/redis/bin/redis-cli –h 192.168.88.129 –p 6379 –a 650901 info replication |
| # Replication  role:master  connected\_slaves:2  slave0:ip=192.168.88.129,port=6380,state=online,offset=15,lag=0  slave1:ip=192.168.88.129,port=6381,state=online,offset=15,lag=0  master\_repl\_offset:15  repl\_backlog\_active:1  repl\_backlog\_size:1048576  repl\_backlog\_first\_byte\_offset:2  repl\_backlog\_histlen:14 |

如果可以发现以上的信息就表示现在6379下有两个从节点。

1. 操作主节点（6379）的数据：set name KuhnWei
   1. 随后退出此客户端，随意登录6380或6381的主机：get name;

这个时候一定是通过主节点进行数据的设置，而后自动同步到所有的从节点上，这样的好处是可以进行数据的备份，如果你现在直接在从节点上操作，则会出现如下错误提示：“(error) READONLY You can't write against a read only slave.”

## 总结

主从设计的最大好处在于：可以自动对数据做备份；

主从设计的最大缺点在于：只能够做备份，而出现灾难之后无法立即恢复。

