Windows安装Redis详细教程

# 说明

Redis全称是“REmote DIctionary Server”，可以直接理解为远程字典服务，是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。

Redis是一个key-value存储系统，它支持存储的value类型包括string（字符串）、list（链表）、set（集合）、zset（sorted set --有序集合）和hash（哈希类型）。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，Redis支持各种不同方式的排序。为了保证效率，数据都是缓存在内存中，并且会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，以及在此基础上实现master-slave（主从）同步。

Redis 是一个高性能的key-value数据库。Redis的出现，很大程度补偿了memcached这类key/value存储的不足，在部 分场合可以对关系数据库起到很好的补充作用。它提供了Java，C/C++，C#，PHP，JavaScript，Perl，Object-C，Python，Ruby，Erlang等客户端，使用很方便。

Redis支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。这使得Redis可执行单层树复制。存盘可以有意无意的对数据进行写操作。由于完全实现了发布/订阅机制，使得从数据库在任何地方同步树时，可订阅一个频道并接收主服务器完整的消息发布记录。同步对读取操作的可扩展性和数据冗余很有帮助。

Redis 最适合处理的是小而热，而且是写频繁，或者读写都比较频繁的热数据。对于大而热的数据，如果其它方式很难解决问题，也可以考虑使用 Redis 解决，但是一定要非常谨慎，这种类型数据很容易引起数据膨胀，导致Redis消耗内存巨大，让系统难以承受，应对这种类型的数据，可以用普通存储 + 缓存的方式。

Redis安装教程基于以下系统软件环境编写：

（1）操作环境Win7（64位）

（2）Redis 安装包（当前教程版本2.8.21）

## 安装步骤

（1）在D盘新建文件夹redis，右键解压Redis压缩包，把所有文件解压到redis文件夹中

（2）配置redis.windows.conf文件，修改port、maxheap、maxmemory和requirepass参数

（3）制作startup.bat服务器启动文件

（4）制作start-client.bat客户端启动文件

## 文件清单

（1）redis-benchmark.exe # 基准测试

（2）redis-check-aof.exe # aof

（3）redischeck-dump.exe # dump

（4）redis-cli.exe # 客户端

（5）redis-server.exe # 服务器

（6）redis.windows.conf # 配置文件

## 运行Redis

（1）windows 运行（快捷键：windows键+R键），输入cmd命令，打开DOC窗口。

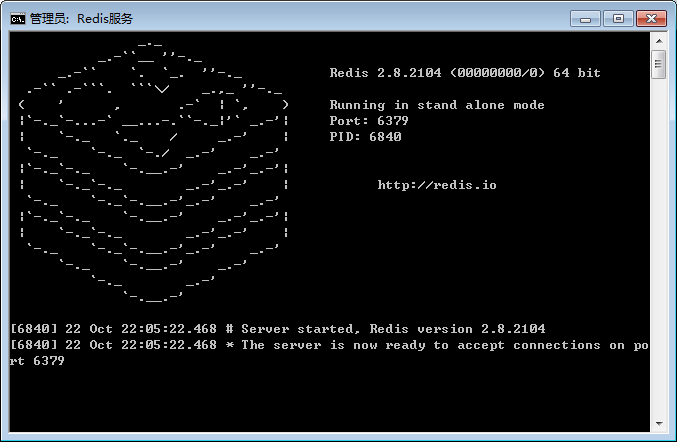
（2）执行命令cd D:\redis进入redis安装目录

（3）执行命令redis-server.exe redis.windows.conf，启动Redis 服务

注：也可以编写一份startup.bat批处理文件，内容：

|  |
| --- |
| @echo off  title Redis服务  redis-server.exe redis.windows.conf |

（4）服务启动成功状态



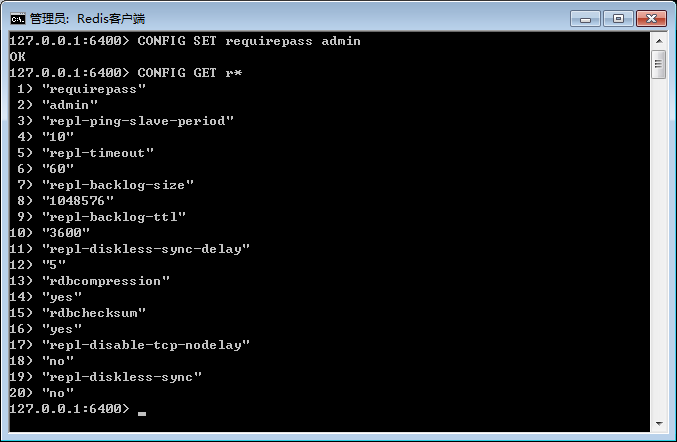
（5）附加几个 bat 批处理脚本，可根据需要灵活配置（可在参数中指定额外配置，例如--port）

|  |  |  |
| --- | --- | --- |
| **变量名** | **类型** | **描述** |
| service-install.bat | 安装服务 | redis-server.exe --service-install redis.windows.conf --port 6400 |
| uninstall-service.bat | 卸载服务 | redis-server --service-uninstall |
| startup.bat | 启动 | redis-server.exe redis.windows.conf |
| start-client.bat | 客户端 | redis-cli.exe -h 127.0.0.1 -p 6400 -a xmgps |

## 运行Redis客户端

（1）运行redis-cli.exe启动自带的客户端工具，也可在启动参数中指定参数后启动，例如：redis-cli.exe -h 127.0.0.1 -p 6400 -a password。

客户端运行成功后，可执行CONFIG SET命令在Redis服务运行时动态地调整 Redis 服务器的配置而无须重启Redis服务。CONFIG GET命令用于取得运行中的 Redis 服务器的配置参数。



## 配置

（1）编辑redis.windows.conf

|  |  |  |
| --- | --- | --- |
| **变量名** | **类型** | **描述** |
| port | 端口 | 监听端口，默认为6379 |
| bind |  | 指定Redis只接收来自于该IP 地址的请求，如果不进行设置，那么将处理所有请求 |
| daemonize |  | 是否以后台daemon方式运行，在windows下不受支持 |
| timeout |  | 设置客户端连接时的超时时间，单位为秒。当客户端在这段时间内没有发出任何指令，那么关闭该连接 |
| loglevel | 日志存储 | 日志显示级别：debug|verbose|notice|warning |
| logfile | 日志存储 | 指定日志输出的文件名，也可指定到标准输出端口 |
| databases |  | 设置最大连接数据库的数量，默认最大是16 |
| save | 持久化策略 | 设置Redis 进行数据库镜像的频率，例如当有10条Keys数据被改变时，300秒刷新到disk一次 |
| rdbcompression | 持久化策略 | 设置在进行镜像备份时，是否进行压缩，默认yes |
| dbfilename | 持久化策略 | 镜像备份文件的文件名，默认dump.rdb |
| dir | 持久化策略 | 镜像备份文件放置的目录，默认./ |
| slaveof | 主从配置 | 设置该数据库为其他数据库的从数据库 |
| masterauth | 主从配置 | 连接主数据库需要的密码 |
| requirepass | 安全性 | 设置客户端连接后进行任何其他指定前需要使用的密码 |
| maxclients | LIMITS | 最大客户端连接数，默认10000 |
| maxheap | LIMITS | 最大内存映射字节数，1gb => 1024\*1024\*1024 bytes |
| maxmemory | LIMITS | 设置redis能够使用的最大内存，单位字节数。当内存满了的时候，如果还接收到set命令，redis将先尝试剔除设置过expire 信息的key，而不管该key 的过期时间还没有到达。在删除时，将按照过期时间进行删除，最早将要被过期的key将最先被删除。如果带有expire 信息的key都删光了，那么将返回错误。这样，redis将不再接收写请求，只接收get请求。 |
| heapdir | 内存存储 | 指定内存映射文件路径；未指定时Windows下启动Redis后就在C:\Users\...\AppData\Local\Redis目录下生成RedisQFork文件, 关闭redis此文件自动删除。 |
| appendonly | APPEND ONLY MODE | 是否开启日志功能，默认no |
| appendfilename | 持久化策略 | 默认appendonly.aof |
| appendfsync | 持久化策略 | always|everysec|no |

# Redis数据存储

redis 的存储分为内存存储、磁盘存储和log文件三部分，配置文件中有三个参数对其进行配置。

（1）save <seconds> <changes>：在指定时间内，达到多少次更新操作时，就将数据同步到数据文件，并且可以添加多个条件配合。

（2）appendonly yes|no：是否在每次更新操作后进行日志记录，如果不开启，可能会在断电时导致一段时间内的数据丢失，因为redis本身同步数据文件是按上面的save条件来同步的，所以有的数据会在一段时间内只存在于内存中。

（3）appendfsync always|everysec|no：no表示等操作系统进行数据缓存同步到磁盘，always表示每次更新操作后手动调用fsync()将数据写到磁盘，everysec表示每秒同步一次。

## RDB快照

Redis借助了fork命令的copy on write机制，将当前数据的快照存成一个数据文件。在生成快照时，将当前进程fork出一个子进程，然后在子进程中循环所有的数据，将数据写成为RDB文件。Redis的RDB文件不会坏掉，因为其写操作是在一个新进程中进行的，当生成一个新的RDB文件时，Redis生成的子进程会先将数据写到一个临时文件中，然后通过原子性rename系统调用将临时文件重命名为RDB文件，这样在任何时候出现故障，Redis的RDB文件都总是可用的。

RDB也有他的不足，一旦数据库出现问题， RDB文件中保存的数据并不一定是全新的，从上次RDB文件生成到Redis停机这段时间的数据全部丢掉了。在某些业务下，这是可以忍受的，我们也推荐这些业务使用RDB的方式进行持久化，因为开启RDB的代价并不高。但是对于另外一些对数据安全性要求极高的应用，无法容忍数据丢失的应用，RDB就无能为力了，所以Redis引入了另一个重要的持久化机制：AOF日志。

## AOF日志

AOF日志的全称是append only file，从名字上我们就能看出来，它是一个追加写入的日志文件。与一般数据库的binlog不同的是，AOF文件是可识别的纯文本，它的内容就是一个个的Redis标准命令。

AOF文件会越来越大，所以Redis又提供了一个功能，叫做AOF rewrite。其功能就是重新生成一份AOF文件，新的AOF文件中一条记录的操作只会有一次，而不像一份老文件那样，可能记录了对同一个值的多次操作。其生成过程和RDB类似，也是fork一个进程，直接遍历数据，写入新的AOF临时文件。在写入新文件的过程中，所有的写操作日志还是会写到原来老的AOF文件中，同时还会记录在内存缓冲区中。当重完操作完成后，会将所有缓冲区中的日志一次性写入到临时文件中。然后调用原子性的rename命令用新的AOF文件取代老的AOF文件。

RDB和AOF操作都是顺序IO操作，性能都很高。而同时在通过RDB文件或者AOF日志进行数据库恢复的时候，也是顺序的读取数据加载到内存中。

## 数据导入

Redis是一个内存数据库，无论是RDB还是AOF，都只是其保证数据恢复的措施。所以Redis在利用RDB和AOF进行恢复的时候，都会读取RDB或AOF文件，重新加载到内存中。

而在利用RDB和利用AOF启动上，其启动时间有一些差别。RDB的启动时间会更短，原因有两个，一是RDB文件中每一条数据只有一条记录，不会像AOF日志那样可能有一条数据的多次操作记录。所以每条数据只需要写一次就行了。另一个原因是RDB文件的存储格式和Redis数据在内存中的编码格式是一致的，不需要再进行数据编码工作，在CPU消耗上要远小于AOF日志的加载。

# Redis数据类型

Redis最为常用的数据类型主要有以下五种：

（1）string

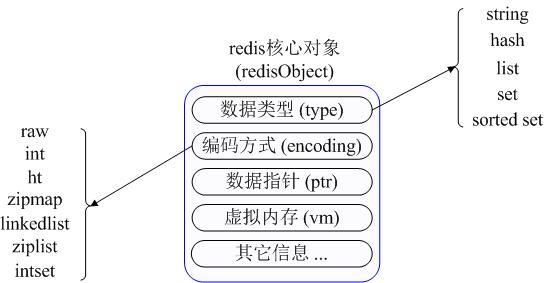
（2）hash

（3）list

（4）set

（5）sorted set

首先Redis内部使用一个redisObject对象来表示所有的key和value，redisObject最主要的信息如下图所示：type代表一个value对象具体是何种数据类型，encoding是不同数据类型在redis内部的存储方式，比如type=string代表value存储的是一个普通字符串，那么对应的encoding可以是raw或者是int，如果是int则代表实际redis内部是按数值型类存储和表示这个字符串的，当然前提是这个字符串本身可以用数值表示，比如"123"、"456"这样的字符串。



## string类型

（1）常用命令

set、get、decr、incr、mget等。

（2）应用场景

string是最常用的一种数据类型，普通的key/value存储都可以归为此类。

（3）实现方式

string在redis内部存储默认就是一个字符串，被redisObject所引用，当遇到incr、decr等操作时会转成数值型进行计算，此时redisObject的encoding字段为int。

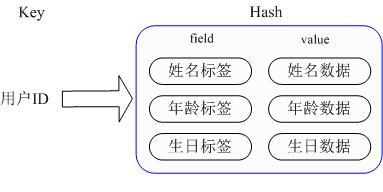
## hash类型

（1）常用命令

hget、hset、hgetall等。

（2）应用场景

hash实际存储的value为一个HashMap，并提供了直接存取这个Map成员的接口，如下图：



也就是说， value是一个Map，这个Map的key是成员的属性名，value是属性值，这样对数据的修改和存取都可以直接通过其内部Map的key（Redis称内部Map的key为field），既不需要重复存储数据，也不会带来序列化和并发修改控制的问题。

（3）实现方式

hash的成员比较少时Redis为了节省内存会采用类似一维数组的方式来紧凑存储，而不会采用真正的HashMap结构，对应redisObject的encoding为zipmap，当成员数量增大时会自动转成真正的HashMap，此时encoding为ht。

## list类型

（1）常用命令

lpush、rpush、lpop、rpop、lrange等。

（2）应用场景

list的应用场景非常多，也是Redis最重要的数据结构之一，比如twitter的关注列表，粉丝列表等都可以用Redis的list结构来实现。

（3）实现方式

list的实现为一个双向链表，同时支持反向查找和遍历，更方便操作。不过带来了部分额外的内存开销，Redis内部的很多实现，包括发送缓冲队列等也都是用的这个数据结构。

## set类型

（1）常用命令

sadd、spop、smembers、sunion 等。

（2）应用场景

set对外提供的功能与list类似是一个列表的功能，特殊之处在于set是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，set是一个很好的选择，并且set提供了判断某个成员是否在一个set集合内的重要接口，这个也是list所不能提供的。

（3）实现方式

set 的内部实现是一个 value永远为null的HashMap，实际就是通过计算hash的方式来快速排重的，这也是set能提供判断一个成员是否在集合内的原因。

## sorted set类型

（1）常用命令

zadd、zrange、zrem、zcard等。

（2）应用场景

sorted set的使用场景与set类似，区别是set不是自动有序的，而sorted set可以通过用户额外提供一个优先级（score）的参数来为成员排序，并且是插入有序的，即自动排序。当你需要一个有序的并且不重复的集合列表，那么可以选择sorted set数据结构，比如twitter的public timeline可以以发表时间作为score来存储，这样获取时就是自动按时间排好序的。

（3）实现方式

sorted set的内部使用HashMap和跳跃表（SkipList）来保证数据的存储和有序，HashMap里放的是成员到score的映射，而跳跃表里存放的是所有的成员，排序依据是HashMap里存的score，使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。

毫无疑问，Redis开创了一种新的数据存储思路，使用Redis，我们不用在面对功能单调的数据库时，把精力放在如何把大象放进冰箱这样的问题上，而是利用Redis灵活多变的数据结构和数据操作，为不同的大象构建不同的冰箱。

# 开发指南

## Redis + Jedis

Jedis是官方提供的唯一Redis Client For Java Provider，可直接从Maven获取。

**<dependency>**

**<groupId>redis.clients</groupId>**

**<artifactId>jedis</artifactId>**

**<version>2.7.3</version>**

**</dependency>**

如果只是简单使用Jedis，以下这么几行代码足够：

Jedis jedis = **new** Jedis("10.11.20.140");

String keys = "name";

// 删数据

jedis.del(keys);

// 存数据

jedis.set(keys, "snowolf");

// 取数据

String value = jedis.get(keys);

System.out.println(value);

Jedis使用commons-pool完成池化实现，Properties配置文件如下。

#最大分配的对象数

redis.pool.maxActive=1024

#最大能够保持idel状态的对象数

redis.pool.maxIdle=200

#当池内没有返回对象时，最大等待时间

redis.pool.maxWait=1000

#当调用borrow Object方法时，是否进行有效性检查

redis.pool.testOnBorrow=true

#当调用return Object方法时，是否进行有效性检查

redis.pool.testOnReturn=true

#IP

redis.ip=127.0.0.1

#Port

redis.port=6379

在静态代码段中完成初始化。

**private** **static** JedisPool pool;

**static** {

    ResourceBundle bundle = ResourceBundle.getBundle("redis");

**if** (bundle == **null**) {

**throw** **new** IllegalArgumentException(

                "[redis.properties] is not found!");

    }

    JedisPoolConfig config = **new** JedisPoolConfig();

    config.setMaxActive(Integer.valueOf(bundle

            .getString("redis.pool.maxActive")));

    config.setMaxIdle(Integer.valueOf(bundle

            .getString("redis.pool.maxIdle")));

    config.setMaxWait(Long.valueOf(bundle.getString("redis.pool.maxWait")));

    config.setTestOnBorrow(Boolean.valueOf(bundle

            .getString("redis.pool.testOnBorrow")));

    config.setTestOnReturn(Boolean.valueOf(bundle

            .getString("redis.pool.testOnReturn")));

    pool = **new** JedisPool(config, bundle.getString("redis.ip"),

            Integer.valueOf(bundle.getString("redis.port")));

}

// 从池中获取一个Jedis对象

Jedis jedis = pool.getResource();

// 释放对象池

pool.returnResource(jedis);

一致性哈希：保留前面的JedisPoolConfig，新增两个Redis的IP（redis1.ip，redis2.ip），完成两个JedisShardInfo实例，并将其丢进List中。

JedisShardInfo jedisShardInfo1 = **new** JedisShardInfo(

                bundle.getString("redis1.ip"), Integer.valueOf(bundle                       .getString("redis.port")));

JedisShardInfo jedisShardInfo2 = **new** JedisShardInfo(

                bundle.getString("redis2.ip"), Integer.valueOf(bundle                       .getString("redis.port")));

List<JedisShardInfo> list = **new** LinkedList<JedisShardInfo>();

list.add(jedisShardInfo1);

list.add(jedisShardInfo2);

初始化ShardedJedisPool代替JedisPool：

ShardedJedisPool pool = **new** ShardedJedisPool(config, list);

改由ShardedJedis，获取Jedis对象：

// 从池中获取一个Jedis对象

ShardedJedis jedis = pool.getResource();