一.NoSQL介绍

        NoSQL（Not Only SQL），意为反SQL运动，是一项合新的数据库革命性运动，早期就有人提出，发展至2009年趋势越发高涨。它指的是非关系型的数据库。随着互联网web2.0网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。

        NoSQL是以key-value形式存储，和传统的关系型数据库不一样，不一定遵循传统数据库的一些基本要求，比如说遵循SQL标准（insert， delete，select）、ACID属性（事务处理）、表结构等等，这类数据库主要有以下特点：非关系型的、分布式的、开源的、水平可扩展的。

支持事务比较简单（不能满足现在的需求）

二.NoSQL特点

1.处理超大量的数据

2.运行在便宜的PC服务器集群上

3.击碎了性能瓶劲（性能高，无须优化）

三.NoSQL适用场景

1.对数据高并发读写（传统数据库上万次的写，硬盘I/O就会有问题）

2.对海量数据的高效率存储和访问（fielfeed：一个月有2.5亿用户动态）

3.对数据的高可扩展性和高可用性（nosql做分布式，很简单。没有固定的表结构，迭代开发时）

四.Redis的介绍

        Redis is an open source,advanced key-value store.It is often referred to as a data structure server since keys can contain strings,hashes,lists,sets and sorted sets.

        Redis是一个开源的，先进的key-value存储。它通常被称为数据结构服务器，因为键可以包含字符串，哈希，链表，集合和有序集合。

        Redis是一个Key-Value存储系统。它支持存储的value类型很多，包括string(字符串)、list(链表)、set(集合)、zset(有序集合)。这些数据类型都支持push/pop、add/remove及取交集和并集及更丰富的操作，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，它也可以**（方式一）**周期性的把更新的数据写入磁盘或者**（方式二）**把修改操作写入追加的记录文件（相当于log文件）。

链表可以做成“栈”（先进后出）和“队列”（先进先出）

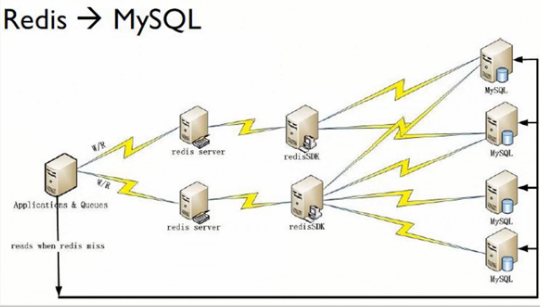
        Redis提供的API语言包括：C、C++、C#、Clojure、Common Lisp、Erlang、Haskell、Java、Javascript、Lua、Objective-C、Perl、PHP、Python、Ruby Scala、Go、Tcl。

        目前全球最大的Redis用户是新浪微博，在新浪有200多台物理机，400多个端口正在运行着Redis，有+4G的数据在Redis上来为微博用户提供服务。

        在新浪微博Redis的部署场景很多，大概分为如下的两种：

        a.应用程序直接访问Redis数据库

        b.应用程序直接访问Redis，只有当Redis访问失败时才访问MySQL



        Redis数据库提供多种灵活的数据结构和数据操作，为不同的数据提供不同的数据类型。

五.Redis具体适用场合

1.取最新N个数据的操作

2.排行榜应用，取TOP N操作

3.需要精确设定过期时间的应用

4.计数器应用

5.Uniq操作，获取某段时间所有数据排重值

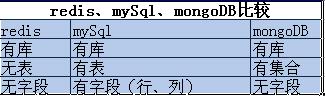
6.实时系统，反垃圾系统

7.Pub/Sub构建实时消息系统

8.构建队列系统

9.缓存

六.redis、mySql、mongoDB的比较



七.Redis的安装与部署

Redis的官方下载站是：http://redis.io/download

步骤一：下载安装包

        wget http://redis.goolecode.com/files/redis-2.4.17.tar.gz

步骤二：编译源程序

**Unix代码  [收藏代码](javascript:void())**

1. tar zxvf redis-2.4.17.tar.gz
2. cd redis-2.4.17
3. make
4. cd src && make install

步骤三：移动文件，便于管理

**Unix代码  [收藏代码](javascript:void())**

1. mkdir -p /usr/local/redis/bin
2. mkdir -p /usr/local/redis/etc
3. mv /lamp/redis-2.4.17/redis.conf /usr/local/redis/etc
4. cd /lamp/redis-2.4.17/src
5. mv mkreleasehdr.sh redis-benchmark redis-check-aof redis-check-dump redis-cli redis-server /usr/local/redis/bin

 步骤四：启动Redis服务

**Unix代码  [收藏代码](javascript:void())**

1. /usr/local/redis/bin/redis-server
2. /usr/local/redis/etc/redis.conf

        Redis服务端的默认连接端口是6379（mySql的端口号是：3306；mongoDB的端口号是：27017，28017）。

        默认情况下，Redis不是在后台运行的，我们需要把开启Redis的后台运行。

**Unix代码  [收藏代码](javascript:void())**

1. vi /usr/local/redis/redis.conf

        将daemonize的值改为yes。

        启动后看进程是否启动

**Unix代码  [收藏代码](javascript:void())**

1. ps -ef | grep redis

        看指定的端口号是否占用

**Unix代码  [收藏代码](javascript:void())**

1. netstat -tunpl | grep 6379

步骤五：客户端连接

**Unix代码  [收藏代码](javascript:void())**

1. /usr/local/redis/bin/redis-cli

步骤六：停止Redis实例

        我们可以使用

**Unix代码  [收藏代码](javascript:void())**

1. /usr/local/redis/bin/redis-cli shutdown

        也可以使用

**Unix代码  [收藏代码](javascript:void())**

1. pkill redis-server

        当然，也可以使用kill all,kill -9。

八.Redis的配置

daemonize：如果需要在后台运行，把该项心为yes

pidfile：配置多个pid的地址，默认在/var/run/redis.pid

bind：绑定ip，设置后只接受来自该ip的请求

port：监听端口，默认为6379

timeout：设置客户端连接时的超时时间，单位为秒

loglevel：分为4级，debug、verbose、notice、warning

logfile：配置log文件地址

databases：设置数据库的个数，默认使用的数据库为0

save：设置redis进行数据库镜像的频率

rdbcompression：在进行镜像备份时，是否进行压缩

Dbfilename：镜像备份文件的文件名

Dir：数据库镜像备份的文件放置路径

Slaveof：设置数据库为其他数据库的从数据库

Masterauth：主数据库连接需要的密码验证

Requirepass：设置登录登录时需要使用的密码

Maxclients：限制同时连接的客户数量

Maxmemory：设置redis能够使用的最大内存

Appendonly：开启append only模式

Appendfsync：设置对appendonly.aof文件同步的频率

vm-enabled：是否开启虚拟内存支持

vm-swap-file：设置虚拟内存的交换文件路径

vm-max-memory：设置redis使用的最大物理内存大小

vm-page-size：设置虚拟内存的页大小

vm-pages：设置交换文件的总的page数量

vm-max-threads：设置VMIO同时使用的线程数量

Glueoutputbuf：把小的输出缓存存放在一起

hash-max-zipmap-entries：设置hash的临界值

Activerehashing：重新hash

一.Redis的数据类型

1.String类型及操作

        String是最简单的类型，一个key对应一个value，string类型是二进制安全的。Redis的string可以包含任何数据，比如jpg图片或者序列化的对象。

**Set**方法：设置key对应的值为string类型的value

        例如：我们添加一个name=lili的键值对

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> set name lili
2. OK

**Setnx**：设置key对应的值为string类型的value，如果key已经存在，返回0，nx是not exist的意思。

        例如：我们添加一个name=lili\_new的键值对。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> get name
2. "lili"
3. redis 127.0.0.1:6379> setnx name lili\_new
4. (integer) 0
5. redis 127.0.0.1:6379> get name
6. "lili"
7. redis 127.0.0.1:6379> setnx age 20
8. (integer) 1

**Setex**：设置key对应的值为string类型的value，并指定此键值对应的有效期。

        例如：我们添加一个haircolor=red的键值对，并指定有效期为10秒

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> setex haircolor 10 red
2. OK
3. redis 127.0.0.1:6379> get haircolor
4. "red"
5. redis 127.0.0.1:6379> get haircolor
6. (nil)

        nil：表示空

**Setrange**：设置指定key的value值的子字符串

        例如：我们希望将lili的126邮箱替换为gmail邮箱

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> get name
2. "lili@126.com"
3. redis 127.0.0.1:6379> setrange name 6 gmail.com
4. (integer) 14
5. redis 127.0.0.1:6379> get name
6. "lili@gmail.com"
8. redis 127.0.0.1:6379> set email lijie@lampbrother.net
9. OK
10. redis 127.0.0.1:6379> get email
11. "lijie@lampbrother.net"
12. redis 127.0.0.1:6379> setrange email 6-li-jie.me
13. (integer) 21
14. redis 127.0.0.1:6379> get email
15. "lijie@li-jie.meer.net"

**mset**：一次设置多个key的值，成功返回ok表示所有的值都设置了，失败返回0表示没有任何值被设置。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> mset key1 lijie1 key2 lijie2
2. OK
3. redis 127.0.0.1:6379> get key1
4. "lijie1"
5. redis 127.0.0.1:6379> get key2
6. "lijie2"

**msetnx**：一次设置多个key的值，成功返回ok表示所有的值都设置了，失败返回0表示没有任何值被设置，但是不会覆盖已经存在的key。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> msetnx key3 lijie3 key2 30
2. (integer) 0
3. redis 127.0.0.1:6379> get key3
4. (nil)
5. redis 127.0.0.1:6379> msetnx key3 lijie3 key4 30
6. (integer) 1
7. redis 127.0.0.1:6379> get key3
8. "lijie3"
9. redis 127.0.0.1:6379> get key4
10. "30"

**get**：获取key对应的string值，如果key不存在返回nil。

**getset**：设置key的值，并返回key的旧值。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> getset key4 40
2. "30"
3. redis 127.0.0.1:6379> get key4
4. "40"

**getrange**：获取key的value值的子字符串。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> get email
2. "lijie@li-jie.meer.net"
3. redis 127.0.0.1:6379> getrange email 0 4
4. "lijie"

**mget**：一次获取多个key的值，如果对应key不存在则对应返回nil。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> mget key1 key2 key3
2. 1) lijie1
3. 2) lijie2
4. 3) lijie3

**incr**：对key的值做加加操作，并返回新的值。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> incr key4
2. (integer) 41
3. redis 127.0.0.1:6379> incr key4
4. (integer) 42
5. redis 127.0.0.1:6379> get key4
6. "42"
8. redis 127.0.0.1:6379> incr key7
9. (integer) 1
10. redis 127.0.0.1:6379> incr key7
11. (integer) 2
12. redis 127.0.0.1:6379> get key7
13. "2"

**incrby**：同incr类似，加指定值，key不存在时候会设置key，并认为原来的value是0。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> incrby key4 5
2. (integer) 47
3. redis 127.0.0.1:6379> get key4
4. "47"
5. redis 127.0.0.1:6379> incrby key4 -5
6. (integer) 42
7. redis 127.0.0.1:6379> get key4
8. "42"

**decr**：对key的值做减减操作。

**decrby**：同decr类似，减指定值。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> get key6
2. "39"
3. redis 127.0.0.1:6379> decr key6
4. (integer) 38
5. redis 127.0.0.1:6379> decrby key6 -3
6. (integer) 41
7. redis 127.0.0.1:6379> decrby key6 3
8. (integer) 38

**append**：给指定key的字符串追加value，返回新字符串值的长度。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> get name
2. "test"
3. redis 127.0.0.1:6379> append name .net
4. (integer) 8
5. redis 127.0.0.1:6379> get name
6. "test.net"

**strlen**：取指定key的value值的长度。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> strlen name
2. (integer) 8

2.hash类型及操作

        Redis hash是一个string类型的field和value的映射表。它的添加、删除操作都是0(1)(平均)。hash特别适合用于存储对象。相较于将对象的每个字段存成单个string类型。将一个对象存储在hash类型中会占用更少的内存，并且可以更方便的存取整个对象。

**hset**：设置hash field为指定值，如果key不存在，则先创建。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> hset user:001 name lijie
2. (integer) 1
3. redis 127.0.0.1:6379> hget user:001 name
4. "lijie"

**hsetnx**：设置hash field为指定值，如果key不存在，则先创建。如果存在返回0。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> hsetnx user:002 name lamp
2. (integer) 1
3. redis 127.0.0.1:6379> hget user:002 name
4. "lamp"
5. redis 127.0.0.1:6379> hsetnx user:002 name lampbrother
6. (integer) 0
7. redis 127.0.0.1:6379> hget user:002 name
8. "lamp"

**hmset**：同时设置hash的多个field。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> hmset user:003 name lijie age 20 sex 1
2. OK
3. redis 127.0.0.1:6379> hget user:003 name
4. "lijie"
5. redis 127.0.0.1:6379> hget user:003 age
6. "20"
7. redis 127.0.0.1:6379> hget user:003 sex
8. "1"

**hget**：获取指定的hash field。

**hmget**：获取全部指定的hash field。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> hmget user:003 name age sex
2. 1) "lijie"
3. 2) "20"
4. 3) "1"

**hincrby**：指定的hash field加上给定值。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> hincrby user:003 age 5
2. (integer) 25
3. redis 127.0.0.1:6379> hget user:003 age
4. "25"

**hexists**：测试指定field是否存在。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> hexists user:003 age
2. (integer) 1
3. redis 127.0.0.1:6379> hexists user:001 sex
4. (ingeger) 0

**hlen**：返回指定hash的field数量

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> hlen user:001
2. (integer) 1
3. redis 127.0.0.1:6379> hlen user:003
4. (integer) 3

**hdel**：删除指定hash的field

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> hget user:003 age
2. "25"
3. redis 127.0.0.1:6379> hdel user:003 age
4. (integer) 1
5. redis 127.0.0.1:6379> hget user:003 age
6. (nil)

**hkeys**：返回hash的所有field

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> hkeys user:001
2. 1) "name"
3. redis 127.0.0.1:6379> hkeys user:003
4. 1) "name"
5. 2) "sex"

**hvals**：返回hash的所有value

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> hvals user:003
2. 1) "lijie"
3. 2) "1"

**hgetall**：获取某个hash中全部的field及value

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> hgetall user:003
2. 1) "name"
3. 2) "lijie"
4. 3) "sex"
5. 4) "1"

**3.list类型及操作**

        List是一个链表结构，主要功能是push、pop、获取一个范围的所有值等等，操作key理解为链表的名字。Redis的list类型其实就是一个每个子元素都是string类型的双向链表。我们可以通过push、pop操作从链表的头部或者尾部添加删除元素，这样list既可以作为栈，又可以作为队列。k

**lpush**:在key对应list的头部添加字符串元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> lpush mylist "world"
2. (integer) 1
3. redis 127.0.0.1:6379> lpush mylist "hello"
4. (integer) 2
5. redis 127.0.0.1:6379> lrange mylist 0 -1
6. 1) "hello"
7. 2) "world"

        0代表头部第一个元素，-1代表尾部第一个元素。lrange mylist 0 -1表示从list中头第一个元素一直到尾第一个元素依次取出来，即第一个元素取到最后一个元素。

**rpush**:在key对应list的尾部添加字符串元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> rpush mylist2 "world"
2. (integer) 1
3. redis 127.0.0.1:6379> rpush mylist2 "hello"
4. (integer) 2
5. redis 127.0.0.1:6379> lrange mylist2 0 -1
6. 1) "world"
7. 2) "hello"

**linsert**:在key对应list的特定位置前或后添加字符串

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> rpush mylist3 "world"
2. (integer) 1
3. redis 127.0.0.1:6379> linsert mylist3 before "world "hello"
4. (integer) 2
5. redis 127.0.0.1:6379> lrange mylist3 0 -1
6. 1) "hello"
7. 2) "world"

**lset**:设置list中指定下标的元素值

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> rpush mylist4 "hello"
2. (integer) 1
3. redis 127.0.0.1:6379> lset mylist4 0 "world"
4. OK
5. redis 127.0.0.1:6379> lrange mylist4 0 -1
6. 1) "world"

**lrem**:从key对应list中删除n个和value相同的元素（n<0从尾删除,n=0全部删除）。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> rpush mylist5 "hello"
2. (integer) 1
3. redis 127.0.0.1:6379> rpush mylist5 "hello"
4. (integer) 2
5. redis 127.0.0.1:6379> lrem mylist5 1 "hello"  //从mylist5中删除1个和hello相同的元素
6. (integer) 1  //代表删除元素的个数

**ltrim**:保留指定key的值范围内的数据

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> rpush mylist8 "one"
2. (integer) 1
3. redis 127.0.0.1:6379> rpush mylist8 "two"
4. (integer) 2
5. redis 127.0.0.1:6379> ltrim mylist8 1 -1
6. (integer) 1
7. redis 127.0.0.1:6379> lrange mylist8 0 -1
8. 1) "two"

**lpop**:从list的头部删除元素，并返回删除元素

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> lrange mylist 0 -1
2. 1) "hello"
3. 2) "world"
4. redis 127.0.0.1:6379> lpop mylist
5. "hello"
6. redis 127.0.0.1:6379> lrange mylist 0 -1
7. 1) "world"

**rpop**:从list的尾部删除元素，并返回删除元素

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> lrange mylist2 0 -1
2. 1) "hello"
3. 2) "world"
4. redis 127.0.0.1:6379> rpop mylist2
5. "world"
6. redis 127.0.0.1:6379> lrange mylist2 0 -1
7. 1) "hello"

**lists类型**

**rpoplpush**：从第一个list的尾部移除元素并添加到第二个list的头部。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> lrange mylist5 0 -1
2. 1) "three"
3. 2) "hello"
4. redis 127.0.0.1:6379> lrange mylist6 0 -1
5. 1) "hello"
6. 2) "foo"
7. redis 127.0.0.1:6379> rpoplpush mylist5 mylist6
8. "hello"
9. redis 127.0.0.1:6379> lrange mylist5 0 -1
10. 1) "three"
11. redis 127.0.0.1:6379> lrange mylist6 0 -1
12. 1) "hello"
13. 2) "hello"
14. 3) "foo"

**lindex**:返回名称为key的list中index位置的元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> lrange mylist5 0 -1
2. 1) "three"
3. 2) "foo"
4. redis 127.0.0.1:6379> lindex mylist5 0
5. "three"
6. redis 127.0.0.1:6379> lindex mylist5 1
7. "foo"

**llen**:返回key对应list的长度

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> lrange mylist6 0 -1
2. 1) "hello"
3. 2) "hello"
4. 3) "foo"
5. redis 127.0.0.1:6379> llen mylist6
6. (integer) 3

4.sets类型

        Set是集合，它是string类型的无序集合。set是通过hash table实现的，添加、删除和查找的复杂度都是O(1)。对集合我们可以取并集、交集、差集。通过这些操作我们可以实现sns中的好友推荐和blog的tag功能。

**sadd**：向名称为key的set中添加元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> sadd myset "hello"
2. (integer) 1  //表示添加成功
3. redis 127.0.0.1:6379> sadd myset "world"
4. (integer) 1  //表示添加成功
5. redis 127.0.0.1:6379> sadd myset "world"
6. (integer) 0  //表示添加不成功
7. redis 127.0.0.1:6379> smembers myset
8. 1) "world"
9. 2) "hello"

**srem**：删除名称为key的set中的元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> sadd myset2 one
2. (integer) 1
3. redis 127.0.0.1:6379> sadd myset2 two
4. (integer) 1
5. redis 127.0.0.1:6379> sadd myset2 three
6. (integer) 1
7. redis 127.0.0.1:6379> smembers myset2
8. 1) "three"
9. 2) "two"
10. 3) "one"
11. redis 127.0.0.1:6379> srem myset2 two
12. (integer) 1
13. redis 127.0.0.1:6379> srem myset2 two
14. (integer) 0
15. redis 127.0.0.1:6379> smembers myset2
16. 1) "three"
17. 2) "one"

**spop**：随机返回并删除名称为key的set中一个元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> sadd myset3 one
2. (integer) 1
3. redis 127.0.0.1:6379> sadd myset3 two
4. (integer) 1
5. redis 127.0.0.1:6379> sadd myset3 three
6. (integer) 1
7. redis 127.0.0.1:6379> sadd myset3 four
8. (integer) 1
9. redis 127.0.0.1:6379> sadd myset3 five
10. (integer) 1
11. redis 127.0.0.1:6379> smembers myset3
12. 1) "four"
13. 2) "three"
14. 3) "two"
15. 4) "one"
16. 5) "five"
17. redis 127.0.0.1:6379> spop myset3
18. "one"
19. redis 127.0.0.1:6379> spop myset3
20. "four"
21. redis 127.0.0.1:6379> spop myset3
22. "three"
23. redis 127.0.0.1:6379> smembers myset3
24. 1) "two"
25. 2) "five"

**sdiff**：返回所有给定key与第一个key的差集。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> smembers myset2
2. 1) "three"
3. 2) "two"
4. redis 127.0.0.1:6379> smembers myset3
5. 1) "two"
6. 2) "one"
7. redis 127.0.0.1:6379> sdiff myset2 myset3
8. 1) "three"

**sdiffstore**：返回所有给定key与第一个key的差集，并将结果存为另一个key。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> smembers myset2
2. 1) "three"
3. 2) "two"
4. redis 127.0.0.1:6379> smembers myset3
5. 1) "two"
6. 2) "one"
7. redis 127.0.0.1:6379> sdiffstore myset4 myset2 myset3
8. (integer) 1
9. redis 127.0.0.1:6379> smembers myset4
10. 1) "three"

**sinter**：返回所有给定key的交集。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> smembers myset2
2. 1) "three"
3. 2) "two"
4. redis 127.0.0.1:6379> smembers myset3
5. 1) "two"
6. 2) "one"
7. redis 127.0.0.1:6379> sinter myset2 myset3
8. 1) "two"

**sinterstore**：返回所有给定key的交集，并将结果存为另一个key。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> smembers myset2
2. 1) "three"
3. 2) "two"
4. redis 127.0.0.1:6379> smembers myset3
5. 1) "two"
6. 2) "one"
7. redis 127.0.0.1:6379> sinterstore myset6 myset2 myset3
8. (integer) 1
9. redis 127.0.0.1:6379> smembers myset6
10. 1) "two"

**sunion**：返回所有给定key的并集。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> smembers myset2
2. 1) "three"
3. 2) "two"
4. redis 127.0.0.1:6379> smembers myset3
5. 1) "two"
6. 2) "one"
7. redis 127.0.0.1:6379> sunion myset2 myset3
8. 1) "three"
9. 2) "one"
10. 3) "two"

**sunionstore**：返回所有给定key的并集。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> smembers myset2
2. 1) "three"
3. 2) "two"
4. redis 127.0.0.1:6379> smembers myset3
5. 1) "two"
6. 2) "one"
7. redis 127.0.0.1:6379> sunionstore myset7 myset2 myset3
8. (integer) 3
9. redis 127.0.0.1:6379> smembers myset7
10. 1) "three"
11. 2) "one"
12. 3) "two"

**smove**：从第一个key对应的set中移除member并添加到第二个对应的set中。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> smembers myset2
2. 1) "three"
3. 2) "two"
4. redis 127.0.0.1:6379> smembers myset3
5. 1) "two"
6. 2) "one"
7. redis 127.0.0.1:6379> smove myset2 myset7 three
8. (integer) 1
9. redis 127.0.0.1:6379> smembers myset2
10. 1) "two"
11. redis 127.0.0.1:6379> smembers myset3
12. 1) "two"
13. 2) "one"
14. 3) "three"

**scard**：返回名称为key的set的元素个数。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> scard myset3
2. (integer) 3

**sismember**：测试member是否是名称为key的set的元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> smembers myset2
2. 1) "two"
3. redis 127.0.0.1:6379> sismember myset2 two
4. (integer) 1
5. redis 127.0.0.1:6379> sismember myset2 one
6. (integer) 0

**srandmember**：随机返回名称为key的set的一个元素，但不删除元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> smembers myset3
2. 1) "two"
3. 2) "one"
4. redis 127.0.0.1:6379> srandmember myset3
5. "two"
6. redis 127.0.0.1:6379> srandmember myset3
7. "one"

5.sorted sets类型

        sorted set是set的一个升级版本，它在set的基础上增加了一个顺序属性，这一属性在添加修改元素的时候可以指定，每次指定后，zset会自动重新按新的值调整顺序。可以理解为有两列的mysql表，一列存value，一列存顺序。操作中key理解为zset的名字。

**zadd**：向名称为key的zset中添加元素member,score用于排序。如果该元素存在，则更新其顺序。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> zadd myzset 1 "one"
2. (integer) 1
3. redis 127.0.0.1:6379> zadd myzset 2 "two"
4. (integer) 1
5. redis 127.0.0.1:6379> zadd myzset 3 "two"
6. (integer) 0
7. redis 127.0.0.1:6379> zrange myzset 0 -1 withscores
8. 1) "one"
9. 2) "1"
10. 3) "two"
11. 4) "3"

**zrem**：删除名称为key的zset中的元素member

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> zrange myzset 0 -1 withscores
2. 1) "one"
3. 2) "1"
4. 3) "two"
5. 4) "3"
6. redis 127.0.0.1:6379> zrem myzset two
7. (integer) 1
8. redis 127.0.0.1:6379> zrange myzset 0 -1 withscores
9. 1) "one"
10. 2) "1"

**zincrby**：如果在名称为key的zset中已经存在元素member，则该元素的score增加increment，否则向该集合中添加该元素，其score的值为increment。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> zadd myzset2 1 "one"
2. (integer) 1
3. redis 127.0.0.1:6379> zadd myzset2 2 "two"
4. (integer) 1
5. redis 127.0.0.1:6379> zincrby myzset2 2 "one"
6. "3" //代表one的顺序号变成了3
7. redis 127.0.0.1:6379> zrange myzset2 0 -1 withscores
8. 1) "two"
9. 2) "2"
10. 3) "one"
11. 4) "3"

**zrank**：返回名称为key的zset中member元素的排名（按score从小到大排序）即下标。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
2. 1) "one"
3. 2) "1"
4. 3) "two"
5. 4) "2"
6. 5) "three"
7. 6) "3"
8. 7) "five"
9. 8) "5"
10. redis 127.0.0.1:6379> zrank myzset3 two
11. (integer) 1 //two的下标是1（从0开始）

**zrevrank**：返回名称为key的zset中member元素的排名（按score从大到小排序）即下标。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
2. 1) "one"
3. 2) "1"
4. 3) "two"
5. 4) "2"
6. 5) "three"
7. 6) "3"
8. 7) "five"
9. 8) "5"
10. redis 127.0.0.1:6379> zrevrank myzset3 two
11. (integer) 2

**zrevrange**：返回名称为key的zset(按score从大到小顺序)中的index从start到end的所有元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> zrevrange myzset3 0 -1 withscores
2. 1) "five"
3. 2) "5"
4. 3) "three"
5. 4) "3"
6. 5) "two"
7. 6) "2"
8. 7) "one"
9. 8) "1"
10. redis 127.0.0.1:6379>

**zrangebyscore**：返回集合中score在给定区间的元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
2. 1) "one"
3. 2) "1"
4. 3) "two"
5. 4) "2"
6. 5) "three"
7. 6) "3"
8. redis 127.0.0.1:6379> zrangebyscore myzset3 2 3 withscores
9. 1) "two"
10. 2) "2"
11. 3) "three"
12. 4) "3"

**zcount**：返回集合中score在给定区间的数量。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
2. 1) "one"
3. 2) "1"
4. 3) "two"
5. 4) "2"
6. 5) "three"
7. 6) "3"
8. 7) "five"
9. 8) "5"
10. redis 127.0.0.1:6379> zcount myzset3 2 3
11. (integer) 2

**zcard**：返回集合中的元素个数。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
2. 1) "one"
3. 2) "1"
4. 3) "two"
5. 4) "2"
6. 5) "three"
7. 6) "3"
8. 7) "five"
9. 8) "5"
10. redis 127.0.0.1:6379> zcard myzset3
11. (integer) 4

**zremrangebyrank**：删除集合中排名在给定区间的元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
2. 1) "one"
3. 2) "1"
4. 3) "two"
5. 4) "2"
6. redis 127.0.0.1:6379> zremrangebyrank myzset3 1 1
7. (integer) 1 //只删除一个
8. redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
9. 1) "one"
10. 2) "1"

**zremrangebyscore**：删除集合中score在给定区间的元素。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
2. 1) "one"
3. 2) "1"
4. 3) "two"
5. 4) "2"
6. 5) "three"
7. 6) "3"
8. redis 127.0.0.1:6379> zremrangebyscore myzset3 1 2
9. (integer) 2 //删除顺序为1和2的
10. redis 127.0.0.1:6379> zrange myzset3 0 -1 withscores
11. 1) "three"
12. 2) "3"

一 .Redis常用命令

        Redis提供了丰富的命令对数据库和各种数据库类型进行操作，这些命令可以在Linux终端使用。

        a.键值相关命令

        b.服务器相关命令

1.键值相关命令

**Keys**：返回满足给定pattern的所有key

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> keys \*
2. 1) "myzset2"
3. 2) "myzset3"
4. 3) "mylist"
5. 4) "myset2"
6. 5) "myset3"
7. 6) "myset4"
8. 7) "list1"
9. redis 127.0.0.1:6379> keys my\*
10. 1) "myzset2"
11. 2) "myzset3"
12. 3) "mylist"
13. 4) "myset2"
14. 5) "myset3"
15. 6) "myset4"

**exists**：确认一个key是否存在。

         如下从结果来看，name键不存在，age键存在。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> exists name
2. (integer) 0
3. redis 127.0.0.1:6379> exists age
4. (integer) 1

**del**：删除一个key

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> del age
2. (integer) 1
3. redis 127.0.0.1:6379> exists age
4. (integer) 0

**expire**：设置一个key的过期时间。

        在下例中，我们设置addr这个key的过期时间是10秒，然后我们不断的用ttl来获取这个key的有效时长，直至为-1说明此值已过期。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> expire addr 10
2. (integer) 1
3. redis 127.0.0.1:6379> ttl addr
4. (integer) 8
5. redis 127.0.0.1:6379> ttl addr
6. (integer) 1
7. redis 127.0.0.1:6379> ttl addr
8. (integer) -1  //代表addr已经过期
9. redis 127.0.0.1:6379> get addr
10. (nil)
11. redis 127.0.0.1:6379> exists addr
12. (integer) 0

**move**：将当前数据库中的key转移到其它数据库中。

        数据库是从0到15，select 0表示选择0号数据库。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> select 0
2. OK
3. redis 127.0.0.1:6379> set age 30
4. OK
5. redis 127.0.0.1:6379> get age
6. "30"
7. redis 127.0.0.1:6379> move age 1  //将age从0数据库移到1数据库
8. (integer) 1
9. redis 127.0.0.1:6379> get age
10. (nil)
11. redis 127.0.0.1:6379[1]> select 1
12. OK
13. redis 127.0.0.1:6379[1]> get age
14. "30"
15. redis 127.0.0.1:6379[1]> select 0
16. OK
17. redis 127.0.0.1:6379> exists age
18. (integer) 0

**persist**：移除给定key的过期时间

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> expire age 300
2. (integer) 1
3. redis 127.0.0.1:6379> ttl age
4. (integer) 294
5. redis 127.0.0.1:6379> persist age
6. (integer) 1
7. redis 127.0.0.1:6379> ttl age
8. (integer) -1  //-1表示取消了过期时间

**randomkey**：随机返回key空间的一个key。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> randomkey
2. "mylist7"
3. redis 127.0.0.1:6379> randomkey
4. "mylist5"

**rename**：重命名key。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> keys \*
2. 1) "age"
3. redis 127.0.0.1:6379> rename age age\_new
4. OK
5. redis 127.0.0.1:6379> keys \*
6. 1) "age\_new"

**type**：返回值的类型。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> type addr
2. string
3. redis 127.0.0.1:6379> type myzset2
4. zset
5. redis 127.0.0.1:6379> type mylist
6. list

2.服务器相关命令

**ping**：测试连接是否存活。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> ping
2. PONG
3. //执行下面命令之前，停止redis服务器
4. redis 127.0.0.1:6379> ping
5. Could not connect to Redis at 127.0.0.1:6379:Connection refused
6. //执行下面命令之前，启动redis服务器
7. redis 127.0.0.1:6379> ping
8. PONG

        第一个ping时，说明此连接正常；第二个ping之前，我们将redis服务器停止，那么ping是失败的；第三个ping之前，我们将redis服务器启动，那么ping是成功的。

**echo**：在命令行打印一些内容。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> echo lamp
2. "lamp"

**select**：选择数据库。Redis数据库编号从0-15，我们可以选择任意一个数据库来进行数据的存取。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> select 1
2. OK
3. redis 127.0.0.1:6379[1]> select 16
4. (error) ERR invalid DB index
5. redis 127.0.0.1:6379[16]>

        当先择16时，报错，说明没有编号为16的这个数据库。

**quit**：退出连接。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> quit
2. [root@localhost redis-2.2.12]#

**dbsize**：返回当前数据库中key的数目。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> dbsize
2. (integer) 18

        结果说明此库中有18个key。

**info**：获取服务器的信息和统计。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> info
2. redis\_version:2.2.12
3. redis\_git\_sha1:00000000
4. redis\_git\_dirty:0
5. arch\_bits:32
6. multiplexing\_api:epoll
7. process\_id:28480
8. uptime\_in\_seconds:2515
9. uptime\_in\_days:0
10. .
11. .
12. .
13. redis 127.0.0.1:6379>

**config get**：实时传储收到的请求。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> config get dir
2. 1) "dir"
3. 2) "/root/4setup/redis-2.2.12"
4. redis 127.0.0.1:6379>

        本例中我们获取了dir这个参数配置的值，如果想获取全部参数据的配置值也很简单，只需要执行"config get \*"即可将全部的值都显示出来。

**flushdb**：删除当前选择数据库中的所有key。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> dbsize
2. (integer) 18
3. redis 127.0.0.1:6379> flushdb
4. OK
5. redis 127.0.0.1:6379> dbsize
6. (integer) 0

        本例中我们将0号数据库中的key都清除了。

**flushall**：删除所有数据库中的所有key。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379[1]> dbsize
2. (integer) 1
3. redis 127.0.0.1:6379> select 0
4. OK
5. redis 127.0.0.1:6379> flushall
6. OK
7. redis 127.0.0.1:6379> select 1
8. OK
9. redis 127.0.0.1:6379[1]> dbsize
10. (integer) 0
11. redis 127.0.0.1:6379[1]> keys \*
12. (empty list or set)
13. redis 127.0.0.1:6379[1]> select 0
14. OK
15. redis 127.0.0.1:6379> keys \*
16. (empty list or set)

        在本例中我们先查看了一个1号数据库中有一个key，然后切换到0号库执行flushall命令，结果1号库中的key也被清除了，说是此命令工作正常。

二.Redis高级实用特性

1.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。

        警告：因为redis速度相当快，所以在一台比较好的服务器下，一个外部的用户可以在一秒钟进行150K次的密码尝试，这意味着你需要指定非常强大的密码来防止暴力破解。

**Unix代码  [收藏代码](javascript:void())**

1. [root@localhost ~]# vi /usr/local/redis/etc/redis.conf

        修改配置文件中的如下配置项

**Unix代码  [收藏代码](javascript:void())**

1. #requirepass foobared
2. requirepass beijing

        我们设置了连接的口令是beijing。

        重启服务

**Unix代码  [收藏代码](javascript:void())**

1. [root@localhost ~]# pkill redis-server
2. [root@localhost ~]# /usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf
3. [root@localhost ~]# /usr/local/redis/bin/redis-cli
4. redis 127.0.0.1:6379> keys \*
5. (error) ERR operation not permitted
6. redis 127.0.0.1:6379> auth beijing
7. OK
8. redis 127.0.0.1:6379> keys \*
9. 1) "name"
10. redis 127.0.0.1:6379> exit
12. [root@localhost ~]# /usr/local/redis/bin/redis-cli -a beijing
13. redis 127.0.0.1:6379> keys \*
14. 1) "name"

2.主从复制

        Redis主从复制配置和使用都非常简单。通过主从复制可以允许多个slave server拥有和master server相同的数据库副本。

1).Redis主从复制特点：

        a.Master可以拥有多个slave

        b.多个slave可以连接同一个master外，还可以连接到其它slave

        c.主从复制不会阻塞master，在同步数据时，master可以继续处理client请求

        d.提高系统的伸缩性

2).Redis主从复制过程：

        a.Slave与Master建立连接，发送sync同步命令

        b.Master会启动一个后台进程，将数据库快照保存到文件中，同时Master主进程会开始收集新的写命令并缓存

        c.后台完成保存后，就将此文件发送给slave

        d.Slave将此文件保存到硬盘上

3).配置主从服务器

        配置slave服务器很简单，只需要在slave的配置文件中加入以下配置：

**Text代码  [收藏代码](javascript:void())**

1. slaveof 192.168.1.1 6379 #指定master的ip和端口
3. masterauth beijing #这是主机的密码

**Unix代码  [收藏代码](javascript:void())**

1. [root@localhost ~]# vi /usr/local/redis/etc/redis.conf
2. //修改其中的slaveof <maserip> <masterport>、masterauth <master-password>
3. [root@localhost ~]# /usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf
4. [root@localhost ~]# /usr/local/redis/bin/redis-cli -a beijing
5. redis 127.0.0.1:6379> keys \*
6. (empty list or set)

        配置好后，在Master设置任何key，在Slaver数据库中也会有，表示主从复制成功。

        如我们在主数据库上设置一对键值对

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> set name master
2. OK
3. redis 127.0.0.1:6379>

        在从数据库上取这个键

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> get name
2. "master"
3. redis 127.0.0.1:6379>

        我们怎么判断哪个是主哪个是从呢？我们只需调用info就可以得到主从信息，我们在从库中执行info如下：

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> info
2. role:slave
3. master\_host:localhost
4. master\_port:6379
5. master\_link\_status:up
6. master\_last\_io\_seconds\_ago:10
7. master\_sync\_in\_progress:0
8. db0:key=1,expires=0
9. redis 127.0.0.1:6379>

        里面的role会显示是Master还是Slave服务器，master\_link\_status的值是up表示正在连接Master服务器

**3.事务处理**

        Redis对事务的支持目前不比较简单。Redis只能保证一个client发起的事务中的命令可以连续的执行，而中间不会插入其他client的命令。当一个client在一个连接中发出multi命令时，这个连接会进入一个事务上下文，该连接后续的命令不会立即执行，而是先放到一个队列中，当执行exec命令时，redis会顺序的执行队列中的所有命令。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> get age
2. "33"
3. redis 127.0.0.1:6379> multi
4. OK
5. redis 127.0.0.1:6379> set age 10
6. QUEUED
7. redis 127.0.0.1:6379> set age 20
8. QUEUED
9. redis 127.0.0.1:6379> exec
10. 1) OK
11. 2) OK
12. redis 127.0.0.1:6379> get age
13. "20"

        如何取消一个事务

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> get age
2. "20"
3. redis 127.0.0.1:6379> multi
4. OK
5. redis 127.0.0.1:6379> set age 30
6. QUEUED
7. redis 127.0.0.1:6379> set age 40
8. QUEUED
9. redis 127.0.0.1:6379> discard
10. OK
11. redis 127.0.0.1:6379> get age
12. "20"

        可以发现这次2个set age命令都没被执行。discard命令其实就是清空事务的命令队列并退出事务上下文，也就是我们常说的事务回滚。

        如下例子可以看到，age由于是个数字，那么它可以有自增运算，但是name是个字符串，无法对其进行自增运算，所以会报错，如果按传统关系型数据库的思路来讲，整个事务都会回滚，但是我们看到redis却是将可以执行的命令提交了，所以这个现象对于习惯于关系型数据库操作的朋友来说是很别扭的，这一点也是redis今天需要改进的地方。

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> incr name
2. (error) ERR value is not an integer or out of range
3. redis 127.0.0.1:6379> multi
4. OK
5. redis 127.0.0.1:6379> incr age
6. QUEUED
7. redis 127.0.0.1:6379> incr name
8. QUEUED
9. redis 127.0.0.1:6379> exec
10. 1) (integer) 21
11. 2) (error) ERR value is not an integer or out of range
12. redis 127.0.0.1:6379> get age
13. "21"

**乐观锁复杂事务控制:**

        乐观锁：大多数是基于数据版本（ version）的记录机制实现的。即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据表添加一个"version"实段来实现读取出数据时，将此版本号一同读出，之后更新时，对此版本号加1。此时，将提交数据的版本号与数据库表对应记录的当前版本号进行比对，如果提交的数据版本号大于数据库当前版本号，则予以更新，否则认为是过期数据。

        Redis乐观锁实例：假设有一个age的key，我们开2个session来对age进行赋值操作，我们来看一下结果如何。

        1)第1步 session1

**Java代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> get age
2. "10"
3. redis 127.0.0.1:6379> watch age
4. OK
5. redis 127.0.0.1:6379> multi
6. OK

        2)第2步 session2

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> set age 30
2. OK
3. redis 127.0.0.1:6379> get age
4. "30"

        3)第3步 session1

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> set age 30
2. QUEUED
3. redis 127.0.0.1:6379> exec
4. (nil)  //因为有wathch age
5. redis 127.0.0.1:6379> get age
6. "30"

        watch命令监视给定的key，当exec时候如果监视的key从调用watch后发生过变化，则整个事务会失败。也可以调用watch多次监视多个key。这样就可以对指定的key加乐观锁了。注意watch的key是对整个连接有效的，事务也一样。如果连接断开，监视和事务都会被自动清除。当然，exec、discard、unwatch命令都会清除连接中的所有监视。

**事务回滚：**redis的事务实现是如此简单，当然会存在一些问题。第一个问题是redis只能保证事务的每个命令连续执行，但是如果事务中的一个命令失败了，并不回滚其他命令，比如使用的命令类型不匹配。

**4.持久化机制**

        Redis是一个支持持久化的内存数据库，也就是说redis需要经常将内存中的数据同步到硬盘来保证持久化。

        Redis支持两种持久化方式：

        a.snapshotting(快照)也是默认方式

        b.Append-only file(缩写aof)的方式

        1).Snapshotting方式

        快照是默认的持久化方式。这种方式是将内存中数据以快照的方式写入到二进制文件中，默认的文件名为dump.rdb。可以通过配置设置自动做快照持久化的方式。我们可以配置redis在n秒内如果超过m个key被修改就自动做快照。

**Text代码  [收藏代码](javascript:void())**

1. save 900 1    #900秒内如果超过1个key被修改，则发起快照保存
2. save 300 10   #300秒内，如超过10个key被修改，则发起快照保存

        2).aof方式

        由于快照方式是在一定间隔时间做一次的，所以如果redis意外down掉的话，就会丢失最后一次快照后的所有修改。

        aof比快照方式有更好的持久化性，是由于在使用aof时，redis会将每一个收到的写命令都通过write函数追加到文件中，当redis重启时会通过重新执行文件中保存的写命令来在内存中重建整个数据库的内容。

        当然由于os会在内核中缓存write做的修改，所以可能不是立即写到磁盘上。这样aof方式的持久化也还是有可能会丢失部分修改。

        可以通过配置文件告诉redis我们想要通过fsync函数强制os写入到磁盘的时机，默认的文件名为appendonly.aof[可通过cat方法看，保存的是命令]。

        vi /usr/local/redis/etc/redis.conf

**Text代码  [收藏代码](javascript:void())**

1. appendonly yes //启动aof持久化方式
2. #appendfsync always  //收到写命令就立即写入磁盘，最慢，但是保证完全的持久化
3. appendfsync everysec //每秒种写入磁盘一次，在性能和持久化方面做了很好的折中
4. #appendfsync no //完全依赖os，性能最好，持久化没保证

**5.发布订阅消息**

        发布订阅(pub/sub)是一种消息通信模式，主要的目的是解除消息发布者和消息订阅者之间的耦合，Redis作为一个pub/sub的server，在订阅者和发布者之间起到了消息路由的功能。订阅者可以通过subscribe和psubscribe命令向redis server订阅自己感兴趣的消息类型，redis将信息类型称为通道(channel)。当发布者通过publish命令向redis server发送特定类型的信息时，订阅该信息类型的全部client都会收到此消息。

        1)第1步 session1

**Java代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> subscribe tv1 tv2
2. Reading message... (press Ctrl+C to quit)
3. 1) "subscribe"
4. 2) "tv1"
5. 3) (integer) 1
6. 1) "subscribe"
7. 2) "tv2"
8. 3) (integer) 2

        2)第2步 session2

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> subscribe tv1
2. Reading message... (press Ctrl+C to quit)
3. 1) "subscribe"
4. 2) "tv1"
5. 3) (integer) 1

        3)第3步 session3

**Unix代码  [收藏代码](javascript:void())**

1. redis 127.0.0.1:6379> publish tv1 test
2. (integer) 2  //说明有两个监听
3. redis 127.0.0.1:6379> publish tv2 haha
4. (integer) 1  //说明只有一个监听
5. redis 127.0.0.1:6379>

        4)第4步 session1

**Unix代码  [收藏代码](javascript:void())**

1. 1) "message"
2. 2) "tv1"
3. 3) "test"
4. 1) "message"
5. 2) "tv2"
6. 3) "haha"

        第4步 session2

**Unix代码  [收藏代码](javascript:void())**

1. 1) "message"
2. 2) "tv1"
3. 3) "test"

**6.虚拟内存的使用**

        Redis的虚拟内存与操作系统的虚拟内存不是一回事，但是思路和目的都是相同的。就是暂时把不经常访问的数据从内存交换到磁盘中，从而腾出宝贵的内存空间用于其他需要访问的数据。尤其是对于redis这样的内存数据库，内存总是不够用的。除了可以将数据分割到多个redis server外，另外能够提高数据库容量的办法就是使用虚拟内存把那些不经常访问的数据交换到磁盘上。

        下面是vm相关配置：

         vi  /usr/local/redis/etc/redis.conf

**Text代码  [收藏代码](javascript:void())**

1. vm-enabled yes                  #开启vm功能
2. vm-swap-file /tmp/redis.swap    #交换出来的value保存的文件路径
3. vm-max-memory 1000000           #redis使用的最大内存上限
4. vm-page-size 32                 #每个页面的大小32字节
5. vm-pages 134217728              #最多使用多少页面
6. vm-max-threads 4                #用于执行value对象换入的工作线程数量

        并设置如下配置，确定要使用虚拟内存

**Text代码  [收藏代码](javascript:void())**

1. really-use-vm yes

**一.redis.php**

**Php代码  [收藏代码](javascript:void())**

1. <?php
2. //实例化
3. $redis = **new** Redis();
4. //连接服务器
5. $redis->connect("localhost");
6. //授权
7. $redis->auth("lamplijie");
8. //相关操作
9. $redis->set("name","lampbrother");
10. $data = $redis->key("\*");
11. var\_dump($data);

页面显示结果：

**Text代码  [收藏代码](javascript:void())**

1. array(1) { [0]=> string(4) "name" }

**二.简单回顾redis的四种数据类型**

        a.string：最简单的数据类型

**Unix代码  [收藏代码](javascript:void())**

1. set user:001:name lijie
2. set user:001:age 20

        b.hash：可以当做表，hash table，比string速度快

**Unix代码  [收藏代码](javascript:void())**

1. hset user:001 name lamp age 20
2. hset user:001 sex nan
3. hset user:002 name lijie age 20
4. hgetall user:001

        c.list：栈、队列

        d.set：并集、交集、差集

        e.zset：set升级版，多了一个顺序

**三.小型的用户管理系统（用户的增删改查、分页、登陆退出、加关注）**

    redis.php

**Php代码  [收藏代码](javascript:void())**

1. <?php
2. //实例化
3. $redis = **new** Redis();
4. //连接服务器
5. $redis->connect("localhost");
6. //授权
7. $redis->auth("lamplijie");

    add.php

**Php代码  [收藏代码](javascript:void())**

1. <form action="reg.php" method="post">
2. 用户名：<input type="text" name="username"/><br/>
3. 密码：<input type="password" name="password"/><br/>
4. 年龄：<input type="text" name="age"/><br/>
5. <input type="submit" value="注册"/>
6. <input type="reset" value="重新填写"/>
7. </form>

    reg.php

**Php代码  [收藏代码](javascript:void())**

1. <?php
2. **require**("redis.php");
3. $username = $\_POST['username'];
4. $password = md5($\_POST['password']);
5. $age = $\_POST['age'];
6. echo $uid = $redis->incr("userid");
7. $redis->hmset("user:".$uid,**array**("uid"=>$uid,"username"=>$username,"password"=>$password,"age"=>$age));
8. $redis->rpush("uid",$uid);
9. $redis->set("username:".$username,$uid);
10. header("location:list.php");

    list.php

**Php代码  [收藏代码](javascript:void())**

1. <a href="add.php">注册</a>
2. <?php
3. **require**("redis.php");
4. **if**(!empty**empty**($\_COOKIE['auth'])){
5. $id = $redis->get("auth:".$COOKIE['auth']);
6. $name = $redis->hget("user:".$id,"username");
7. ?>
8. 欢迎您，<?php echo $name?>,<a href="logout.php">退出</a>
9. <?php
10. }**else**{
11. ?>
12. <a href="login.php">登陆</a>
13. <?php
14. }
15. //用户总数
16. $count = $redis->lsize("uid");
18. //页大小
19. $page\_size = 3;
21. //当前页码
22. $page\_num = (!empty**empty**($\_GET['page']))?$\_GET['page']:1;
24. //页总数
25. $page\_count = ceil($count/$page\_size);
27. $ids = $redis->lrange("uid",($page\_num-1)\*$page\_size,(($page\_num-1)\*$page\_size+$page\_size-1));
29. //var\_dump($ids);
31. /\*
32. for($i=1;$i<=($redis->get("userid"));$i++) {
33. $data[] = $redis->hgetall("user:".$i);
34. }\*/
36. **foreach**($ids **as** $v){
37. $data[] = $redis->hgetall("user:".$v);
38. }
40. //var\_dump($data);
41. //$data = array\_filter($data);
42. ?>
43. <table border="1">
44. <tr>
45. <th>uid</th>
46. <th>username</th>
47. <th>age</th>
48. <th>操作</th>
49. </tr>
50. <?php **foreach**($data **as** $v){?>
51. <tr>
52. <td><?php echo $v['uid']?></td>
53. <td><?php echo $v['username']?></td>
54. <td><?php echo $v['age']?></td>
55. <td>
56. <a href="del.php?id=<?php echo $v['uid']?>">删除</a>
57. <a href="mod.php?id=<?php echo $v['uid']?>">编辑</a>
58. <?php **if**(!**empty**empty($\_COOKIE['auth']) && $id!=$v['uid']){?>
59. <a href="addfans.php?id=<?php echo $v['uid']?>&uid=<?php echo $id?>">加关注</a>
60. <?php } ?>
61. </td>
62. </tr>
63. <?php}?>
64. <tr>
65. <td colspan="4">
66. <a href="?page=<?php echo(($page\_num-1)<=1)?1:($page\_num-1) ?>">上一页</a>
67. <a href="?page=<?php echo(($page\_num+1)>=$page\_count)?$page\_count:($page\_num+1) ?>">下一页</a>
68. <a href="?page=1">首页</a>
69. <a href="?page=<?php echo $page\_count ?>">尾页</a>
70. 当前<?php echo $page\_num ?>页
71. 总共<?php echo $page\_count ?>页
72. 总共<?php echo $count ?>个用户
73. </td>
74. </tr>
75. </table>
77. <table border=1>
78. <caption>我关注了谁</caption>
79. <?php $data = $redis->smembers("user:".$id.":following");?>
80. **foreach**($data **as** $v) {
81. $row = $redis->hgetall("user:".$v);
82. <tr>
83. <td><?php echo $row['uid']?></td>
84. <td><?php echo $row['username']?></td>
85. <td><?php echo $row['age']?></td>
86. </tr>
87. <?php
88. }
89. ?>
90. </table>
92. <table border=1>
93. <caption>我的粉丝</caption>
94. <?php
95. $data = $redis->smembers("user:".$id.":followers");
96. **foreach**($data **as** $v) {
97. $row = $redis->hgetall("user:".$v);
98. ?>
99. <tr>
100. <td><?php echo $row['uid']?></td>
101. <td><?php echo $row['username']?></td>
102. <td><?php echo $row['age']?></td>
103. </tr>
104. <?php
105. }
106. ?>
107. </table>

    del.php

**Php代码  [收藏代码](javascript:void())**

1. <?php
2. **require**("redis.php");
3. $uid = $\_GET['id'];
4. $redis->del("user:".$uid);
5. $redis->lrem("uid",$uid);
6. header("localhost:list.php");

    mod.php

**Php代码  [收藏代码](javascript:void())**

1. <?php
2. **require**("redis.php");
3. $uid = $\_GET['id'];
4. $data = $redis->hgetall("user:".$uid);
5. ?>
6. <form action="doedit.php" method="post">
7. <input type="hidden" value="<?php echo $data['uid']?>" name="uid"/>
8. 用户名：<input type="text" name="username" value="<?php echo $data['username']?>"/><br/>
9. 年龄：<input type="text" name="age" value="?php echo $[data['age']?>"/><br/>
10. <input type="submit" value="修改"/>
11. <input type="reset" value="重新填写"/>
12. </form>

    doedit.php

**Php代码  [收藏代码](javascript:void())**

1. <?php
2. $uid = $\_POST['uid'];
3. $username = $\_POST['username'];
4. $age = $\_POST['age'];
5. $a = $redis->hmset("user:".$uid,**array**("username"=>$username,"age"=>$age));
6. **if**($a) {
7. header("location:list.php");
8. }**else** {
9. header(location:mod.php?id=".$uid);
10. }

    login.php

**Php代码  [收藏代码](javascript:void())**

1. <?php
2. **require**("redis.php");
3. $username = $\_POST['username'];
4. $pass = $\_POST['password'];
5. $id = $redis->get("username:".$username);
6. **if**(!empty**empty**($id)) {
7. $password = $redis->hget("user:".$id,"password");
8. **if**(md5($pass) == $password) {
9. $auth = md5(time().$username.rand());
10. $redis->set("auth:".$auth,$id);
11. setcookie("auth", $auth, time() + 86400);
12. header("location:list.php");
13. }
14. }
15. ?>
16. <form action="" method="post">
17. 用户名：<input type="text" name="username"/><br/>
18. 密码：<input type="password" name="password"/><br/>
19. <input type="submit" value="登陆"/>
20. </form>

    logout.php

**Php代码  [收藏代码](javascript:void())**

1. <?php
2. setcookie("auth","",time()-1);
3. header("location:list.php");

    addfans.php

**Php代码  [收藏代码](javascript:void())**

1. <?php
2. $id = $\_GET['id'];
3. $uid = $\_GET['uid'];
4. **require**("redis.php");
5. $redis->sadd("user:".$uid.":following",$id);
6. $redis->sadd("user:".$id.":followers",$uid);
7. header("location:list.php");

当然，采用sdiff user:1:following user:2:following语句，用户1可以向用户2推荐关注（即用户1的关注与用户2的关注的差集）。

**一.在Linux上安装Redis服务**（详细可参看：<http://bijian1013.iteye.com/blog/2070773>）

步骤一：Redis的官方下载安装包

        官网地址：http://redis.io/download

步骤二：编译源程序

**Unix代码  [收藏代码](javascript:void())**

1. tar zxvf redis-2.4.17.tar.gz
2. cd redis-2.4.17
3. make
4. cd src && make install

步骤三：移动文件，便于管理

**Unix代码  [收藏代码](javascript:void())**

1. mkdir -p /usr/local/redis/bin
2. mkdir -p /usr/local/redis/etc
3. mv /lamp/redis-2.4.17/redis.conf /usr/local/redis/etc
4. cd /lamp/redis-2.4.17/src
5. mv mkreleasehdr.sh redis-benchmark redis-check-aof redis-check-dump redis-cli redis-server /usr/local/redis/bin

 步骤四：启动Redis服务

**Unix代码  [收藏代码](javascript:void())**

1. /usr/local/redis/bin/redis-server
2. /usr/local/redis/etc/redis.conf

        Redis服务端的默认连接端口是6379（mySql的端口号是：3306；mongoDB的端口号是：27017，28017）。

        默认情况下，Redis不是在后台运行的，我们需要把开启Redis的后台运行。

**Unix代码  [收藏代码](javascript:void())**

1. vi /usr/local/redis/redis.conf

        将daemonize的值改为yes。

        启动后看进程是否启动

**Unix代码  [收藏代码](javascript:void())**

1. ps -ef | grep redis

        看指定的端口号是否占用

**Unix代码  [收藏代码](javascript:void())**

1. netstat -tunpl | grep 6379

步骤五：客户端连接

**Unix代码  [收藏代码](javascript:void())**

1. /usr/local/redis/bin/redis-cli

步骤六：将Redis作为Linux服务随机启动

        vi /etc/rc.local, 使用vi编辑器打开随机启动配置文件，并在其中加入下面一行代码。

**Unix代码  [收藏代码](javascript:void())**

1. /usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf

步骤七：客户端连接验证

        新打开一个Session输入：/usr/local/redis/bin/redis-cli，如果出现下面提示，就可以开始Redis之旅了。

**Unix代码  [收藏代码](javascript:void())**

1. [root@localhost bin]# /usr/local/redis/bin/redis-cli
2. 127.0.0.1:6379>

步骤八：停止Redis实例

        我们可以使用

**Unix代码  [收藏代码](javascript:void())**

1. /usr/local/redis/bin/redis-cli shutdown

        也可以使用

**Unix代码  [收藏代码](javascript:void())**

1. pkill redis-server

        当然，也可以使用kill all,kill -9。

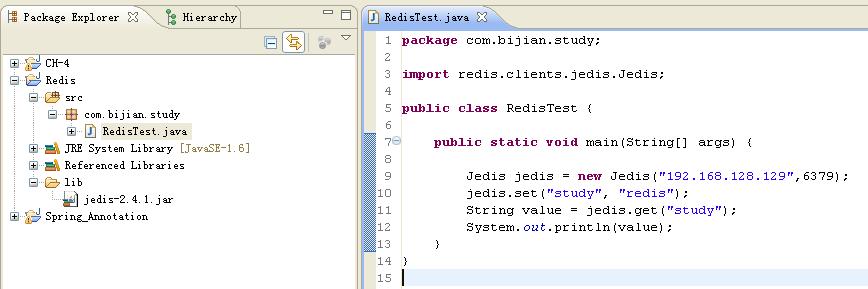
**二.编写JAVA程序**

        在Eclipse里新建一个java project，导入jedis-\*.jar包。

**Java代码  [收藏代码](javascript:void())**

1. **package** com.bijian.study;
3. **import** redis.clients.jedis.Jedis;
5. **public** **class** RedisTest {
7. **public** **static** **void** main(String[] args) {
9. Jedis jedis = **new** Jedis("192.168.128.129",6379);
10. jedis.set("study", "redis");
11. String value = jedis.get("study");
12. System.out.println(value);
13. }
14. }

        工程结构如下：

  
        运行结果是控制台输出redis。

        也可以打开一个Session，连接客户端后，输入get study，亦可获得redis，

如下所示：

**Unix代码  [收藏代码](javascript:void())**

1. 127.0.0.1:6379> get study
2. "redis"
3. 127.0.0.1:6379>