1.NoSQL 介绍

a) NoSQL (NoSQL = Not Only SQL)，意即“不仅仅是 SQL”，是一项全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。泛指非关系型的数据库。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 数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。

b) 特点：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,SELECT,UPDATE,DELTE 等语句），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

c) 适用场景：1.对数据高并发读写 2.对海量数据的高效率储存和访问 3 对数据的高扩展性和高可用性

2.Redis 的介绍

a) Redis 是一个开源（BSD 许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。它支持多种类型的数据结构，如 字符串（strings），散列（hashes）， 列表（lists）， 集合（sets），有序集合（sorted sets） 与范围查询， bitmaps， hyperloglogs 和 地理空间（geospatial） 索引半径查询。 Redis 内置了 复制（replication），LUA 脚本（Lua scripting）， LRU 驱动事件（LRU eviction），事务（transactions） 和不同级别的磁盘持久化

（persistence），并通过 Redis 哨兵（Sentinel）和自动分区（Cluster）提供高可用性（high availability）。

b) 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:队列，栈），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厚的操作，redis 支持各种不同方式的排序。为了保证效率，数据都是缓存到内存中，它也可以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中。

c) 提供大量语言的 API

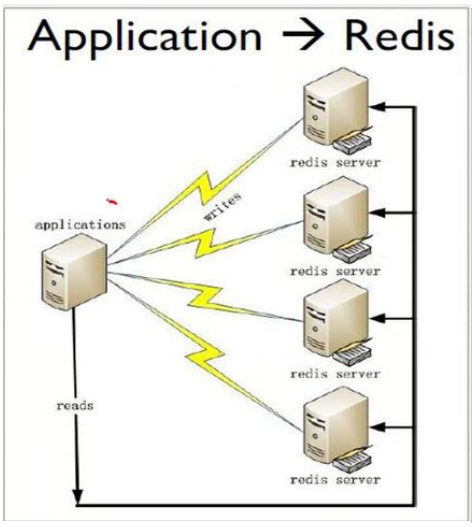
3.Redis 适用场景

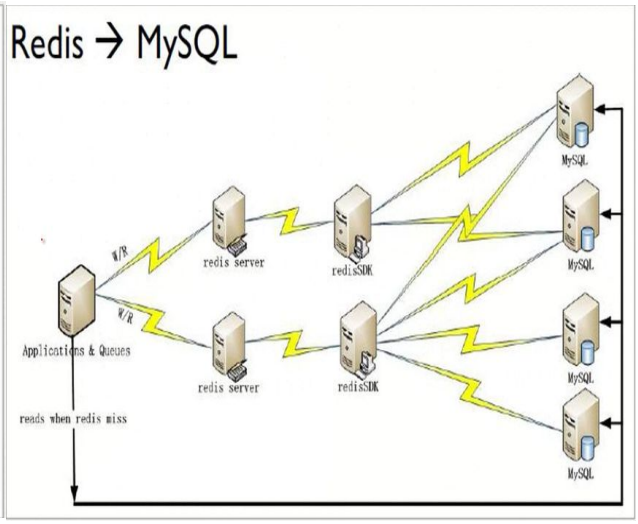
a) 目前全球最大的 Redis 用户是新浪微博，在新浪有 200 多台物理机，在 400 多个端口正在运行着 redis，有+4G 的数据在 redis 上来为微博用户提供服务。

b) 在新浪微博 redis 的部署场景很多，大概分为如下 2 种：

1.应用程序直接访问 redis 数据库

2.应用程序直接访问 redis 数据库，只有当 redis 访问失败时才访问 MySQL





c) Redis 数据库提供多种灵活的数据结构和数据操作，为不同的数据构建不同类型

d) 具体的场景：

1.取最新 N 个数据的操作

2.排行榜应用，取 TOPN 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除重值

6.实时系统，反垃圾系统

7.Pub/Sub 构建实时消息系统（独有）发布和订阅

8.构建队列系统

9.缓存

4.Redis 的安装和部署

a) 下载安装包 wget http://download.redis.io/releases/redis-3.0.7.tar.gz

b) 编译安装

1.tar zxvf redis-3.0.7.tar.gz

2.cd redis-3.0.7

3.make

4.cd src && make install

c) 移动文件，方便管理

1. mkdir -p /Data/apps/redis/etc;mkdir -p /Data/apps/redis/bin;

2. mv /root/redis-3.0.7/src/mkreleasehdr.sh redis-benchmark redis-check-aof redis-check-dump redis-cli redis-server /Data/apps/redis/bin/

3. mv /root/redis-3.0.7/redis.conf /Data/apps/redis/etc/

d) 启动 redis 服务时要指定配置文件，否则会报 :WARNING overcommit_memory is set to 0! Background save may fail under low memory condition。

/Data/apps/redis/bin/redis-server /Data/apps/redis/etc/redis.conf

Redis 服务器的默认端口是 6379

redis.conf 配置 :daemonize yes (后台运行 redis) 是否作为守护进程运行(&不可行)

查看服务： ps -ef | grep redis 查看端口：netstat -tunpl | grep 6379

e) 客户端连接

/Data/apps/redis/bin/redis-cli auth 验证字符串 keys *

f) 停止 redis 实例

/Data/apps/redis/bin/redis-cli shutdown

pkill redis-server

g) 配置文件

1. redis.conf 配置参数：

#是否作为守护进程运行

daemonize yes

#如以后台进程运行，则需指定一个 pid，默认为/var/run/redis.pid

pidfile redis.pid

#绑定主机 IP，默认值为 127.0.0.1

#bind 127.0.0.1

#Redis 默认监听端口

port 6379

#客户端闲置多少秒后，断开连接，默认为 300（秒）

timeout 300

#日志记录等级，有 4 个可选值，debug，verbose（默认值），notice，warning

loglevel verbose

#指定日志输出的文件名，默认值为 stdout，也可设为/dev/null 屏蔽日志

logfile stdout

#可用数据库数，默认值为 16，默认数据库为 0

databases 16

#保存数据到 disk 的策略

#当有一条 Keys 数据被改变是，900 秒刷新到 disk 一次

save 900 1

#当有 10 条 Keys 数据被改变时，300 秒刷新到 disk 一次

save 300 10

#当有 1w 条 keys 数据被改变时，60 秒刷新到 disk 一次

save 60 10000

#当 dump .rdb 数据库的时候是否压缩数据对象

rdbcompression yes

#本地数据库文件名，默认值为 dump.rdb

dbfilename dump.rdb

#本地数据库存放路径，默认值为 ./

dir /var/lib/redis/

########### Replication ######################

#Redis 的复制配置

slaveof <masterip> <masterport> 当本机为从服务时，设置主服务的 IP 及端口

masterauth <master-password> 当本机为从服务时，设置主服务的连接密码

#连接密码

requirepass foobared

#最大客户端连接数，默认不限制

maxclients 128

#最大内存使用设置，达到最大内存设置后，Redis 会先尝试清除已到期或即将到期的 Key，当此方法处理后，任到达最大内存设置，将无法再进行写入操作。

maxmemory <bytes>

#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 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。默认值为 no

appendonly no

#更新日志文件名，默认值为 appendonly.aof

#appendfilename

#更新日志条件 ,共有 3 个可选值。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(默认值 ）。

appendfsync always

appendfsync everysec

appendfsync no

################ VIRTUAL MEMORY ###########

#是否开启 VM 功能，默认值为 no

vm-enabled no

vm-enabled yes

#虚拟内存文件路径，默认值为/tmp/redis.swap，不可多个 Redis 实例共享

vm-swap-file /tmp/redis.swap

将所有大于 vm-max-memory 的数据存入虚拟内存,无论 vm-max-memory 设置多小,所有索引数据都是内存存储的 (Redis 的索引数据就是 keys),也就是说,当 vm-max-memory 设置为 0 的时候,其实是所有 value 都存在于磁盘。默认值为 0。

vm-max-memory 0

vm-page-size 32

vm-pages 134217728

vm-max-threads 4

############# ADVANCED CONFIG ###############

glueoutputbuf yes

hash-max-zipmap-entries 64

hash-max-zipmap-value 512

#是否重置 Hash 表

activerehashing yes

注意：Redis 官方文档对 VM 的使用提出了一些建议:

当你的 key 很小而 value 很大时,使用 VM 的效果会比较好.因为这样节约的内存比较大.

当你的 key 不小时,可以考虑使用一些非常方法将很大的 key 变成很大的 value,比如你

可以考虑将 key,value 组合成一个新的 value.

最好使用 linux ext3 等对稀疏文件支持比较好的文件系统保存你的 swap 文件.

vm-max-threads 这个参数,可以设置访问 swap 文件的线程数,设置最好不要超过机器的核数.如果设置为 0,那么所有对 swap 文件的操作都是串行的.可能会造成比较长时间的延迟,但是对数据完整性有很好的保证.

2. 调整系统内核参数

如果内存情况比较紧张的话，需要设定内核参数：

echo 1 > /proc/sys/vm/overcommit_memory

这里说一下这个配置的含义：/proc/sys/vm/overcommit_memory

该文件指定了内核针对内存分配的策略，其值可以是 0、1、2。

0，表示内核将检查是否有足够的可用内存供应用进程使用；如果有足够的可用内存，内存申请允许；否则，内存申请失败，并把错误返回给应用进程。

1，表示内核允许分配所有的物理内存，而不管当前的内存状态如何。

2，表示内核允许分配超过所有物理内存和交换空间总和的内存

Redis 在 dump 数据的时候，会 fork 出一个子进程，理论上 child 进程所占用的内存和 parent 是一样的，比如 parent 占用的内存为 8G，这个时候也要同样分配 8G 的内存给 child, 如果内存无法负担 ,往往会造成 redis 服务器的 down 机或者 IO 负载过高，效率下降。所以这里比较优化的内存分配策略应该设置为 1（表示内核允许分配所有的物理内存，而不管当前的内存状态如何）

一、 运行服务

redis-server /etc/redis/redis.conf 开启

redis-cli shutdown 关闭

二、 测试

1) 可在后台启动 redis 服务后，用 redis-benchmark 命令测试

2) 通过 redis-cli 命令实际操作测试

三、 保存/备份

数据备份可以通过定期备份该文件实现。

因为 redis 是异步写入磁盘的，如果要让内存中的数据马上写入硬盘可以执行如下命令：

redis-cli save 或者 redis-cli -p 6380 save（指定端口）

注意，以上部署操作需要具备一定的权限，比如复制和设定内核参数等。

执行 redis-benchmark 命令时也会将内存数据写入硬盘。

四、 开启端口号

1) 打开/etc/sysconfig/iptables，

2) 在-【A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT】后面 ,加上 【A INPUT -p tcp -m state --state NEW -m tcp --dport 6379 -j ACCEPT】

//这里的 6379 是 Redis 默认端口号

3) 保存，重启防火墙：/etc/init.d/iptables restart

5.Redis 的数据类型

a) 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 jpg 图片或者序列化的对象。

1. Set 设置 key 对应的值为 string 类型的 value。

例如：添加一个 name=zhangsan 的键值对

127.0.0.1:6379> set name zhangsan

OK

2. Setnx 设置 key 对应的值为 string 类型的 value,如果 key 已经存在，返回 0，存在了同名的 key 就不会覆盖更新，nx 是 not exist 的意思，整体表示：不存在才设置，设置成功返回 1.

例如：添加一个 name=zhangsan_new 的键值对

127.0.0.1:6379> setnx name zhangsan_new

(integer) 0

127.0.0.1:6379> setnx age 20

(integer) 1

3. 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，不设置有效则表示永久有效

例如：添加一个 color=red 的键值对，并指定有效期为 10 秒

127.0.0.1:6379> setex color 10 red

OK

4.Setrange 设置指定 key 的 value 值的子字符串，替换成功返回字符串长度

例如：将 zhangsan 的 126 邮箱替换为 sina 邮箱

127.0.0.1:6379> set mail zhangsan@126.com

OK

127.0.0.1:6379> get mail

"zhangsan@126.com"

127.0.0.1:6379> setrange mail 9 sina.com // 9 表示字符位置，由左往右从 0 开始

(integer) 17

127.0.0.1:6379> get mail

"zhangsan@sina.com"

备注：1. 只对应替换的长度

set name zhangsan@sina.com

OK

127.0.0.1:6379> setrange name 9 qq.com

(integer) 17

127.0.0.1:6379> get name

"zhangsan@qq.comom"

5. mset 一次设置多个 key 的值，成功返回 ok 表示所有设置成功，失败返回 0 表示没有任何值被设置成功。

127.0.0.1:6379> mset name lisi age 24 sex nan

OK

6. msetnx 一次设置多个 key 的值，成功返回 ok 表示所有设置成功，失败返回 0 表示没有任何值被设置成功。但是不会覆盖已经存在的 key，一个 key 不成功所有都不会成功。

127.0.0.1:6379> msetnx name lisi age 24 sex nan address sanya

(integer) 0

7. get 获取 key 对应的 string 值，如果 key 不存在返回 nil 。

8. getset 设置 key 值，并返回 key 的旧值 。

127.0.0.1:6379> getset name wangwu

"lisi"

9. getrange 获取 key 的 value 值的子字符串

127.0.0.1:6379> getrange name 1 5

"angwu"

10.mget 一次获取多个 key 的值，如果对应的 key 不存在则对应返回 nil

127.0.0.1:6379> mget name age sex address

1) "wangwu"

2) "24"

3) "nan"

4) (nil)

11.incr 对 key 的值做加加操作，并返回新的值

127.0.0.1:6379> incr age

(integer) 25

12.incrby 同 incr 类似，加指定值，key 不存在时候会设置 key,并认为原来的值为 0

127.0.0.1:6379> incrby age 5

(integer) 30

127.0.0.1:6379> incrby age -2

(integer) 28

127.0.0.1:6379> incrby height 175

(integer) 175

13. decr 对 key 的值做减减操作，并返回新的值

127.0.0.1:6379> decr height

(integer) 174

14. decrby 同 incr 类似，减指定值

127.0.0.1:6379> decrby height 5

(integer) 169

127.0.0.1:6379> decrby height -2

(integer) 171

127.0.0.1:6379> decrby width 10

(integer) -10

127.0.0.1:6379> decrby width -2

(integer) -8

15. append 给指定 key 的字符串追加 value,返回新字符串值的长度

127.0.0.1:6379> get height

"171"

127.0.0.1:6379> append height cm

(integer) 5

127.0.0.1:6379> get height

"171cm"

16. strlen 获取指定 key 的 value 值的长度

127.0.0.1:6379> strlen height

(integer) 5

b)　Hashes 类型

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。相对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空间，并且很方便存取整个对象。

1．　hSet　设置 hash field 对应的值，如果 key 不存在，则创建并设置值。

127.0.0.1:6379> hset user:001 name zhangsan

(integer) 1

//相当于 hset　hash 表名称 hash 表字段 字段值

2．hSetnx　设置 hash field 对应的值，如果 key 不存在，则创建并设置值。如果存在返回 0.

127.0.0.1:6379> hsetnx user:002 name lisi

(integer) 1

127.0.0.1:6379> hsetnx user:002 name wangwu

(integer) 0

3．hmet　同时设置 hash 的多个 field 对应的值。

127.0.0.1:6379> hmset user:002 age 25 height 178cm name zhaoliu

OK

4．hget 获取指定的 hash field 的值

127.0.0.1:6379> hget user:002 age

"25"

5．hmget 获取指定 hash 多个 field 的值

127.0.0.1:6379> hmget user:002 age name height

1) "25"

2) "zhaoliu"

3) "178cm"

6．hincrby 指定的 hash field 加上给定值

127.0.0.1:6379> hincrby user:002 age 6

(integer) 31

127.0.0.1:6379> hincrby user:002 age -8

(integer) 23

7．hexists 测试指定 field 是否存在

127.0.0.1:6379> hexists user:002 age

(integer) 1

127.0.0.1:6379> hexists user:002 address

(integer) 0

8．hlen 返回指定 hash 的 field 数量

127.0.0.1:6379> hlen user:002

(integer) 3

9．hdel 删除指定 hash 的 field

127.0.0.1:6379> hdel user:002 age

(integer) 1

10．hkeys 返回 hash 的所有 field

127.0.0.1:6379> hkeys user:002

1) "name"

2) "height"

11．hvals 返回 hash 的所有 value

127.0.0.1:6379> hvals user:002

1) "zhaoliu"

2) "178cm"

12．hgetall 获取某一个 hash 中全部的 field 及 value

127.0.0.1:6379> hgetall user:002

1) "name"

2) "zhaoliu"

3) "height"

4) "178cm"

c) Lists 类型

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，操作中 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 操作作为链表的头部或者尾部添加删除元素，这样 listj 既是可以作为栈，又可以作为链表。

1. lpush 在 key 对应 list 的头部添加字符串元素 （先进后出）

127.0.0.1:6379> lpush mylist hell

(integer) 1

127.0.0.1:6379> lpush mylist world

(integer) 2

127.0.0.1:6379> lpush mylist !

(integer) 3

127.0.0.1:6379> lrange mylist 0 -1　//从头部（0）取到尾部（-1）

1) "!"

2) "world"

3) "hell"

2. rpush　在 key 对应 list 的尾部添加字符串元素　（先进先出）

127.0.0.1:6379> rpush mylist we

(integer) 4

127.0.0.1:6379> rpush mylist love

(integer) 5

127.0.0.1:6379> lrange mylist 0 -1　//从头取到尾部

1) "!"

2) "world"

3) "hell"

4) "we"

5) "love"

3.linsert 在 key 对应 list 的特定位置前或者后添加字符串

127.0.0.1:6379> lpush newlist hello

(integer) 1

```
127.0.0.1:6379> lpush newlist world

(integer) 2

127.0.0.1:6379> linsert newlist before world is

(integer) 3

127.0.0.1:6379> lrange newlist 0 -1

1) "is"

2) "world"

3) "hello"

127.0.0.1:6379> linsert newlist before hello one

(integer) 4

127.0.0.1:6379> lrange newlist 0 -1

1) "is"

2) "world"

3) "one"

4) "hello"

127.0.0.1:6379> linsert newlist after hello two

(integer) 5

127.0.0.1:6379> lrange newlist 0 -1

1) "is"

2) "world"

3) "one"

4) "hello"
```

5) "two"

4.lset 设置 list 中指定下标的元素值

127.0.0.1:6379> rpush nlist cheng

(integer) 1

127.0.0.1:6379> rpush nlist hui

(integer) 2

127.0.0.1:6379> lrange nlist 0 -1

1) "cheng"

2) "hui"

127.0.0.1:6379> lset nlist 0 chen

OK

127.0.0.1:6379> lrange nlist 0 -1

1) "chen"

2) "hui"

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

127.0.0.1:6379> lrange alist 0 -1

1) "three"

2) "two"

3) "two"

4) "two"

5) "one"

6) "one"

7) "one"

8) "one"

127.0.0.1:6379> lrem alist 2 one //从 alist 删除 2 个 one

(integer) 2

127.0.0.1:6379> lrem alist 4 two

(integer) 3

127.0.0.1:6379> lrem alist 1 four

(integer) 0

127.0.0.1:6379> lrange alist 0 -1

1) "three"

2) "one"

3) "one"

6. ltrim 保留指定 key 的值范围内的数据

127.0.0.1:6379> lrange blist 0 -1

1) "one"

2) "two"

3) "three"

4) "four"

127.0.0.1:6379> ltrim blist 1 -2 //保留下标从 1 到-2 的值

OK

127.0.0.1:6379> lrange blist 0 -1

1) "two"

2) "three"

7. lpop 从 list 的头部删除元素，并返回删除元素

127.0.0.1:6379> lrange clist 0 -1

1) "one"

2) "two"

3) "three"

127.0.0.1:6379> lpop clist

"one"

127.0.0.1:6379> lrange clist 0 -1

1) "two"

2) "three"

8. rpop 从 list 的尾部删除元素，并返回删除元素

127.0.0.1:6379> lrange clist 0 -1

1) "two"

2) "three"

127.0.0.1:6379> rpop clist

"three"

127.0.0.1:6379> lrange clist 0 -1

1) "two"

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

127.0.0.1:6379> lrange dlist 0 -1

1) "two"

2) "one"

127.0.0.1:6379> lrange elist 0 -1

1) "mysql"

2) "java"

3) "php"

127.0.0.1:6379> rpoplpush elist dlist //从 elist 尾部弹出并将其压入 dlist 头部

"php"

127.0.0.1:6379> lrange elist 0 -1

1) "mysql"

2) "java"

127.0.0.1:6379> lrange dlist 0 -1

1) "php"

2) "two"

3) "one"

10.lindex 返回名称为 key 的 list 中 index 位置的元素

127.0.0.1:6379> lrange dlist 0 -1

1) "php"

2) "two"

3) "one"

127.0.0.1:6379> lindex dlist 1

"two"

11.llen 返回 key 对应 list 的长度（几个元素）

127.0.0.1:6379> lrange dlist 0 -1

1) "php"

2) "two"

3) "one"

127.0.0.1:6379> llen dlist

(integer) 3

d) 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 0（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。

1. sadd 向名称为 key 的 set 中添加元素

127.0.0.1:6379> sadd myset one

(integer) 1

127.0.0.1:6379> sadd myset two //添加成功返回 1，失败返回 0

(integer) 1

127.0.0.1:6379> sadd myset two

(integer) 0

127.0.0.1:6379> smembers myset //查看集合名为 myset 的所有成员

1) "one"

2) "two"

2. srem 删除名称为 key 的 set 中的元素 （ remove ）

127.0.0.1:6379> smembers myset

1) "one"

2) "two"

127.0.0.1:6379> srem myset one // 删除成功返回 1，失败返回 0

(integer) 1

127.0.0.1:6379> smembers myset

1) "two"

3. spop 随机返回并删除名称为 key 的 set 中一个元素 （随机 = 无序 ）

127.0.0.1:6379> smembers myset

1) "three"

2) "one"

3) "four"

4) "two"

127.0.0.1:6379> spop myset

"three"

127.0.0.1:6379> spop myset

"one"

4. sdiff 返回所有给定 key 与第一个 key 的差集 （ different ）

127.0.0.1:6379> smembers amyset

1) "one"

2) "two"

127.0.0.1:6379> smembers bmyset

1) "one"

2) "two"

127.0.0.1:6379> sdiff amyset bmyset // 没有差集返回

(empty list or set)

127.0.0.1:6379> srem bmyset two //删除

(integer) 1

127.0.0.1:6379> sadd bmyset three //添加

(integer) 1

127.0.0.1:6379> sdiff amyset bmyset //以 amyset 为基准

1) "two"

127.0.0.1:6379> sdiff bmyset amyset //以 bmyset 为基准

1) "three"

5. 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

127.0.0.1:6379> sdiff bmyset amyset

1) "three"

127.0.0.1:6379> sdiffstore bamyset bmyset amyset

(integer) 1

127.0.0.1:6379> smembers bamyset

1) "three"

6. sinter 返回所有给定 key 的交集

127.0.0.1:6379> smembers bmyset

1) "three"

2) "one"

127.0.0.1:6379> smembers amyset

1) "one"

2) "two"

127.0.0.1:6379> sinter amyset bmyset //前后无关系

1) "one"

127.0.0.1:6379> sinter bmyset amyset

1) "one"

7. sinterstore 返回所有给定 key 的交集，并将结果存为另一个 key

127.0.0.1:6379> sinter bmyset amyset

1) "one"

127.0.0.1:6379> sinterstore nbamyset bmyset amyset

(integer) 1

127.0.0.1:6379> smembers nbamyset

1) "one"

8. sunion 返回所有给定 key 的并集

127.0.0.1:6379> smembers amyset

1) "one"

2) "two"

127.0.0.1:6379> smembers bmyset

1) "three"

2) "one"

127.0.0.1:6379> sunion amyset bmyset

1) "three"

2) "one"

3) "two"

9. sunionstore 返回所有给定 key 的并集，并将结果存为另一个 key

127.0.0.1:6379> sunion amyset bmyset

1) "three"

2) "one"

3) "two"

127.0.0.1:6379> sunionstore uabmyset amyset bmyset

(integer) 3

127.0.0.1:6379> smembers uabmyset

1) "three"

2) "one"

3) "two"

10. 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

127.0.0.1:6379> smembers bmyset

1) "three"

2) "one"

127.0.0.1:6379> smembers amyset

1) "one"

2) "two"

127.0.0.1:6379> smove amyset bmyset two //把 amyset 的 two 成员移到 bmyset

(integer) 1

127.0.0.1:6379> smembers amyset

1) "one"

127.0.0.1:6379> smembers bmyset

1) "three"

2) "one"

3) "two"

11. scard 返回名称为 key 的 set 的元素个数

127.0.0.1:6379> smembers bmyset

1) "three"

2) "one"

3) "two"

127.0.0.1:6379> scard bmyset

(integer) 3

12. sismember 测试 member 是否是名称为 key 的 set 的元素

127.0.0.1:6379> sismember bmyset one

(integer) 1

127.0.0.1:6379> sismember bmyset five

(integer) 0

13. srandmember 随机返回名称为 key 的 set 的一个元素，但不删除元素

127.0.0.1:6379> smembers bmyset

1) "three"

2) "one"

3) "two"

127.0.0.1:6379> srandmember bmyset

"two"

127.0.0.1:6379> srandmember bmyset

"two"

e) Sorted sets 类型

Sorted sets 是 set 的一个升级版本，它在 set 的基础上添加一个顺序属性，这个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。

1. zadd 向名称为 key 的 zset 中添加元素

127.0.0.1:6379> zadd myzset 1 one

(integer) 1

127.0.0.1:6379> zadd myzset 2 two

(integer) 1

127.0.0.1:6379> zadd myzset 3 two // 集合中不能有相同成员，会覆盖顺序

(integer) 0

127.0.0.1:6379> zrange myzset 0 -1 withscores //withscores 输出顺序标识

1) "one"

2) "1"

3) "two"

4) "3"

2. zrem 删除名称为 key 的 zset 中的元素 member

127.0.0.1:6379> zrange myzset 0 -1 withscores

1) "one"

2) "1"

3) "two"

4) "3"

127.0.0.1:6379> zrem myzset one // 删除

(integer) 1

127.0.0.1:6379> zrange myzset 0 -1 withscores

1) "two"

2) "3"

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

127.0.0.1:6379> zrange myzset 0 -1 withscores

1) "five"

2) "1"

3) "two"

4) "3"

5) "one"

6) "4"

127.0.0.1:6379> zincrby myzset 1 two // 改变 two 顺序

"4"

127.0.0.1:6379> zincrby myzset 1 five

"2"

127.0.0.1:6379> zincrby myzset 2 three // 不会改变其他成员顺序

"2"

127.0.0.1:6379> zrange myzset 0 -1 withscores

1) "five"

2) "2"

3) "three"

4) "2"

5) "one"

6) "4"

7) "two"

8) "4"

4. zrank 返回名称为 key 的 zset 中 member 元素的排名（索引值）

（按 score 从小到大排序）即下标

127.0.0.1:6379> zrange bzset 0 -1 withscores

1) "one"

2) "1"

3) "three"

4) "2"

5) "five"

6) "3"

7) "two"

8) "3"

9) "six"

10) "4"

127.0.0.1:6379> zrank bzset five

(integer) 2

5. 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

127.0.0.1:6379> zrevrank bzset two

(integer) 1

6. 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

127.0.0.1:6379> zrange bzset 0 -1 withscores

1) "one"

2) "1"

3) "three"

4) "2"

5) "five"

6) "3"

7) "two"

8) "3"

9) "six"

10) "4"

127.0.0.1:6379> zrevrange bzset 0 -1 withscores

1) "six"

2) "4"

3) "two"

4) "3"

5) "five"

6) "3"

7) "three"

8) "2"

9) "one"

10) "1"

7. zrangebyscore　　返回集合中 score 在给定区间的元素（顺序）

127.0.0.1:6379> zrange czset 0 -1 withscores

1) "five"

2) "1"

3) "four"

4) "2"

5) "three"

6) "3"

7) "two"

8) "4"

127.0.0.1:6379> zrangebyscore czset 2 4 withscores // 删除顺序 2 到 4

1) "four" //成员

2) "2" // 顺序值

3) "three"

4) "3"

5) "two"

6) "4"

8. zcount 返回集合中 score 在给定区间的数量

127.0.0.1:6379> zcount czset 2 4

(integer) 3

9. zcard 返回集合中元素个数

127.0.0.1:6379> zcard czset

(integer) 4

10. zremrangebyrank 删除集合中 score 在给定区间的元素

127.0.0.1:6379> zrange czset 0 -1 withscores

1) "five"

2) "1"

3) "four"

4) "2"

5) "three"

6) "3"

7) "two"

8) "4"

127.0.0.1:6379> zremrangebyrank czset 1 2 // 降序排序并删除索引值 1 到 2 成员

(integer) 2

127.0.0.1:6379> zrange czset 0 -1 withscores

1) "five"

2) "1"

3) "two"

4) "4"

6.Redis 的常用命令

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。

1.键值相关命令

2.服务器相关命令

a) 键值相关命令

1．Keys 返回满足给定 pattern 的所有 key

127.0.0.1:6379> keys *list

1) "mylist"

2) "alist"

3) "nlist"

4) "newlist"

5) "wlist"

6) "elist"

7) "dlist"

8) "blist"

9) "clist"

2．del 删除一个 key

127.0.0.1:6379> del clist

(integer) 1

127.0.0.1:6379> exists clist

(integer) 0

3．expire 设置一个 key 的过期时间

127.0.0.1:6379> expire age 10

(integer) 1

127.0.0.1:6379> exists age // 检查键是否存在

(integer) 1

127.0.0.1:6379> ttl age // 获取 key 有效时长

(integer) -2

127.0.0.1:6379> exists age

(integer) 0

4．move 将当前数据库中的 key 转移到其他数据库中

Redis 默认有 0～15 数据库。默认进入 0 数据库

127.0.0.1:6379> select 0　　// 选择数据库

OK

127.0.0.1:6379> set age 25

OK

127.0.0.1:6379> get age

"25"

127.0.0.1:6379> move age 1　// 转移到数据库 1

(integer) 1

127.0.0.1:6379> get age

(nil)

127.0.0.1:6379> select 1

OK

127.0.0.1:6379[1]> get age

"25"

5．persist 移除给定 key 的过期时间（取消过期时间）

127.0.0.1:6379> expire sex 300

(integer) 1

127.0.0.1:6379> ttl sex　//获取 key 有效时长

(integer) 293

127.0.0.1:6379> persist sex

(integer) 1

127.0.0.1:6379> ttl sex

(integer) -1

6．randomkey 随机返回 key 空间的一个 key

127.0.0.1:6379> randomkey

"dlist"

127.0.0.1:6379> randomkey

"newlist"

127.0.0.1:6379> randomkey

"alist"

7． rename 重命名 key

127.0.0.1:6379> rename blist glist

OK

127.0.0.1:6379> exists blist

(integer) 0

127.0.0.1:6379> get blist

(nil)

8．type 返回 key 类型

127.0.0.1:6379> type blist

none

127.0.0.1:6379> type glist

list

b) 服务器相关命令

1. ping 测试连接是否存活

127.0.0.1:6379> exit

[root@localhost bin]# pkill redis-server

[root@localhost bin]# netstat -tunpl | grep 6379

[root@localhost bin]# ./redis-server ../etc/redis.conf //指定配置文件

[root@localhost bin]# ./redis-cli

127.0.0.1:6379> ping

PONG

2. 在命令行打印一些内容

127.0.0.1:6379> echo hello

"hello"

3. 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。

127.0.0.1:6379> select 1

OK

127.0.0.1:6379[1]> select 16

(error) ERR invalid DB index

4. dbsize 返回当前数据库中 key 的数目

127.0.0.1:6379> dbsize

(integer) 4

5. Info 获取服务的信息和统计

6. config get 实时转储收到的请求

127.0.0.1:6379> config get *

1) "dbfilename"

2) "dump.rdb"

.....

127.0.0.1:6379> config get dir

1) "dir"

2) "/Data/apps/redis/bin"

7. flushdb 删除当前选择数据库中的所有 key

127.0.0.1:6379[2]> dbsize

(integer) 3

127.0.0.1:6379[2]> flushdb

OK

127.0.0.1:6379[2]> dbsize

(integer) 0

8. flushall 删除所有数据库中的所有 key

127.0.0.1:6379[2]> dbsize

(integer) 3

127.0.0.1:6379[2]> select 1

OK

127.0.0.1:6379[1]> dbsize

(integer) 4

127.0.0.1:6379[1]> flushall

OK

127.0.0.1:6379[1]> dbsize

(integer) 0

127.0.0.1:6379[1]> select 2

OK

127.0.0.1:6379[2]> dbsize

(integer) 0

7.Redis 的高级应用

1. **安全性**
2. **主从复制**
3. **事务处理**
4. **持久化机制**
5. **发布订阅消息**
6. **虚拟内存的使用**

a) 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

注意：

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 15 万次的密码尝试，这意味着你需要指定非常复杂的密码来防止爆力破解。

步骤：

1. 在配置文件找到选项 # requirepass foobared

   requirepass　自定义密码字符串（beijng）

2. 修改配置文件重启服务

   [root@localhost bin]# netstat -tunpl | grep 6379

   tcp 0　0 0.0.0.0:6379 0.0.0.0:*　LISTEN　11741/./redis-serve

   tcp 0　0 :::6379　:::*　LISTEN　11741/./redis-serve

   [root@localhost bin]# pkill redis

   [root@localhost bin]# netstat -tunpl | grep 6379

   [root@localhost bin]# redis-server ../etc/redis.conf

   [root@localhost bin]# ./redis-cli

   127.0.0.1:6379> select 2

   (error) NOAUTH Authentication required.

3. 使用 auth 授权

   127.0.0.1:6379> auth beijing

   OK

   127.0.0.1:6379> select 2

   OK

4. 直接在登录客户端时指定授权密码，使用 -a 密码串

   [root@localhost bin]# redis-cli -a beijing

   127.0.0.1:6379> select 2

   OK

b) 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，还可以连接到其他 slave(为了防止 master 不能提供服务，当 master 宕机时，该 slave 直接转化为 master 角色）

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（不会影响客户端向 master 写入数据等操作）

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。（写进程与快照进程相互不影响）

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

a) 配置 slave 服务很简单，只需要在 slave 的配置文件中加入以下配置：

   i. slaveof 192.168.1.1 6379 # 指定 master 的 ip 和 端口

   ii. masterauth beijing #授权主机登录密码

步骤：

1.查看主机 ip 和从机 ip，保证之间能通（ifconfig eth0，使用 ping 测试是否连通）

备注：在虚拟机操作，在克隆虚拟机时遇到问题和解决方案：

问题：ifconfig eth0

eth1: error fetching interface information: Device not found

解决方案：

vi /etc/udev/rules.d/70-persistent-net.rules

将 NAME="eth1" 改成 NAME="eth0" ,并将 ATTR{address}=="00:0c:29:48:85:9c" 和/etc/sysconfig/network-scripts/ifcfg-eth0 的 HWADDR="00:0C:29:48:85:9C"对应。然后 reboot

2.测试之间 ping 通

3.配置 slave 服务

iii. slaveof 192.168.1.1 6379 # 指定 master 的 ip 和 端口

iv. masterauth beijing #授权主机登录密码

4.在 master 设置的键值，在 slave 服务可以获得，表示成功

5.使用 info 查看当前主机状态

Replication

role:slave // 从服务器

master_host:192.168.1.224

master_port:6379

master_link_status:up //与主服务器连接上

c) 事务处理

Redis 对事务的支持目前还比较简单。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

接中发起 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，而是先放在一个队列中，当执行 exec 命令时，redis 会顺序的执行队列中的所有命令。

1.创建事务

127.0.0.1:6379> get age

"10"

127.0.0.1:6379> multi

OK

127.0.0.1:6379> set age 15

QUEUED

127.0.0.1:6379> set age 20

QUEUED

127.0.0.1:6379> exec

1) OK

2) OK

127.0.0.1:6379> get age

"20"

2.取消事务

discard 命令就是清空事务的命令队列并退出事务上下文，也就是事务回滚。

127.0.0.1:6379> set name zhangsan

OK

127.0.0.1:6379> multi

OK

127.0.0.1:6379> set name lisi

QUEUED

127.0.0.1:6379> discard //取消事务

OK

127.0.0.1:6379> get name

"zhangsan"

3.Redis 事务回滚局限性

127.0.0.1:6379> get name

"zhangsan"

127.0.0.1:6379> get age

"20"

127.0.0.1:6379> incr name //字符串不能自增

(error) ERR value is not an integer or out of range

127.0.0.1:6379> multi //开始事务

OK

127.0.0.1:6379> incr age //让 age 自增

QUEUED

127.0.0.1:6379> incr name //让name自增 (此测试事务中的任何一步失败后会不会回滚整体)

QUEUED

127.0.0.1:6379> exec //执行事务

1) (integer) 21

2) (error) ERR value is not an integer or out of range

127.0.0.1:6379> get age //结果事务没有进行回滚，只是依次执行事务中的操作

"21"

127.0.0.1:6379> get name

"zhangsan"

总结：

本实例中看到，age 由于是一个数字，那么它是可以自增运算，但是 name 是个字符串无法进行自增运算，所以会报错，如果按传统的关系型数据库会将整个事务进行回滚，但是在 redis 数据库还是会将命令进行提交执行，所以 redis 的事务处理有一定的局限性。

d) 乐观锁复杂事务控制（类似于版本控制器）

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，一般是通过为数据库表添加一个”version“字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，如果提交数据版本号大于数据库当前版本号，则给予更新，否则认为是过期数据。

1.redis 乐观锁实例：

WATCH 命令可以监控一个或多个键，一旦其中有一个键被修改（或删除），之后的事务就不会执行。监控一直持续到 EXEC 命令（事务中的命令是在 EXEC 之后才执行的，所以在 MULTI 命令后可以修改 WATCH 监控的键值）。

127.0.0.1:6379> set age 10

OK

127.0.0.1:6379> watch age //监听

OK

127.0.0.1:6379> set age 20

OK

127.0.0.1:6379> multi //开启事务

OK

127.0.0.1:6379> set age 100

QUEUED

127.0.0.1:6379> set age 101

QUEUED

127.0.0.1:6379> exec

(nil)

127.0.0.1:6379> get age

"20"

分析：执行 WATCH 命令后、事务执行前修改了 key 的值，所以最后事务中的命令没有执行，EXEC 命令返回空结果。

2.Watch 命令会监视给定的 key，当 exec 时候如果监视的 key 从调用 watch 后发生过变化，则整体事务会失败。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，监听和事务都会被自动清除。当然了 exec，discard，unwatch 命令都会清楚连接中的所有监视。

e) 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：

1.snapshotting（快照）也是默认方式。

2.Append-only file （缩写 aof）的方式。

i. 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 被修改就会自动做快照。

实例：

save 900 1 #900 秒内如果操作 1 个 key 被修改，则发起快照保存

save 300 10 #300 秒内如果操作 10 个 key 被修改，则发起快照保存

[root@localhost bin]# cat dump.rdb //二进制文件

REDIS0006þageÞnamzhangsanlissanyarenÿ¹1L[root@localhost bin]#

ii. 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，redi 会将每一个收到的写命令都通过 write 函数追加到 appendonly.aof 文件中，当 redis 重启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。

Appendonly yes //启用 aof 持久化方式

#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。

#appendfsync everysec //每秒钟写入磁盘一次，在性能和持久化方面做了很好的折中。

#appendfsync no //完全依赖 os，性能最好，持久化没有保证

f) 发布及订阅消息

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 将消息类型称为通道。当发布者通过 publish 命令向 redis 发送特定类型的消息时，订阅该消息类型的全部 client 都会收到此消息。

实例：打开三个会话终端（session）

步骤一：

第一个终端会话：

[root@localhost bin]# who am i

root pts/1 2016-03-02 08:45 (192.168.1.100)

[root@localhost bin]# ./redis-cli -a beijing //登录授权

127.0.0.1:6379> subscribe tv1 //订阅 tv1

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

1) "subscribe"

2) "tv1"

3) (integer) 1

第二个终端会话：

[root@localhost bin]# who am i

root pts/0 2016-03-02 13:21 (192.168.1.100)

[root@localhost bin]# ./redis-cli -a beijing

127.0.0.1:6379> subscribe tv1 tv2 //订阅 tv1 tv2

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

1) "subscribe"

2) "tv1"

3) (integer) 1

1) "subscribe"

2) "tv2"

3) (integer) 2

步骤二：

第三个终端会话：（测试一）

[root@localhost bin]# who am i

root pts/2 2016-03-02 13:21 (192.168.1.100)

[root@localhost bin]# ./redis-cli -a beijing

127.0.0.1:6379> publish tv1 zhangsan　　//向 tv1 发广播

(integer) 2

结果：

第一会话终端：

127.0.0.1:6379> subscribe tv1

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

1) "subscribe"

2) "tv1"

3) (integer) 1

1) "message"

2) "tv1"

3) "zhangsan"

第二个会话终端：

127.0.0.1:6379> subscribe tv1 tv2

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

1) "subscribe"

2) "tv1"

3) (integer) 1

1) "subscribe"

2) "tv2"

3) (integer) 2

1) "message"

2) "tv1"

3) "zhangsan"

第三个会话终端：（测试二）

127.0.0.1:6379> publish tv2 wangwu

(integer) 1

第一个会话终端：

没有接收到任何信息

第二个会话终端：

127.0.0.1:6379> subscribe tv1 tv2

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

1) "subscribe"

2) "tv1"

3) (integer) 1

1) "subscribe"

2) "tv2"

3) (integer) 2

1) "message"

2) "tv1"

3) "zhangsan"

1) "message"

2) "tv2"

3) "wangwu"

g) 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中。

虚拟内存配置：

```
vm-enabled yes                  #开启 vm 功能
vm-swap-file /tmp/redis.swap    #交换出来的 value 保存的文件路径
vm-max-momery 1000000           #使用最大的内存上限
vm-page-size 32                 # 每个页面的大小 32 字节
vm-pages 134217728              #最多使用多少页面
vm-max-threads 4                # 用于执行 value 对象换入的工作线程数量
really-use-vm yes               # 确定使用虚拟内存
```

8.Redis 连接 PHP

a) 安装 php 扩展模块

b) Php redis 使用手册

1.安装 php 扩展

a) 下载 phpredis 扩展 （到官网查看） http://pecl.php.net/package/redis

   i. wget https://codeload.github.com/phpredis/phpredis/zip/develop

   ii. unzip develop #解压

   iii. cd phpredis-develop/

iv. 为 redis 准备编译环境

[root@localhost phpredis-develop]# /Data/apps/php/bin/phpize

```
-rwxr-xr-x 1 root root 452643 3月   2 15:45 configure
```

phpredis-develop 目录下生成一个配置文件。

v. 在 configure 文件目录下，配置 PHP 扩展 redis 模块

./configure --with-php-config=/Data/apps/php/bin/php-config

vi. 编译 make

vii. 安装 make install

Redis.so： /Data/apps/php/lib/php/extensions/no-debug-zts-20121212/

viii. 配置 php.ini 加载 redis 模块

vim /Data/apps/php/etc/php.ini 添加一行

extension = “redis.so”

ix. 重启服务

b) Php-redis 使用手册

i. Redis-PHP 扩展类使用手册.pdf

ii.

9.Redis -- PHP 实现小型用户信息系统

a) 类型介绍

i. String 最简单的数据类型

ii. Hash 可当成表 hash table 比 string 速度快

iii. List 栈 队列

iv. Set 并集 交集 差集

v. Zset set 的升级版，多了一个顺序

b) 用户信息增删改查

Redis.php

```
<?php
    //实例化
    $redis = new Redis();
    //连接服务器
    $redis->connect("localhost");
    //授权
    $redis->auth("lamplijie");
```

Add.php

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

Reg.php

```
<?php
    require("redis.php");
    $username = $_POST['username'];
    $password = md5($_POST['password']);
    $age = $_POST['age'];
    $uid = $redis->incr("userid"); //自增
    $redis->hmset("user:".$uid,array("uid"=>$uid,"username"=>$username,"password"=>
$password,"age"=>$age));
    header("location:list.php");
```

List.php

```
<a href="add.php">注册</a>
<?php
    require("redis.php");
    for($i=1;$i<=($redis->get("userid"));$i++){
        $data[] = $redis->hgetall("user:".$i);
```

```
    }
    //var_dump($data);
    $data = array_filter($data);
?>
<table border=1>
    <tr>
        <th>uid</th>
        <th>username</th>
        <th>age</th>
        <th>操作</th>
    <tr>
<?php foreach($data as $v){?>
    <tr>
        <td><?php echo  $v['uid']?></td>
        <td><?php echo  $v['username']?></td>
        <td><?php echo  $v['age']?></td>
        <td><a  href="del.php?id=<?php  echo  $v['uid']?>">删 除 </a> <a href="mod.php?id=<?php echo $v['uid']?>">编辑</a></td>
    </tr>
<?php }?>
</table>
```

Mod.php

```
<?php
    require("redis.php");
    $uid = $_GET['id'];
    $data = $redis->hgetall("user:".$uid);
?>
<form action="doedit.php" method="post">
    <input type="hidden" value="<?php echo $data['uid']?>" name="uid" />
    用 户 名 ： <input type="text" name="username" value="<?php echo $data['username']?>" /><br />
    年龄：<input type="text" name="age" value="<?php echo $data['age']?>" /><br />
    <input type="submit" value="修改" />
    <input type="reset" value="重新填写" />
</form>
```

Doedit.php

```
<?php
```

```
require("redis.php");
$uid = $_POST['uid'];
$username = $_POST['username'];
$age = $_POST['age'];
$a = $redis->hmset("user:".$uid,array("username"=>$username,"age"=>$age));
if($a){
    header("location:list.php");
}else{
    header("location:mod.php?id=".$uid);
}
```

Del.php

```
<?php
    require("redis.php");
    $uid = $_GET['id'];
    $redis->del("user:".$uid);
    header("location:list.php");
```

c) 用户登录+分页+关注

Redis.php

```
<?php
    //实例化
    $redis = new Redis();
    //连接服务器
    $redis->connect("localhost");
    //授权
    $redis->auth("lamplijie");
```

Add.php

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

Reg.php

```
<?php
    require("redis.php");
    $username = $_POST['username'];
    $password = md5($_POST['password']);
    $age = $_POST['age'];
    $uid = $redis->incr("userid");
    $redis->hmset("user:".$uid,array("uid"=>$uid,"username"=>$username,"password"=>
$password,"age"=>$age));
    $redis->rpush("uid",$uid);
    $redis->set("username:".$username,$uid);
    header("location:list.php");
```

List.php

```
<a href="add.php">注册</a>

<?php
    require("redis.php");
    if(!empty($_COOKIE['auth'])){
    $id = $redis->get("auth:".$_COOKIE['auth']);
    $name = $redis->hget("user:".$id,"username");
?>

欢迎您，<?php echo $name?>，<a href="logout.php">退出</a>


<?php
}else{
?>


<a href="login.php">登陆</a>

<?php
}
    //用户总数
    $count = $redis->lsize("uid");
    //页大小
    $page_size = 3;
    //当前页码
    $page_num = (!empty($_GET['page']))?$_GET['page']:1;
    //页总数
    $page_count = ceil($count/$page_size);
    $ids                                                                    =
```

```
$redis->lrange("uid",($page_num-1)*$page_size,(($page_num-1)*$page_size+$page_size-1))
;

    //var_dump($ids);
    /*for($i=1;$i<=($redis->get("userid"));$i++){
        $data[] = $redis->hgetall("user:".$i);
    }*/
    foreach($ids as $v){
        $data[] = $redis->hgetall("user:".$v);
    }
    //var_dump($data);
    //$data = array_filter($data);
?>
<table border=1>
    <tr>
        <th>uid</th>
        <th>username</th>
        <th>age</th>

        <th>操作</th>

    <tr>
<?php foreach($data as $v){?>
    <tr>
        <td><?php echo  $v['uid']?></td>
        <td><?php echo  $v['username']?></td>
        <td><?php echo  $v['age']?></td>

        <td><a  href="del.php?id=<?php  echo  $v['uid']?>"> 删 除 </a>  <a

href="mod.php?id=<?php echo $v['uid']?>">编辑</a>


        <?php if(!empty($_COOKIE['auth'])&&$id!=$v['uid']){?>

        <a href="addfans.php?id=<?php echo $v['uid']?>&uid=<?php echo $id?>">加关注

</a></td>
        <?php }?>
    </tr>
<?php }?>
<tr>
    <td colspan="4">

        <a href="?page=<?php echo (($page_num-1)<=1)?1:($page_num-1)  ?>">上一页

</a>
        <a                          href="?page=<?php                          echo
```

```
(($page_num+1)>=$page_count)?$page_count:($page_num+1)    ?>">下一页</a>

            <a href="?page=1">首页</a>

            <a href="?page=<?php echo $page_count?>">尾页</a>

            当前<?php echo $page_num?>页

            总共<?php echo $page_count?>页

            总共<?php echo $count?>个用户

        </td>
</tr>
</table>

<table border=1>

        <caption>我关注了谁</caption>

        <?php
            $data = $redis->smembers("user:".$id.":following");
            foreach($data as $v){
                $row = $redis->hgetall("user:".$v);
        ?>
        <tr>
            <td><?php echo $row['uid']?></td>
            <td><?php echo $row['username']?></td>
            <td><?php echo $row['age']?></td>
        </tr>
        <?php
            }
        ?>
</table>

<table border=1>

        <caption>我的粉丝</caption>

        <?php
            $data = $redis->smembers("user:".$id.":followers");
            foreach($data as $v){
                $row = $redis->hgetall("user:".$v);
        ?>
        <tr>
            <td><?php echo $row['uid']?></td>
```

```
            <td><?php echo $row['username']?></td>
            <td><?php echo $row['age']?></td>
        </tr>
        <?php
            }
        ?>
</table>
```

Mod.php

```
<?php
    require("redis.php");
    $uid = $_GET['id'];
    $data = $redis->hgetall("user:".$uid);
?>
<form action="doedit.php" method="post">
    <input type="hidden" value="<?php echo $data['uid']?>" name="uid" />
    用 户 名 ： <input type="text" name="username" value="<?php echo $data['username']?>" /><br />
    年龄：<input type="text" name="age" value="<?php echo $data['age']?>" /><br />
    <input type="submit" value="修改" />
    <input type="reset" value="重新填写" />
</form>
```

Doedit.php

```
<?php
    require("redis.php");
    $uid = $_POST['uid'];
    $username = $_POST['username'];
    $age = $_POST['age'];
    $a = $redis->hmset("user:".$uid,array("username"=>$username,"age"=>$age));
    if($a){
        header("location:list.php");
    }else{
        header("location:mod.php?id=".$uid);
    }
```

Del.php

```
<?php
    require("redis.php");
    $uid = $_GET['id'];
    $redis->del("user:".$uid);
    $redis->lrem("uid",$uid);
    header("location:list.php");
```

Login.php

```
<?php
    require("redis.php");
    $username = $_POST['username'];
    $pass = $_POST['password'];
    $id = $redis->get("username:".$username);
    if(!empty($id)){
        $password = $redis->hget("user:".$id,"password");
        if(md5($pass) == $password){
            $auth = md5(time().$username.rand());
            $redis->set("auth:".$auth,$id);
            setcookie("auth",$auth,time()+86400);
            header("location:list.php");
        }
    }


?>
<form action="" method="post">
    用户名:<input type="text" name="username" /><br />

    密码：<input type="password" name="password" /><br />

    <input type="submit" value="登陆" />
</form>
```

Logout.php

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

addfans.php

```
<?php
$id = $_GET['id'];
$uid = $_GET['uid'];
require("redis.php");
$redis->sadd("user:".$uid.":following",$id);
$redis->sadd("user:".$id.":followers",$uid);
header("location:list.php");
```