一. Redis

1. 什么是 Redis

Redis 是 Remote Dictionary Server(远程数据服务)的缩写
由意大利人 antirez(Salvatore Sanfilippo) 开发的一款 内存高速缓存数据库
该软件使用 C 语言编写,它的数据模型为 key-value
它支持丰富的数据结构(类型),比如 String list hash set sorted set。
可持久化(随时把备份到硬盘中一份),保证了数据安全。

同一个 select 查询语句,每天需要被执行查询 100 万次,为了减轻数据库的负载,就把查询好的数据给缓存起来(存储在内存中),每天的第一个用户执行从 mysql 中获得数据并存储到内存中,第二个 到 第 100 万个用户就直接从内存中获得数据。

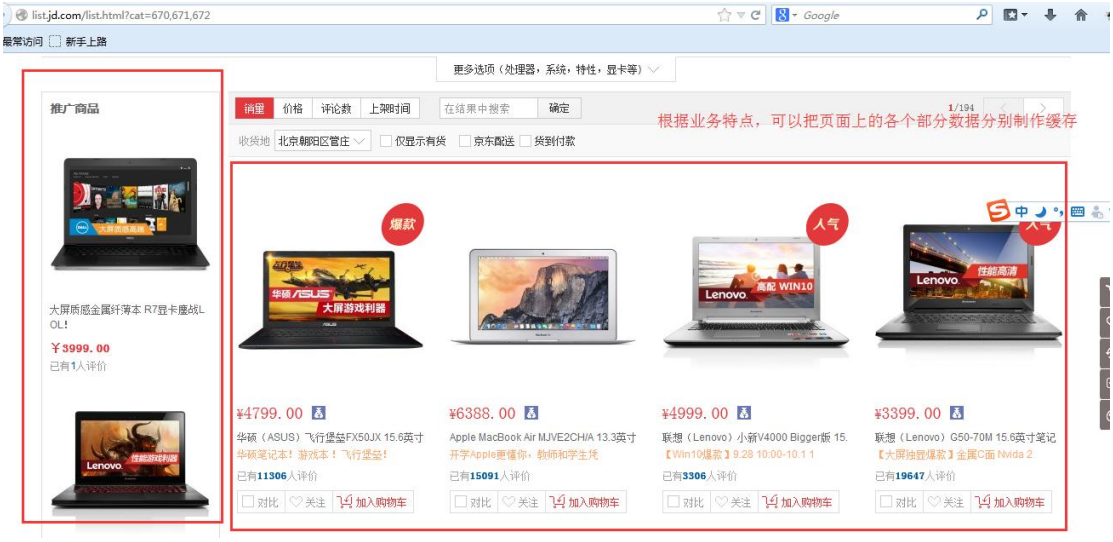
使用**缓存**减轻数据库的负载。
在开发网站的时候如果有一些数据在短时间之内不会发生变化,而它们还要被频繁访问,为了提高用户的请求速度和降低网站的负载,就把这些数据放到一个读取速度更快的介质上(或者是通过较少的计算量就可以获得该数据),该行为就称作对该数据的缓存。

该介质可以是文件、数据库、内存,内存介子经常用于数据缓存。

缓存的两种形式:
页面缓存经常用在 CMS(content manage system)内存管理系统里边(Smarty 缓存)
数据缓存经常会用在页面的具体**数据**里边

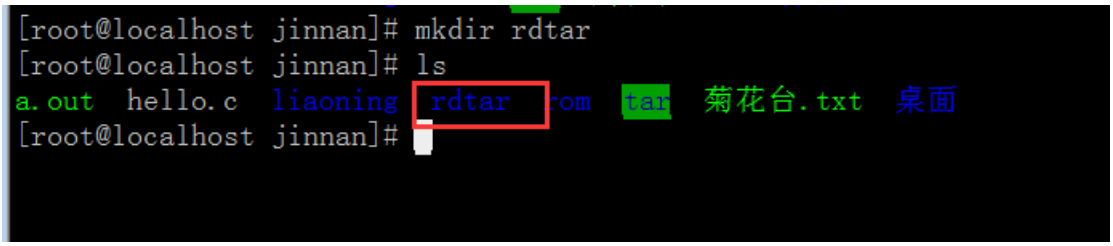
新闻信息(数据不变化、有实时性)页面适合做页面缓存:

商品展示页面(数据有各种分类)，为了降低数据库负载，他们比较适合做各个小部分的数据缓存，数据更新也只是更新每个小块的数据缓存：

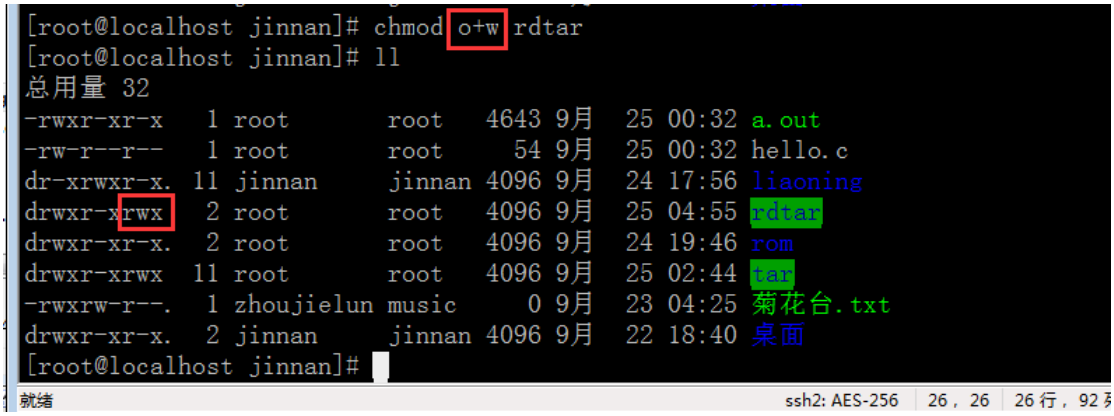


3. 安装 redis

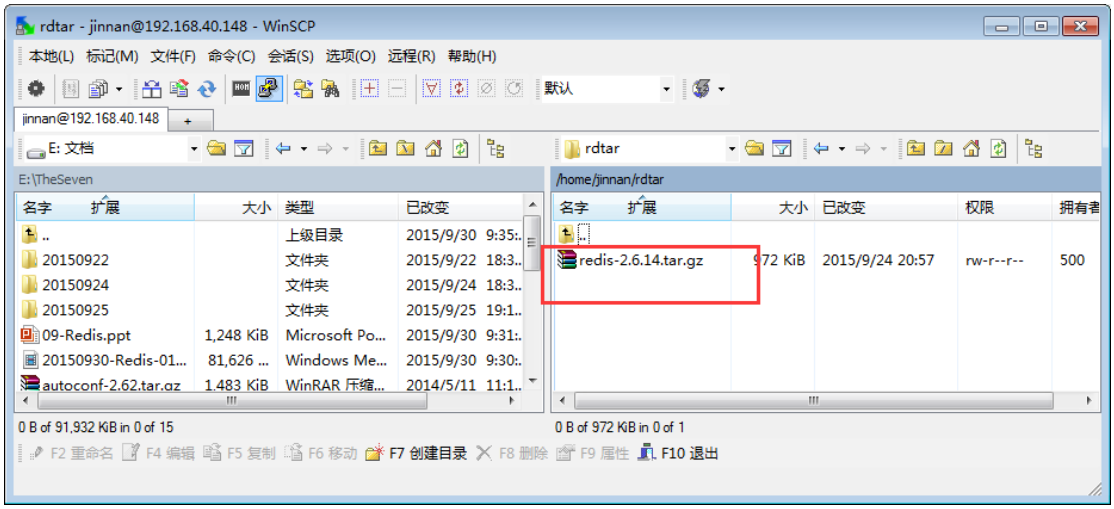
把 redis 需要的软件上传到此目录：



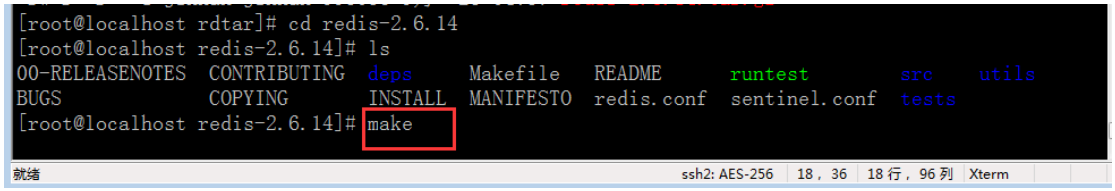
把其他组用户的 w 写权限赋予给 rdtar 目录：



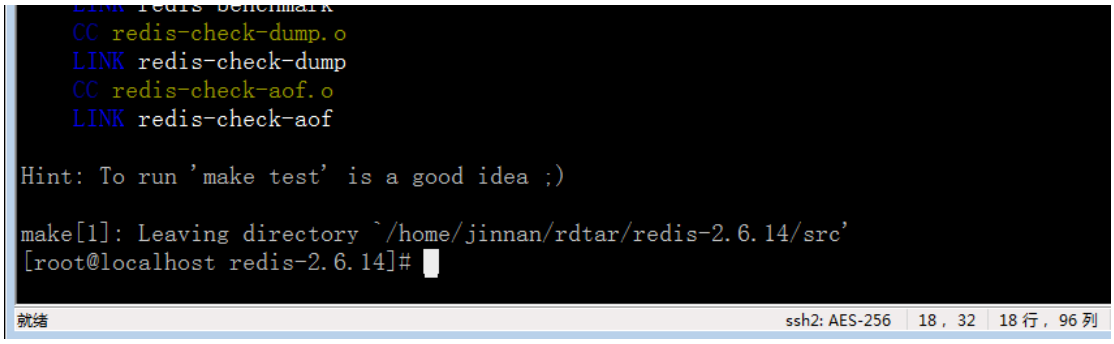
把需要安装的软件上传到服务器：



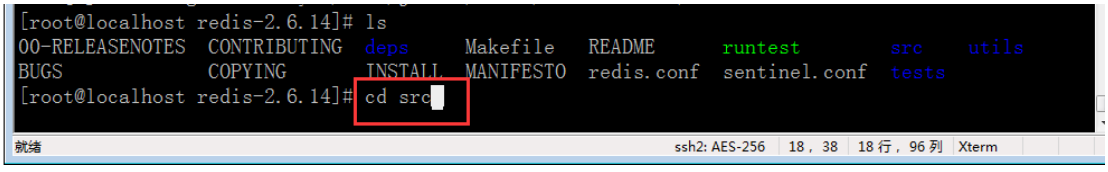
进入解压目录直接 make 即可：

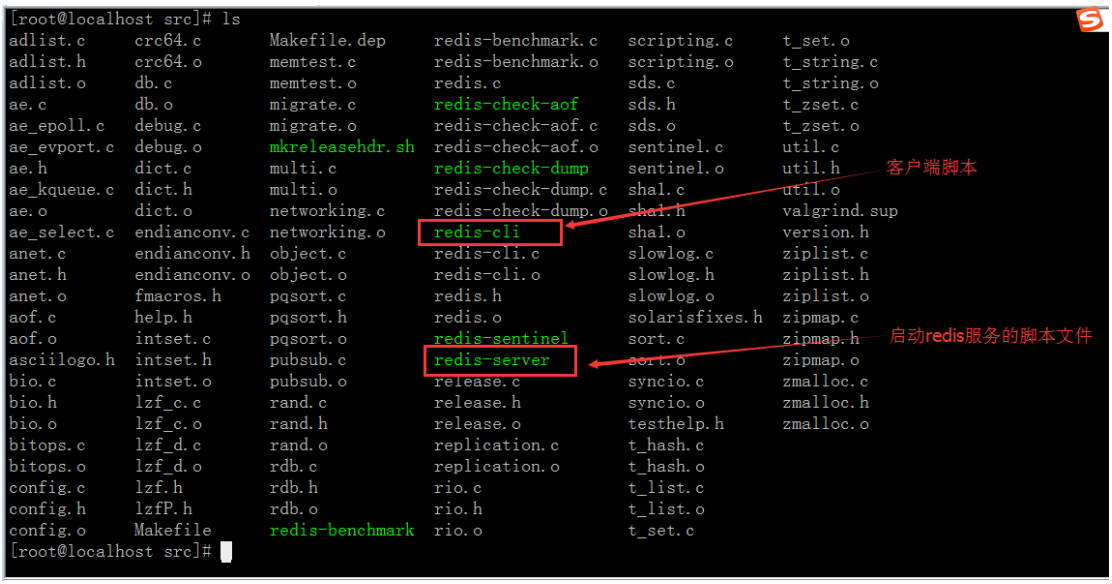


make 执行成功：



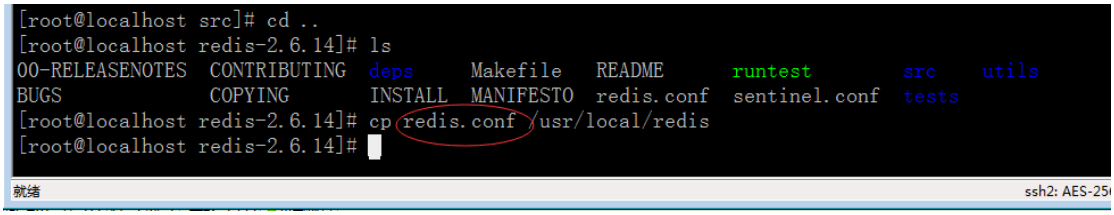
进入 src 目录：



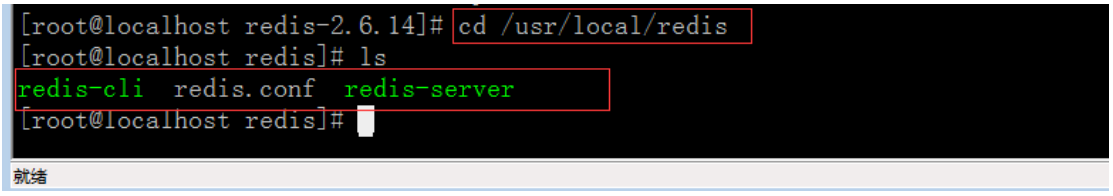


创建 redis 运行目录，并拷贝需要的两个文件：

进入 redis 直接解压目录并拷贝 配置文件到运行目录：



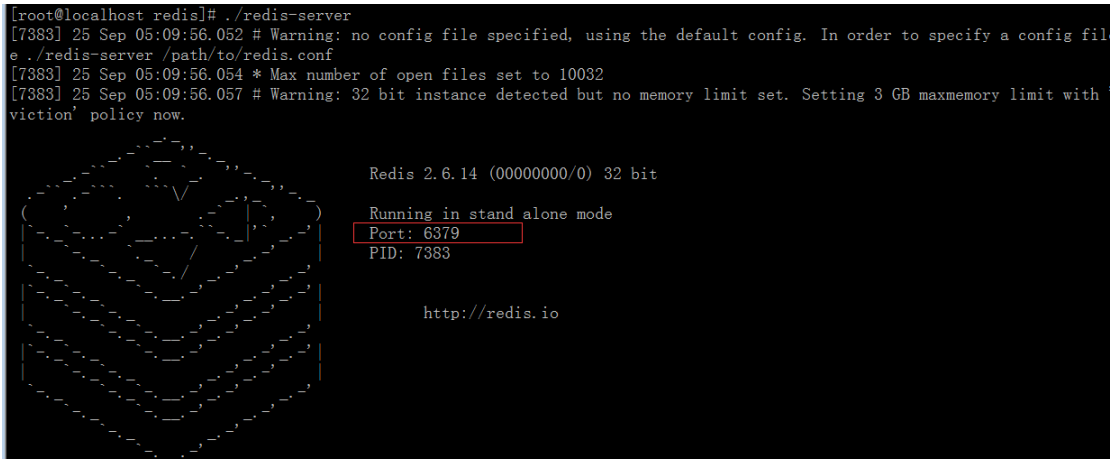
进入 redis 运行目录，内部有需要的三个文件(刚才拷贝的)：



默认为前台启动 redis 服务：

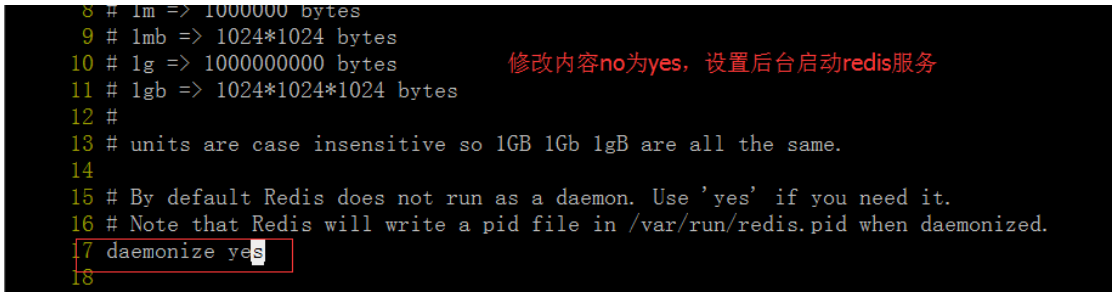
前台启动服务：始终有一个终端脚本被挂起执行(终端脚本被关闭后立即停止服务，不推荐)

后台启动服务：服务以隐藏的方式执行，没有终端脚本，可以通过 ps -A | grep 名称 查看是否有该服务。



Ctrl+c 关闭前台服务

修改配置文件(/usr/local/redis/redis.conf)，设置后台启动 redis 服务：



带着 redis.conf 配置文件参数，后台启动服务：

4．简单使用

```
[root@localhost redis]# ./redis-cli
redis 127.0.0.1:6379> set name tom
OK
redis 127.0.0.1:6379> set age 23
OK
redis 127.0.0.1:6379> set height 170
OK
redis 127.0.0.1:6379> get name
"tom"
redis 127.0.0.1:6379> get age
"23"
redis 127.0.0.1:6379> get height
"170"
redis 127.0.0.1:6379>
```

设置变量信息

读取变量信息

二．具体使用

redis 中数据的模型为：key/value
类似在 php 中定义变量：名称 = 值；

1． key 的操作

在 redis 里边，除了"\n"和空格 不能作为名字的组成内容外，其他内容都可以作为 key 的名字部分。名字长度不做要求。

redis 中 key 的组成内容较随意(没有\n 和空格即可)：

```
redis 127.0.0.1:6379> set 123_abc_!@#*()<>IUIYUH beijing2008
OK
redis 127.0.0.1:6379>
```

redis 中一共有 16 个数据库(redis.conf)：

```
79 # syslog-facility local0
80
81 # Set the number of databases. The default database is DB 0, you can se
82 # a different one on a per-connection basis using SELECT <dbid> where
83 # dbid is a number between 0 and 'databases'-1
84 databases 16
```

通过模糊方式查看当前数据库全部的 key 名称信息(*星 代表任意名字信息)：

```
redis 127.0.0.1:6379> keys *
1) "name"
2) "age"
3) "123_abc_!@#*()<>IUIYUH"
4) "number"
5) "height"
redis 127.0.0.1:6379>
```

就绪

查看当前数据库名称以 n 开始的 key 的名字：

```
redis 127.0.0.1:6379> keys n*
1) "name"
2) "number"
```

```
//① keys键操作
exists key              测试指定 key 是否存在
del key1 key2 ....keyN  删除给定 key
type key                返回给定 key 的 value 类型
keys pattern            返回匹配指定模式的所有 key
rename oldkey newkey    改名字
dbsize                  返回当前数据库的 key 数量
expire key seconds      为 key 指定过期时间
ttl key                 返回 key 的剩余过期秒数
select db-index         选择数据库
move key db-index       将 key 从当前数据库移动到指定数据库
flushdb                 删除当前数据库中所有 key
flushall                删除所有数据库中的所有 key
```

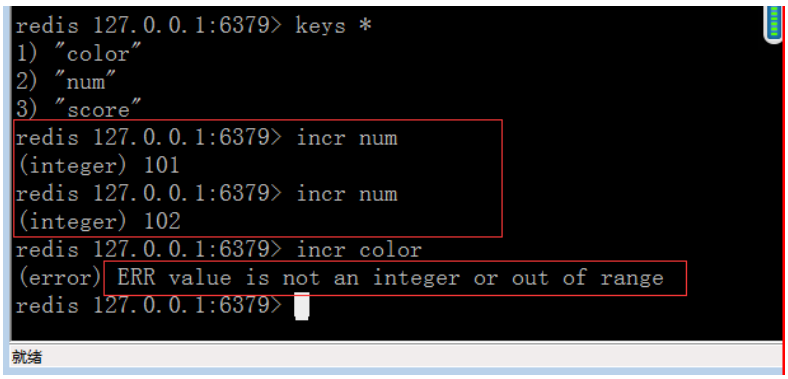
2. String 类型操作

string 是 redis 最基本的类型
redis 的 string 可以包含任何数据。包括 jpg 图片或者序列化的对象。
单个 value 值最大上限是 1G 字节。

```
//② string类型操作
set key value           设置 key 对应的值为 string 类型的 value
mset key1 value1 ... keyN valueN        一次设置多个 key 的值
mget key1 key2 ... keyN                 一次获取多个 key 的值
incr key                对 key 的值做加加操作,并返回新的值
decr key                同上，但是做的是减减操作
incrby key integer      同 incr，加指定值
decrby key integer      同 decr，减指定值
append key value        给指定 key 的字符串值追加 value
substr key start end    返回截取过的 key 的字符串值
```

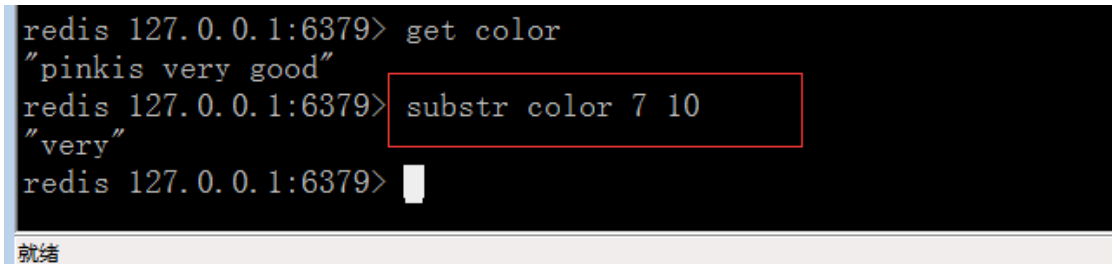
incr： increment 增长
该指令可以对 key 进行累加 1 操作，默认是累加 1 操作,类似 i++操作
该指令可以针对 新 key 或已有 key 进行操作

新 key：**创建**该 key 并累加 1，其值为 1
已有 key：key 的信息值类型要求必须为整型的
已有 key 的信息必须为“整型”的才允许 incr 操作：

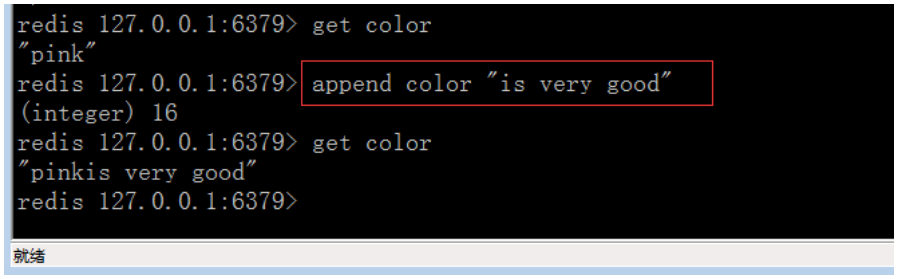


decr 的操作模式与 incr 一致，不过其实减 1 操作

substr：对内容进行截取，**包括 start 和 end** 标记位置内容

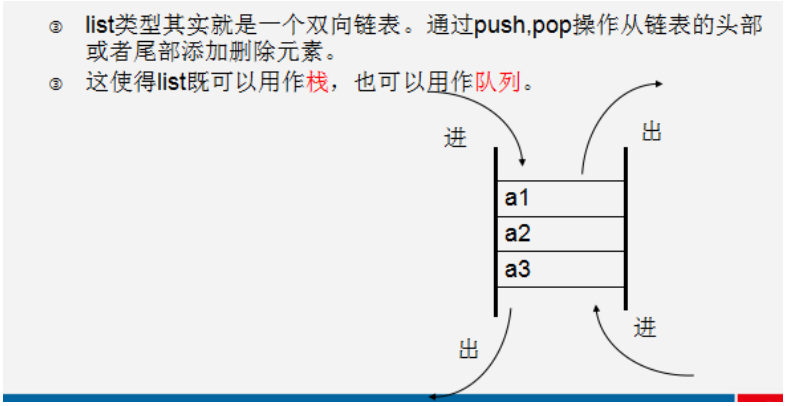


给 key 追加内容(如果被操作内容是空格分隔的多个信息，避免混淆，要使用引号)：



3．数据类型 List 链表

list 类型其实就是一个双向链表。通过 push,pop 操作从链表的头部或者尾部添加删除元素。
这使得 list 既可以用作栈，也可以用作队列。
上进上出 ： 栈
上进下出 ： 队列

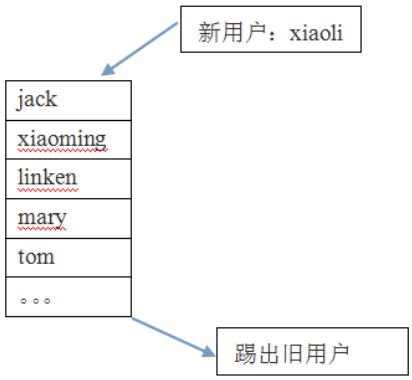


该 list 链表类型应用场合：

获得最新的 10 个登录用户信息：select * from user order by logintime desc limit 10;

以上 sql 语句可以实现用户需求，但是数据多的时候，全部数据都要受到影响查询，对数据库的负载比较高。必要情况还需要给关键字段(id 或 logintime)设置索引，索引也比较耗费系统资源

如果通过 list 链表实现以上功能，可以在 list 链表中只保留最新的 10 个数据，每进来一个新数据就删除一个旧数据。每次就可以从链表中直接获得需要的数据。极大节省各方面资源消耗



通过 list 链表保存登录系统的最新 5 个用户信息：
jim xiaoli jack xiaoming linken mary tom

给 newlogin 链表添加 5 个元素：

```
redis 127.0.0.1:6379[2]>
redis 127.0.0.1:6379[2]> lpush newlogin tom
(integer) 1
redis 127.0.0.1:6379[2]> lpush newlogin mary
(integer) 2
redis 127.0.0.1:6379[2]> lpush newlogin linken
(integer) 3
redis 127.0.0.1:6379[2]> lpush newlogin xiaoming
(integer) 4
redis 127.0.0.1:6379[2]> lpush newlogin jack
(integer) 5
redis 127.0.0.1:6379[2]> keys *
1) "newlogin"
redis 127.0.0.1:6379[2]>
```

newlogin 链表只保留 5 个元素，每进来一个新的旧删除一个旧的：

```
redis 127.0.0.1:6379[2]> lpush newlogin xiaoli
(integer) 6
redis 127.0.0.1:6379[2]> rpop newlogin
"tom"
redis 127.0.0.1:6379[2]>
```

链表只保留 5 个有顺序的元素存在：

```
redis 127.0.0.1:6379[2]> lrange newlogin 0 100
1) "xiaoli"
2) "jack"
3) "xiaoming"
4) "linken"
5) "mary"
redis 127.0.0.1:6379[2]>
```

按照指定区间对元素进行截取：

```
redis 127.0.0.1:6379[2]> lrange newlogin 0 100
1) "jim"
2) "xiaoli"
3) "jack"
4) "xiaoming"
5) "linken"
redis 127.0.0.1:6379[2]> ltrim newlogin 1 3
OK
redis 127.0.0.1:6379[2]> lrange newlogin 0 100
1) "xiaoli"
2) "jack"
3) "xiaoming"
redis 127.0.0.1:6379[2]>
```

就绪

4．set 集合类型

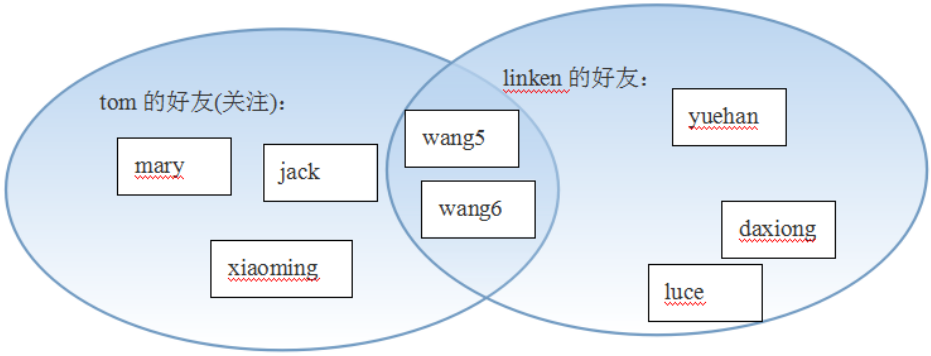
```
38 //④ set类型操作
39 sadd key member                  添加一个 string 元素到 key 对应的 set 集合中，成功返回 1，
40                                  如果元素已经在集合中 返回 0,key 对应的 set 不存在返回错误
41 srem key member [member]        从 key 对应 set 中移除给定元素，成功返回 1
42 smove p1 p2 member              从 p1 对应 set 中移除 member 并添加到 p2 对应 set 中
43 scard key                       返回 set 的元素个数
44 sismember key member            判断 member 是否在 set 中
45 sinter key1 key2...keyN         返回所有给定 key 的交集
46 sunion key1 key2...keyN         返回所有给定 key 的并集
47 sdiff key1 key2...keyN          返回所有给定 key 的差集
48 smembers key                    返回 key 对应 set 的所有元素，结果是无序的
```

redis 的 set 是 string 类型的无序集合。
set 元素最大可以包含(2 的 32 次方-1)个元素。
关于 set 集合类型除了基本的**添加、删除**操作，其他有用的操作还包含集合的取并集(union)，交集(intersection)，差集(difference)。通过这些操作可以很容易的实现 sns 中的好友推荐功能。

注意：每个集合中的各个元素**不能重复**。

该类型应用场合：qq 好友推荐。

tom 朋友圈(与某某是好友)：mary jack xiaoming wang5 wang6
linken 朋友圈(与某某是好友)：yuehan daxiong luce wang5 wang6



把 tom 的好友通过集合给设置好：

```
redis 127.0.0.1:6379[2]> sadd tomFri mary
(integer) 1
redis 127.0.0.1:6379[2]> sadd tomFri jack
(integer) 1
redis 127.0.0.1:6379[2]> sadd tomFri xiaoming
(integer) 1
redis 127.0.0.1:6379[2]> sadd tomFri wang5
(integer) 1
redis 127.0.0.1:6379[2]> sadd tomFri wang6
(integer) 1
redis 127.0.0.1:6379[2]> keys *
1) "tomFri"
2) "newlogin"
redis 127.0.0.1:6379[2]>
```

就绪

linken 的好友集合：

```
redis 127.0.0.1:6379[2]> sadd linkenFri yuehan
(integer) 1
redis 127.0.0.1:6379[2]> sadd linkenFri daxiong
(integer) 1
redis 127.0.0.1:6379[2]> sadd linkenFri luce
(integer) 1
redis 127.0.0.1:6379[2]> sadd linkenFri wang5
(integer) 1
redis 127.0.0.1:6379[2]> sadd linkenFri wang6
(integer) 1
redis 127.0.0.1:6379[2]> keys *
1) "linkenFri"
2) "tomFri"
3) "newlogin"
redis 127.0.0.1:6379[2]>
```

就绪

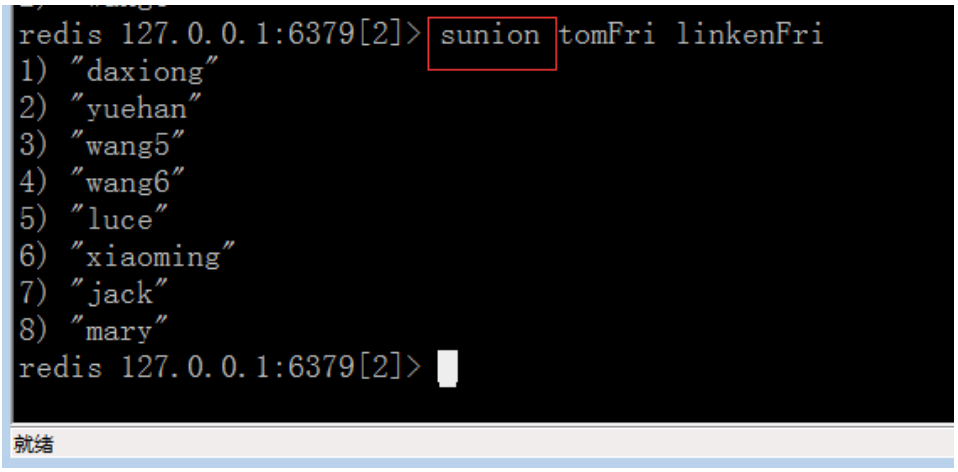
集合计算：

求交集：

```
redis 127.0.0.1:6379[2]> sinter tomFri linkenFri
1) "wang5"
2) "wang6"
redis 127.0.0.1:6379[2]>
```

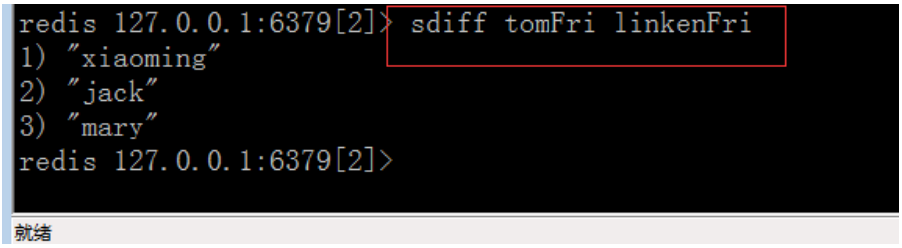
就绪

求并集：

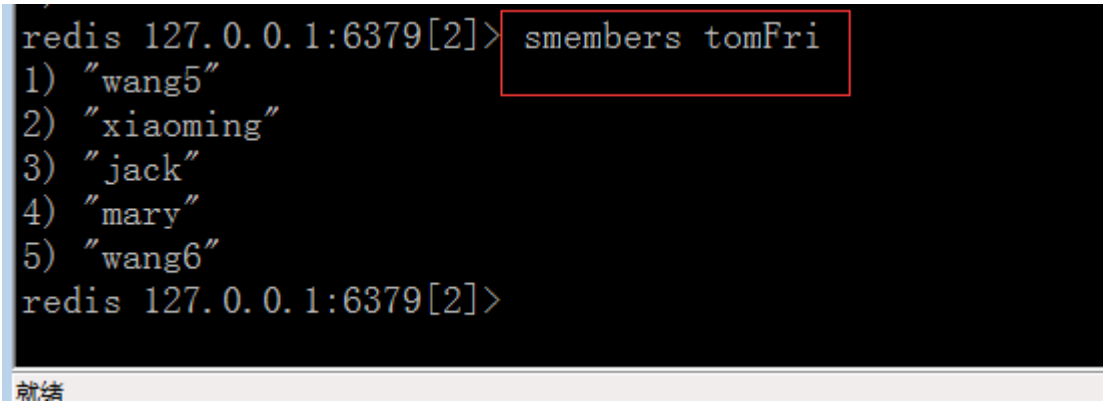


求差集(前者对后者求差集，结果只有前者的信息没有后者)：

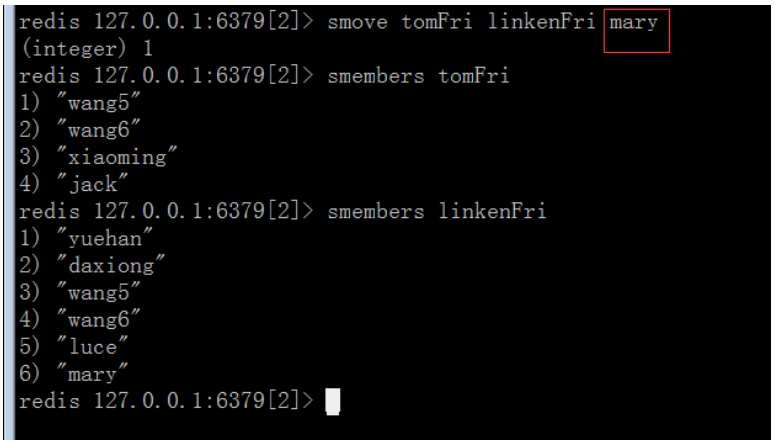
例如：tom 对 linken 求差集，结果只有 tom 的信息结果：



查看集合内部的全部元素信息：



把 mary 从 tom 集合里边移动到 linken 的集合中去：



总结：

1. redis 的安装和使用

key 的使用

key 具体操作：

exists　keys *　rename

del　dbsize　select 0-15

flushdb　flushall

数据类型：

String:

get　set　mget　mset　incr　decr　append　substr

List:

lpush　rpop　lrange

rpush　lpop　ltrim

Set 集合(集合运算)：

sadd　sinter　sunion　sdiff

smembers　scard　sismember

5. Sort Set 排序集合类型

该 Sort Set 是两种类型(list 和 set)的集中体现,称为排序集合类型。

和 set 一样 sorted set 也是 string 类型元素的集合，
不同的是每个元素都会关联一个**权**。
通过**权/值**可以有序的获取集合中的元素

```
50 //⑤ sorted set排序类型
51 zadd key score member          添加元素到集合，元素在集合中存在则更新对应 score
52 zrem key member                删除指定元素，1 表示成功，如果元素不存在返回 0
53 zincrby key incr member        按照 incr 幅度增加对应 member 的 score 值，返回 score 值
54 zrank key member               返回指定元素在集合中的排名（下标）,集合中元素是按 score 从小到大排序的
55 zrevrank key member            同上,但是集合中元素是按 score 从大到小排序
56 zrange key start end           类似 lrange 操作从集合中去指定区间的元素。返回的是有序结果
57 zrevrange key start end        同上，返回结果是按 score 逆序的
58 zcard key                      返回集合中元素个数
59 zscore key element             返回给定元素对应的 score
60 zremrangebyrank key min max    删除集合中排名在给定区间的元素
61
```

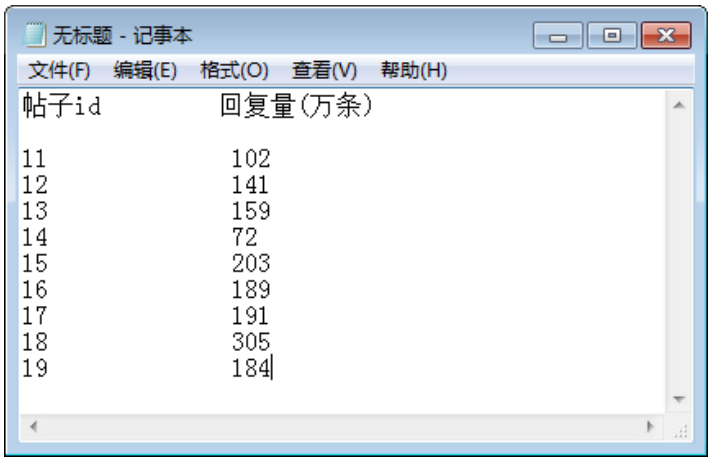
该 Sort set 类型适合场合：

获得最热门(回复量)前 5 个帖子信息：

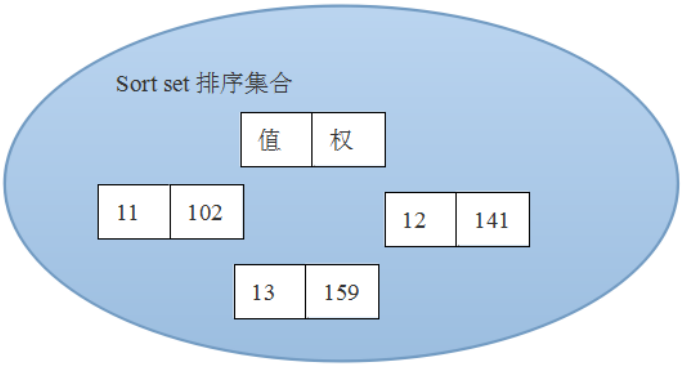
select * from message order by backnum desc limit 5;

(以上需求可以通过简单 sql 语句实现，但是 sql 语句比较耗费 mysql 数据库资源)

案例：利用 sort set 实现获取最热门的前 5 帖子信息



排序集合中的每个元素都是值、权的组合
（之前的 set 集合类型每个元素就只是一个 值）



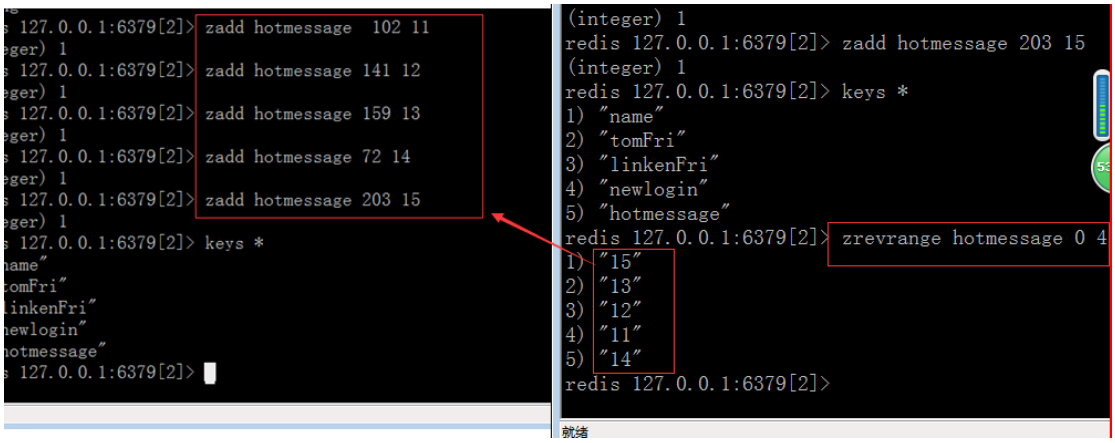
我们只做一个 sort set 排序集合，里边只保留 5 个元素信息，该 5 个元素是回复量最高的，
每个帖子被回复的时候，都有机会进入该集合里边，但是只有回复量最高的前 5 个帖子会存在于在集合，回复量低的就被删除。

制作一个排序集合类型的 key，内部 5 个元素：

```
redis 127.0.0.1:6379[2]> zadd hotmessage  102 11
(integer) 1
redis 127.0.0.1:6379[2]> zadd hotmessage 141 12
(integer) 1
redis 127.0.0.1:6379[2]> zadd hotmessage 159 13
(integer) 1
redis 127.0.0.1:6379[2]> zadd hotmessage 72 14
(integer) 1
redis 127.0.0.1:6379[2]> zadd hotmessage 203 15
(integer) 1
redis 127.0.0.1:6379[2]> keys *
1) "name"
2) "tomFri"
3) "linkenFri"
4) "newlogin"
5) "hotmessage"
redis 127.0.0.1:6379[2]>
```

就绪

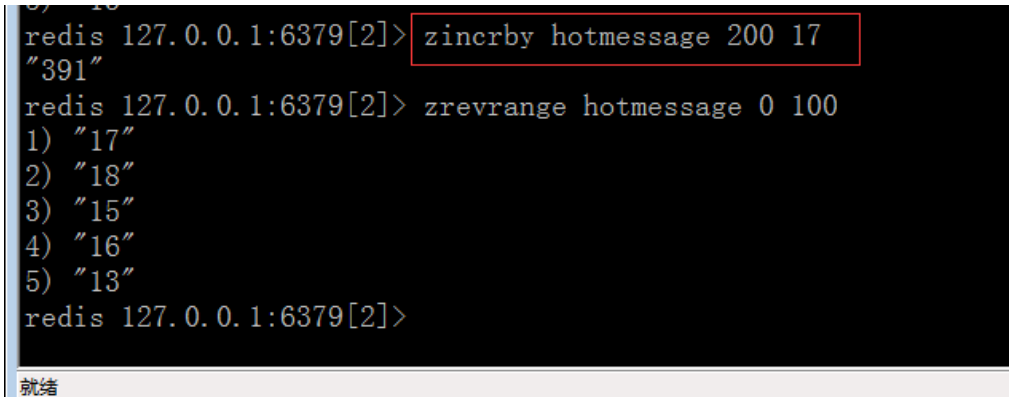
按照权值由高到低的顺序获得对应元素：



每进一个新元素，就删除一个权值最低的元素(保证集合中只有 5 个元素)：

```
redis 127.0.0.1:6379[2]> zadd hotmessage 189 16
(integer) 1
redis 127.0.0.1:6379[2]> zremrangebyrank hotmessage 0 0
(integer) 1
redis 127.0.0.1:6379[2]> zrevrange hotmessage 0 100
1) "15"
2) "16"
3) "13"
4) "12"
5) "11"
redis 127.0.0.1:6379[2]>
```

给指定值为 17 的元素的权累加 200 信息：


redis 127.0.0.1:6379[2]> zincrby hotmessage 200 17
"391"
redis 127.0.0.1:6379[2]> zrevrange hotmessage 0 100
1) "17"
2) "18"
3) "15"
4) "16"
5) "13"
redis 127.0.0.1:6379[2]>
就绪

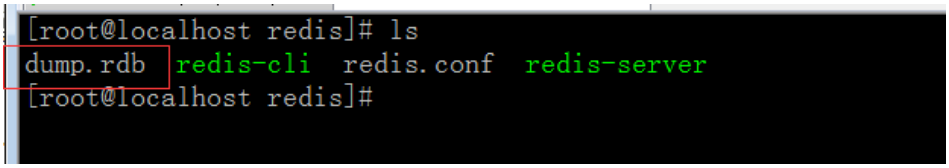
三．持久化功能

redis(nosql 产品)为了内部数据的安全考虑，会把本身的数据以**文件**形式保存到硬盘中一份，在服务器重启之后会自动把硬盘的数据恢复到内存(redis)的里边。
数据保存到硬盘的过程就称为“持久化”效果。

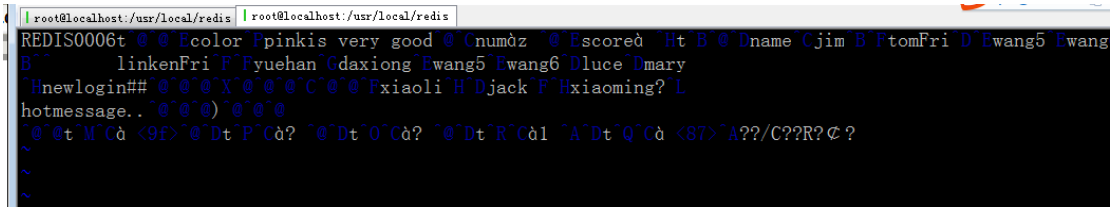
1． snap shotting 快照持久化

该持久化默认开启，一次性把 redis 中全部的数据保存一份存储在硬盘中，如果数据非常多(10-20G)就不适合频繁进行该持久化操作。

快照持久化备份文件：



备份文件 dump.rdb 内部的数据：



该方式备份机制(频率)：

save 900 1　　#900 秒内如果超过 1 个 key 被修改，则发起快照保存
save 300 10　　#300 秒超过 10 个 key 被修改，发起快照
save 60 10000　　#60 秒超过 10000 个 key 被修改，发起快照
以上三个备份频率需要同时存在：
数据变化非常快的时候，就快点做备份(保证数据安全)
数据变化慢的时候，就慢点做备份(节省服务器资源)

快照持久化备份文件的名称和目录设置：

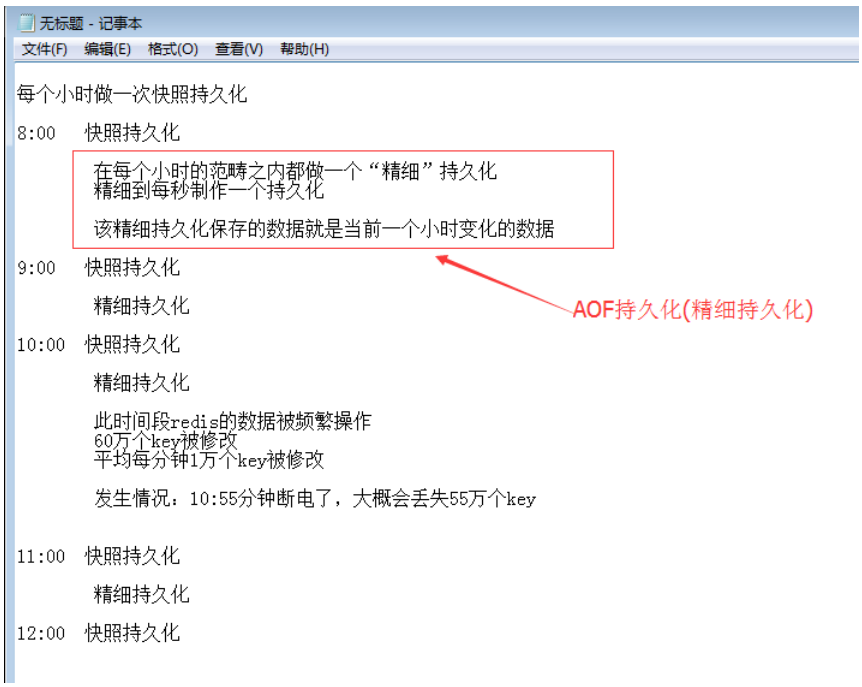
```
141
142 # The filename where to dump the DB
143 dbfilename dump.rdb
144
145 # The working directory.
146 #
147 # The DB will be written inside this directory, with the filename specified
148 # above using the 'dbfilename' configuration directive.
149 #
150 # The Append Only File will also be created inside this directory.
151 #
152 # Note that you must specify a directory here, not a file name.
153 dir ./
154
155 ################################# REPLICATION #################################
```

1.1 手动发起快照持久化

```
[root@localhost redis]# ll
总用量 8676
-rw-r--r-- 1 root root     280 9月  25 10:54 dump.rdb
-rwxr-xr-x 1 root root 3911782 9月  25 05:05 redis-cli
-rw-r--r-- 1 root root   25674 9月  25 05:14 redis.conf
-rwxr-xr-x 1 root root 4934060 9月  25 05:05 redis-server
[root@localhost redis]# ./redis-cli bgsave
Background saving started
[root@localhost redis]# ll
总用量 8676
-rw-r--r-- 1 root root     294 9月  25 10:57 dump.rdb
-rwxr-xr-x 1 root root 3911782 9月  25 05:05 redis-cli
-rw-r--r-- 1 root root   25674 9月  25 05:14 redis.conf
-rwxr-xr-x 1 root root 4934060 9月  25 05:05 redis-server
[root@localhost redis]#
就绪                                                    ssh2: AES-2
```

2. append only file （AOF 持久化）

本质：把用户执行的每个“写”指令（添加、修改、删除）都备份到文件中，还原数据的时候就是执行具体写指令而已。



该 AOF 持久化默认没有开启，现在就开启使用：
（可以自定义该持久化备份文件的名称）

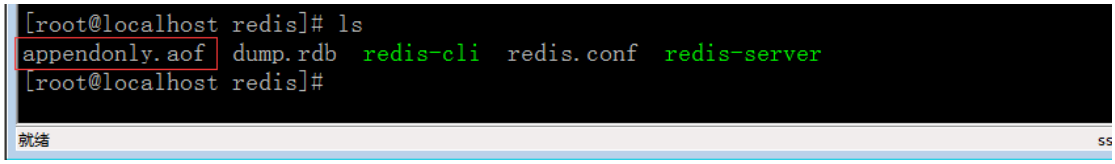
```
365 #
366 # Please check http://redis.io/topics/persistence for more information.
367
368 appendonly yes
369
370 # The name of the append only file (default: "appendonly.aof")
371 # appendfilename appendonly.aof
372
```

配置文件 redis.conf 被修改后，为了有效果，需要重启 redis 服务：
（杀掉旧进程，根据新配置文件启动新进程）

```
[root@localhost redis]# ls
dump.rdb  redis-cli  redis.conf  redis-server
[root@localhost redis]# ps -A | grep redis
 7407 ?        00:00:16 redis-server
 7416 pts/0    00:00:00 redis-cli
[root@localhost redis]# kill -9 7407
[root@localhost redis]# ./redis-server redis.conf
[root@localhost redis]# ps -A | grep redis
 7416 pts/0    00:00:00 redis-cli
 8323 ?        00:00:00 redis-server
[root@localhost redis]#
```

就绪 ssh2: A

AOF 持久化开启后会自动生成一个备份文件：



注意：AOF 持久化开启后会自动清除目前 redis 中的全部数据

AOF 备份的频率：

数据最安全　　服务器性能低

数据较安全　　服务器性能中等

数据不安全　　服务器性能高(优良)

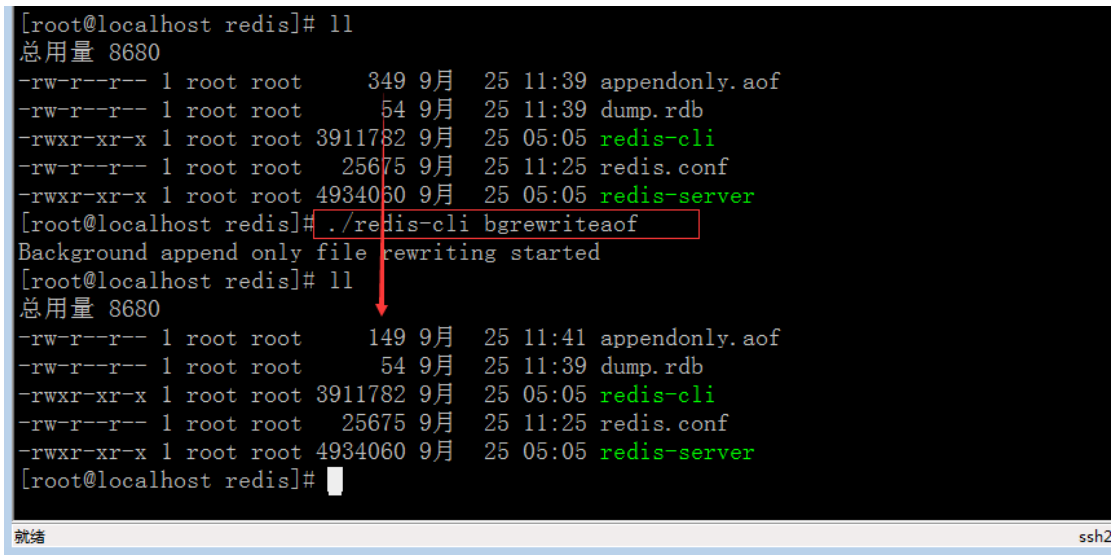
appendfsync always　//每次收到写命令就立即强制写入磁盘，最慢的，但是保证完全的持久化，不推荐使用

appendfsync everysec　//每秒钟强制写入磁盘一次，在性能和持久化方面做了很好的折中，推荐

appendfsync no　//完全依赖 os，性能最好,持久化没保证

2.1 为 aof 备份文件做优化压缩处理

例如：可以把多个 incr 指令换为一个 set 指令



持久化相关指令：

redis的持久化相关指令

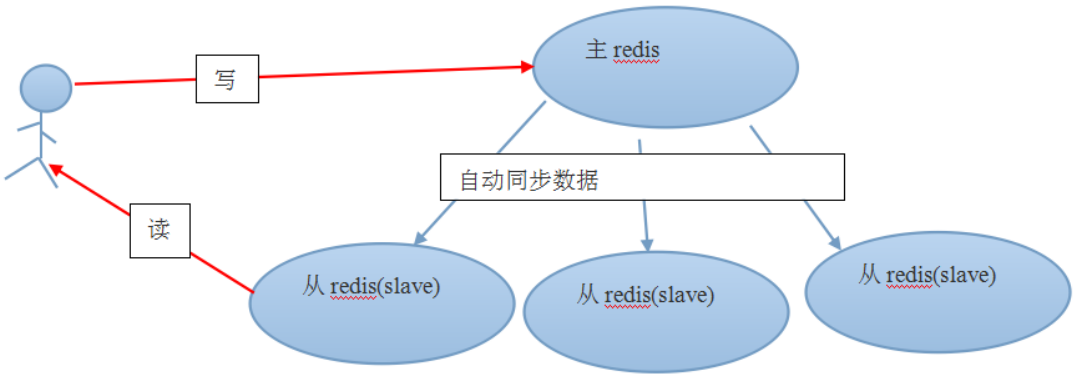
- bgsave 异步保存数据到磁盘(快照保存)
- lastsave 返回上次成功保存到磁盘的unix时间戳
- shutdown 同步保存到服务器并关闭redis服务器
- bgrewriteaof 当日志文件过长时优化AOF日志文件存储

- ./redis-cli bgrewriteaof
- ./redis-cli bgsave
- ./redis-cli -h 127.0.0.1 -p 6379 bgsave #手动发起快照

四. redis 的主从模式

mysql 为了降低每个服务器负载，可以设置读写分类（有写服务器、有读取服务器）

为了降低每个 redis 服务器的负载，可以多设置几个，并做主从模式
一个服务器负载“写”（添加、修改、删除）数据，其他服务器负载“读”数据
主服务器数据会“自动”同步给从服务器



在 redis.conf 里边设置并成为 192.168.40.148 服务器的从服务器：

```
156
157 # Master-Slave replication. Use slaveof to make a Redis instance a copy of
158 # another Redis server. Note that the configuration is local to the slave
159 # so for example it is possible to configure the slave to save the DB with a
160 # different interval, or to listen to another port, and so on.
161 #
162 slaveof 192.168.40.148 6379
163
164 # If the master is password protected (using the "requirepass" configuration
165 # directive below) it is possible to tell the slave to authenticate before
166 # starting the replication synchronization process, otherwise the master will
```

redis.conf 配置文件修改后杀掉旧进程，启动新进程：

```
[root@localhost redis]# ps -A | grep redis
 8323 ?        00:00:01 redis-server
 8335 pts/0    00:00:00 redis-cli
[root@localhost redis]# kill -9 8323
[root@localhost redis]# ./redis-server redis.conf
[root@localhost redis]#
```

就绪 ssh2

此时就可以看到主服务器自动同步给从服务器的数据

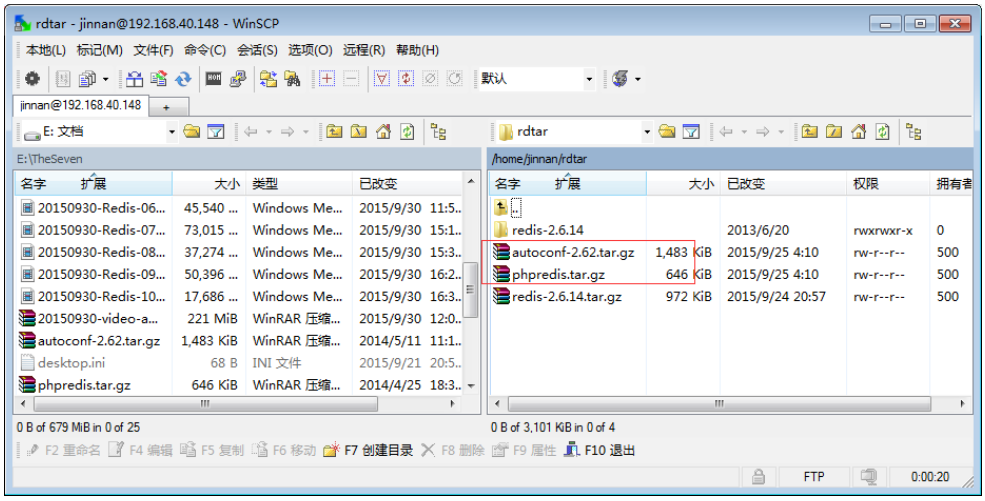
从服务器默认只读：

```
194 # Still a read only slave exports by default all the administrative commands
195 # such as CONFIG, DEBUG, and so forth. To a limited extend you can improve
196 # security of read only slaves using 'rename-command' to shadow all the
197 # administrative / dangerous commands.
198 slave-read-only yes
199
200 # Slaves send PINGs to server in a predefined interval. It's possible to change
201 # this interval with the repl_ping_slave_period option. The default value is 10
202 # seconds.
```

五． php 与 redis 结合

1． 安装 php 的 redis 扩展

上传 redis 扩展软件和依赖软件：



解压缩 phpredis 软件：

```
[root@localhost rdtar]# ls
autoconf-2.62.tar.gz  phpredis.tar.gz  redis-2.6.14  redis-2.6.14.tar.gz
[root@localhost rdtar]# tar zxf phpredis.tar.gz
[root@localhost rdtar]# ls
autoconf-2.62.tar.gz  phpredis  phpredis.tar.gz  redis-2.6.14  redis-2.6.14.tar.gz
[root@localhost rdtar]#
```

顺序：redis 与其他软件(xml、gd、jpeg 等等)都是 php 的扩展(php 依赖扩展软件)

正确的安装顺序是先安装依赖软件、之后在安装 php 软件

此时 redis 与 php 的安装顺序有前后颠倒的意味，但是 php 允许 redis 反方向安装进来。

在 phpredis 的解压目录下运行/usr/local/php/bin/phpize，以便 redis 反方向安装进 php 里边：

```
[root@localhost phpredis]# ls
acinclude.m4      config.log      CREDITS          Makefile            package.xml          redis_session.h
aclocal.m4        config.m4       debian           Makefile.fragments  php_redis.h          rpm
arrays.markdown   config.nice     debian.control   Makefile.global     README.markdown      run-tests.php
autom4te.cache    config.status   include          Makefile.objects    redis_array.c        serialize.list
build             config.sub      install-sh       missing             redis_array.h        tests
common.h          configure       library.c        mkdeb-apache2.sh    redis_array_impl.c
config.guess      configure.in    library.h        mkdeb.sh            redis_array_impl.h
config.h          config.w32      libtool          mkinstalldirs       redis.c
config.h.in       COPYING         ltmain.sh        modules             redis_session.c
[root@localhost phpredis]# /usr/local/php/bin/phpize
```

执行 phpize 时提示有软件(autoconf)依赖没哟解决：

```
[root@localhost phpredis]# /usr/local/php/bin/phpize
Configuring for:
PHP Api Version:         20090626
Zend Module Api No:      20090626
Zend Extension Api No:   220090626
Cannot find autoconf.  Please check your autoconf installation and the
$PHP_AUTOCONF environment variable.  Then, rerun this script.

[root@localhost phpredis]#
```

解压缩 autoconf 软件并安装：

```
[root@localhost rdtar]# ls
autoconf-2.62.tar.gz  phpredis  phpredis.tar.gz  redis-2.6.14  redis-2.6.14.tar.gz
[root@localhost rdtar]# tar zxf autoconf-2.62.tar.gz
[root@localhost rdtar]# ls
autoconf-2.62  autoconf-2.62.tar.gz  phpredis  phpredis.tar.gz  redis-2.6.14  redis-2.6.14.tar.gz
[root@localhost rdtar]# cd autoconf-2.62
[root@localhost autoconf-2.62]# ./configure && make && make install
```

就绪 ssh2: AES-256 20，68 20 行，

autoconf 依赖软件安装成功：

```
 /usr/bin/install -c -m 644 '././autoheader.1' '/usr/local/share/man/man1/autoheader.1'
 /usr/bin/install -c -m 644 '././autom4te.1' '/usr/local/share/man/man1/autom4te.1'
 /usr/bin/install -c -m 644 '././autoreconf.1' '/usr/local/share/man/man1/autoreconf.1'
 /usr/bin/install -c -m 644 '././autoscan.1' '/usr/local/share/man/man1/autoscan.1'
 /usr/bin/install -c -m 644 '././autoupdate.1' '/usr/local/share/man/man1/autoupdate.1'
 /usr/bin/install -c -m 644 '././ifnames.1' '/usr/local/share/man/man1/ifnames.1'
 /usr/bin/install -c -m 644 '././config.guess.1' '/usr/local/share/man/man1/config.guess.1'
 /usr/bin/install -c -m 644 '././config.sub.1' '/usr/local/share/man/man1/config.sub.1'
make[3]: Leaving directory `/home/jinnan/rdtar/autoconf-2.62/man'
make[2]: Leaving directory `/home/jinnan/rdtar/autoconf-2.62/man'
make[1]: Leaving directory `/home/jinnan/rdtar/autoconf-2.62'
[root@localhost autoconf-2.62]#
```

之后继续安装 phpize：

```
[root@localhost phpredis]# /usr/local/php/bin/phpize
Configuring for:
PHP Api Version:         20090626
Zend Module Api No:      20090626
Zend Extension Api No:   220090626
[root@localhost phpredis]#
```

可以看见，phpize 安装成功(前提是 autoconf 依赖软件先安装好)

下边开始安装 phpredis：

带着 php-config 参数值给 phpredis 做配置：

```
  LIBS        libraries to pass to the linker, e.g. -l<library>
  CPPFLAGS    C/C++/Objective C preprocessor flags, e.g. -I<include dir> if
              you have headers in a nonstandard directory <include dir>
  CPP         C preprocessor

Use these variables to override the choices made by `configure' or to help
it to find libraries and programs with nonstandard names/locations.

[root@localhost phpredis]# ./configure --with-php-config=/usr/local/php/bin/php-config
```

就绪 ssh2: AES-256 20

configure 指令执行成功：

```
checking whether stripping libraries is possible... yes
checking if libtool supports shared libraries... yes
checking whether to build shared libraries... yes
checking whether to build static libraries... no

creating libtool
appending configuration tag "CXX" to libtool
configure: creating ./config.status
config.status: creating config.h
config.status: config.h is unchanged
[root@localhost phpredis]#
```

就绪

之后执行 make && make install：

```
creating libtool
appending configuration tag "CXX" to libtool
configure: creating ./config.status
config.status: creating config.h
config.status: config.h is unchanged
[root@localhost phpredis]# make && make install
```

就绪 ssh2: AES-256

make && make install 执行成功：

```
See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------

Build complete.
Don't forget to run 'make test'.

Installing shared extensions:     /usr/local/php/lib/php/extensions/no-debug-non-zts-20090626/
[root@localhost phpredis]#
```

就绪 ssh2: AES-256 20 , 28

给 php 生成好的 redis 扩展文件：

```
Installing shared extensions:     /usr/local/php/lib/php/extensions/no-debug-non-zts-20090626/
[root@localhost phpredis]# ls /usr/local/php/lib/php/extensions/no-debug-non-zts-20090626/
redis.so
[root@localhost phpredis]#
```

redis针对php的扩展文件

就绪 ssh2: AES-256 20 , 28 20

在/usr/local/php/lib/php.ini 配置文件中开启 redis

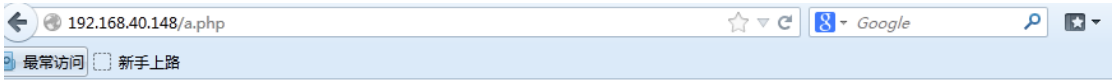
```
970 ;extension=php_pdo_sqlite.dll
971 ;extension=php_pgsql.dll
972 ;extension=php_phar.dll
973 ;extension=php_pspell.dll
974 ;extension=php_shmop.dll
975 extension=redis.so
976
977 ; The MIBS data available in the PHP distribution
```

之后重启 apache：

```
"/usr/local/php/lib/php.ini" 1895L, 68919C written
[root@localhost phpredis]# /usr/local/http2/bin/apachectl restart
[root@localhost phpredis]#
```

就绪

通过浏览器访问 php 代码(phpinfo())：

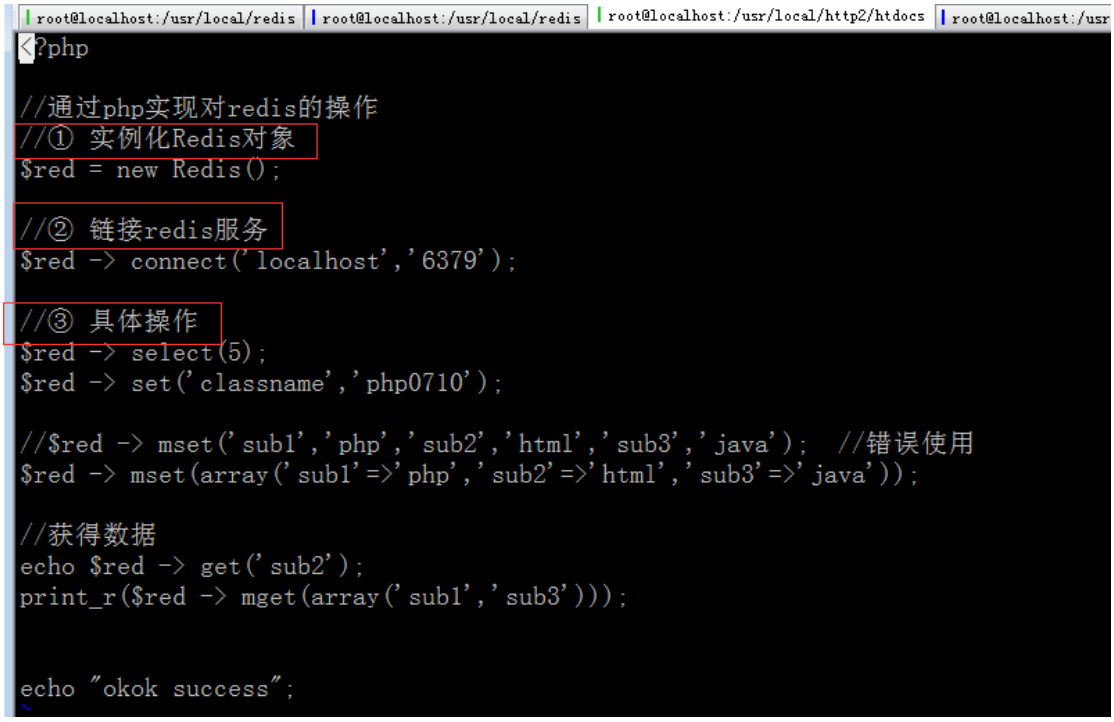


redis

Redis Support	enabled
Redis Version	2.2.4

2．通过 php 操作 redis

在 php 里边，redis 就是一个功能类 Redis，Redis 类里边有许多成员方法(名字基本与 redis 指令的名字一致，参数也一致)



3.php 中 redis 的可操作方法有哪些

获得 Redis 类内部一共的方法(利用反射 Reflection 实现):
php 大部分操作都是正向的:类、实例化对象、对象调用成员
其实类可以反向操作:类、反过来感知类的成员、反方向感知方法是否是公开的/私有的/受保护的/最终的

总结:

1. 数据类型 Sort Set
   获得最热门前 5 个帖子信息
   (优秀学员奥赛班分班使用)
2. 持久化
   1) 快照持久化(默认开启)
   2) AOF 追加持久化(精细),本质备份用户的“写指令”

3. 主从模式
4. php 实现 redis 操作
   1） 给 php 安装 redis 扩展
   2） php 对 redis 操作

作业：
1. 利用 list 链表数据类型实现网站首页面显示前 10 个登录用户信息。

问题：
php 代码都在是 wamp 环境下开发的，该环境 php 不支持 redis 使用
解决：
在 linux 系统里边创建两个 php 文件(user_add.php 和 user_look.php)
user_add.php 负责给 redis 写数据
user_look.php 负责 redis 读数据

在 windows 下触发 linux 中的 php 执行
file_get_contents('http://192.168.40.148/user_add.php?user_id=105');
file_get_contents('http://192.168.40.148/user_look.php');