Redis

第一章 Redis 简介

借用Redis官网上的介绍

> Redis is an open source, BSD licensed, 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是开源,BSD许可,高级的key-value存储系统. 可以用来存储字符串,哈希结构,链表,集合,因此,常用来提供数据结构服务.

redis和memcached相比,的独特之处:

1. redis可以用来做存储(storge), 而memcached是用来做缓存(cache)
   这个特点主要因为其有"持久化"的功能,redis为了读取速度,可以将数据存储于内存当中,为了数据的持久化,可以把数据同步存储到硬盘当中.
2. 存储的数据有"结构",对于memcached来说,存储的数据,只有1种类型"字符串",
   而redis则可以存储字符串,链表,哈希结构,集合,有序集合.

1.1 redis安装

我们去官网下载最新稳定版本,解压源码并进入目录,和其他软件不同,我们不必运行configure,可以直接make安装(如果是32位的机器,需使用make 32 bit),编译完毕,编译文件夹的src下新增加了一系列文件.

> 可选步骤: make test 测试编译情况,可能出现: need tcl >8.5这种情况, 只需要使用yum install tcl

在make install 过程当中,我们可以添加参数指定安装目录 `make PREFIX=/usr/local/redis install`.安装完毕,安装文件夹下新添加bin目录,目录中文件如下

- redis-benchmark 性能测试工具
- redis-check-aof 日志文件检测工(比如断电造成日志损坏,可以检测并修复)
- redis-check-rdb 快照文件检测工具,效果类上
- redis-cli 客户端
- redis-server 服务端

> redis以日志的形式将缓存当中的数据存储到硬盘上,其中aof文件存储的是语句日志,rdb文件存储的是数据日志.

1.2 redis的配置和连接

从README文件可知,我们启动的时候可以直接启动,也可以携带参数启动;

- 复制配置文件

```
cp /usr/local/src/redis-4.0.2/redis.conf /usr/local/redis
```

- 修改配置文件中的daemonize为yes用于静默启动

放到后台运行

- 启动redis

```
./bin/redis-server ./redis.conf
```

- 连接redis

```
./bin/redis-cli [-h localhost -p 6379]
```

第二章 Redis 使用

2.1 Redis 对key的通用操作

所有类型的键都可以使用

```
del key1 key2 ... keyn
```

作用 :删除一个或者多个键

返回值:不存在的key忽略,返回真正删除的key的数量

`rename key newkey`

作用:给key赋一个新的key名称,如果newkey已经存在,则newkey的原值会被覆盖

`move key db`

作用:移动某一个键到其他仓库;

redis在初始化的过程当中会建立0到15共16个仓库,我们可以通过select来决定切换至某个仓库,可以用move将键迁移到某个仓库.

`keys pattern`

作用:查询相应的key;

在redis里,允许模糊查询key,有3个通配符`*,?,[]`,其中

- ◦ 通配任意多个字符
- ? 通配单个字符
- [] 通配括号内的某个字符串

> keys* 慎用

`randomkey`

作用: 返回随机key

`exists key`

作用:判断key是否存在,返回1/0;

存在返回1，不存在返回0

`type key`

不存在返回none

作用:返回key存储的值的类型,有string,list,set,order set,hash

`ttl key`

作用:查询key的生命周期,返回秒数,对于不存在或者过期的key,返回-2,对于不自动失效的key,返回-1;

其他的返回key的生命周期

`expire key 整型值`

作用:设置key的声明周期,以秒为单位

pexpire以毫秒为单位设置key的声明周期

> 与上述两个指令相同作用的指令有 pexpire,pttl,单位都为毫秒

pttl

以毫秒为单位返回 key 的剩余的过期时间

`persist key`

作用:把指定key设置为永久有效

2.2 字符串类型的操作

`set key value [ex 秒数]/[px 毫秒数] [nx]/[xx]`

作用:设置一个键值对,可以配置失效时间和修改条件

- nx 表示key不存在时,执行操作
- xx 表示key存在时,执行操作

> 注意:如果ex,px同时写,以后面的有效期为准

`mset`

作用:一次性设置多个键值,例如mset key1 v1 key2 v2...

`get`

`mget key1,key2,...keyn`

作用: 获取多个key的值

`setrange key offset value`

作用:把字符串的offset偏移字节改成value

```
redis 127.0.0.1:6379>set greet hello
OK
redis 127.0.0.1:6379>setrange greet 2 x
(integer) 5
redis 127.0.0.1:6379>get greet
hexllo
```

偏移从0开始

> 注意:如果偏移量>字符长度,该字符自动补为\X00

`getrange key start stop`

作用:获取字符串中[start,stop]范围的值

> 注意:对于字符串的下标,左数从0开始,右数从-1开始

`append key value`

作用:把value追加到key的原值上

`getset key newvalue`

作用:获取并返回旧值,设置新值

`incr key`

作用:指定的key增加1,并返回加1之后的值

`decr key`

`incrby key num`

'decrby key num'

`incrbyfloat key floatnumber`

`getbit key offset`

作用:获取值的二进制表示,对应位上的值(从左,从0编号)

```
//A对应的asci码位65 a对应的asci码位97 即A位 0100 0001,a为 0110 0001

redis 127.0.0.1:6379> set char A
OK
redis 127.0.0.1:6379> getbit char 1
(integer) 1
redis 127.0.0.1:6379> getbit char 2
(integer) 0
redis 127.0.0.1:6379> getbit char 7
(integer) 1
```

`setbit key offset value`

作用:设置对应二进制位上的值,返回值为该位上的旧值

```
127.0.0.1:6379[1]> setbit char 2 1
(integer) 0
127.0.0.1:6379[1]> get char
"a"
```

> 如果offset过大,则会在中间填充0,offset最大能到2^32-1,可以推出最大的字符串为512M

`bitop operation destkey key1 [key2 ...]`

作用:对key1,key2,...keyN做运算,结果保存到destkey当中,运算可以是AND,OR,NOT,XOR

```
redis 127.0.0.1:6379> setbit lower 7 0
(integer) 0
redis 127.0.0.1:6379> setbit lower 2 1
(integer) 0
redis 127.0.0.1:6379> get lower
" "
redis 127.0.0.1:6379> set char Q
OK
redis 127.0.0.1:6379> get char
"Q"
redis 127.0.0.1:6379> bitop or char char lower
(integer) 1
redis 127.0.0.1:6379> get char
"q"
```

2.3 setbit 实际应用

场景:有1亿个用户,每个用户都要登陆/做任意操作,我们统计用户的操作,标记为活跃或者不活跃,每周或者每月按照活跃度;

假设设计数据库 ,我们需要一张用户信息表,记录用户的uid和logintime,虽然本表能帮助我们存储用户的登陆信息,但是由于用户量庞大,表内容会急剧膨胀,不利于查找和维护;

对于这种情况,我们可以使用位图法帮助我们记录数据;

1. 记录用户登陆

每天按照日期生成一个位图,用户登陆后,把user_id上的bit的值置为1,1亿的用户每天仅需要12M的数据存储;

```
Log1113:  '011001..............0'

......
log1114 :   '011001..............0'
Log1115 :  '0110000.............1'
```

2.4 list 链表结构

> list类似于php中的索引数组,我们可以通过方法操作链表中的数据.
> 它能完成什么?
> 流行的 Twitter 社交网络，使用 Redis 列表来存储用户最新的微博 (tweets)。

`lpush key value or rpush` rpush把值插入到链表的尾部

作用:把值插入到链接头部

`rpop key`

作用:返回并删除链表尾元素

`lrange key start stop`

作用:返回链表中的[start,stop]中的元素,其中左数从0开始,右数从-1开始;

`lrem key count value`

作用:
count > 0 : 从表头开始向表尾搜索，移除与 VALUE 相等的元素，数量为 COUNT 。
count < 0 : 从表尾开始向表头搜索，移除与 VALUE 相等的元素，数量为 COUNT 的绝对值。
count = 0 : 移除表中所有与 VALUE 相等的值。

`lindex key index`

作用:返回index索引上的值

`llen key`

作用:计算链表的元素个数;

`ltrim key`
作用 : LTRIM 命令类似于 LRANGE，但是不同于展示指定范围的元素，而是将其作为列表新值存储。所有范围外的元素都将被删除

2.5 集合 set 相关命令

集合的性质:唯一性,无序性,确定性;

集合适用于表达对象间关系。例如，我们可以很容易的实现标签。对这个问题的最简单建模，就是有一个为每个需要标记的对象的集合。集合中保存着与对象相关的标记的 ID。

因此我们在string和list的命令中所使用的range访问并不能在set中生效,因为set具有无序性的特点,我们无法通过下表或者范围访问部分元素

所以我们想要查看set中的元素,要么随机选取一个,要么全部选取;

`sadd key value1 value2`

作用: 往集合key中增加元素

`srem value1 value2`

作用: 删除集合中值为 value1 value2的元素,返回值为 忽略不存在的元素后,真正删除掉的元素的个数

`spop key`

作用: 返回并删除集合中key中1个随机元素

随机--体现了无序性

`srandmember key`

作用: 返回集合key中,随机的1个元素.

`smembers key`

作用: 返回集中中所有的元素

`scard key`

作用: 返回集合中元素的个数

`sismember key value`

作用: 判断value是否在key集合中,是返回1,否返回0

`smove source dest value`

作用:把source中的value删除,并添加到dest集合中

`sinter key1 key2 key3`

作用: 求出key1 key2 key3 三个集合中的交集,并返回

```
redis 127.0.0.1:6379> sadd s1 a b c
(integer) 3
redis 127.0.0.1:6379> sadd s2 b c d
(integer) 3
redis 127.0.0.1:6379> sadd s3 c d e
(integer) 3
redis 127.0.0.1:6379> sinter s1 s2 s3
```

```
1) “c"
redis 127.0.0.1:6379> sinter s3 s1 s2
1) "c"
```

`sinterstore dest key1 key2 key3`

作用: 求出key1 key2 key3 三个集合中的交集,并赋给dest

`suinon key1 key2.. Keyn`

sunion key1 key2 .. keyn

作用: 求出key1 key2 keyn的并集,并返回

`sdiff key1 key2 key3`

作用: 求出key1与key2 key3的差集

2.6 有序集合

集合本来是无序的,redis在存储集合中的元素时添加了权重,让集合变得有序

`zadd key score1 value1 score2 value2 ..`

作用:添加元素

```
redis 127.0.0.1:6379> zadd stu 18 lily 19 hmm 20 lilei 21 lilei
(integer) 3
```

`zcard key`

作用:返回元素个数

`zcount key min max`

作用:返回score在[min,max] 区间内元素的数量

`zrank key member`

作用:查询member的排名(按分数升续, 0名开始)

`zrevrank key memeber`

作用:查询 member的排名(按分数降续, 0名开始)

`zrange key start stop [WITHSCORES]`

作用:把集合排序后,返回名次[start,stop]的元素,默认是升续排列,Withscores 是把score也打印出来

`zrevrange key start stop`

作用:把集合降序排列,取名字[start,stop]之间的元素

`zrangebyscore key min max [withscores] limit offset N`

作用: 集合(升续)排序后,取score在[min,max]内的元素,并跳过 offset个, 取出N个

```
redis 127.0.0.1:6379> zadd stu 1 a 3 b 4 c 9 e 12 f 15 g
(integer) 6
redis 127.0.0.1:6379> zrangebyscore stu 3 12 limit 1 2 withscores
1) "c"
2) "4"
3) "e"
4) "9"
```

`zrem key value1 value2 ..`

作用: 删除集合中的元素

`zremrangebyscore key min max`

作用: 按照socre来删除元素,删除score在[min,max]之间的

```
redis 127.0.0.1:6379> zremrangebyscore stu 4 10
(integer) 2
redis 127.0.0.1:6379> zrange stu 0 -1
1) "f"
```

`zremrangebyrank key start end`

作用: 按排名删除元素,删除名次在[start,end]之间的

```
redis 127.0.0.1:6379> zremrangebyrank stu 0 1
(integer) 2
redis 127.0.0.1:6379> zrange stu 0 -1
1) "c"
2) "e"
3) "f"
4) "g"
```

2.7 Hash 哈希数据类型相关命令

`hset key field value`

作用: 把key中 filed域的值设为value

> 注:如果没有field域,直接添加,如果有,则覆盖原field域的值

`hmset key field1 value1 [field2 value2 field3 value3 ......fieldn valuen]`

作用: 设置field1->N 个域, 对应的值是value1->N

> 对应PHP理解为 $key = array(file1=>value1, field2=>value2 ....fieldN=>valueN)

`hgetall key`

作用:返回key中,所有域与其值

`hget key field`

作用: 返回key中field域的值

`hmget key field1 field2 fieldN`

作用: 返回key中field1 field2 fieldN域的值

`hlen key`

作用: 返回key中元素的数量

`hkeys key`

作用: 返回key中所有的field

`hvals key`

作用: 返回key中所有的value

`hexists key field`

作用: 判断key中有没有field域

`hincrby key field value`

作用: 是把key中的field域的值增长整型值value

`hincrby float key field value`

作用: 是把key中的field域的值增长浮点值value

`hdel key field`

作用: 删除key中 field域

2.8 flush

flushdb: 删除当前数据库
flushall : 删除所有库中的数据

2.9 scan

我们可以keys * 列出所有数据, 但是如果在生产环境中,redis存储几十万个key,即使用关键词匹配后,也有上万个, 调用keys将会对内存,IO都产生很大的冲击,甚至宕机.
redis作者本人有调用keys导致宕机的经历, 2.8以后增加了scan命令.
scan是以游标的形式来获取相应的key,而非一次性取出.

```
127.0.0.1:6379> mset k1 v1 k2 v2 k3 v3 k4 v4 k5 v5
127.0.0.1:6379> mset k6 v6 k7 v7 k8 v8 k9 v9 k10 v10 k11 v11 k12 v12
127.0.0.1:6379> scan 0 match k*
1) "13"   # 游标已经偏移至13位置,并返回10个key
2)  1) "k12"
    2) "k7"
    3) "k4"
    4) "k10"
    5) "k3"
    6) "k2"
    7) "k1"
    8) "k9"
    9) "k11"
   10) "k6"
127.0.0.1:6379> scan 13 #继续从游标13处扫描,又得2个key,游标返回0,说明扫描完毕
1) "0"
2) 1) "k5"
      2) "k8"
```

第三章 Redis中的事务

Redis支持简单的事物

Redis和mysql事务的对比

操作	mysql	redis
开启	start transaction	multi
语句	普通sql	普通命令
失败	rollback 回滚	discard
成功	commit	exec

思考:

我正在买票,成功之后 Ticket -1 , money -100

而票只有1张, 如果在我multi之后,和exec之前, 票被别人买了---即ticket变成0了.

我该如何观察这种情景,并不再提交

悲观的想法:

世界充满危险,肯定有人和我抢, 给 ticket上锁, 只有我能操作. [悲观锁]

> mysql等数据库可以采用悲观锁在事务开启的时候就保护好票数

乐观的想法:

没有那么多人和我抢,因此,我只需要注意--有没有人更改ticket的值就可以了 [乐观锁]

Redis的事务中,启用的是乐观锁,只负责监测key没有被改动.

被改动了就无法成功执行事务

具体采用watch命令

```
redis 127.0.0.1:6379> watch ticket
OK
redis 127.0.0.1:6379> multi
OK
redis 127.0.0.1:6379> decr ticket
QUEUED
redis 127.0.0.1:6379> decrby money 100
QUEUED
redis 127.0.0.1:6379> exec
(nil)   // 返回nil,说明监视的ticket已经改变了,事务就取消了.
redis 127.0.0.1:6379> get ticket
"0"
redis 127.0.0.1:6379> get money
"200"
```

unwatch 取消监听

第四章 安装PHP的redis扩展

① 下载并解压 redis 扩展

我们这里下载最新的stable版

> *wget http://pecl.php.net/get/redis-2.2.8.tgz*

解压

> *tar zxf redis-2.2.8.tgz*

② 生成configure文件

> *cd redis-2.2.8*
> */usr/local/php/bin/phpize*

③ 配置编译安装参数

> *./configure --with-php-config=/usr/local/php/bin/php-config*

④ 编译并安装

> *make && make install*

⑤ 为 php.ini 配置文件添加 redis 扩展

如果你忘记了你的php.ini的配置文件路径

> */usr/local/php/bin/php -i | grep php.ini*
>
> extension=/usr/local/php/lib/php/extensions/no-debug-non-zts-20131226/redis.so

最后如果你不确定你的redis是否添加进入了扩展,你可以使用php命令进行查看.

> */usr/local/php/bin/php -m | grep redis*

第五章 设计标签系统

接下来我们要设计一个书签系统,用来存放我们每本书籍的书签.目的是为了方便通过书签筛选书籍.

建库

> *create database itbool charset utf8;*

①创建书籍表

```
create table books (
    bookid int unsigned not null auto_increment primary key comment '书籍id',
    title char(25) not null default '' comment '书籍名称'
) engine=myisam default charset=utf8 comment='书籍表';
```

书籍数据插入:

```
insert into books values
(5,'php圣经'),
(6,'ruby实战'),
(7,'mysql运维'),
(8,'ruby服务端编程');
```

②创建标签表

```
create table tags (
    tagid int unsigned not null auto_increment primary key comment '标签id',
    bookid int unsigned not null comment '书籍id',
    name char(25) not null default '' comment '标签名称'
) engine=myisam default charset=utf8 comment='标签表';
```

标签数据插入:

```
insert into tags values
(10,5,'php'),
(11,5,'web'),
(12,6,'web'),
(13,6,'ruby'),
(14,7,'database'),
(15,8,'ruby'),
(16,8,'server');
```

③项目需求

查询出即带有web标签又带有php标签的书

> *select* \ from tags inner join tags as tmp on tags.bookid = tmp.bookid where tags.name = 'php' and tmp.name = 'web';*

如果你想直接看到书名,则需要再连接一张books表

> *select* \ from tags inner join tags as tmp on tags.bookid = tmp.bookid
> inner join books on books.bookid = tags.bookid
> where tags.name = 'php' and tmp.name = 'web';*

5.1 redis实现标签功能

同时含有多个标签,或者不含某个标签.我们将其放在redis的集合模型中,利用集合所有的特性运算,就能很快得到解决.

①设计书籍key-value

开启redis服务器

> */MyComputer/Program-Files/redis/bin/redis-server /MyComputer/Program-Files/redis/redis.conf*

用redis客户端连接

> */MyComputer/Program-Files/redis/bin/redis-cli -h 127.0.0.1 -p 6379*

```
set book:5:title 'PHP圣经'
set book:6:title 'ruby实战'
set book:7:title 'mysql运难'
set book:8:title 'ruby服务端编程'
```

标签数据存放在集合中

```
sadd tag:php 5
sadd tag:web 5 6
sadd tag:database 7
sadd tag:ruby 6 8
sadd tag:server 8
```

②项目需求

找出既有php标签又有web标签的书籍(求集合的交集)

> *sinter tag:php tag:web*

找出含有php标签或者web标签的书籍(求集合的并集)

> *sunion tag:php tag:web*

找出含有ruby标签但是不含web标签的书籍(求差集)

> *sdiff tag:ruby tag:web*