像keys或者smembers命令，需要遍历数据集合中的所有元素。在一个大的数据库中使用，可能会阻塞服务器较长的一段时间，造成性能问题，因此不适用与生产环境。

在Redis2.8.0中引入了scan类命令，可用于迭代整个数据库或者是哈希键、集合键和有序集合键这样的数据集合。

scan类命令支持增量式迭代，它们每次执行都只会返回少量元素，所以这些命令可以用于生产环境。

scan类命令使用游标迭代数据集合，命令每次被调用之后，都会向用户返回一个新的游标，用户在下次迭代时，使用这个新游标作为scan类命令的游标参数，以此来延续之前的迭代过程。

第一次迭代时，游标参数置为0，经过若干次迭代之后，如果返回的游标为0，表示完成了一次完整迭代。

scan类命令包括：scan命令用于迭代当前的数据库，返回其中的键；sscan命令用于迭代集合键，返回集合中的元素；hscan用于迭代哈希键，返回其中的字段以及字段值；zscan用于迭代有序集合键，返回其中的元素及其分数。

**一：用法**

scan类命令的用法如下：

SCAN cursor **[**MATCH pattern**]** **[**COUNT count**]**

HSCAN key cursor **[**MATCH pattern**]** **[**COUNT count**]**

SSCAN key cursor **[**MATCH pattern**]** **[**COUNT count**]**

ZSCAN key cursor **[**MATCH pattern**]** **[**COUNT count**]**

COUNT 选项指定每次迭代返回元素的最大值，默认值为10。

注意，对于一个使用哈希表作为底层实现的大数据集来说，scan类命令每次最多会返回count个元素；如果数据集合不是以哈希表作为底层实现的话，则scan类命令无视count选项，直接返回数据集合中的所有元素。

并非每次迭代都要使用相同的COUNT值。

MATCH选项，可以提供一个glob风格的模式参数，让scan类命令只返回和给定模式相匹配的元素。

注意，对元素的模式匹配，是在命令从数据集中取出元素之后，向客户端返回元素之前的进行的，所以如果被迭代的数据集中只有少量元素和模式相匹配，那么迭代命令可能不返回任何元素。

例子如下：

127.0.0.1**:**6379**>** scan 0

1**)** "7"

2**)** 1**)** "msg3"

2**)** "msg"

3**)** "msg5"

4**)** "msg6"

5**)** "msg4"

6**)** "msg2"

7**)** "msg8"

8**)** "msg11"

9**)** "msg7"

10**)** "msg10"

127.0.0.1**:**6379**>** scan 7

1**)** "0"

2**)** 1**)** "msg9"

2**)** "msg1"

scan命令只能对被返回的元素提供有限的保证：

a：从完整遍历开始直到完整遍历结束期间，一直存在于数据集内的所有元素都会被完整遍历返回；

b：同一个元素可能会被返回多次；如果一个元素是在迭代过程中被添加到数据集的，又或者是在迭代过程中从数据集中被删除的，那么这个元素可能会被返回， 也可能不会， 这是未定义的。

c：不保证每次执行都返回某个给定数量的元素。甚至可能会返回零个元素，但只要命令返回的游标不是 0，应用程序就不应该将迭代视作结束。

**二：实现**

scan命令是通过src/db.c中的scanCommand函数实现的，该函数的代码如下：

void scanCommand**(**redisClient **\***c**)** **{**

unsigned long cursor**;**

**if** **(**parseScanCursorOrReply**(**c**,**c**->**argv**[**1**],&**cursor**)** **==** REDIS\_ERR**)** **return;**

scanGenericCommand**(**c**,NULL,**cursor**);**

**}**

该函数首先通过parseScanCursorOrReply，解析客户端命令参数中的游标参数，赋值到cursor中。如果解析失败，则直接返回；解析成功后，调用scanGenericCommand进行处理。

下面是解析游标参数的函数parseScanCursorOrReply的实现：

int parseScanCursorOrReply**(**redisClient **\***c**,** robj **\***o**,** unsigned long **\***cursor**)** **{**

char **\***eptr**;**

*/\* Use strtoul() because we need an \*unsigned\* long, so*

*\* getLongLongFromObject() does not cover the whole cursor space. \*/*

errno **=** 0**;**

**\***cursor **=** strtoul**(**o**->**ptr**,** **&**eptr**,** 10**);**

**if** **(**isspace**(((**char**\*)**o**->**ptr**)[**0**])** **||** eptr**[**0**]** **!=** '\0' **||** errno **==** ERANGE**)**

**{**

addReplyError**(**c**,** "invalid cursor"**);**

**return** REDIS\_ERR**;**

**}**

**return** REDIS\_OK**;**

**}**

该函数主要是调用strtoul对存放在字符串对象o中的字符串进行解析，如果该字符串是有效的数字表示，则将其转换成整数后放到参数cursor中，并返回REDIS\_OK，否则反馈给客户端"invalid cursor"，并返回REDIS\_ERR。

下面是主要的实现函数scanGenericCommand的代码，该函数不止是scan命令的主要实现函数，还是sscan、hscan和zscan命令的主要实现函数：

void scanCallback**(**void **\***privdata**,** const dictEntry **\***de**)** **{**

void **\*\***pd **=** **(**void**\*\*)** privdata**;**

list **\***keys **=** pd**[**0**];**

robj **\***o **=** pd**[**1**];**

robj **\***key**,** **\***val **=** **NULL;**

**if** **(**o **==** **NULL)** **{**

sds sdskey **=** dictGetKey**(**de**);**

key **=** createStringObject**(**sdskey**,** sdslen**(**sdskey**));**

**}** **else** **if** **(**o**->**type **==** REDIS\_SET**)** **{**

key **=** dictGetKey**(**de**);**

incrRefCount**(**key**);**

**}** **else** **if** **(**o**->**type **==** REDIS\_HASH**)** **{**

key **=** dictGetKey**(**de**);**

incrRefCount**(**key**);**

val **=** dictGetVal**(**de**);**

incrRefCount**(**val**);**

**}** **else** **if** **(**o**->**type **==** REDIS\_ZSET**)** **{**

key **=** dictGetKey**(**de**);**

incrRefCount**(**key**);**

val **=** createStringObjectFromLongDouble**(\*(**double**\*)**dictGetVal**(**de**),**0**);**

**}** **else** **{**

redisPanic**(**"Type not handled in SCAN callback."**);**

**}**

listAddNodeTail**(**keys**,** key**);**

**if** **(**val**)** listAddNodeTail**(**keys**,** val**);**

**}**

void scanGenericCommand**(**redisClient **\***c**,** robj **\***o**,** unsigned long cursor**)** **{**

int i**,** j**;**

list **\***keys **=** listCreate**();**

listNode **\***node**,** **\***nextnode**;**

long count **=** 10**;**

sds pat**;**

int patlen**,** use\_pattern **=** 0**;**

dict **\***ht**;**

*/\* Object must be NULL (to iterate keys names), or the type of the object*

*\* must be Set, Sorted Set, or Hash. \*/*

redisAssert**(**o **==** **NULL** **||** o**->**type **==** REDIS\_SET **||** o**->**type **==** REDIS\_HASH **||**

o**->**type **==** REDIS\_ZSET**);**

*/\* Set i to the first option argument. The previous one is the cursor. \*/*

i **=** **(**o **==** **NULL)** **?** 2 **:** 3**;** */\* Skip the key argument if needed. \*/*

*/\* Step 1: Parse options. \*/*

**while** **(**i **<** c**->**argc**)** **{**

j **=** c**->**argc **-** i**;**

**if** **(!**strcasecmp**(**c**->**argv**[**i**]->**ptr**,** "count"**)** **&&** j **>=** 2**)** **{**

**if** **(**getLongFromObjectOrReply**(**c**,** c**->**argv**[**i**+**1**],** **&**count**,** **NULL)**

**!=** REDIS\_OK**)**

**{**

**goto** cleanup**;**

**}**

**if** **(**count **<** 1**)** **{**

addReply**(**c**,**shared**.**syntaxerr**);**

**goto** cleanup**;**

**}**

i **+=** 2**;**

**}** **else** **if** **(!**strcasecmp**(**c**->**argv**[**i**]->**ptr**,** "match"**)** **&&** j **>=** 2**)** **{**

pat **=** c**->**argv**[**i**+**1**]->**ptr**;**

patlen **=** sdslen**(**pat**);**

*/\* The pattern always matches if it is exactly "\*", so it is*

*\* equivalent to disabling it. \*/*

use\_pattern **=** **!(**pat**[**0**]** **==** '\*' **&&** patlen **==** 1**);**

i **+=** 2**;**

**}** **else** **{**

addReply**(**c**,**shared**.**syntaxerr**);**

**goto** cleanup**;**

**}**

**}**

*/\* Step 2: Iterate the collection.*

*\**

*\* Note that if the object is encoded with a ziplist, intset, or any other*

*\* representation that is not a hash table, we are sure that it is also*

*\* composed of a small number of elements. So to avoid taking state we*

*\* just return everything inside the object in a single call, setting the*

*\* cursor to zero to signal the end of the iteration. \*/*

*/\* Handle the case of a hash table. \*/*

ht **=** **NULL;**

**if** **(**o **==** **NULL)** **{**

ht **=** c**->**db**->**dict**;**

**}** **else** **if** **(**o**->**type **==** REDIS\_SET **&&** o**->**encoding **==** REDIS\_ENCODING\_HT**)** **{**

ht **=** o**->**ptr**;**

**}** **else** **if** **(**o**->**type **==** REDIS\_HASH **&&** o**->**encoding **==** REDIS\_ENCODING\_HT**)** **{**

ht **=** o**->**ptr**;**

count **\*=** 2**;** */\* We return key / value for this type. \*/*

**}** **else** **if** **(**o**->**type **==** REDIS\_ZSET **&&** o**->**encoding **==** REDIS\_ENCODING\_SKIPLIST**)** **{**

zset **\***zs **=** o**->**ptr**;**

ht **=** zs**->**dict**;**

count **\*=** 2**;** */\* We return key / value for this type. \*/*

**}**

**if** **(**ht**)** **{**

void **\***privdata**[**2**];**

*/\* We set the max number of iterations to ten times the specified*

*\* COUNT, so if the hash table is in a pathological state (very*

*\* sparsely populated) we avoid to block too much time at the cost*

*\* of returning no or very few elements. \*/*

long maxiterations **=** count**\***10**;**

*/\* We pass two pointers to the callback: the list to which it will*

*\* add new elements, and the object containing the dictionary so that*

*\* it is possible to fetch more data in a type-dependent way. \*/*

privdata**[**0**]** **=** keys**;**

privdata**[**1**]** **=** o**;**

**do** **{**

cursor **=** dictScan**(**ht**,** cursor**,** scanCallback**,** privdata**);**

**}** **while** **(**cursor **&&**

maxiterations**--** **&&**

listLength**(**keys**)** **<** **(**unsigned long**)**count**);**

**}** **else** **if** **(**o**->**type **==** REDIS\_SET**)** **{**

int pos **=** 0**;**

int64\_t ll**;**

**while(**intsetGet**(**o**->**ptr**,**pos**++,&**ll**))**

listAddNodeTail**(**keys**,**createStringObjectFromLongLong**(**ll**));**

cursor **=** 0**;**

**}** **else** **if** **(**o**->**type **==** REDIS\_HASH **||** o**->**type **==** REDIS\_ZSET**)** **{**

unsigned char **\***p **=** ziplistIndex**(**o**->**ptr**,**0**);**

unsigned char **\***vstr**;**

unsigned int vlen**;**

long long vll**;**

**while(**p**)** **{**

ziplistGet**(**p**,&**vstr**,&**vlen**,&**vll**);**

listAddNodeTail**(**keys**,**

**(**vstr **!=** **NULL)** **?** createStringObject**((**char**\*)**vstr**,**vlen**)** **:**

createStringObjectFromLongLong**(**vll**));**

p **=** ziplistNext**(**o**->**ptr**,**p**);**

**}**

cursor **=** 0**;**

**}** **else** **{**

redisPanic**(**"Not handled encoding in SCAN."**);**

**}**

*/\* Step 3: Filter elements. \*/*

node **=** listFirst**(**keys**);**

**while** **(**node**)** **{**

robj **\***kobj **=** listNodeValue**(**node**);**

nextnode **=** listNextNode**(**node**);**

int filter **=** 0**;**

*/\* Filter element if it does not match the pattern. \*/*

**if** **(!**filter **&&** use\_pattern**)** **{**

**if** **(**sdsEncodedObject**(**kobj**))** **{**

**if** **(!**stringmatchlen**(**pat**,** patlen**,** kobj**->**ptr**,** sdslen**(**kobj**->**ptr**),** 0**))**

filter **=** 1**;**

**}** **else** **{**

char buf**[**REDIS\_LONGSTR\_SIZE**];**

int len**;**

redisAssert**(**kobj**->**encoding **==** REDIS\_ENCODING\_INT**);**

len **=** ll2string**(**buf**,sizeof(**buf**),(**long**)**kobj**->**ptr**);**

**if** **(!**stringmatchlen**(**pat**,** patlen**,** buf**,** len**,** 0**))** filter **=** 1**;**

**}**

**}**

*/\* Filter element if it is an expired key. \*/*

**if** **(!**filter **&&** o **==** **NULL** **&&** expireIfNeeded**(**c**->**db**,** kobj**))** filter **=** 1**;**

*/\* Remove the element and its associted value if needed. \*/*

**if** **(**filter**)** **{**

decrRefCount**(**kobj**);**

listDelNode**(**keys**,** node**);**

**}**

*/\* If this is a hash or a sorted set, we have a flat list of*

*\* key-value elements, so if this element was filtered, remove the*

*\* value, or skip it if it was not filtered: we only match keys. \*/*

**if** **(**o **&&** **(**o**->**type **==** REDIS\_ZSET **||** o**->**type **==** REDIS\_HASH**))** **{**

node **=** nextnode**;**

nextnode **=** listNextNode**(**node**);**

**if** **(**filter**)** **{**

kobj **=** listNodeValue**(**node**);**

decrRefCount**(**kobj**);**

listDelNode**(**keys**,** node**);**

**}**

**}**

node **=** nextnode**;**

**}**

*/\* Step 4: Reply to the client. \*/*

addReplyMultiBulkLen**(**c**,** 2**);**

addReplyBulkLongLong**(**c**,**cursor**);**

addReplyMultiBulkLen**(**c**,** listLength**(**keys**));**

**while** **((**node **=** listFirst**(**keys**))** **!=** **NULL)** **{**

robj **\***kobj **=** listNodeValue**(**node**);**

addReplyBulk**(**c**,** kobj**);**

decrRefCount**(**kobj**);**

listDelNode**(**keys**,** node**);**

**}**

cleanup**:**

listSetFreeMethod**(**keys**,**decrRefCountVoid**);**

listRelease**(**keys**);**

**}**

在函数scanGenericCommand中，参数o表示要迭代的数据集合，首先对o进行校验，如果o不是NULL，或者o的类型不是REDIS\_SET、REDIS\_HASH、REDIS\_ZSET其中之一的话，则直接报错退出；

然后，将i置为命令中第一个可选参数的索引；对于scan命令，i为2，其他scan命令，i为3；接下来开始解析可选参数count和match的值，如果没有可选参数count，则将其设置为10；如果出现解析错误的情况，则向客户端反馈syntaxerr，并在清理工作之后返回；

接下来开始遍历数据。如果要遍历的对象，不是通过哈希表实现的，比如哈希对象和有序集合可以通过压缩表实现，集合对象可以通过intset实现，这种情况下，则无视count值，直接返回对象中的所有元素；另外，对于使用字典实现哈希对象，以及跳跃表实现的有序集合来说，需要返回字典中的键值对，因此需要将count\*=2；

如果对象是通过字典实现的，则通过dictScan对字典进行遍历，针对字典对象的每一个键值对，调用scanCallback，该函数针对不同的对象类型，从字典中取出相应的值，放入到链表keys中。这种情况下，最多调用count次dictScan；另外，为了避免字典处于罕见的错误状态（字典在rehash时的某种状态，这种状态下dictScan可能不返回或者返回极少的元素），最多调用10\*count次dictScan；

如果对象不是通过字典实现，并且是集合对象的话，则集合对象是通过intset实现的，此时调用intsetGet，将集合中的所有元素都遍历一遍，插入到链表keys中，并置cursor为0；

如果对象不是通过字典实现，并且是哈希对象或者有序集合对象的话，则它们是通过压缩表实现的，此时调用ziplistGet，将压缩列表中的所有元素都遍历一遍，插入到链表keys中，并置cursor为0；

接下来，开始对keys中的数据进行过滤，针对keys中的每一个元素，如果命令参数中指定了过滤选项的话，如果该元素是字符串的话，则直接使用stringmatchlen进行匹配，如果元素是以整数进行存储的话，则先将其转换成字符串，再用stringmatchlen进行匹配。匹配不成功，则置filter为1，表示需要舍弃该元素；

如果是遍历数据库的话，还需要用expireIfNeeded判断该键是否因超时需要删除，如果是，则也置filter为1；

如果filter为1，则将该元素从keys中删除；

如果对象为哈希对象或者有序集合的话，则keys中不但保存了其中的key，还保存了对应的键，所以下一次过滤操作需要隔一个元素开始，而且，如果当前元素被舍弃掉的话，还需要删除该元素的后继元素（键元素对应的值）；

过滤完成后，将cursor和keys中的所有元素都反馈给客户端；最后做清理工作。

scan的详细解释可以参考：

http://redis.io/commands/scan，它的中文翻译见：http://doc.redisfans.com/key/scan.html