Redis源代码分析(上)

文 / 阮若夷

Redis是一个开源的Key-Value的内存数据库，以支持丰富的数据结构而著称，支持主从复制、持久化等高可用特性，可以和程序无缝地结合，本文从分析Redis的源代码入手，帮助大家了解这一款数据库的工作机理。

Redis是一个开源的Key-Value的内存数据库，与它类似的Key-Value数据库有memcached、Tokyo Cabinet等。Redis以支持丰富的数据结构而著称，还支持主从复制、持久化等高可用特性，可以和程序无缝地结合，而无须像使用关系型数据库一样需要ORM转换成关系型，或者像memcached一样只能使用简单的Key-Value。

memcached使用libevent这个已经不那么轻量级的网络事件库，而Redis本身不依赖任何第三方的函数库，无论是网络事件、散列表，数据结构都是自己实现的，全部代码只有2万行，算是一个中小型的项目，代码清晰，阅读起来非常流畅，甚至都无须Debug调试来辅助理解。

本文将分析Redis源代码，版本为2.2.2。下载源代码解压redis.tar.gz包后，进入redis目录，从src/redis.c的主函数开始我们的代码旅行。推荐读者使用cscope，这样可以很方便从函数之间跳转。

redis-server的启动过程

先从redis-server启动说起，自main函数开始遍历各个关键函数，了解一下Redis的主框架（如图1所示）。

首先initServerConfig函数会设置一些默认的参数，比如监听端口为6379，默认的DB个数为16等。PopulateCommandTable会把命令和函数数组转化成hash table结构，这个后文会详细描述。如果启动参数里有redis.conf，LoadServerConfig还会读入redis.conf里的参数，覆盖默认值。

initServer会给redisServer这个数据结构做初始化，申请各自成员的空间，其中有些是list结构，有些是dict结构。此后会添加一个时间事件，其函数为serverCron，并且每100ms执行一次，后文会详述这个函数的作用。接下来是启动监听，注册一个监听的文件事件，把accept行为注册到只读的监听文件描述符上。如果有激活aof功能，还会打开aof文件。接着会判断数据目录是否存在镜像文件或者aof文件，如果存在，redis会将数据载入到内存中。最后进入主循环。

主循环主要处理刚才注册的时间事件和文件事件。如何保证时间事件每100ms执行一次，又能即时处理网络交互的文件事件呢?

Redis处理得比较巧妙。先执行aeSearchNearestTimer，确定距离下次时间事件执行还有多少时间。假设第一次执行直到下次时间事件还有100ms，先执行文件事件，epool_wait的超时时间就设置为100ms，如果10ms后，有网络交互后经过一系列处理后消耗20ms，该次循环结束。aeSearchNearestTimer会再次计算距离下次时间事件的间隔为100-10-20=70ms，于是epoll_wait的超时时间为70ms，70ms之内如果没有处理文件事件，则执行时间事件。这样既保证了即时处理文件事件，又能在文件事件处理完毕后，按时处理时间事件。

时间事件serverCron会处理很多函数，例如定时打出日志展现Redis目前的状况，查看是否

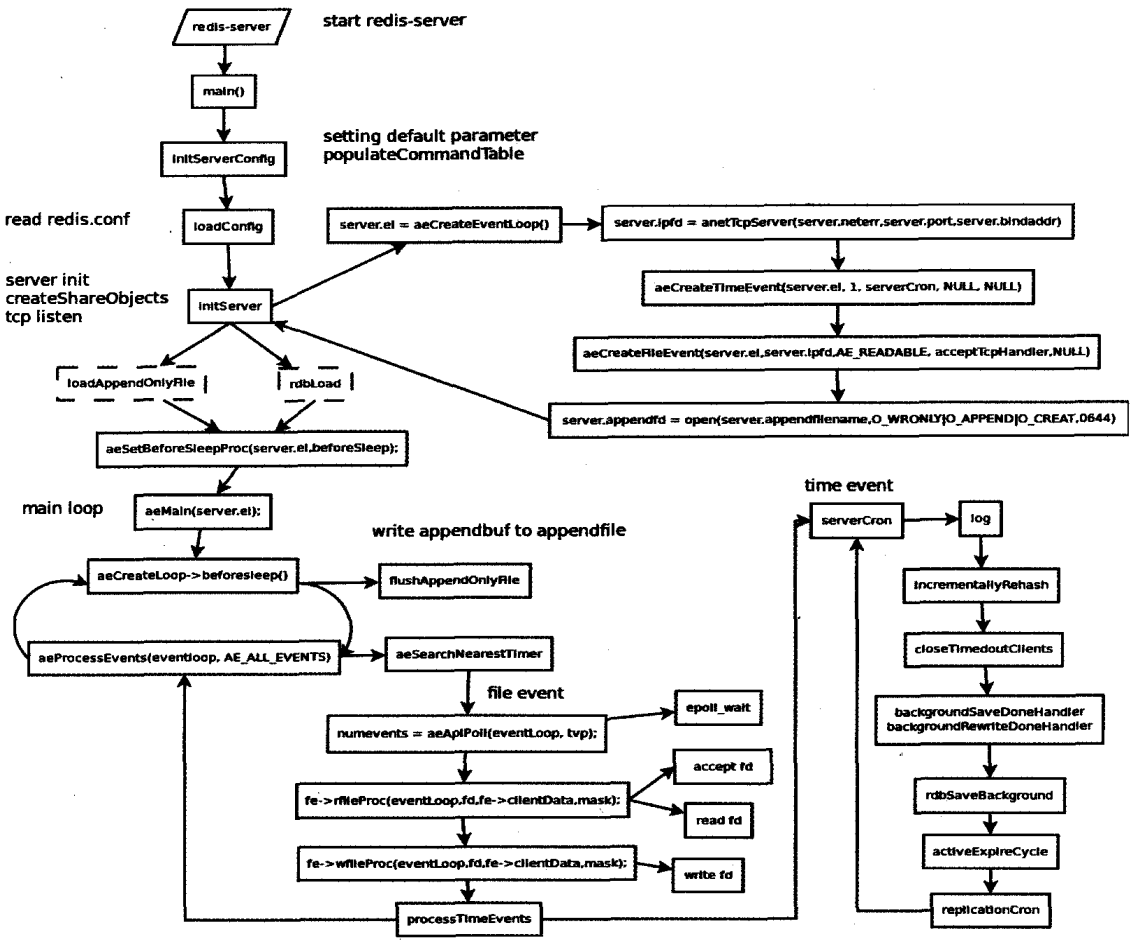

图1 redis启动示意图

需要rehash来迁移keys到新的bucket，这个后文会详细讲。

hash table实现

Key-Value数据库的KV查询的实现有很多种，比如功能全面的btree，而Redis的作者选择了简单的hash来实现，使用hash就意味着无法使用范围查询等功能，但择以更好的hash函数可达到更快的速度，而且代码实现更简单。

在Redis里hash无处不在，全局的Key-Value查询、内部的hash数据结构、命令与函数指针的关系都是使用hash。Hash的实现在src/dict.c、src/dict.h里。

dict为hash table的主结构体（如图2所示），dictht是为rehash而存在的中间数据结构，bucket就是hash算法里的桶，而dictEntry就为每个key-value结构体。dictht ht[2]指向2个dictht。存在2个ht的目的是为了在rehash时，可以平滑迁移bucket里的数据，而不像client的dict要把老的hash table里的一次性全部数据迁移到新的hash table，这在造成一个密集型的操作，在业务高峰期不可取。

每次Key-Value查询过程就是把要查询的key经过hash函数执行后的值与dictht->sizemask求位与，这样就获得一个大于等于0小于等于sizemask的值，定位到了bucket数组的位置。bucket数组的元素是dictEntry的指针。而dictEntry包含next指针，发生hash conflict时，直接以链表的形式加到链表的头部，所以查询是一个O(N)的操作，需要遍历dictEntry链表，而插入只插入到链表的头部，只是一个O（1）的时间复杂度。dictht->used表示这个hash table里已经插入的key的个数，也就是dictEntry的个

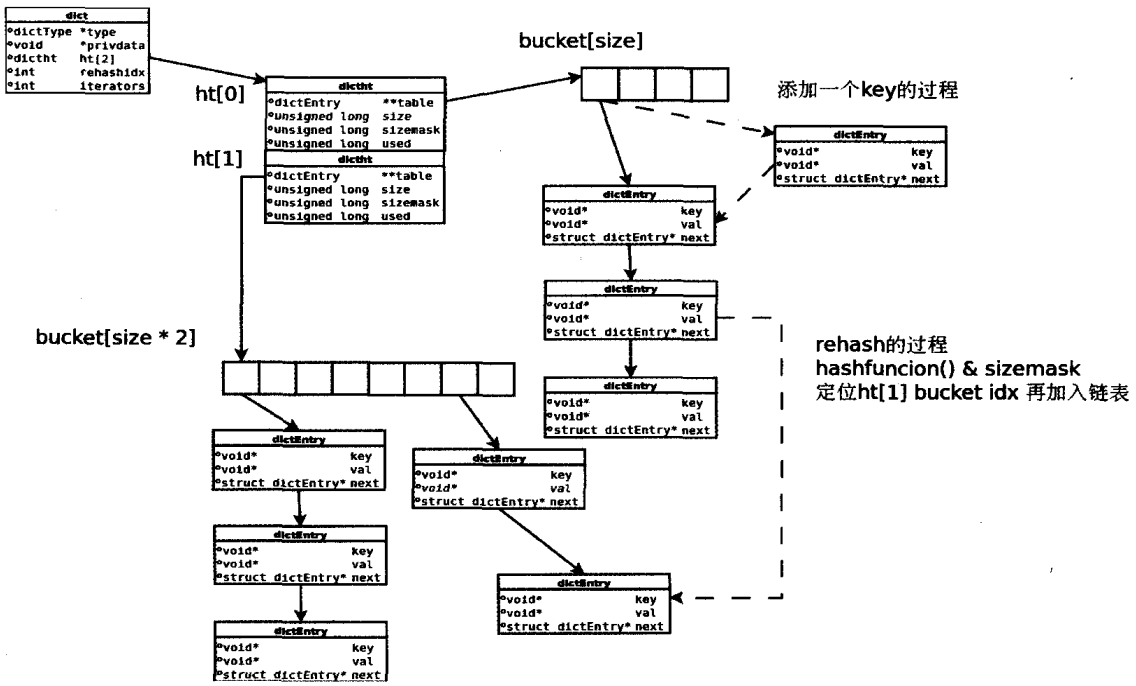

图2 redis-dict示意图

数，每次dictAdd成功会+1，dictDel成功会-1。

随着key不断添加，如果保持bucket数组大小不变，则每个bucket元素的单链表越来越长，查找、删除效率越来越低。

当dict->used/dict->size >= dict_force_resize_ratio（默认是5）时，就认为链表较长了，于是就有了expand和rehash，创建一个新的hash table（ht[1]），expand ht[1]的bucket数组长度为ht[0]上的两倍，rehash会把ht[0]上所有的key移动到ht[1]上。随着bucket数量增多，每个dictEntry链表的长度就缩短了。hash查找时，O（1）不会因为bucket数组大小改变而变化，而遍历链表从O（N)变为O（N/2）的时间复杂度。

rehash并不是一次性迁移所有的key，而是随着dictAdd、dictFind函数的执行过程，调度_dictRehashStep函数，一次一个将bucket下的key从ht[0]迁移到ht[1]。dict->rehashidx决定哪个bucket需要被迁移。当前bucket下的key都被迁移后，dict->rehashidx++，然后迁移下一个bucket，直到所有的bucket下的key被迁走。

除了dict_add、dict_find出发rehash，另外redis运行过程中，会调用dictRehash-Milliseconds函数，一次rehash 100个bucket，直到消耗了1秒才结束rehash，这样即使没有发生查询行为也会进行rehash的迁移。

rehash运行的具体过程如下：遍历dict->rehashidx对应的bucket下dictEntry链表的每个key，对key进行hash函数运算后与ht[1]->sizemask求位与，确定ht[1]的新bucket位置，然后加入到dictEntry链表里，然后ht[0].used--，ht[1].used++。当ht[0].used=0，释放ht[0]的table，再赋值ht[0]= ht[1]。

在rehash的过程中，如果有新的key加入，直接加到ht[1]。如果key的查找，会先查ht[0]再查询ht[1]。如果key的删除，在ht[0]找到后则删除返回，否则继续到ht[1]里寻找。在rehash的过程中，不会再检测是否需要expand。由于ht[1]是ht[0]size的两倍，每次dictAdd时都会迁移一个bucket，所以不会出现ht[1]溢出了，而ht[0]还有数据的状况。

使用hash table的地方

举个最常见使用的例子，RedisServer存储全局key-value的RedisDB。

每个Key-Value的数据都会存储在RedisDB这个结构里，而RedisDB就是一个hash table。从图3上我们可以看出key为“hello”，value为

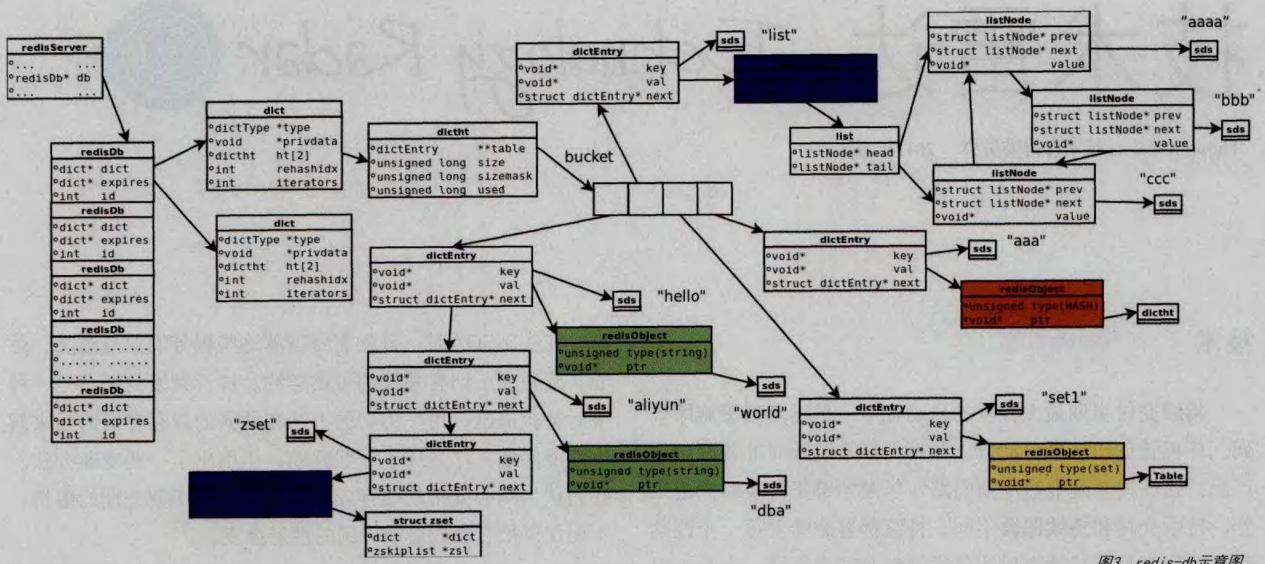

图3 redis-db示意图

“world”的存储格式。key为“list”，value为一个字符串链表（["aaa","bbb","ccc"]）的存储型式，另外的set、hash、zset就不一一表示了。

另外常见的命令（例如get、set等）会调用的函数指针，这个数据结构也是以hash table的形式存储的。每次客户端输入“set aa bb”等数据时，解析得到字符串“set”后，会根据“set”作为一个key，查找到value；一个函数指针（setCommand），再把“aa”、“bb”作为参数传给这个函数。这个hash table存储在redisServer->command里，每次redis-server启动时，会对readonlyCommandTable这个数组进行加工（populateCommandTable）转化成redisServer->command这个hash table方便查询，而非遍历readonlyCommandTable查找要执行的函数。

一个redis命令的历险

我们通过客户端发出一个普通的“set a 1”的命令来遍历Redis，来遍历Redis几个关键函数，并熟悉协议处理的过程。

你可以通过telnet或者redis_cli、各种程序语言bind的lib库发送请求给RedisServer。前者以一种裸协议的请求发送到服务端，而后两者会对请求进行协议组装，帮助更好的解析（常见的固定位置放置请求长度和协议类型）。我们来看看请求和回复的协议格式。

请求协议：

```
*参数的个数\r\n
$第一个参数的长度\r\n
第一个参数\r\n
...
$第N个参数的长度\r\n
第N个参数\r\n
```

例如请求“get a”，经过协议组装后的请求为：

```
2\r\n
$3\r\n
get\r\n
$1\r\n
a\r\n
```

以上过程可以帮助我们了解到遍历关键函数的Redis命令的处理过程。（未完待续）

阮若夷

目前就职于阿里云计算运维部，是阿里云关系型数据库云服务（RDS）和鹰眼监控的开发者。

责任编辑：高松（gaosong@csdn.net）