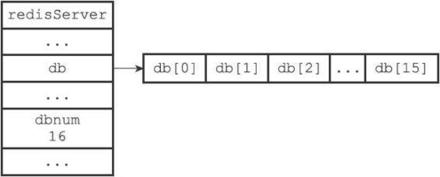
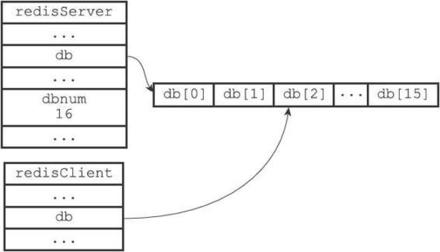
数据库详解

# 数据库

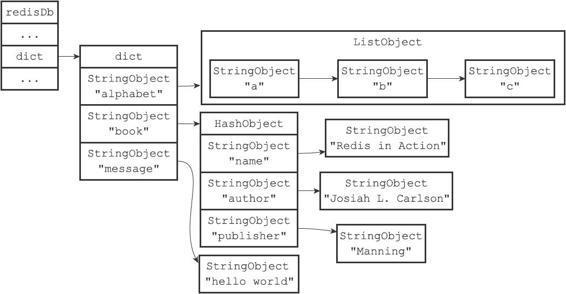
Redis中的所有数据库都保存在redisServer结构中的数组中，数据库的数量有dbnum属性保存。默认情况下Redis数据库会创建16个数据库。



Redis客户端 redisClient结构都会有一个db属性指向客户端可以使用的数据库，默认情况下是0号数据库，你也可以使用SELECT命令，切换到其他数据库上。



redisDb结构中的dict属性，保存着整个数据库的键空间。



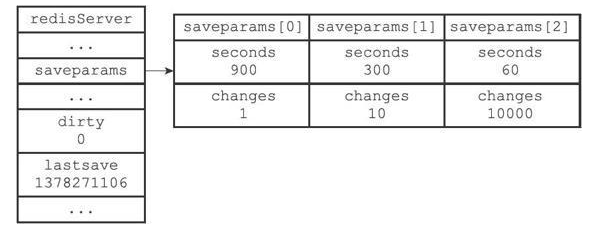
Redis中针对数据库本身的命令，其实都是操作键空间来完成的，在对键空间进行读写操作后，Redis会根据键是否存在来设置键空间命中率次数和不命中次数，并且在读取后，会设置键的最后一次使用时间，来就是键的闲置时间。

redisDb结构中还有一个expires节点，它指向一个字典空间，该字典的键指向数据库中的某个键，而值则记录了数据库键的过期时间，过期时间是一个以毫秒为单位的UNIX时间戳，Redis使用惰性删除和定期删除两种策略来删除过期的键：惰性删除策略只在碰到过期键时才进行删除操作，定期删除策略则每隔一段时间主动查找并删除过期键。

save和bgsave两个持久化命令，都不会持久化，已经过期的键，AOF模式写会生成一个过期键的删除命令到AOF文件的尾部，Redis 2.8 中新增了客户端订阅指定键变化的通知功能。

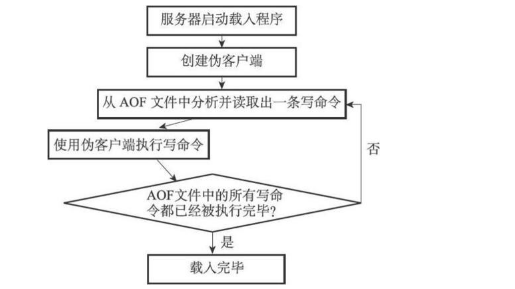
# RDB持久化

Redis数据持久化RDB是将Redis数据库当前的状态，经过压缩保存在一个二进制文件中，当Redis服务器重新启动时会检查是否存在RDB文件，并加载它以恢复数据库原来的状态。在RDB模式下，有save和bgsave两个命令进行持久化操作，save是在主服务进程进行持久化，创建RDB文件，因为是在服务进程中运行所以会阻塞redis运行，bgsave是从主进程中fork出一个子进程然后进行持久化，redis提供自动间隔性保存功能，它是通过redisServer结构中的，saveparams结构数组，来记录，在n秒内有m个变化来自动触发的，dirty和lastsave属性分别记录了，本次周期中修改的次数和最后的保存时间。



# AOF持久化

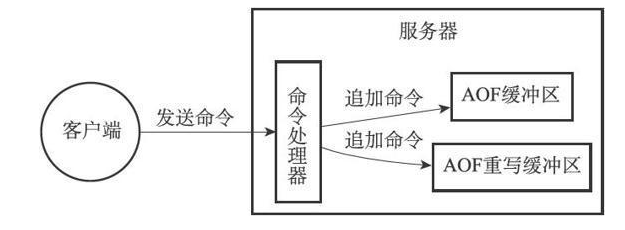
AOF(Append Only File) 是通过保存，Redis服务器所执行的写命令的记录来保存数据库的状态的。它在持久化的时候通过执行追加，文件写入，文件同步三个步骤来完成工作。



Redis，通过提供AOF文件重写公共来避免当个AOF文件过大导致的问题（重写的文件和原来的文件保存的状态是相同的，但是去除了无用的冗余命令，所以文件要小的多）。

AOF重写功能的原理是，通过读取服务器当前的状态而不是读取现有的AOF文件，利用redisServer的Db属性构建写入命令序列，这样就不会有冗余的命令了。

在执行BGREWRITEAOF命令时，Redis服务器会维护一个AOF重写缓冲区，该缓冲区会在子进程创建新AOF文件期间，记录服务器执行的所有写命令。当子进程完成创建新AOF文件的工作之后，服务器会将重写缓冲区中的所有内容追加到新AOF文件的末尾，使得新旧两个AOF文件所保存的数据库状态一致。最后，服务器用新的AOF文件替换旧的AOF文件，以此来完成AOF文件重写操作。

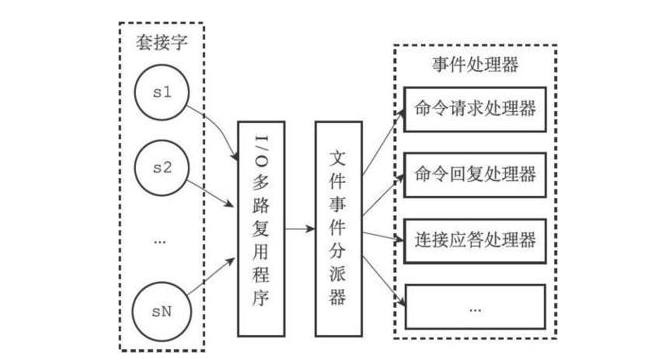


# 事件

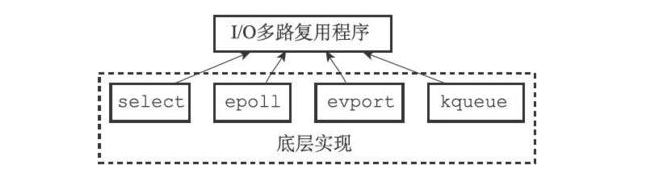
Redis 有文件事件和时间事件两类事件，它们分别用于处理网络套接字和Redis服务器定时操作。

## 文件事件

文件事件使用了reactor模型，利用I/O多路复用，在单线程的Redis上处理多个套接字。



I/O多路复用程序有多个API相同的底层库可选。



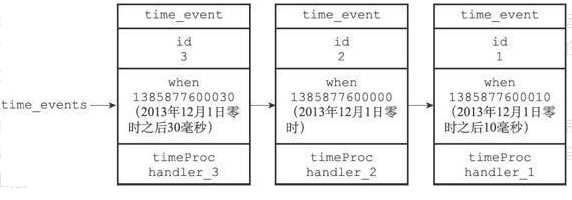
文件事件处理器的有多种类型，它们都是通过与AE\_READABLE或AE\_WRITABLE事件进行关联然后出发的。

|  |  |
| --- | --- |
| 处理器 | **图示** |
| 连接应答处理器 |  |
| 命令请求处理器 |  |
| 命令回复处理器 |  |

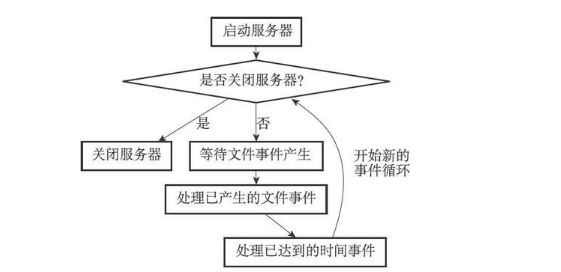
## 时间事件

Redis的时间事件分为定时性的和周期性的事件两种，默认情况下Redis服务器只会运行一个周期性的serverCron时间事件，它用于更新如内存占用等统计信息，清理过期的键值对，持久化，同步等操作。

时间事件的底层是使用无序链表来保存着时间事件的数据，执行时顺序遍历整个链表，查找已满足执行条件的事件，因为Redis仅仅使用了一个时间事件，所以整个遍历不会影响性能。



Redis文件事件和时间事件之间是合作关系，服务器会轮流处理这两种事件，并且处理事件的过程中也不会进行抢占



# 客户端

Redis利用IO多路复用技术使用单进程服务器链接多个客户端，进行通信，对每个连接的客户端创建一个redisClient结构用于保存客户端的状态。所以的redisClient结构都会作为客户端链表的节点连接在 redisServer结构的clients属性上，每一个新增的客户端会添加到链表的尾部。

|  |
| --- |
| typedef struct redisClient {  int fd; // 文件描述符，值为-1的时候代表是 Lua脚本或AOF持久化的伪客户端  robj \*name; // Redis默认下客户端没有名字  int flag; // 不同的标志位值可以表示不同的客户端角色和当前状态  sds querybuf;// 输入缓存区记录的发送的命令请求，其大小如果超过1GB，就会关闭连接  robj \*\*argv; // 命令参数  int argc; // 命令参数的个数  struct redisCommand \*cmd; // 命令实现结构  char buf[REDIS\_REPLY\_CHUNCK\_BTYES]; // 输出缓存区，默认16KB  int bufpos; // 记录buf数组字节数  list \*reply; // 可变输出缓存区，用于buf 16KB空间不够的情况  int authenticated; // 客户端验证，值为1是表示已经通过验证，0则未验证(默认)  time\_t ctime; // 客户端的创建时间  time\_t lastinteraction; // 记录客户端与服务器的最后通讯时间  time\_t obuf\_soft\_limit\_reached\_time; // 输出缓存区第一次到达软性限制的时间  } redisClient;  其中比较重要的一个就是 redisCommand类型的 cmd 属性了，它是redis客户端命令执行的重要一步，在redis服务器将客户端请求的命令保存到 argv和argc中后，会通过redis的命令字典表，通过命令名称找到其对应的redisCommand结构，这个结构实际上就是包括命令实现函数在内的命令信息，然后在讲 cmd属性指向这个结构，并执行，这样就完成了客户端命令请求的执行。 |

# 服务器

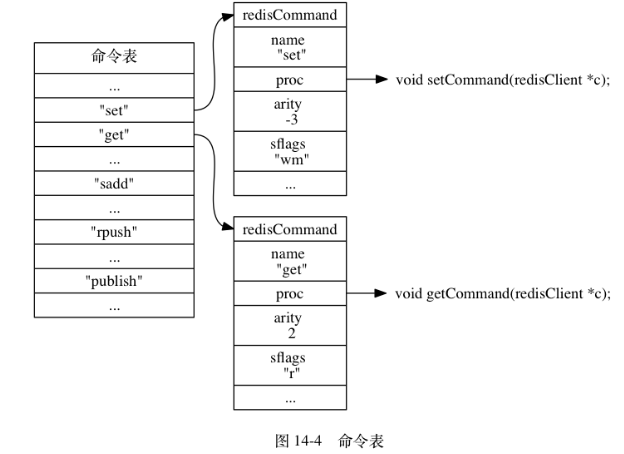
## 命令请求执行过程

当一个命令从客户端发送到服务器后，服务器要经过下面的处理过程，最终才能完成一条命令请求的执行。

**向服务器发送命令请求**：客户端将命令请求的字符串转换成命令协议格式，然后通过套接字连接发送给服务器。

**服务器读取命令请求**：从套接字中读取命令请求，然后在存储到redisClient的querybuf输入缓存区属性中，并且从中解析出命令的参数及其个数，再保存到redisClient的argv和argc属性中。

**查找命令实现**：服务器从redisClient的argv属性中读取第一个元素也就是它要执行的命令的名称，然后通过这个名字在命令表中查找相应的命令。命令表是键为命令名，值为命令redisCommand结构的字典。当查找到对应的命令后，redis服务器会将redisClient的cmd属性的指针指向对应命令的redisCommand上。命令查找不区分大小写



**执行预备操作**：在查找到相应的命令后，redis并不会立即指向相应的实现函数，而是需要先进行一系列的预备检查工作，其中主要的步骤是：

* 检查客户端发送的命令是否存在于命令表中，通过检查cmd属性是否为Null
* 检查命令参数的个数和给定的是否相同。
* 如果命令需要身份验证，那么检查客户端是否已经验证。
* 如果客户端正在使用订阅频道或订阅模式功能，那么除了其相关的命令外其他命令都会被拒绝执行。

**调用命令实现函数**：到了这一步，服务器就可以将redisClient结构本身作为参数传递到其cmd属性所指向的redisCommand结构的proc属性函数中了。

client->cmd->proc(client);

具体的实现函数会从redisClient中读取它需要的参数信息，然后会将命令结构赋值到redisClient的输出缓存区buf属性中。

**执行后续工作**: 完成执行命令后，服务器还需要一些后续步骤如：主从模式下的命令传递，AOF持久化和慢查询日志的检查。

**服务器将命令回复 OK 发送给客户端**：当命令完成执行完后，redisClient会关联一个命令回复处理器，等到套接字可用的时候，服务器就会将输出缓存区的内容写入到套接字中。客户端接收服务器返回的命令回复 OK ， 并将这个回复打印，客户端通过套接字收到命令回复后，安装命令协议格式解析回复，并且打印字符串到输出端上。

## 初始化服务器过程

#1**初始化服务器状态**：redis通过调用initServerConfig函数完成初始化状态，它主要是进行：设置默认端口号和持久化条件，创建命令表和设置服务器ID，默认频率等信息。

#2**载入服务器配置：**这个步骤就是根据用户启动redis server指定选项或配置文件，中对服务器的配置，进行设置服务器的可选参数。诸如：端口号，数据库数量等。

#3**初始化服务器数据结构：**这步中redis将调用initServer函数去，初始化那些非常重要的数据结构，如：记录客户端的server.clients，server.db 和pubsub功能下的server.pubsub\_channels等结构。最后initServer还会设置共享对象，启动时间事件函数和启动I/O准备网络通信。

#4**还原数据库状态：**如果redis开启了持久化功能，那么在这步中就会读取文件恢复数据库的状态。（AOF和RDB）

#5**执行事件循环**：最后一步执行事件循环，等待客户端发送的命令请求。