# 数据库

## 服务器中的数据库

Redis服务器将所有数据库都保存在db数组中，数组的每一个元素就是一个数据库。

在初始化服务器时，服务器会根据dbnum决定应该创建多少数据库

（dbnum由服务器配置的database决定，默认情况下为16）

Struct redisServer{

//…

redisDb \*db; //数据库数组

int dbnum; //服务器的数据库数量

//…

}

## 切换数据库（SELECT命令原理）

通过SELECT可以切换数据库。

在服务器内部，客户端状态redisClient结构的db属性记录了客户端当前的目标数。

Typedef struct redisClient{

//…

redisDb \*db; //记录当前客户端正在使用数据库

}

## 数据库键空间

Redis是一个键值对数据库服务器，服务器中的每一个数据库都由redisDb结构表示，其中redisDb结构中的dict字典保存 了数据库中的所有键值对，这个字典称为键空间：

Typedef struct redisDb{

//…

dict \*dict; //数据库键空间，保存着数据库中所有键值对

//

}redisDb;

## 设置键的生存时间和过期时间

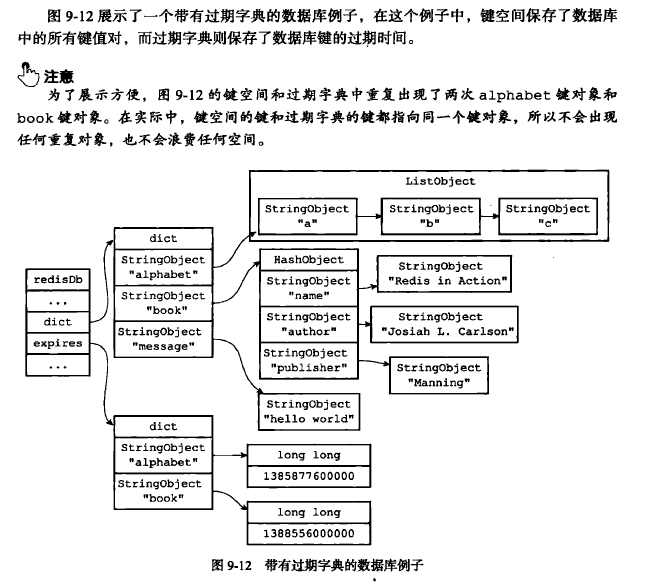
RedisDb结构的expires字典保存了数据库中所有键的过期时间，我们称这个字典为过期字典。

Typedef struct redisDb{

//…

dict \*expires; //过期字典，保存键的过期时间

}



## Redis的过期键删除策略

### 定时删除

定时删除策略：

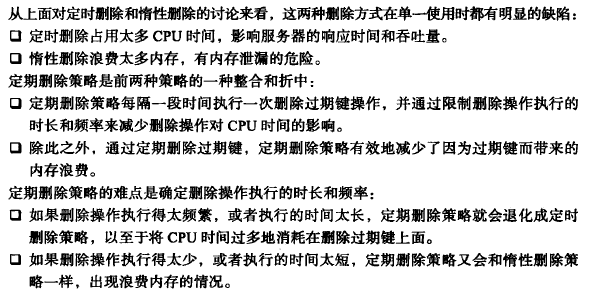
1. 对内存时最友好的：通过使用定时器，定时删除策略可以保证过期键会尽可能快地删除，并释放过期键所占用的内存。
2. 对cpu时间时最不友好的，在过期键较多的情况下，伤处过期键会占用相当一部分CPU时间。在内存不紧张，cpu时间紧张的情况下，将cpu时间用在删除和当前任务无关的过期键上，会对服务器响应时间和吞吐量造成影响。

### 惰性删除

惰性删除策略：

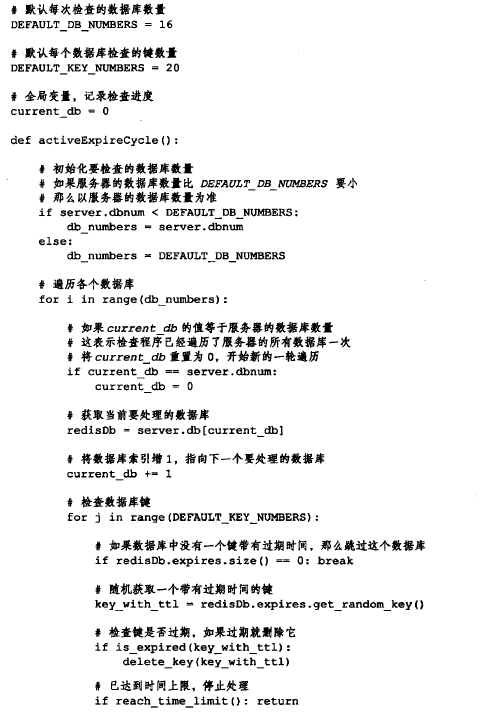
1. 对CPU时间是友好的，程序只有在取出键的时候才对键进行过期检查，这就保证了不会在删除其他无关过期键上浪费cpu时间。
2. 对内存不友好，如果一个键过期了，又保留在数据库那么它所占用的内存就不会释放。

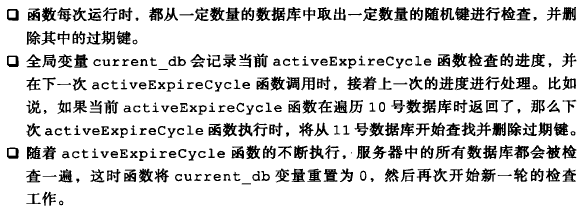
### 定期删除策略



### 定期删除的实现

伪代码：





## AOF,RDB和复制功能对过期键的处理

### RDN

1. 生成RDB文件：已过期键不会被保存
2. 载入RDB文件：

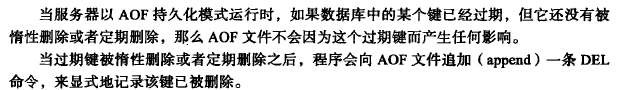
主服务器模式：忽略过期键

从服务器模式：

无论是否过期，一律载入数据库。不过主从服务器在同步数据时，从服务器的数据库会被清空，所以一般来讲，过期键对载入RDB文件的从服务器不会造成影响。

### AOF

1. AOF文件写入：

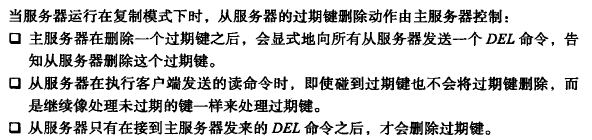


1. AOF重写



### 复制

一句话：过期键删不删，主服务器说的算

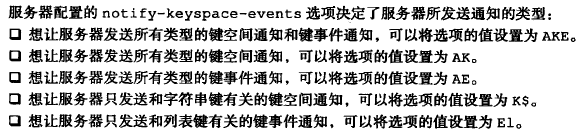


## 数据库通知

数据库通知有键空间通知和键事件通知。

键空间通知：关注” 某键执行了什么命令 “

键事件通知：关注“ 某命令你个被什么键执行了 “



### 发送通知

发送数据库通知的功能使用notifyKeyspaceEvent函数实现的。

Void notifyKeyspaceEvent( int type, char \*event, robj \*key, int dbid )

type:

发送通知的类型，服务器会很具这个值判断是否时notify-keyspace-events选线设置的通知类型，从而决定是否发送

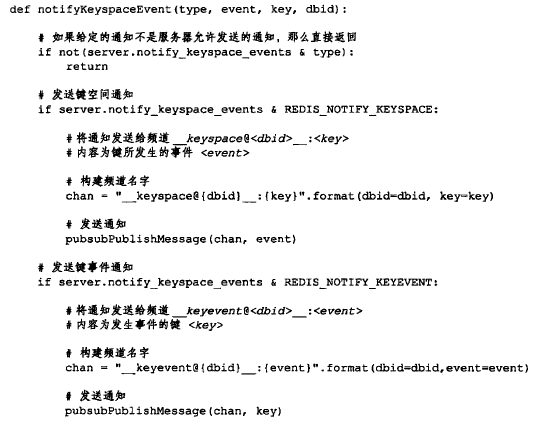
event: 事件名称

key： 产生事件的键

dbid： 产生事件的数据库号码

### 发送通知的实现

伪代码：



NotifyKeyspaceEvent函数执行以下操作：

