# 数据结构与对象

## 简单动态字符串SDS

### 1.1.1 SDS的定义

Struct sdshdr {

int len;

int free;

char buf[];

};

1.1.2 SDS与c字符串的区别

* 常数复杂度获取字符串长度
* 杜绝缓冲区溢出
* 减少修改字符串时带来的内存重分配次数
  + 空间预分配
  + 惰性空间释放
* 二进制安全
* 兼容部分C字符串函数

## 链表

由于C语言没有实现链表结构，Redis实现了自己的链表。

Typedef struct listNode {

struct listNode \*prev;

struct listNode \*next;

void \* value;

} listNode;

## 1.3 字典

Redis的字典使用hash表作为底层实现，一个hash表里面可以有多个hash表节点，而每个hash表节点就保存了字典中的一个键值对。

### 1.3.1 Redis里面使用的哈希算法

|  |
| --- |
| hash = dict->type->hashFunction(key);  index = hash & dict->ht[x].sizemask; |

采用的hash算法是murmurhash2

### 1.3.2 与hash相关

#### 1.3.2.1 解决键冲突

链地址法。

#### 1.3.2.2 rehash

在对hash表进行扩展或者收缩操作时，程序需要将现有hash表包含的所有键值对rehash到新hash表里面，并且这个rehash过程并不是一次性完成的，而是渐进式地完成的。

## 跳跃表

跳跃表是一种有序数据结构，它通过每个节点中维持多个指向其他节点的指针，从而达到快速访问节点的目的。

* 跳跃表是有序集合的底层实现之一。
* Redis的跳跃表实现由zskiplist和zskiplistNode两个结构组成，其中zskiplist用于保存跳跃表信息(比如表头节点、表尾节点、长度)，而zskiplistNode则用于表示跳跃表节点。
* 每个跳跃表节点的层高都是1至32之间的随机数。
* 在同一个跳跃表中，多个节点可以包含相同的分值，但每个节点的成员对象必须是唯一的。
* 跳跃表中的节点按照分值大小进行排序，当分值相同时，节点按照成员对象的大小进行排序。

## 1.5 整数集合

* 整数集合是集合键的底层实现之一。
* 整数集合的底层实现为数组，这个数组以有序、无重复的方式保存集合元素，在有需要时，程序会根据新添加的元素的类型，改变这个数组的类型。
* 升级操作作为整数集合带来了操作上的灵活性，并且尽可能的节约了内存。
* 整数集合只支持升级操作，不支持降级操作。

## 1.6 压缩列表

* 压缩列表是一种为节约内存而开发的顺序型数据结构。
* 压缩列表被用作列表键和hash键的底层实现之一。
* 压缩列表可以包含多个节点，每个节点可以保存一个字节数组或者整数值。
* 添加新节点到压缩列表，或者从压缩列表中删除节点，可能会引发连锁更新操作，但这种操作出现的机率并不高。

## 1.7 对象

# 二、单机数据库

## 2.1数据库

* Redis服务器的所有数据库都保存在redisServer.db数组中，而数据库的数量则由redisServer.dbnum属性保存。
* 客户端通过修改目标数据库指针，让它指向redisServer.db数组中的不同元素来切换不同的数据库。
* 数据库主要由dict和expires两个字典构成，其中dict字典负责保存键值对，而exoires字典则负责保存键的过期时间。
* 因为数据库由字典构成，所以对数据库的操作都是建立在字典操作之上。
* 数据库的键总是一个字符串对象，而值则可以是任意一种Redis对象类型，包括字符串对象、hash表对象、集合对象、列表对象和有序集合对象，分别对应字符串键、集合键、列表键和有序集合键。
* Expires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键。
* 执行SAVE命令或者BGSAVE命令所产生的RDB文件不会包含已经过期的键。
* 执行BGREWRITEAOF命令所产生的重写AOF文件不会包含已经过期的键。
* 当一个过期键被删除之后，服务器会追加一条DEL命令到现有的AOF文件的末尾，显式的删除过期键。
* 当主服务器删除一个过期键之后，它会向所有从服务器发送一条DEL命令，显示的删除过期键。
* 从服务器即使发现过期键也不会自作主张的删除它，而是等待主节点发来DEL命令，这种统一、中心化的过期键删除策略可以保证主从服务器数据的一致性。
* 当Redis命令对数据库进行修改之后，服务器会根据配置向客户端发送数据库通知。

## 2.2 RDB持久化

* RDB文件用于保存和还原Redis服务器所有数据库中的所有键值对数据。
* SAVE命令由服务器进程直接执行保存操作，所以该命令会阻塞服务器。
* BGSAVE命令由子进程执行保存操作，所以该命令不会阻塞服务器。
* 服务器状态中会保存所有用save选项设置的保存条件，当任意一个保存条件被满足时，服务器会自动执行BGSAVE命令。
* RDB文件是一个经过压缩的二进制文件，由多个部分组成。
* 对于不同类型的键值对，RDB文件会使用不同的方式来保存它们。

## 2.3 AOF持久化

* AOF文件通过保存所有修改数据库的写命令请求来记录服务器的数据库状态。
* AOF文件中的所有命令都以Redis命令请求协议的格式保存。
* 命令请求会先保存到AOF缓冲区里面之后再定期写入并同步到AOF文件。
* Appendfsync选项的不同值对AOF持久化功能的安全性以及Redis服务器的性能有很大的影响。
* 服务器只要载入并重新执行保存在AOF文件中的命令，就可以还原数据库本来的状态。
* AOF重写可以产生一个新的AOF文件，这个新的AOF文件和原有的AOF文件保存的数据库状态一样，但体积更小。
* AOF重写是一个有歧义的名字，该功能是通过读取数据库中的键值对来实现的，程序无需对现有的AOF文件进行任何读入、分析或者写入操作。
* 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的AOF文件的工作之后，服务器会将重写缓冲区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。

## 2.4事件

* Redis服务器是一个事件驱动程序，服务器处理的事件分为时间事件和文件事件两类。
* 文件事件处理器是基于Reactor模式实现的网络通信程序。
* 文件事件是对套接字操作的抽象：每次套接字变为可应达(acceptable)、可写(writable)或者可读(readable)时，相应的文件事件就会产生。
* 文件事件分为AE\_READABLE事件(读事件)和AE\_WRITABLE事件(写事件)两类。
* 时间事件分为定时事件和周期性事件：定时事件只在指定的时间到达一次，而周期性事件则每隔一段时间达到一次。
* 服务器在一般情况下只执行serverCron函数一个时间事件，并且这个事件是周期性事件。
* 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。
* 时间事件的实际处理时间通常会比设定的到达时间晚一些。

## 2.5客户端

* 服务器状态结构使用clients链表连接起多个客户端状态，新添加的客户端状态会被放到链表的末尾。
* 客户端状态的flags属性使用不同标志来表示客户端的角色，以及客户端当前所处的状态。
* 输入缓冲区记录了客户端发送的命令请求，这个缓冲区的大小不能超过1GB。
* 命令的参数和参数个数会被记录在客户端状态的argv和argc属性里面，而cmd属性则记录了客户端要执行命令的实现函数。
* 客户端有固定大小缓冲区和可变大小缓冲区两种缓冲区可用，其中固定大小缓冲区的最大大小为16KB，而可变大小缓冲区的最大大小不能超过服务器设置的硬性限制值。
* 输出缓冲区限制值有两种，如果输出缓冲区的大小超过了服务器设置的硬性限制，那么客户端会被立即关闭；除此之外，如果客户端在一定的时间内，一直超过服务器设置的软性限制，那么客户端也会被关闭。
* 当一个客户端通过网络连接连上服务器时，服务器会为这个客户端创建相应的客户端状态。网络连接关闭、发送了不合协议格式的命令请求、成为CLIENT KILL命令的目标、空转时间超时、输出缓冲区的大小超过限制，以上这些原因都会造成客户端被关闭。
* 处理Lua脚本的伪客户端在服务器初始化时创建，这个客户端会一直存在，直到服务器关闭。
* 载入AOF文件时使用的伪客户端在载入工作开始时动态创建，载入工作完毕之后关闭。

## 2.6服务器

* 一个命令请求从发送到完成主要包括以下步骤：1）客户端命令请求发送给服务器；2）服务器读取命令请求，并分析出命令参数；3）命令执行器根据参数查找命令的实现函数，然后执行实现函数并得出命令回复；4）服务器将命令回复返回给客户端。
* serverCron函数默认每隔100毫秒执行一次，它的工作主要包括更新服务器状态信息，处理服务器接收的SIGTERM信号，管理客户端资源和数据库状态，检查并执行持久化操作等等。
* 服务器从启动到能够处理客户端的命令请求需要执行以下步骤：1）初始化服务器状态；2）载入服务器配置；3）初始化服务器数据结构；4）还原数据库状态；5）执行事件循环。

# 三、多数据库的实现

## 3.1复制

* Redis 2.8以前的复制功能不能高效地处理断线后重复制的情况，但Redis 2.8新添加的部分重同步功能可以解决这个问题。
* 部分重同步通过复制偏移量、复制积压缓冲区、服务器运行ID三个部分来实现的。
* 在复制操作刚开始的时候，从服务器会成为主服务器的客户端，并通过向主服务器发送命令请求来执行复制步骤，而在复制操作的后期，主服务器会互相成为对方的客户端。
* 主服务器通过向从服务器传播命令来更新服务器的状态，保持主从服务器一致，而从服务器则通过向主服务器发送命令来进行心跳检测，以及命令丢失检测。

## 3.2 Sentinel

* Sentinel只是一个运行在特殊模式下的Redis服务器，它使用了和普通模式不同的命令表，所以Sentinel模式能够使用的命令和普通Reids服务器能够使用的命令不同。
* Sentinel会读入用户指定的配置文件，为每个要被监视的主服务器创建相应的实例结构，并创建连向主服务器的命令连接和订阅连接，其中命令连接用于向主服务器发送命令请求，而订阅连接则用于接收指定频道的消息。
* Sentinel通过向主服务器发送INFO命令来获得主服务器属下所有从服务器的地址信息，并为这些从服务器创建相应的实例结构，以及连向这些从服务器的命令连接和订阅连接。
* 在一般情况下，Sentinel以每十秒一次的频率向被监视的主服务器和从服务器发送INFO命令，当主服务器处于下线状态，或者Sentinel正在对主服务器进行故障转移操作时，Sentinel向服务器发送INFO命令的频率改为每秒一次。
* 对于监视同一个主服务器和从服务器的多个Sentinel来说，它们会以每两秒一次的频率，通过向被监视服务器的\_\_sentinel\_\_:hello频道发送消息来向其他Sentinel创建相应的实例结构，以及命令连接。
* Sentinel只会与服务器和从服务器创建命令连接和订阅连接，Sentinel与Sentinel之间则只会创建命令连接。
* Sentinel以每秒一次的频率向实例发送PING命令，并根据PING命令的回复来判断实例是否在线，当一个实例在指定时长中连续向Sentinel发送无效回复时，Sentinel会将这个实例判断为主观下线。
* 当Sentinel将一个主服务器判断为主管下线时，它会向同样监视这个主服务器的其他Sentinel进行询问，看它们是否同意这个主服务器已经进入主观下线状态。
* 当Sentinel收集到足够多的主观下线投票之后，它会将主服务器判断为客观下线，并发起一次针对主服务器的故障转移操作。

## 3.3 集群

* 节点通过握手来将其他节点添加到自己所处的集群中。
* 集群中的16384个槽可以分别指派给集群中的各个节点，每个节点都会记录哪些槽指派给了自己，而哪些槽又被指派给了其他节点。
* 节点在接到一个命令请求时，会先检查这个命令请求要处理的键所在的槽是否由自己负责，如果不是的话，节点将向客户端返回一个MOVED错误，MOVED错误携带的信息可以指引客户端转向至正在负责相关槽的节点。
* 如果节点A正在迁移槽i至节点B，那么当节点A没能在自己的数据库中找到命令指定的数据库键时，节点A会向客户端返回一个ASK错误，指引客户端到节点B继续找指定的数据库键。
* MOVED错误表示槽的负责权已经从一个节点转移到了另一个节点，而ASK错误只是两个节点在迁移槽的过程中使用的一种临时措施。
* 集群里的从节点用于复制主节点，并在主节点下线时，代替主节点继续处理命令请求。
* 集群中的节点通过发送和接收消息来进行通信，常见的消息包括MEET、PING、PONG、PUBLISH、FAIL五种。