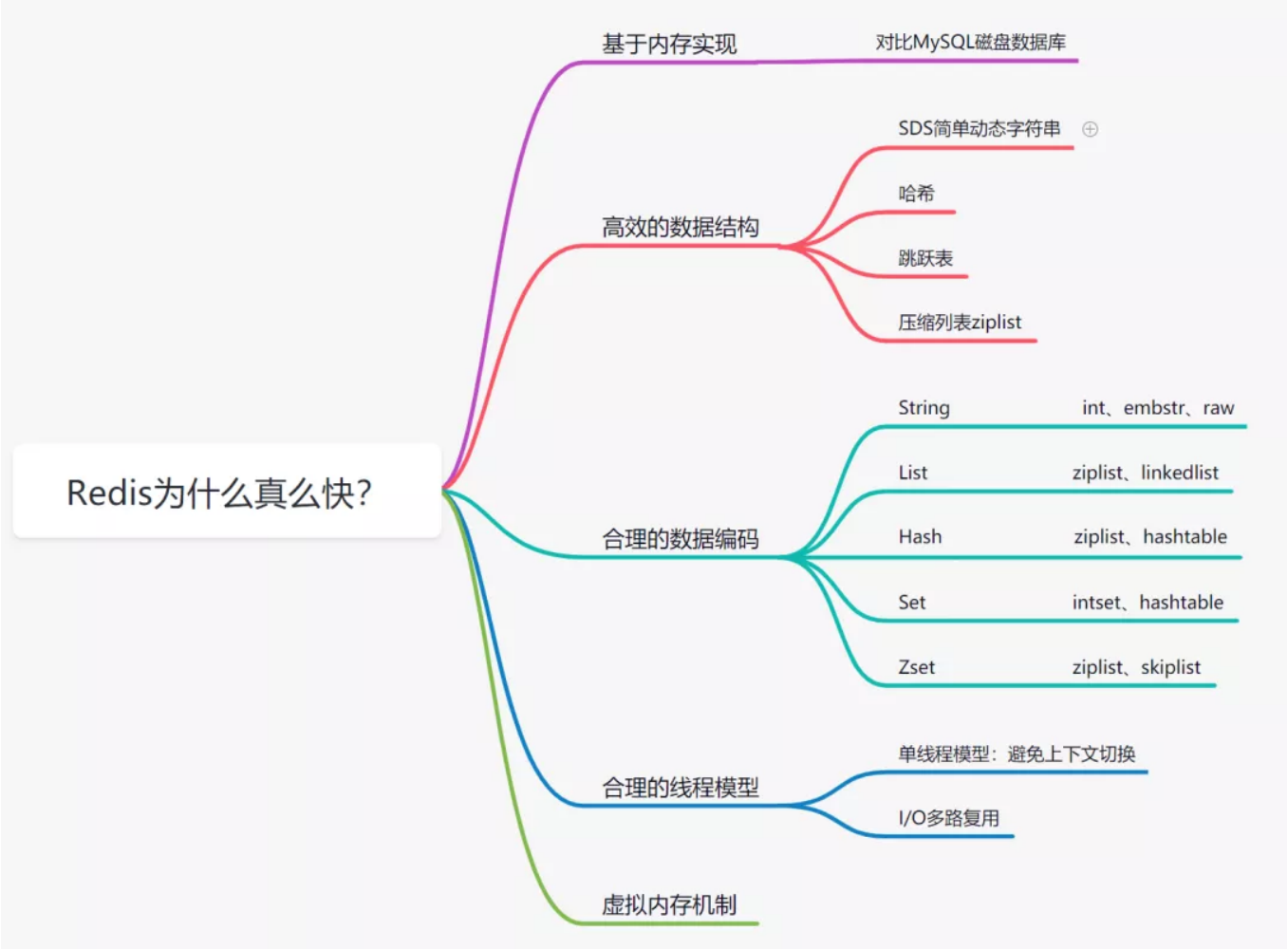
大厂经典面试题：Redis为什么这么快?

Original 捡田螺的小男孩 捡田螺的小男孩 6/27

前言

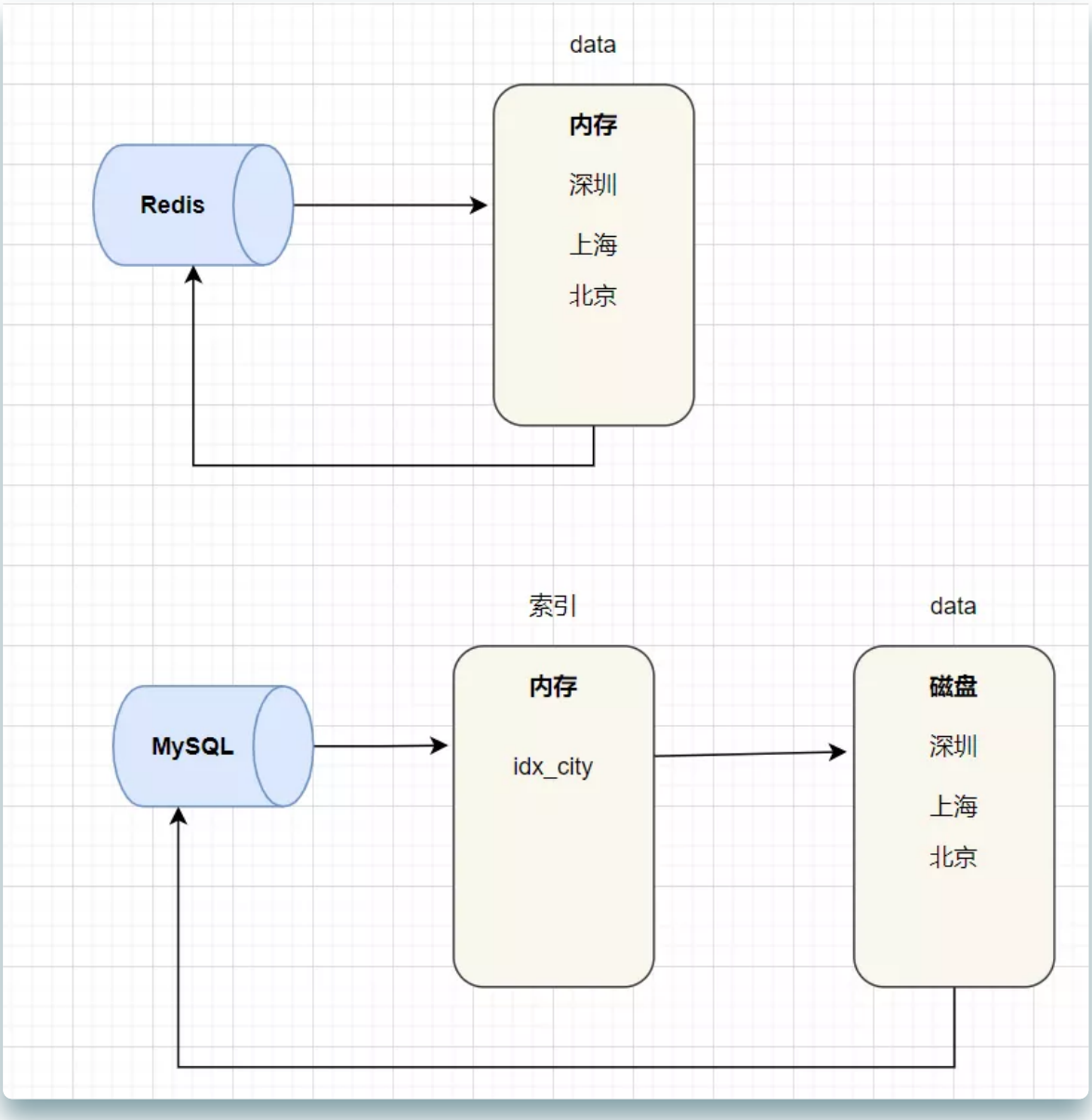
大家好呀，我是捡田螺的小男孩。我们都知道Redis很快，它QPS可达10万（每秒请求数）。**Redis为什么这么快呢**,本文将跟大家一起学习。



- 公众号：**捡田螺的小男孩**

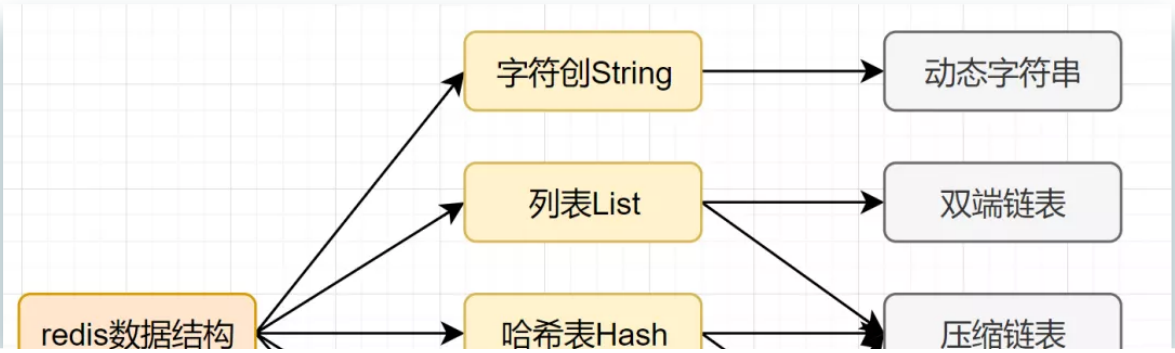
基于内存实现

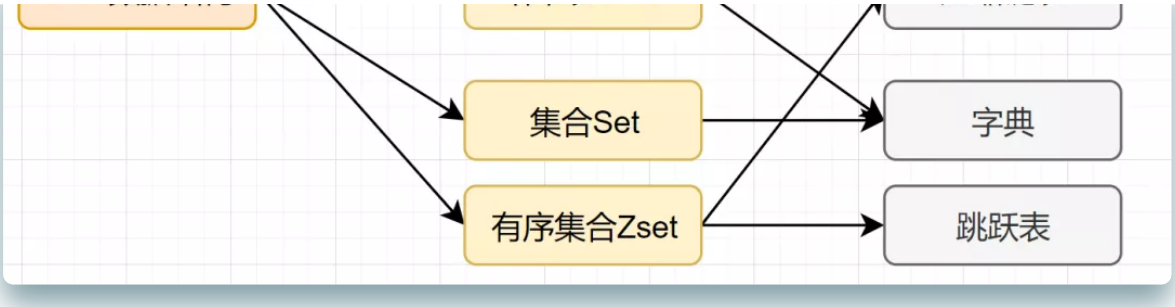
我们都知道内存读写是比磁盘读写快很多的。Redis是基于内存存储实现的数据库，相对于数据存在磁盘的数据库，就省去磁盘磁盘I/O的消耗。MySQL等磁盘数据库，需要建立索引来加快查询效率，而Redis数据存放在内存，直接操作内存，所以就很快。



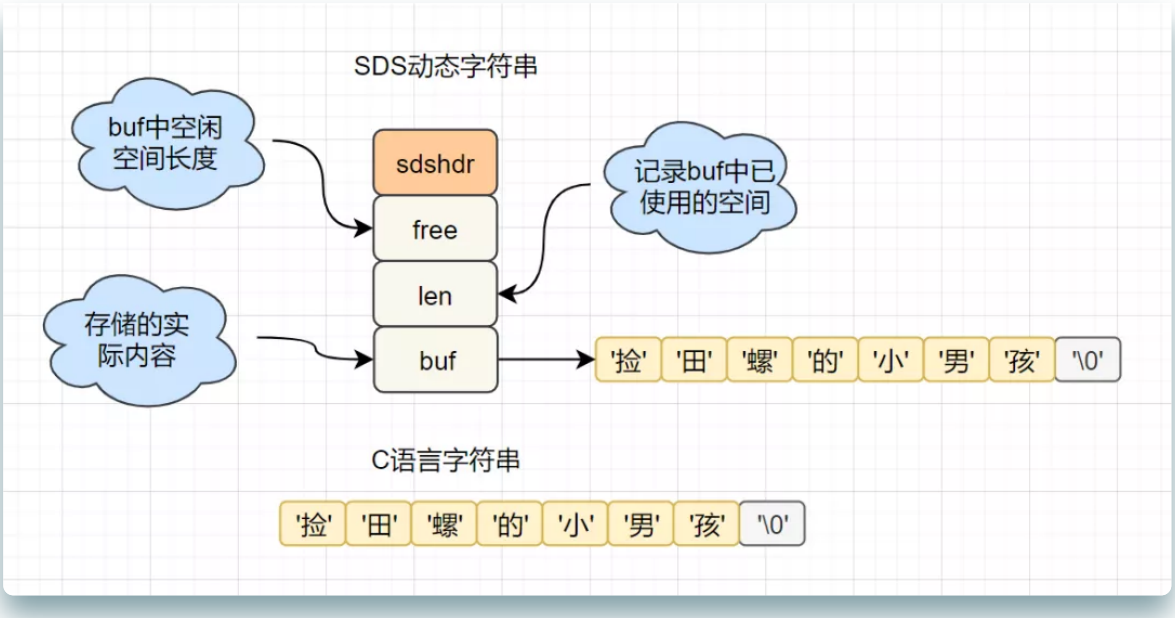
高效的数据结构

我们知道，MySQL索引为了提高效率，选择了B+树的数据结构。其实合理的数据结构，就是可以让你的应用/程序更快。先看下Redis的数据结构&内部编码图：





SDS简单动态字符串




```
struct sdshdr { //SDS简单动态字符串
    int len;    //记录buf中已使用的空间
    int free;   // buf中空闲空间长度
    char buf[]; //存储的实际内容
}
```

字符串长度处理

在C语言中，要获取 捡田螺的小男孩 这个字符串的长度，需要从头开始遍历，复杂度为O（n）；在Redis中， 已经有一个**len**字段记录当前字符串的长度啦，直接获取即可，时间复杂度为O(1)。

减少内存重新分配的次数

在C语言中，修改一个字符串，需要重新分配内存，修改越频繁，内存分配就越频繁，而分配内存是会**消耗性能**的。而在Redis中，SDS提供了两种优化策略：空间预分配和惰性空间释放。

空间预分配

当SDS简单动态字符串修改和空间扩充时，除了分配必需的内存空间，还会额外分配未使用的空间。分配规则是酱紫的：

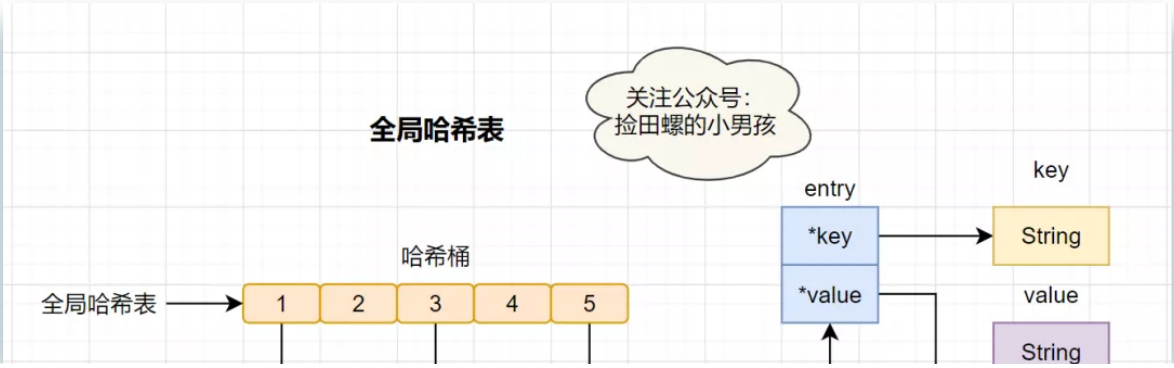
- SDS修改后，len的长度小于1M，那么将额外分配与len相同长度的未使用空间。比如len=100，重新分配后，buf 的实际长度会变为100(已使用空间)+100(额外空间)+1(空字符)=201。
- SDS修改后, len长度大于1M，那么程序将分配1M的未使用空间。

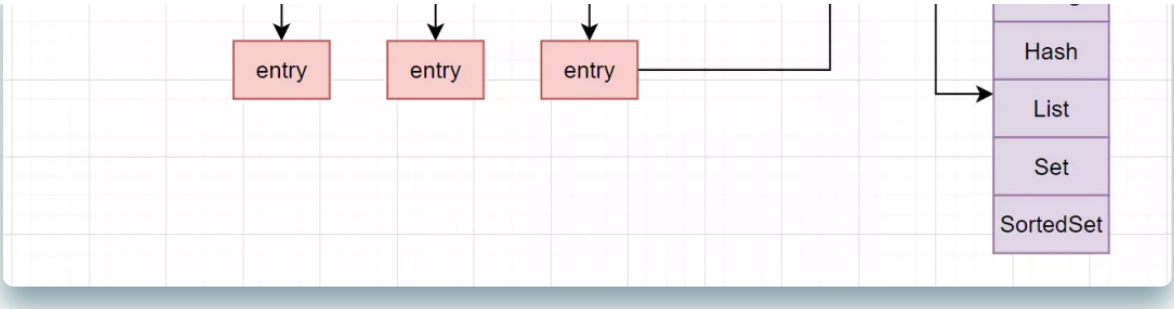
惰性空间释放

当SDS缩短时，不是回收多余的内存空间，而是用free记录下多余的空间。后续再有修改操作，直接使用free中的空间，减少内存分配。

哈希

Redis 作为一个K-V的内存数据库，它使用用一张全局的哈希来保存所有的键值对。这张哈希表，有多个哈希桶组成，哈希桶中的entry元素保存了 `*key` 和 `*value` 指针，其中 `*key` 指向了实际的键， `*value` 指向了实际的值。



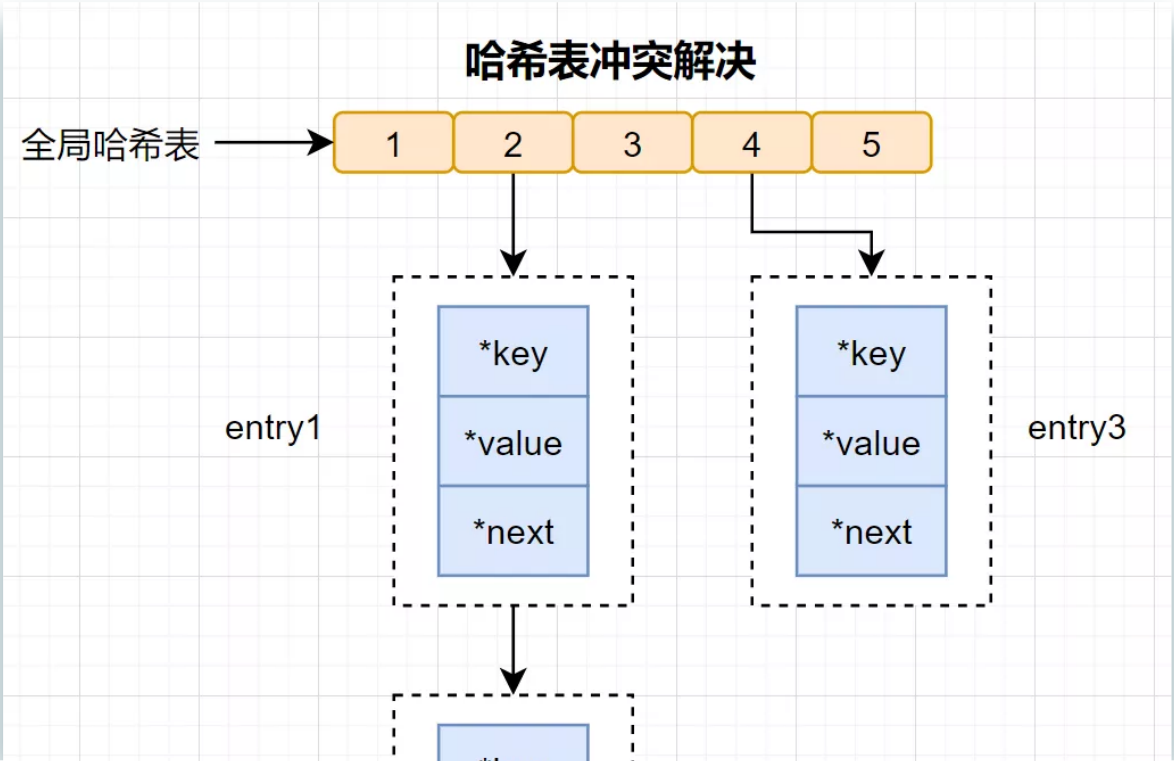


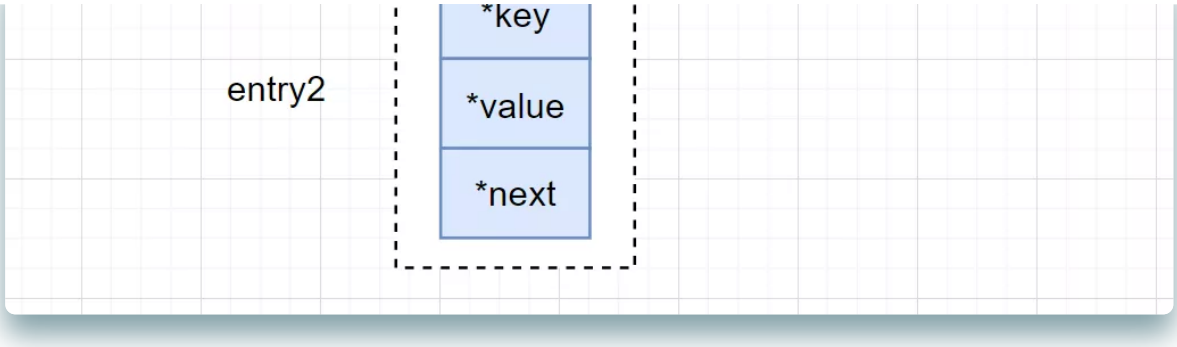
哈希表查找速率很快的，有点类似于Java中的**HashMap**，它让我们在**O(1)** 的时间复杂度快速找到键值对。首先通过key计算哈希值，找到对应的哈希桶位置，然后定位到entry，在entry找到对应的数据。

有些小伙伴可能会有疑问：你往哈希表中写入大量数据时，不是会遇到**哈希冲突**问题嘛，那效率就会降下来啦。

> ★ **哈希冲突：** 通过不同的key，计算出一样的哈希值，导致落在同一个哈希桶中。

Redis为了解决哈希冲突，采用了**链式哈希**。链式哈希是指同一个哈希桶中，多个元素用一个链表来保存，它们之间依次用指针连接。



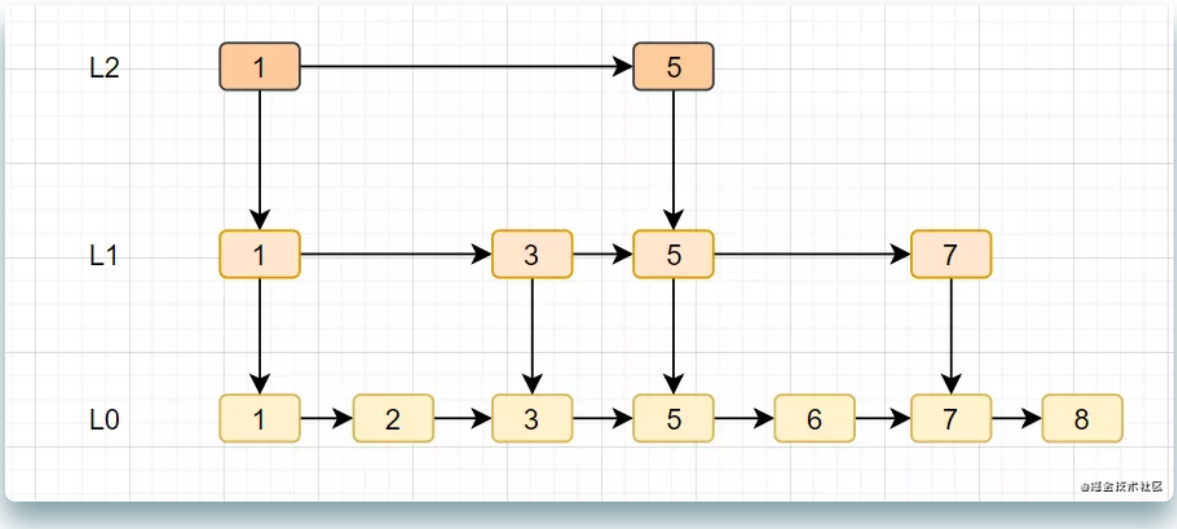


有些小伙伴可能还会有疑问：哈希冲突链上的元素只能通过指针逐一查找再操作。当往哈希表插入数据很多，冲突也会越多，冲突链表就会越长，那查询效率就会降低了。

为了保持高效，Redis 会对哈希表做**rehash操作**，也就是增加哈希桶，减少冲突。为了**rehash更高效**，Redis还默认使用了**两个全局哈希表**，一个用于当前使用，称为主哈希表，一个用于扩容，称为备用哈希表。

跳跃表

跳跃表是Redis特有的数据结构，它其实就是在**链表的基础上，增加多级索引**，以提高查找效率。跳跃表的简单原理图如下：



- 每一层都有一条有序的链表，最底层的链表包含了所有的元素。
- 跳跃表支持平均 O（logN）,最坏 O（N）复杂度的节点查找，还可以通过顺序性操作批量处理节点。

压缩列表ziplist

压缩列表ziplist是列表键和字典键的的底层实现之一。它是由一系列特殊编码的内存块构成的列表， 一个ziplist可以包含多个entry， 每个entry可以保存一个长度受限的字符数组或者整数，如下：

- zlbytes： 记录整个压缩列表占用的内存字节数
- zltail: 尾节点至起始节点的偏移量
- zllen : 记录整个压缩列表包含的节点数量
- entryX: 压缩列表包含的各个节点
- zlend : 特殊值0xFF(十进制255)，用于标记压缩列表末端

由于内存是**连续分配**的，所以遍历速度很快。。

合理的数据编码

Redis支持多种数据基本类型，每种基本类型对应不同的数据结构，每种数据结构对应不一样的编码。为了提高性能，Redis设计者总结出，数据结构最适合的编码搭配。

Redis是使用对象（redisObject）来表示数据库中的键值，当我们在 Redis 中创建一个键值对时，至少创建两个对象，一个对象是用做键值对的键对象，另一个是键值对的值对象。

```
//关注公众号：捡田螺的小男孩
typedef struct redisObject{
    //类型
   unsigned type:4;
   //编码
   unsigned encoding:4;
   //指向底层数据结构的指针
   void *ptr;
    //...
 }robj;
```

redisObject中，**type** 对应的是对象类型，包含String对象、List对象、Hash对象、Set对象、zset对象。**encoding** 对应的是编码。

- String：如果存储数字的话，是用int类型的编码;如果存储非数字，小于等于39字节的字符串，是embstr；大于39个字节，则是raw编码。
- List：如果列表的元素个数小于512个，列表每个元素的值都小于64字节（默认），使用ziplist编码，否则使用linkedlist编码
- Hash：哈希类型元素个数小于512个，所有值小于64字节的话，使用ziplist编码,否则使用hashtable编码。
- Set：如果集合中的元素都是整数且元素个数小于512个，使用intset编码，否则使用hashtable编码。
- Zset：当有序集合的元素个数小于128个，每个元素的值小于64字节时，使用ziplist编码，否则使用skiplist（跳跃表）编码

合理的线程模型

单线程模型：避免了上下文切换

Redis是单线程的，其实是指**Redis的网络IO和键值对读写**是由一个线程来完成的。但Redis的其他功能，比如持久化、异步删除、集群数据同步等等，实际是由额外的线程执行的。

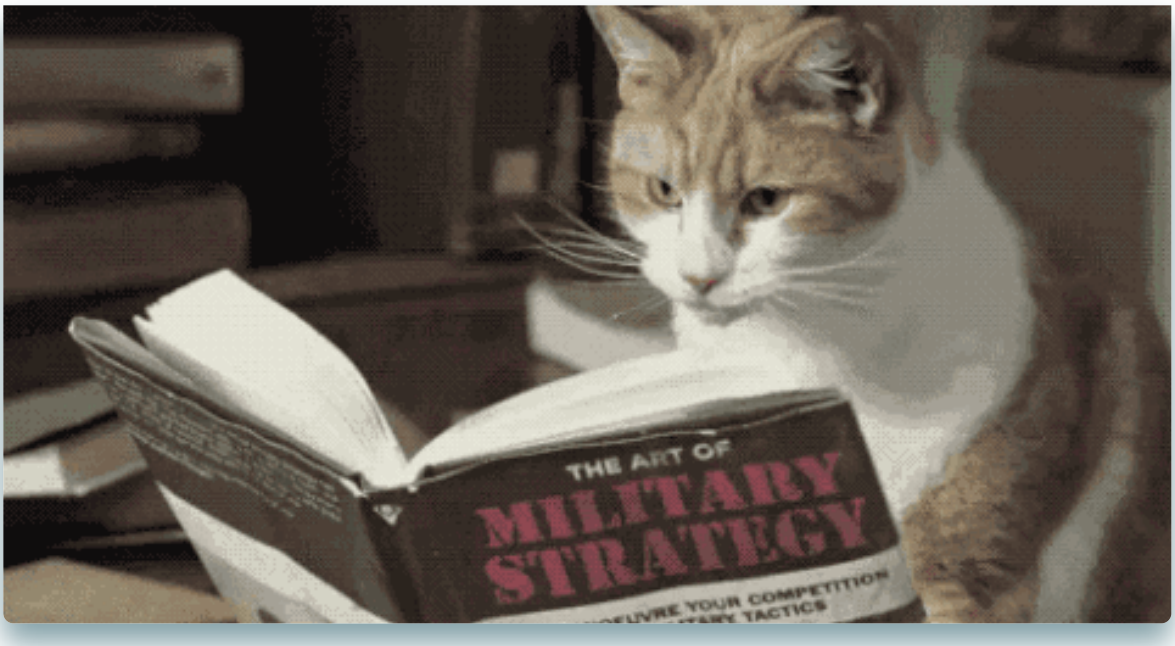
Redis的单线程模型，避免了**CPU不必要的上下文切换**和**竞争锁的消耗**。也正因为是单线程，如果某个命令执行过长（如hgetall命令），会造成阻塞。Redis是面向快速执行场景的内存数据库，所以要慎用如lrange和smembers、hgetall等命令。

什么是**上下文切换**？举个粟子：

- 比如你在看一本英文小说，你看到某一页，发现有个单词不会读，你加了个书签，然后去查字典。查完字典后，你回来从书签那里继续开始读，这个流程就很舒畅。
- 如果你一个人读这本书，肯定没啥问题。但是如果你去查字典的时候，别的小伙伴翻了一下你的书，然后溜了。你再回来看的时候，发现书不是你看的那一页了，你得花时间找到你的那一页。

- 一本书，你一个人怎么看怎么打标签都没事，但是人多了翻来翻去，这本书各种标记就很乱了。可能这个解释很粗糙，但是道理应该是一样的。

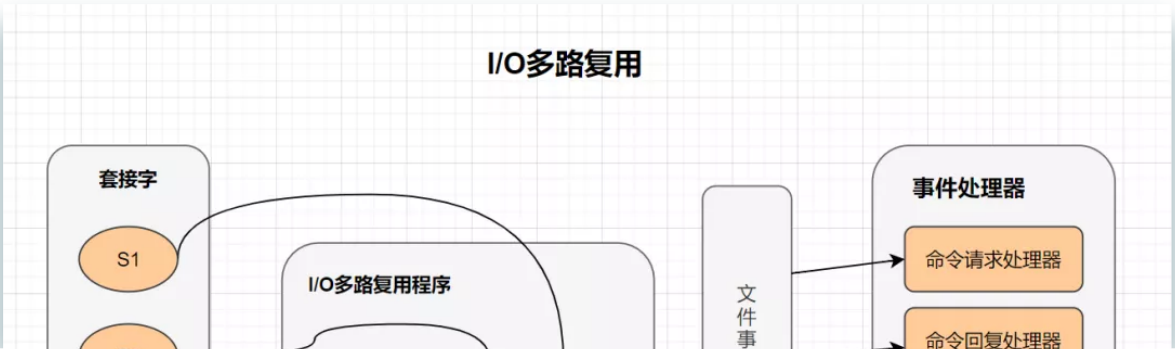
”

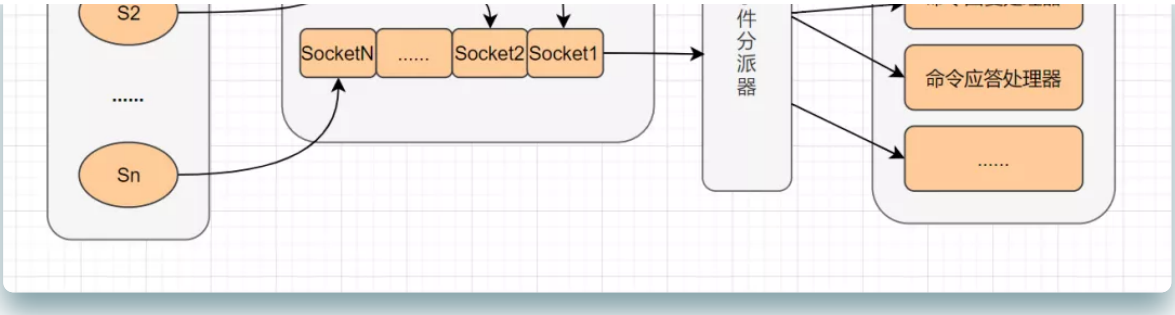


I/O 多路复用

什么是I/O多路复用？

- I/O： 网络 I/O
- 多路： 多个网络连接
- 复用：复用同一个线程。
- IO多路复用其实就是一种同步IO模型，它实现了一个线程可以监视多个文件句柄；一旦某个文件句柄就绪，就能够通知应用程序进行相应的读写操作；而没有文件句柄就绪时,就会阻塞应用程序，交出cpu。





多路I/O复用技术可以让单个线程高效的处理多个连接请求，而Redis使用用epoll作为I/O多路复用技术的实现。并且Redis自身的事件处理模型将epoll中的连接、读写、关闭都转换为事件，不在网络I/O上浪费过多的时间。

虚拟内存机制

Redis直接自己构建了VM机制 ，不会像一般的系统会调用系统函数处理，会浪费一定的时间去移动和请求。

Redis的虚拟内存机制是啥呢？

虚拟内存机制就是暂时把不经常访问的数据(冷数据)从内存交换到磁盘中，从而腾出宝贵的内存空间用于其它需要访问的数据(热数据)。通过VM功能可以实现冷热数据分离，使热数据仍在内存中、冷数据保存到磁盘。这样就可以避免因为内存不足而造成访问速度下降的问题。

参考资料

[1] Redis之VM机制: *https://www.codenong.com/cs106843764/*

[2] 一文揭秘单线程的Redis为什么这么快?: *https://zhuanlan.zhihu.com/p/57089960*

[3] 洞察|Redis是单线程的，但Redis为什么这么快？: *https://zhuanlan.zhihu.com/p/42272979*

捡田螺的小男孩

专注后端技术栈，热爱分享，热爱交朋友，热爱工作总结。毕业于华南理工大学，软件工程...

94篇原创内容

Official Account

跪求大家帮忙点个赞、在看、转发，感谢！