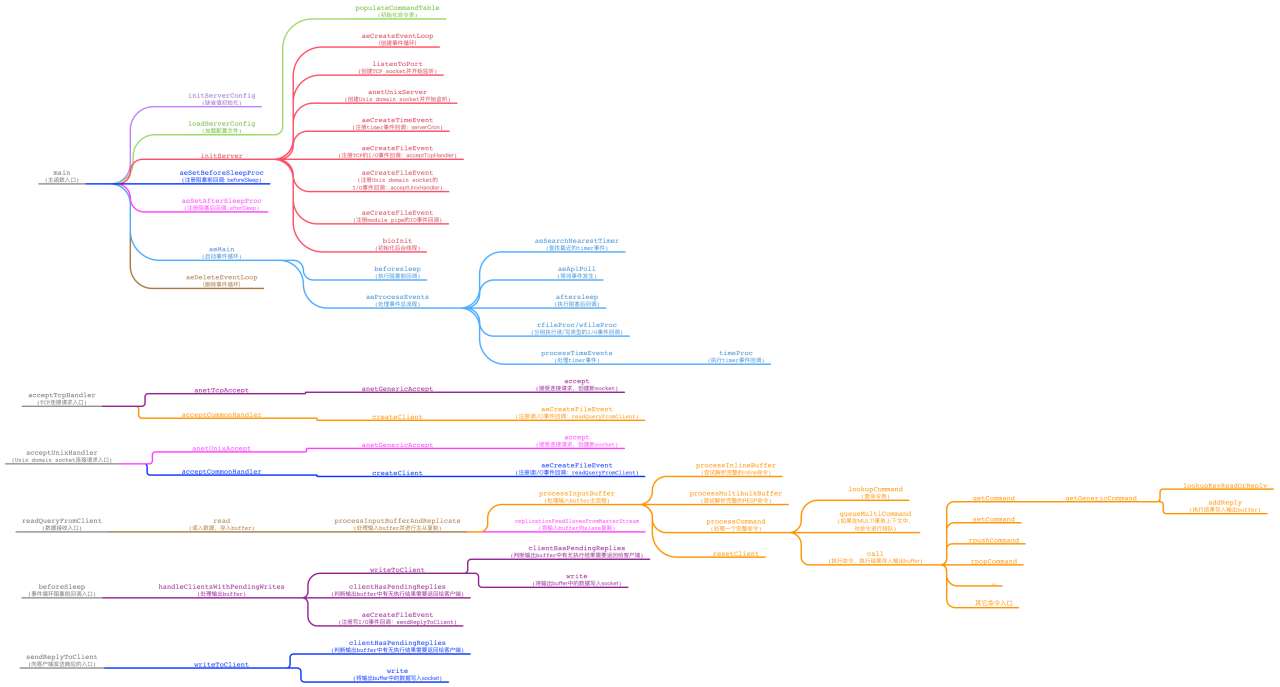
Redis

1. Redis

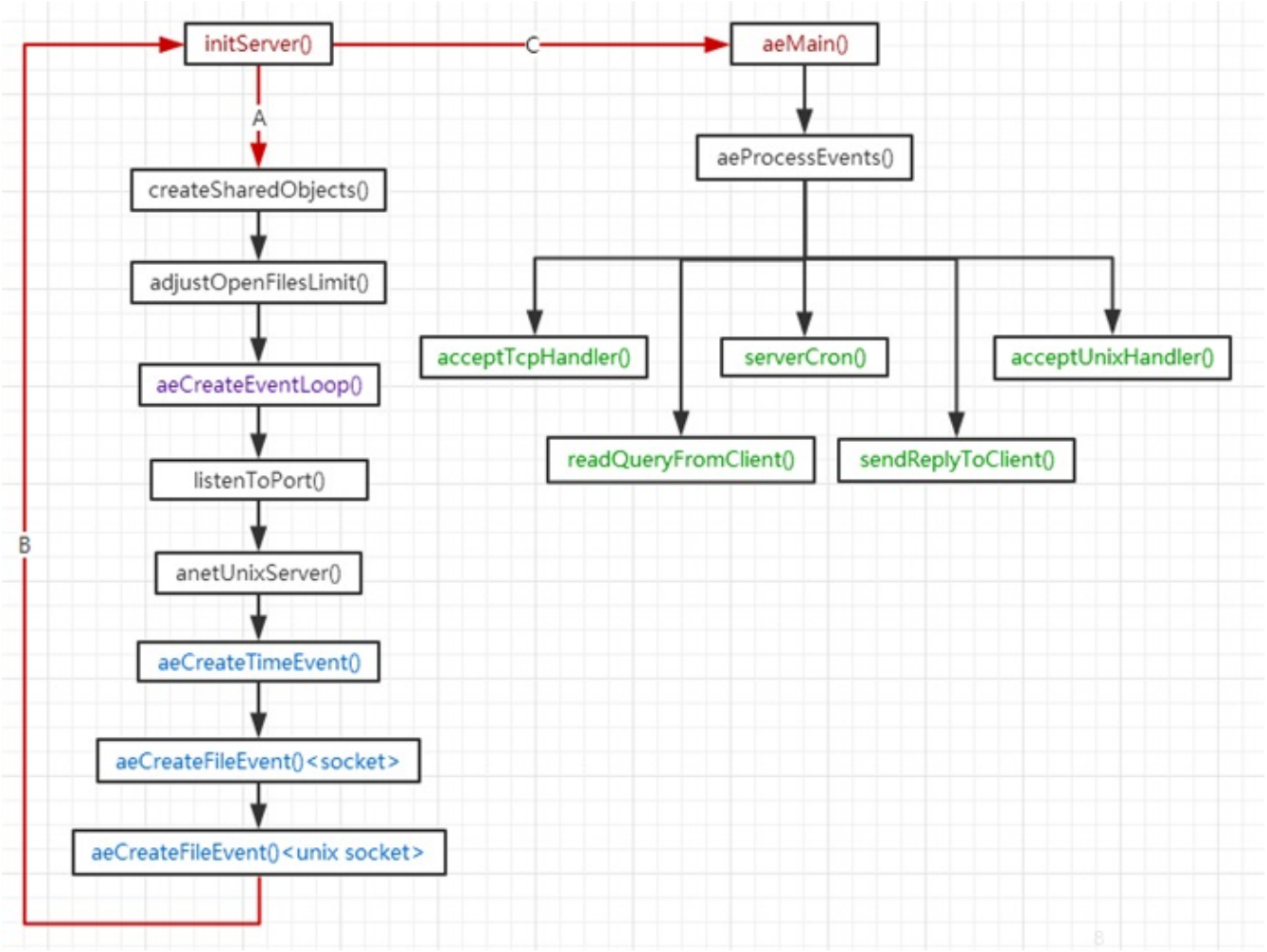
https://cloud.tencent.com/developer/article/1408753



2. redis http://www.ituring.com.cn/article/265187

```
//--打散argv和environ所存储内容的空间
spt_init(argc, argv);
//--主要在初始化全局的server对象中的各个数据成员
initServerConfig();
//--继续初始化全局的server对象，并在里面初始化了全局的shared对象
//--监听端口和uinx socket文件
//--启动bio线程
//--lua环境
initServer();
//--检查系统的THP和overcommit_memory
linuxMemoryWarnings();
//--检查tcp_backlog和系统的somaxconn参数值
checkTcpBacklogSettings();
//--根据RDB或者AOF文件加载旧数据，优先AOF文件
loadDataFromDisk();
//--进入事件循环处理
aeMain(server.el);
```

3. redishttp://www.ituring.com.cn/article/196415



4. redis1 https://www.cnblogs.com/kernel_hcy/archive/2011/05/15/2046963.html

5. *redis https://blog.csdn.net/jasper_xulei/article/details/18364313

6. Redis https://blog.csdn.net/yswKnight/article/details/78158540

7. redis https://blog.csdn.net/cjk_cynosure/article/details/70124975

8. redis 4.0module () https://blog.csdn.net/gochenguowei/article/details/85226346

9. Redis radix tree https://blog.csdn.net/yunqiinsight/article/details/89394215

10. Radix tree http://www.360doc.com/content/19/0305/18/496343_819431105.shtml

11. Redis RDB File Format http://rdb.fnordig.de/file_format.html

Value Type

A one byte flag indicates encoding used to save the Value.

- `0` = String Encoding
- `1` = List Encoding
- `2` = Set Encoding
- `3` = Sorted Set Encoding
- `4` = Hash Encoding
- `9` = Zipmap Encoding
- `10` = Ziplist Encoding
- `11` = Intset Encoding
- `12` = Sorted Set in Ziplist Encoding
- `13` = Hashmap in Ziplist Encoding (Introduced in RDB version 4)
- `14` = List in Quicklist encoding (Introduced in RDB version 7)

12. Redis(5)——quicklist http://zhangtielei.com/posts/blog-redis-quicklist.html

13. Redis(1)——dict http://zhangtielei.com/posts/blog-redis-dict.html

在讨论任何一个系统的内部实现的时候，我们都要先明确它的设计原则，这样我们才能更深刻地理解它为什么会进行如此设计的真正意图。在本文接下来的讨论中，我们主要关注以下几点：

- 存储效率（memory efficiency）。Redis是专用于存储数据的，它对于计算机资源的主要消耗就在于内存，因此节省内存是它非常非常重要的一个方面。这意味着Redis一定是非常精细地考虑了压缩数据、减少内存碎片等问题。
- 快速响应时间（fast response time）。与快速响应时间相对的，是高吞吐量（high throughput）。Redis是用于提供在线访问的，对于单个请求的响应时间要求很高，因此，快速响应时间是比高吞吐量更重要的目标。有时候，这两个目标是矛盾的。
- 单线程（single-threaded）。Redis的性能瓶颈不在于CPU资源，而在于内存访问和网络IO。而采用单线程的设计带来的好处是，极大简化了数据结构和算法的实现。相反，Redis通过异步IO和pipelining等机制来实现高速的并发访问。显然，单线程的设计，对于单个请求的快速响应时间也提出了更高的要求。

14. Redis(2)——sds http://zhangtielei.com/posts/blog-redis-sds.html

15. Redis(3)——robj http://zhangtielei.com/posts/blog-redis-robj.html

有10种encoding，在这里我们先简单解释一下，在这个系列后面的文章中，我们应该还有机会碰到它们。

- OBJ_ENCODING_RAW: 最原生的表示方式。其实只有string类型才会用这个encoding值（表示成sds）。
- OBJ_ENCODING_INT: 表示成数字。实际用long表示。
- OBJ_ENCODING_HT: 表示成dict。
- OBJ_ENCODING_ZIPMAP: 是个旧的表示方式，已不再用。在小于Redis 2.6的版本中才有。
- OBJ_ENCODING_LINKEDLIST: 也是个旧的表示方式，已不再用。
- OBJ_ENCODING_ZIPLIST: 表示成ziplist。
- OBJ_ENCODING_INTSET: 表示成intset。用于set数据结构。
- OBJ_ENCODING_SKIPLIST: 表示成skiplist。用于sorted set数据结构。
- OBJ_ENCODING_EMBSTR: 表示成一种特殊的嵌入式的sds。
- OBJ_ENCODING_QUICKLIST: 表示成quicklist。用于list数据结构。

16. Redis(4)——ziplist http://zhangtielei.com/posts/blog-redis-ziplist.html

17. Redis(6)——skiplist http://zhangtielei.com/posts/blog-redis-skiplist.html

在分析之前，我们还需要着重指出的是，执行插入操作时计算随机数的过程，是一个很关键的过程，它对skiplist的统计特性有着很重要的影响。这并不是一个普通的服从均匀分布的随机数，它的计算过程如下：

- 首先，每个节点肯定都有第1层指针（每个节点都在第1层链表里）。
- 如果一个节点有第i层(i>=1)指针（即节点已经在第1层到第i层链表中），那么它有第(i+1)层指针的概率为p。
- 节点最大的层数不允许超过一个最大值，记为MaxLevel。

这个计算随机层数的伪码如下所示：

```
randomLevel()
    level := 1
    // random()返回一个[0...1)的随机数
    while random() < p and level < MaxLevel do
        level := level + 1
    return level
```

randomLevel()的伪码中包含两个参数，一个是p，一个是MaxLevel。在Redis的skiplist实现中，这两个参数的取值为：

```
p = 1/4
MaxLevel = 32
```

skiplist与平衡树、哈希表的比较

- skiplist和各种平衡树（如AVL、红黑树等）的元素是有序排列的，而哈希表不是有序的。因此，在哈希表上只能做单个key的查找，不适宜做范围查找。所谓范围查找，指的是查找那些大小在指定的两个值之间的所有节点。
- 在做范围查找的时候，平衡树比skiplist操作要复杂。在平衡树上，我们找到指定范围的小值之后，还需要以中序遍历的顺序继续寻找其它不超过大值的节点。如果不对平衡树进行一定的改造，这里的中序遍历并不容易实现。而在skiplist上进行范围查找就非常简单，只需要在找到小值之后，对第1层链表进行若干步的遍历就可以实现。
- 平衡树的插入和删除操作可能引发子树的调整，逻辑复杂，而skiplist的插入和删除只需要修改相邻节点的指针，操作简单又快速。
- 从内存占用上来说，skiplist比平衡树更灵活一些。一般来说，平衡树每个节点包含2个指针（分别指向左右子树），而skiplist每个节点包含的指针数目平均为1/(1-p)，具体取决于参数p的大小。如果像Redis里的实现一样，取p=1/4，那么平均每个节点包含1.33个指针，比平衡树更有优势。
- 查找单个key，skiplist和平衡树的时间复杂度都为O(log n)，大体相当；而哈希表在保持较低的哈希值冲突概率的前提下，查找时间复杂度接近O(1)，性能更高一些。所以我们平常使用的各种Map或dictionary结构，大都是基于哈希表实现的。
- 从算法实现难度上来比较，skiplist比平衡树要简单得多。

18. Redis(7)——intset http://zhangtielei.com/posts/blog-redis-intset.html

19. RedisLFU https://blog.csdn.net/zhujibcom/article/details/90740523

20. Redis https://blog.csdn.net/sinat_35261315/article/details/78976272

过期键删除策略

对于过期键值对的删除有三种策略，分别是

- 定时删除，设置一个定时器和回调函数，时间一到就调用回调函数删除键值对。优点是及时删除，缺点是需要为每个键值对都设置定时器，比较麻烦(其实可以用timer_fd的，参考muduo定时任务的实现)
- 惰性删除，只有当再次访问该键时才判断是否过期，如果过期将其删除。优点是不需要为每个键值对进行时间监听，缺点是如果这个键值对一直不被访问，那么即使过期也会一直残留在数据库中，占用不必要的内存
- 周期删除，每隔一段时间执行一次删除过期键值对的操作。优点是既不需要监听每个键值对导致占用CPU，也不会一直不删除导致占用内存，缺点是不容易确定删除操作的执行时长和频率

Redis采用惰性删除和周期删除两种策略，通过配合使用，服务器可以很好的合理使用CPU时间和避免内不能空间的浪费

21. redis(): serverCron https://www.cnblogs.com/flypighhblog/p/7757996.html

22. redis—AE https://blog.csdn.net/yuweiping5247/article/details/83786492

23.Redisredis https://blog.csdn.net/qq_35219282/article/details/80894759

24. redis RDB https://www.cnblogs.com/daoluanxiaozi/p/3625285.html

25. redis6——aof rewrite https://blog.csdn.net/chosen0ne/article/details/44461497/

26. redis4.0RDB-AOF https://yq.aliyun.com/articles/193034

27. redis4.0aofrewrite https://yq.aliyun.com/articles/177819?spm=a2c4e.11153940.0.0.2c137063Lv9qW9

28. http://wiki.jikexueyuan.com/project/redis/master-slave-replication.html

29. redis -slave https://www.jianshu.com/p/e10d21ecdd0b

30. Redis Replication https://521-wf.com/archives/414.html

31. Redis4.0()-PSYNC2 https://yq.aliyun.com/articles/245528?utm_content=m_34929

在之前的版本，redis重启后，复制信息是完全丢失;所以从实例重启后，只能进行全量重新同步。

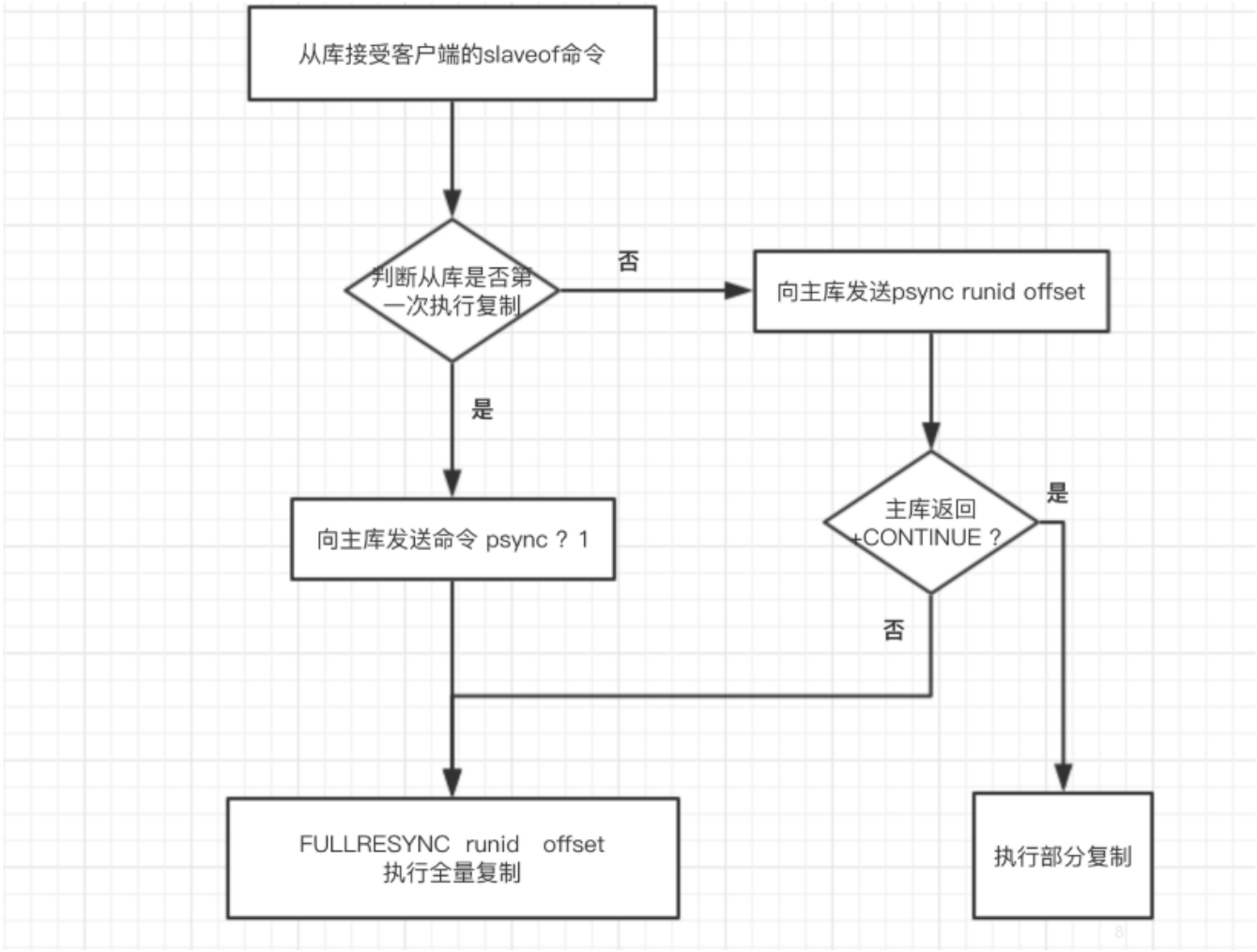
redis4.0为实现重启后，仍可进行部分重新同步，主要做以下3点：

redis关闭时，把复制信息作为辅助字段(AUX Fields)存储在RDB文件中；以实现同步信息持久化；

redis启动加载RDB文件时，会把复制信息赋给相关字段；

redis重新同步时，会上报repl-id和repl-offset同步信息，如果和主实例匹配，且offset还在主实例的复制积压缓冲区内，则只进行部分重新同步。

32. redis–redis https://www.cnblogs.com/wdliu/p/9407179.html



33. Java(23-4) - Redis Replication https://www.wandouip.com/t5i358316/