01-带你快速攻略Redis源码的整体架构

你好，我是蒋德钧。从今天这节课开始，我们将开启“Redis代码之旅”，一起来掌握Redis的核心设计思想。

不过，在正式开始我们的旅程之前，还需要先做个“攻略”，也就是要了解和掌握Redis代码的整体架构。

这是因为，一旦掌握了Redis代码的整体架构，就相当于给Redis代码画了张全景图。有了这张图，我们再去学习Redis不同功能模块的设计与实现时，就可以从图上快速查找和定位这些功能模块对应的代码文件。而且，有了代码的全景图之后，我们还可以对Redis各方面的功能特性有个全面了解，这样也便于更加全面地掌握Redis的功能，而不会遗漏某一特性。

那么，我们究竟该如何学习Redis的代码架构呢？我的建议是要掌握以下两方面内容：

- **代码的目录结构和作用划分**，目的是理解Redis代码的整体架构，以及所包含的代码功能类别；
- **系统功能模块与对应代码文件**，目的是了解Redis实例提供的各项功能及其相应的实现文件，以便后续深入学习。

实际上，当你掌握了以上两方面的内容之后，即使你要去了解和学习其他软件系统的代码架构，你都可以按照“先面后点”的方法来推进。也就是说，先了解目录结构与作用类别，再对应功能模块与实现文件，这样可以帮助你快速地掌握一个软件系统的代码全景。

所以，在后续的学习过程中，你要仔细跟住我的脚步，并且手边最好能备着一台可以方便查看源码的电脑，针对我提到的源码文件、关键模块或是代码运行，一定要实际阅读一遍或是实操一遍，这样你就能对Redis的代码架构建立更深刻的认识。

好了，话不多说，下面我们就一起来完成Redis代码之旅的攻略吧。

Redis目录结构

首先，我们来了解下Redis的目录结构。

为什么要从目录结构开始了解呢？其实，这是我自己**阅读代码的一个小诀窍**：在学习一个大型系统软件的代码时，要想快速地对代码有个初步认知，了解系统源码的整体目录结构就是一个行之有效的方法。这是因为，系统开发者通常会把完成同一或相近功能的代码文件，按目录结构来组织。能划归到同一个目录下的代码文件，一般都是具有相近功能目标的。

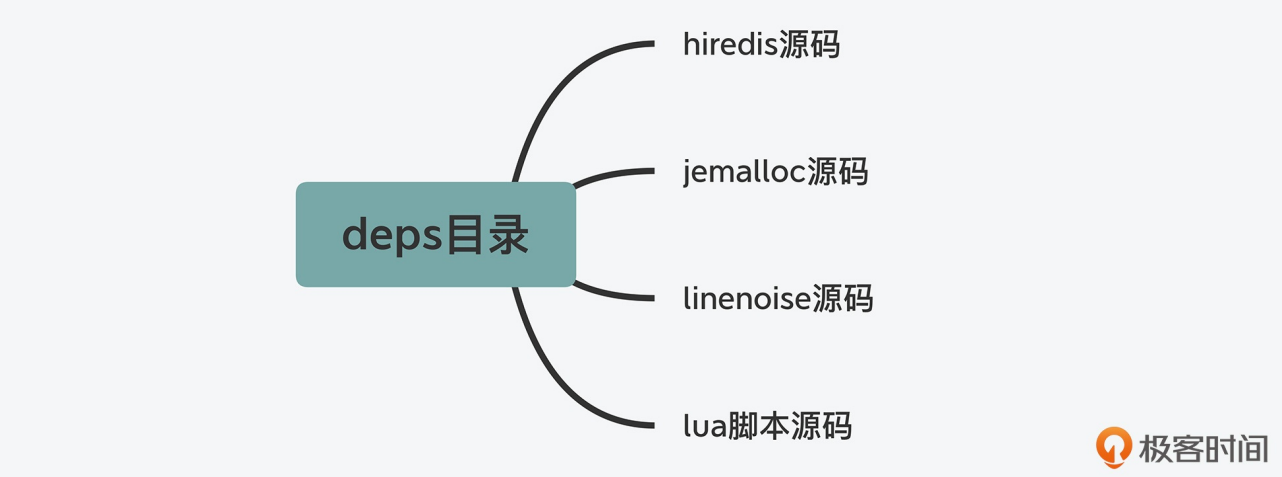
所以，从代码的目录结构开始学习，可以让我们从目录命名和目录层次结构中，直接了解到一个系统的主要组成部分。

那么对于Redis来说，在它的源码总目录下，一共包含了deps、src、tests、utils四个子目录，这四个子目录分别对应了Redis中发挥不同作用的代码，下面我们具体来看看。

deps目录

这个目录主要**包含了Redis依赖的第三方代码库**，包括Redis的C语言版本客户端代码hiredis、jemalloc内存分配器代码、readline功能的替代代码linenoise，以及lua脚本代码。

这部分代码的一个显著特点，就是**它们可以独立于Redis src目录下的功能源码进行编译**，也就是说，它们可以独立于Redis存在和发展。下面这张图显示了deps目录下的子目录内容。



那么，为什么在Redis源码结构中会有第三方代码库目录呢？其实主要有两方面的原因。

一方面，Redis作为一个用C语言写的用户态程序，它的不少功能是依赖于标准的glibc库提供的，比如内存分配、行读写（readline）、文件读写、子进程/线程创建等。但是，glibc库提供的某些功能实现，效率并不高。

我举个简单的例子，glibc库中实现的内存分配器的性能就不是很高，它的内存碎片化情况也比较严重。因此为了避免对系统性能产生影响，Redis使用了jemalloc库替换了glibc库的内存分配器。可是，jemalloc库本身又不属于Redis系统自身的功能，把它和Redis功能源码放在一个目录下并不合适，所以，Redis使用了专门的deps目录来保存这部分代码。

另一方面，有些功能是Redis运行所需要的，但是这部分功能又会独立于Redis进行开发和演进。这种类型最为典型的功能代码，就是Redis的客户端代码。

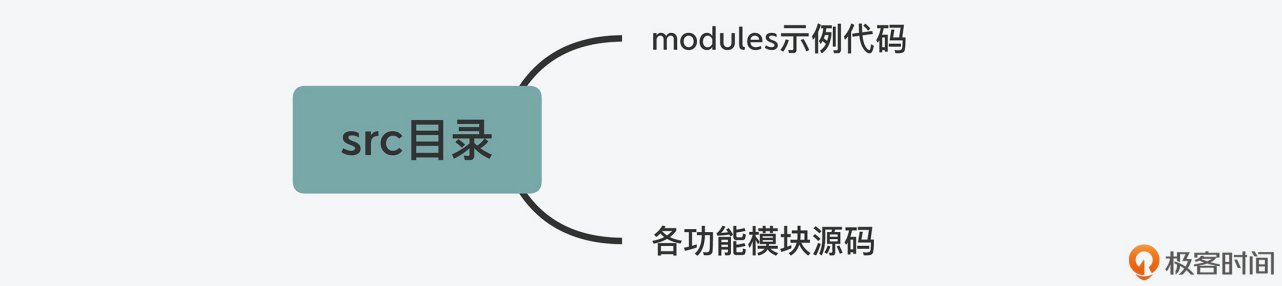
Redis作为Client-Server架构的系统，访问Redis离不开客户端的支撑。此外，Redis自身功能中的命令行redis-cli、基准测试程序redis-benchmark以及哨兵，都需要用到客户端来访问Redis实例。

不过你应该也清楚，针对客户端的开发，只要保证客户端和实例交互的过程满足RESP协议就行，客户端和实例的功能可以各自迭代演进。所以在Redis源码结构中，C语言版本的客户端hiredis，就被放到了deps目录中，以便开发人员自行开发和改进客户端功能。

好了，总而言之，对于deps目录来说，你只需要记住它主要存放了三类代码：一是Redis依赖的、实现更加高效的功能库，如内存分配；二是独立于Redis开发演进的代码，如客户端；三是lua脚本代码。后续你在学习这些功能的设计实现时，就可以在deps目录找到它们。

src目录

这个目录里面**包含了Redis所有功能模块的代码文件，也是Redis源码的重要组成部分**。同样，我们先来看下src目录下的子目录结构。



我们会发现，src目录下只有一个modules子目录，其中包含了一个实现Redis module的示例代码。剩余的源码文件都是在src目录下，没有再分下一级子目录。

因为Redis的功能模块实现是典型的C语言风格，不同功能模块之间不再设置目录分隔，而是通过头文件包含来相互调用。这样的代码风格在基于C语言开发的系统软件中，也比较常见，比如Memcached的源码文件也是在同一级目录下。

所以，当你使用C语言来开发软件系统时，就可以参考Redis的功能源码结构，用一个扁平的目录组织所有的源码文件，这样模块相互间的引用也会很方便。

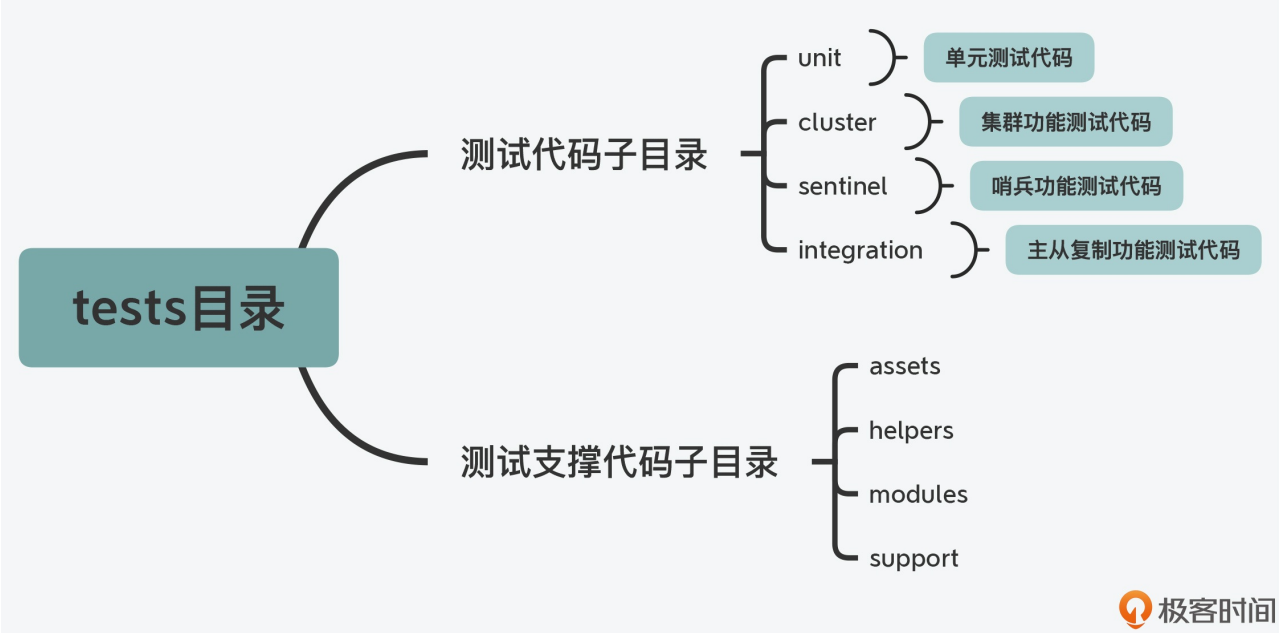
tests目录

在软件产品的开发过程中，除了第三方依赖库和功能模块源码以外，我们通常还需要在系统源码中，添加用于功能模块测试和单元测试的代码。而在Redis的代码目录中，就将这部分代码用一个tests目录统一管理了起来。

Redis实现的测试代码可以分成四部分，分别是**单元测试**（对应unit子目录），**Redis Cluster功能测试**（对应cluster子目录）、**哨兵功能测试**（对应sentinel子目录）、**主从复制功能测试**（对应integration子目录）。这些子目录中的测试代码使用了Tcl语言（通用的脚本语言）进行编写，主要目的就是方便进行测试。

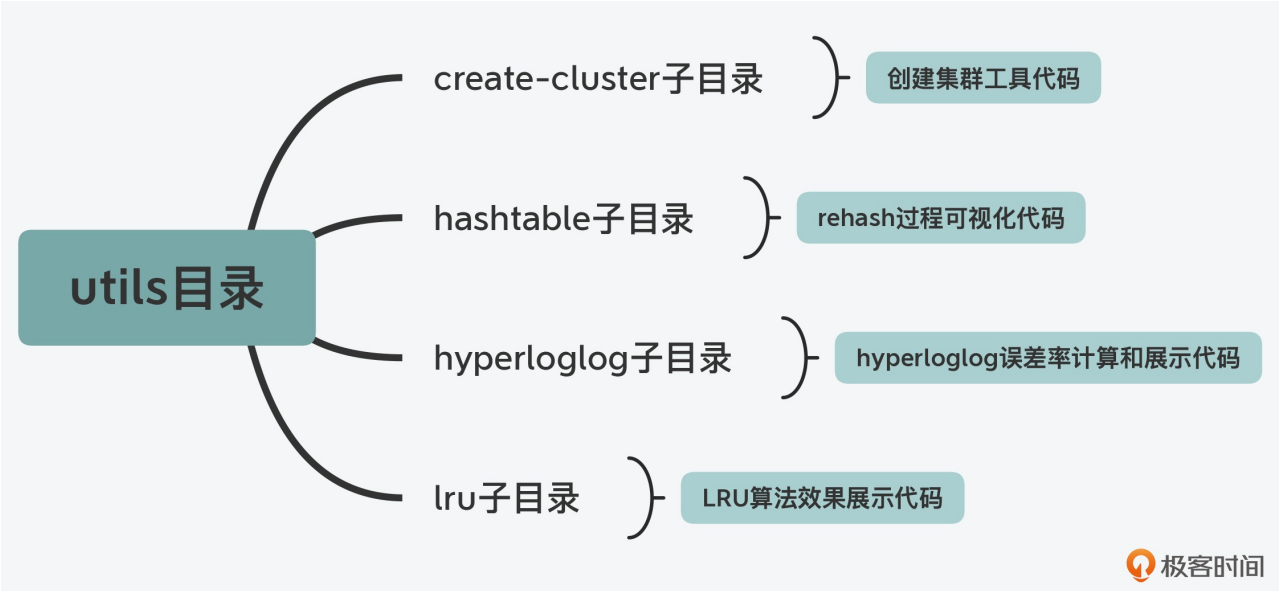
另外，每一部分的测试都是一个测试集合，覆盖了相应功能模块中的多项子功能测试。比如，在单元测试的目录中，我们可以看到有针对过期key的测试（expire.tcl）、惰性删除的测试（lazyfree.tcl），以及不同数据类型操作的测试（type子目录）等。而在Redis Cluster功能测试的目录中，我们可以看到有针对故障切换的测试（failover.tcl）、副本迁移的测试（replica-migration.tcl）等。

不过在tests目录中，除了有针对特定功能模块的测试代码外，还有一些代码是**用来支撑测试功能**的，这些代码在assets、helpers、modules、support四个目录中。这里我画了这张图，展示了tests目录下的代码结构和层次，你可以参考下。



utils目录

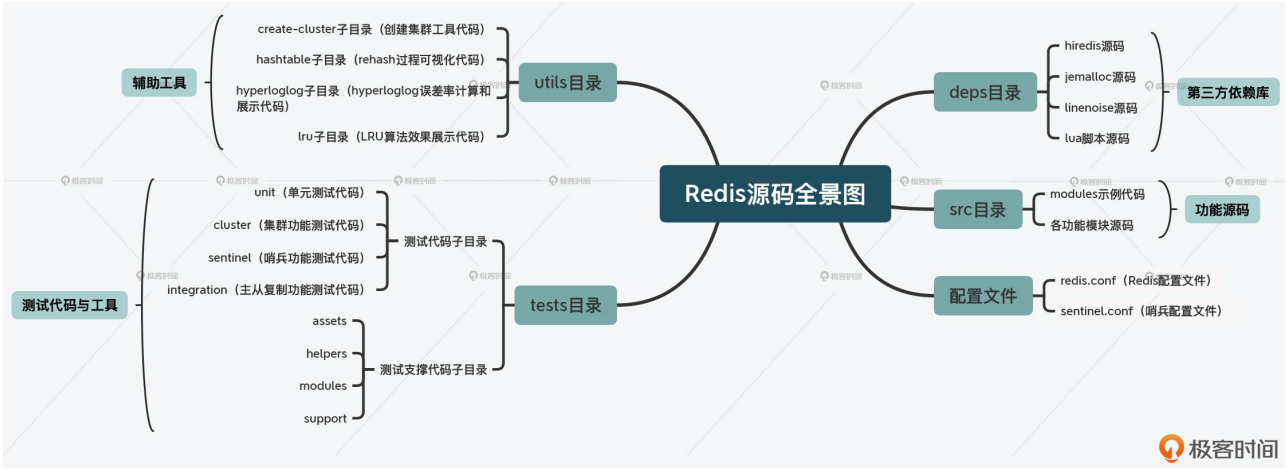
在Redis开发过程中，还有一些功能属于辅助性功能，包括用于创建Redis Cluster的脚本、用于测试LRU算法效果的程序，以及可视化rehash过程的程序。在Redis代码结构中，这些功能代码都被归类到了utils目录中统一管理。下图展示了utils目录下的主要子目录，你可以看下。



所以，当我们在开发系统时，就可以学习Redis的代码结构，也把和系统相关的辅助性功能划归到utils目录中统一管理。

好，除了deps、src、tests、utils四个子目录以外，Redis源码总目录下其实还包含了两个重要的配置文件，一个是**Redis实例的配置文件redis.conf**，另一个是**哨兵的配置文件sentinel.conf**。当你需要查找或修改Redis实例或哨兵的配置时，就可以直接定位到源码总目录下。

最后呢，你也可以再次整体回顾下Redis源码的总体结构层次，如下图所示。



好，在了解了Redis的代码目录和层次以后，接下来，我们还需要重点学习下功能模块的源码文件（即src目录下的文件内容），这有助于我们在后续课程中学习Redis的相关设计思想时，能够快速找到对应的源码文件。

Redis功能模块与源码对应

Redis代码结构中的src目录，包含了实现功能模块的123个代码文件。在这123个代码文件中，对于某个功能来说，一般包括了实现该功能的**C语言文件（.c文件）**和对应的**头文件（.h文件）**。比如，dict.c和dict.h就是用于实现哈希表的C文件和头文件。

> 注意：在课程中，如果没有特殊说明，我介绍的源码都是基于Redis 5.0.8版本的。

那么，我们该如何将这123个文件和Redis的主要功能对应上呢？

其实，**Redis代码文件的命名非常规范，文件名中就体现了该文件实现的主要功能。**比如，对于rdb.h和rdb.c这两个代码文件来说，从文件名上，你就可以看出来它们是实现内存快照RDB的对应代码。

所以这里，为了让你能快速定位源码，我分别按照Redis的服务器实例、数据库操作、可靠性和可扩展性保证、辅助功能四个维度，把Redis功能源码梳理成了四条代码路径。你可以根据自己想要了解的功能维度，对应地学习相关代码。

服务器实例

首先我们知道，Redis在运行时是一个网络服务器实例，因此相应地就需要有代码实现服务器实例的初始化和主体控制流程，而这是由server.h/server.c实现的，Redis整个代码的main入口函数也是在server.c中。**如果你想了解Redis是如何开始运行的，那么就可以从server.c的main函数开始看起。**

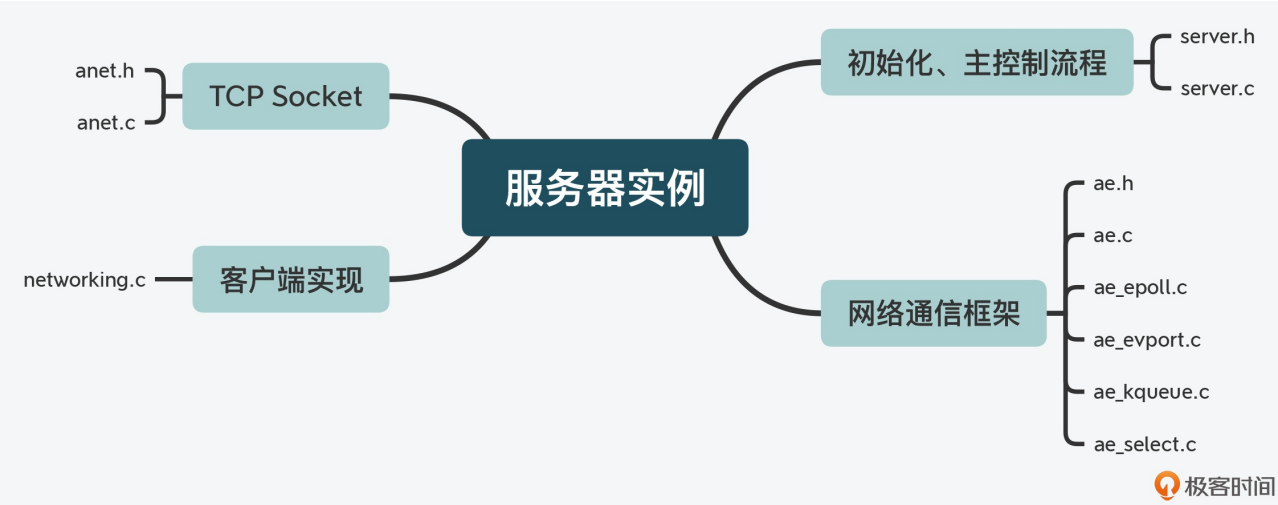
当然，对于一个网络服务器来说，它还需要提供网络通信功能。Redis使用了**基于事件驱动机制的网络通信框架**，涉及的代码文件包括ae.h/ae.c，ae_epoll.c，ae_evport.c，ae_kqueue.c，ae_select.c。关于事件驱动框架的具体设计思路与实现方法，我会在第10讲中给你详细介绍。

而除了事件驱动网络框架以外，与网络通信相关的功能还包括**底层TCP网络通信**和**客户端实现**。

Redis对TCP网络通信的Socket连接、设置等操作进行了封装，这些封装后的函数实现在anet.h/anet.c中。这些函数在Redis Cluster创建和主从复制的过程中，会被调用并用于建立TCP连接。

除此之外，客户端在Redis的运行过程中也会被广泛使用，比如实例返回读取的数据、主从复制时在主从库间传输数据、Redis Cluster的切片实例通信等，都会用到客户端。Redis将客户端的创建、消息回复等功能，实现在了networking.c文件中，如果你想了解客户端的设计与实现，可以重点看下这个代码文件。

这里我也给你总结了与服务器实例相关的功能模块及对应的代码文件，你可以看下。



那么，在了解了Redis服务器实例的主要功能代码之后，我们再从Redis内存数据库这一特性维度，来梳理下与它相关的代码文件。

数据库数据类型与操作

Redis数据库提供了丰富的键值对类型，其中包括了String、List、Hash、Set和Sorted Set这五种基本键值类型。此外，Redis还支持位图、HyperLogLog、Geo等扩展数据类型。

而为了支持这些数据类型，Redis就使用了多种数据结构来作为这些类型的底层结构。比如，String类型的底层数据结构是SDS，而Hash类型的底层数据结构包括哈希表和压缩列表。

不过，因为Redis实现的底层数据结构非常多，所以这里我把这些底层结构和它们对应的键值对类型，以及相应的代码文件列在了下表中，你可以用这张表来快速定位代码文件。

底层数据结构	对应数据类型	对应源码文件
SDS	String	sds.h/sds.c/sdsalloc.h
双向链表	List	adlist.h/adlist.c
压缩列表	List、Hash、Sorted Set	ziplist.h/ziplist.c
QuickList	List、Hash、Sorted Set	quicklist.h/quicklist.c
整数列表	Set	intset.h/intset.c
Zipmap	Hash	zipmap.h/zipmap.c
哈希表	Hash	dict.h/dict.c
HyperLogLog	HyperLogLog	hyperloglog.c
GeoHash	Geo	geo.h/geo.c, geohash.h/geohash.c, geohash_helper.h/geohash_helper.c
位图	位图	bitops.c
Stream	时序数据	stream.h/t_stream.c

除了实现了诸多的数据类型以外，Redis作为数据库，还实现了对键值对的新增、查询、修改和删除等操作接口，这部分功能是在**db.c文件**实现的。

当然，Redis作为内存数据库，其保存的数据量受限于内存大小。因此，内存的高效使用对于Redis来说就非常重要。

那么你可能就要问了：**Redis是如何优化内存使用的呢?**

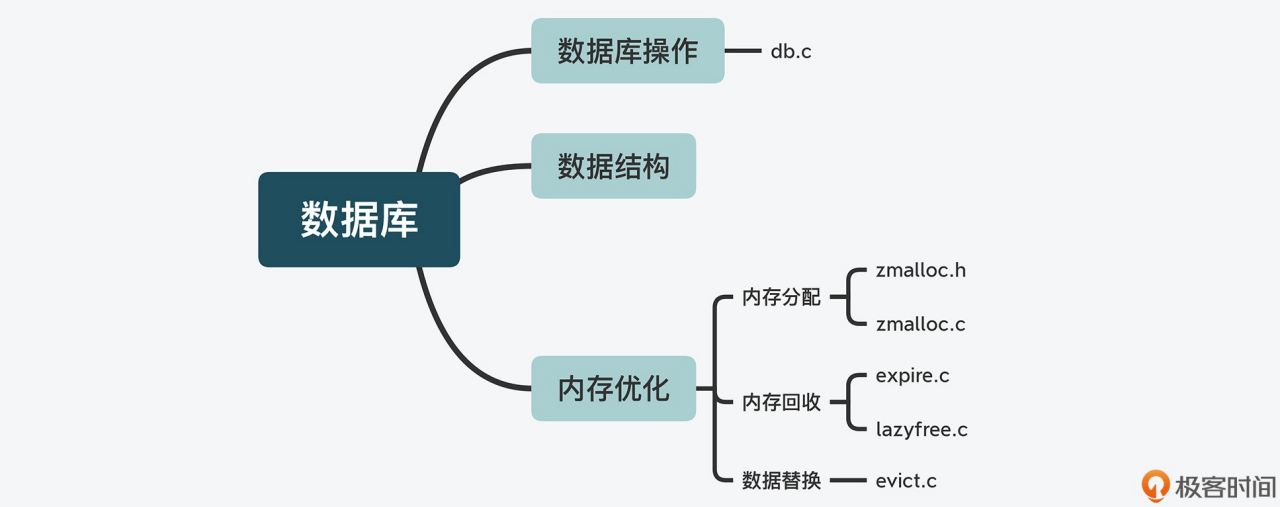
实际上，Redis是从三个方面来优化内存使用的，分别是内存分配、内存回收，以及数据替换。

首先，在**内存分配**方面，Redis支持使用不同的内存分配器，包括glibc库提供的默认分配器tcmalloc、第三方库提供的jemalloc。Redis把对内存分配器的封装实现在了zmalloc.h/zmalloc.c。

其次，在**内存回收**上，Redis支持设置过期key，并针对过期key可以使用不同删除策略，这部分代码实现在expire.c文件中。同时，为了避免大量key删除回收内存，会对系统性能产生影响，Redis在lazyfree.c中实现了异步删除的功能，所以这样，我们就可以使用后台IO线程来完成删除，以避免对Redis主线程的影响。

最后，针对**数据替换**，如果内存满了，Redis还会按照一定规则清除不需要的数据，这也是Redis可以作为缓存使用的原因。Redis实现的数据替换策略有很多种，包括LRU、LFU等经典算法。这部分的代码实现在了evict.c中。

同样，这里我也把和Redis数据库数据类型与操作相关的功能模块及代码文件，总结成了一张图，你可以看下。



高可靠性和高可扩展性

首先，虽然Redis一般是作为内存数据库来使用的，但是它也提供了可靠性保证，这主要体现在Redis可以对数据做持久化保存，并且它还实现了主从复制机制，从而可以提供故障恢复的功能。

这部分的代码实现比较集中，主要包括以下两个部分。

- **数据持久化实现**

Redis的数据持久化实现有两种方式：**内存快照RDB** 和 **AOF日志**，分别实现在了 **rdb.h/rdb.c** 和 **aof.c** 中。

注意，在使用RDB或AOF对数据库进行恢复时，RDB和AOF文件可能会因为Redis实例所在服务器宕机，而未能完整保存，进而会影响到数据库恢复。因此针对这一问题，Redis还实现了**对这两类文件的检查功能**，对应的代码文件分别是redis-check-rdb.c和redis-check-aof.c。

- **主从复制功能实现**

Redis把主从复制功能实现在了**replication.c文件**中。另外你还需要知道的是，Redis的主从集群在进行恢复时，主要是依赖于哨兵机制，而这部分功能则直接实现在了sentinel.c文件中。

其次，与Redis实现高可靠性保证的功能类似，Redis高可扩展性保证的功能，是通过**Redis Cluster**来实现的，这部分代码也非常集中，就是在**cluster.h/cluster.c代码文件**中。所以这样，我们在学习Redis Cluster的设计与实现时，就会非常方便，不用在不同的文件之间来回跳转了。

辅助功能

Redis还实现了一些用于支持系统运维的辅助功能。比如，为了便于运维人员查看分析不同操作的延迟产生来源，Redis在latency.h/latency.c中实现了操作延迟监控的功能；为了便于运维人员查找运行过慢的操作命令，Redis在slowlog.h/slowlog.c中实现了慢命令的记录功能，等等。

此外，运维人员有时还需要了解Redis的性能表现，为了支持这一目标，Redis实现了对系统进行性能评测的功能，这部分代码在redis-benchmark.c中。如果你想要了解如何对Redis开展性能测试，这个代码文件也值得一读。

小结

今天是我们了解Redis源码架构和设计思想的“热身课”，这里我们需要先明确一点，就是理解代码结构，可以为我们提供Redis功能模块的全景图，并方便我们快速查找和定位某个具体功能模块的实现源码，这样也有助于提升代码阅读的效率。

我在一开始，先给你介绍了一个**小诀窍**：通过目录命名和层次，来快速掌握一个系统软件的代码结构。而通过学习Redis的目录结构，我们也学到了一个**重要的编程规范**：在开发系统软件时，使用不同的目录对代码进行划分。

常见的目录包括保存第三方库的deps目录、保存测试用例的tests目录，以及辅助功能和工具的常用目录utils目录。按照这个规范来组织你的代码，就可以提升代码的可读性和可维护性。

另外，在学习Redis功能模块的代码结构时，面对123个代码文件，我也给你分享了一种我一直比较推崇的方法：**分门别类**。也就是说，按照一定的维度将所要学习的内容进行分类描述或总结。

在课程中，我是按照服务器实例、数据库数据类型与操作、高可靠与高可扩展保证，以及辅助功能四个维度，给你梳理了四条代码路径。这四条代码路径也基本涵盖了Redis的主要功能代码，可以方便你去有逻辑、有章法地学习掌握Redis源码，不至于遗漏重要代码。

那么在最后，我还想说一点，就是在你学习了Redis源码结构的同时，也希望你能把这个方法应用到其他的代码学习中，提高学习效率。

每课一问

Redis从4.0版本开始，能够支持后台异步执行任务，比如异步删除数据，你能在Redis功能源码中，找到实现后台任务的代码文件么？

欢迎在留言区分享你的思考和操作过程，我们一起交流讨论。如果觉得有收获的话，也欢迎你把今天的内容分享给更多的朋友。

精选留言：

- Kaito 2021-07-28 00:48:01
  重新看了一下源码目录，结合这篇文章的内容，整理了一下代码分类（忽略.h头文件），这也更清晰一些：

  数据类型：
  - String（t_string.c、sds.c、bitops.c）
  - List（t_list.c、ziplist.c）
  - Hash（t_hash.c、ziplist.c、dict.c）
  - Set（t_set.c、intset.c）
  - Sorted Set（t_zset.c、ziplist.c、dict.c）
  - HyperLogLog（hyperloglog.c）
  - Geo（geo.c、geohash.c、geohash_helper.c）
  - Stream（t_stream.c、rax.c、listpack.c）

  全局：
  - Server（server.c、anet.c）
  - Object（object.c）
  - 键值对（db.c）
  - 事件驱动（ae.c、ae_epoll.c、ae_kqueue.c、ae_evport.c、ae_select.c、networking.c）
  - 内存回收（expire.c、lazyfree.c）
  - 数据替换（evict.c）
  - 后台线程（bio.c）
  - 事务（multi.c）
  - PubSub（pubsub.c）
  - 内存分配（zmalloc.c）
  - 双向链表（adlist.c）

  高可用&集群：
  - 持久化：RDB（rdb.c、redis-check-rdb.c)、AOF（aof.c、redis-check-aof.c）
  - 主从复制（replication.c）
  - 哨兵（sentinel.c）
  - 集群（cluster.c）

  辅助功能：
  - 延迟统计（latency.c）
  - 慢日志（slowlog.c）
  - 通知（notify.c）
  - 基准性能（redis-benchmark.c）

  下面解答课后问题：

  Redis 从 4.0 版本开始，能够支持后台异步执行任务，比如异步删除数据，你能在 Redis 功能源码中，找

到实现后台任务的代码文件么？

后台任务的代码在 bio.c 中。

Redis Server 在启动时，会在 server.c 中调用 bioInit 函数，这个函数会创建 3 类后台任务（类型定义在 bio.h 中）：

#define BIO_CLOSE_FILE 0 // 后台线程关闭 fd
#define BIO_AOF_FSYNC 1 // AOF 配置为 everysec，后台线程刷盘
#define BIO_LAZY_FREE 2 // 后台线程释放 key 内存

这 3 类后台任务，已经注册好了执行固定的函数（消费者）：

- BIO_CLOSE_FILE 对应执行 close(fd)
- BIO_AOF_FSYNC 对应执行 fsync(fd)
- BIO_LAZY_FREE 根据参数不同，对应 3 个函数（freeObject/freeDatabase/freeSlowsMap）

之后，主线程凡是需要把一个任务交给后台线程处理时，就会调用 bio.c 的 bioCreateBackgroundJob 函数（相当于发布异步任务的函数），并指定该任务是上面 3 个的哪一类，把任务挂到对应类型的「链表」下（bio_jobs[type]），任务即发布成功（生产者任务完成）。

消费者从链表中拿到生产者发过来的「任务类型 + 参数」，执行上面任务对应的方法即可。当然，由于是「多线程」读写链表数据，这个过程是需要「加锁」操作的。

如果要找「异步删除数据」的逻辑，可以从 server.c 的 unlink 命令为起点，一路跟代码进去，就会看到调用了 lazyfree.c 的 dbAsyncDelete 函数，这个函数最终会调到上面提到的发布异步任务函数 bioCreateBackgroundJob，整个链条就串起来了。 [36赞]

- 悟空聊架构 2021-07-27 00:34:30

回答每课一问：
Redis 从 4.0 版本开始，能够支持后台异步执行任务，比如异步删除数据，你能在 Redis 功能源码中，找到实现后台任务的代码文件么？

我翻看了 3.0 的源码，发现 3.0 就支持后台任务了。在文件 src\bio.c 里面有一个后台任务的函数：bioProcessBackgroundJobs，支持两种后台任务：关闭文件和 AOF 文件的 fsync 也是放到后台执行的。

（fsync 就是执行命令后将命令写到日志中，提供了三种策略：Always，同步写回，Everysec，每秒写回，No，操作系统控制的写回。）

疑问：根据 3.0 源码，Redis 3.0 其实就已经有后台任务了，老师在文中说的 4.0 才开始支持后台任务，我没理解。

然后我又去翻了下 4.0 的源码，增加一种后台任务：BIO_LAZY_FREE。

当任务类型等于 BIO_LAZY_FREE 时，针对不同的传参，可以释放对象、数据库、跳跃表。

对于释放可以稍微说一下，释放的源码在这个文件里面：\src\lazyfree.c，相对 3.0 来说，这个文件是新增加的。

关于对象的释放，我们可以联想到 Java 的垃圾回收算法：可达性分析算法，但是 Redis 的垃圾回收算法

用的是引用计数算法，另外 PHP 的垃圾回收算法用的也是引用计数（扩展下：用了多色标记的方式，来识别垃圾，详细参考这里：https://mp.weixin.qq.com/s/n6PGIgfZ8vXUZ1rkU5Otog），所以别再说引用计数不能用做垃圾回收了哦。

而对于 Redis 释放对象来说，会减少引用的次数，调用的是这个函数：decrRefCount(o); 根据函数的名字也容易理解。

吐槽下：Github 上下载源码总是下载失败，为了其他同学们方便下载，我整理了多套源码的下载地址，都是国内的网盘链接，只有几MB 大小，下载比较快的。

http://www.passjava.cn/#/12.Redis/00.DownloadRedis [5赞]

- 可怜大灰狼 2021-07-26 18:13:40
我的第一反应应该是从unlink命令入手查找。首先肯定是server.c中redisCommandTable[]中的unlinkCommand，找到了lazyfree.c中dbAsyncDelete方法，然后找到了bio.c中bioCreateBackgroundJob方法，很显然bio.h中加了一种后台IO任务类型：BIO_LAZY_FREE=2。我记得我看3.0代码还只有BIO_CLOSE_FILE和BIO_AOF_FSYNC [4赞]

- Darren 2021-07-26 17:58:32
bio.c
在5.x的源码中，后台异步执行又3个子线程
#define BIO_NUM_OPS 3
#define BIO_CLOSE_FILE 0 /* Deferred close(2) syscall. */
#define BIO_AOF_FSYNC 1 /* Deferred AOF fsync. */
#define BIO_LAZY_FREE 2 /* Deferred objects freeing. */

bioInit方法中通过pthread_create创建BIO_NUM_OPS子线程，不同线程的任务在static list *bio_jobs[BIO_NUM_OPS]中存储。 [4赞]

- lison 2021-07-26 19:49:29
老师，有幸听了您的集训班，想咨询下 后续章节是否有具体版本和编译工具的介绍，后续好和您保持同步 [2赞]

- 小五 2021-07-27 13:12:02
1 Redis 支持 3 大类型的后台任务，它们定义在 bio.h 文件中：
/* Background job opcodes 后台作业操作码
* 1 处理关闭文件
* 2 AOF 异步刷盘
* 3 lazyfree
*/
#define BIO_CLOSE_FILE 0 /* Deferred close(2) syscall. */
#define BIO_AOF_FSYNC 1 /* Deferred AOF fsync. */
#define BIO_LAZY_FREE 2 /* Deferred objects freeing. */

在 Redis 服务器启动时，会创建以上三类后台线程，然后阻塞等待任务的到来。处理关闭文件和 AOF 异步刷盘异步任务比较好理解，lazyfree 类型的异步任务场景就比较多了，比如下面几种情况：
1）删除数据：配置 lazyfree_lazy_user_del ，使用 unlink , 都可能将删除封装成任务放到 bio_jobs 任务队列中
2) 定期删除时，如果配置 lazyfree_lazy_expire ，那么可能将删除封装成任务放到 bio_jobs 任务队列中

2 后台创建的以上三种 bio 后台线程会不断轮询 bio_jobs 任务队列中的任务，并分门别类的处理对应的任务。逻辑操作定义在 bio.c 文件中

3 在 Redis 6.0 中增加了 io 多线程，在 networking.c 中定义了 io 线程的任务队列，以及创建 io_threads_num 个 io 线程，这些 io 线程会不断轮询 IO 读写任务。 [1赞]

- Kang 2021-07-26 18:24:21
请问下老师，各个源码目录的作用是从那里获取到的，mysql的话也会有相应解释吗 [1赞]

- 陌 2021-07-29 19:54:08
关于作业，如果一开始对 Redis 源码不熟悉的话，我们可以借用 GDB 工具来回答 Redis 有哪些后台任务:

1) 添加 -g 的编码参数，向编译文件中添加调试信息，以便使用 GDB：

make CFLAGS="-g -O0"

2) cd src && gdb redis-server

3) 在 aeMain 函数处打一个断点，然后再使得程序运行至此处:
break aeMain
run

4) 查看线程信息:
info threads

这时候我们就能够看到 4 个线程的相关信息，分别是 redis-server、bio_close_file、bio_aof_fsync、bio_lazy_free，然后就可以按线程名称再去源码中查找了。

- Ethan New 2021-07-28 17:46:41
评论区也太强了吧，瑟瑟发抖

- 冯磊 2021-07-28 08:49:08
等我复习下c语言，马上回来！

- 曾轼麟 2021-07-27 18:13:22
以unlink为例子，其实是有几个步骤的：
1、首先在redis启动的时候会调用bioInit初始化异步执行的线程，等待注册任务执行（截至到6.x版本目前只初始化三个线程，对应三种异步类型，每个线程只处理指定类型的任务，概念上和线程池不一样）
2、当执行unlink的时候，先删除key在字典上的指针，如果不需要异步直接使用dictFreeUnlinkedEntry释放内存，如果需要直接跳过释放步骤
3、通过lazyfreeGetFreeEffort函数计算删除key的代价，如果代价超过阈值则注册一个bioCreateLazyFreeJob 并标记类型为BIO_LAZY_FREE等待异步执行
4、异步线程执行job释放内存

实际代码流程如下所示：
1、unlink主流程（允许异步）： unlinkCommand -> delGenericCommand -> dbAsyncDelete -> dictUnlink -> 注册bioCreateBackgroundJob
2、unlink主流程（不用异步）： unlinkCommand -> delGenericCommand -> dbAsyncDelete -> dictUn

link -> dictFreeUnlinkedEntry(直接释放内存)

对应的代码在lazyfree.c和bio.c文件中，此外异步执行一共支持三种类型 BIO_CLOSE_FILE（异步关闭文件） BIO_AOF_FSYNC（AOF异步同步） BIO_LAZY_FREE（懒删除）

- sljoai 2021-07-27 13:04:54
  没什么C语言开发经验，请问一下大家都是采用哪种开发工具来查看及调试Redis源码的?

- Leven 2021-07-26 22:47:25
  请问老师，没有c语言基础适合吗

- 春华秋实 2021-07-26 19:16:33
  一讲这么长，必须买