01-基本架构：一个键值数据库包含什么？

你好，我是蒋德钧。

我们知道，Redis是典型的键值数据库，所以今天，我准备手把手地带你构建一个简单的键值数据库。为啥要这么做呢？

还记得我在开篇词说过吗？Redis本身比较复杂，如果我们一上来就直接研究一个个具体的技术点，比如“单线程”“缓存”等，虽然可以直接学习到具体的内容，甚至立马就能解决一些小问题，但是这样学，很容易迷失在细枝末节里。

从我自己的经验来看，更好的学习方式就是先建立起“**系统观**”。这也就是说，如果我们想要深入理解和优化Redis，就必须要对它的总体架构和关键模块有一个全局的认知，然后再深入到具体的技术点。这也是我们这门课坚持的一种讲课方式。

我相信，经过这样一个过程，我们在实践中定位和解决问题时，就会轻松很多，而且你还可以把这个学习方式迁移到其他的学习活动上。我希望你能彻底掌握这个学习思路，让自己的学习、工作效率更高。

说远了，还是回到我们今天的课程主题上。今天，在构造这个简单的键值数据库时，我们只需要关注整体架构和核心模块。这就相当于医学上在正式解剖人体之前，会先解剖一只小白鼠。我们通过剖析这个最简单的键值数据库，来迅速抓住学习和调优Redis的关键。

我把这个简单的键值数据库称为SimpleKV。需要注意的是，GitHub上也有一个名为SimpleKV的项目，这跟我说的SimpleKV不是一回事，我说的只是一个具有关键组件的键值数据库架构。

好了，你是不是已经准备好了，那我们就一起来构造SimpleKV吧。

开始构造SimpleKV时，首先就要考虑里面可以存什么样的数据，对数据可以做什么样的操作，也就是数据模型和操作接口。它们看似简单，实际上却是我们理解Redis经常被用于缓存、秒杀、分布式锁等场景的重要基础。

理解了数据模型，你就会明白，为什么在有些场景下，原先使用关系型数据库保存的数据，也可以用键值数据库保存。例如，用户信息（用户ID、姓名、年龄、性别等）通常用关系型数据库保存，在这个场景下，一个用户ID对应一个用户信息集合，这就是键值数据库的一种数据模型，它同样能完成这一存储需求。

但是，如果你只知道数据模型，而不了解操作接口的话，可能就无法理解，为什么在有些场景中，使用键值数据库又不合适了。例如，同样是在上面的场景中，如果你要对多个用户的年龄计算均值，键值数据库就无法完成了。因为它只提供简单的操作接口，无法支持复杂的聚合计算。

那么，对于Redis来说，它到底能做什么，不能做什么呢？只有先搞懂它的数据模型和操作接口，我们才能真正把“这块好钢用在刀刃上”。

接下来，我们就先来看可以存哪些数据。

可以存哪些数据？

对于键值数据库而言，基本的数据模型是key-value模型。 例如，“hello”：“world”就是一个基本的KV

对，其中，“hello”是key，“world”是value。SimpleKV也不例外。在SimpleKV中，key是String类型，而value是基本数据类型，例如String、整型等。

但是，SimpleKV毕竟是一个简单的键值数据库，对于实际生产环境中的键值数据库来说，value类型还可以是复杂类型。

不同键值数据库支持的key类型一般差异不大，而value类型则有较大差别。我们在对键值数据库进行选型时，一个重要的考虑因素是**它支持的value类型**。例如，Memcached支持的value类型仅为String类型，而Redis支持的value类型包括了String、哈希表、列表、集合等。**Redis能够在实际业务场景中得到广泛的应用，就是得益于支持多样化类型的value**。

从使用的角度来说，不同value类型的实现，不仅可以支撑不同业务的数据需求，而且也隐含着不同数据结构在性能、空间效率等方面的差异，从而导致不同的value操作之间存在着差异。

只有深入地理解了这背后的原理，我们才能在选择Redis value类型和优化Redis性能时，做到游刃有余。

可以对数据做什么操作?

知道了数据模型，接下来，我们就要看它对数据的基本操作了。SimpleKV是一个简单的键值数据库，因此，基本操作无外乎增删改查。

我们先来了解下SimpleKV需要支持的3种基本操作，即PUT、GET和DELETE。

- PUT：新写入或更新一个key-value对；
- GET：根据一个key读取相应的value值；
- DELETE：根据一个key删除整个key-value对。

需要注意的是，**有些键值数据库的新写/更新操作叫SET**。新写入和更新虽然是用一个操作接口，但在实际执行时，会根据key是否存在而执行相应的新写或更新流程。

在实际的业务场景中，我们经常会碰到这种情况：查询一个用户在一段时间内的访问记录。这种操作在键值数据库中属于SCAN操作，即**根据一段key的范围返回相应的value值**。因此，**PUT/GET/DELETE/SCAN是一个键值数据库的基本操作集合**。

此外，实际业务场景通常还有更加丰富的需求，例如，在黑白名单应用中，需要判断某个用户是否存在。如果将该用户的ID作为key，那么，可以增加EXISTS操作接口，用于判断某个key是否存在。对于一个具体的键值数据库而言，你可以通过查看操作文档，了解其详细的操作接口。

当然，当一个键值数据库的value类型多样化时，就需要包含相应的操作接口。例如，Redis的value有列表类型，因此它的接口就要包括对列表value的操作。后面我也会具体介绍，不同操作对Redis访问效率的影响。

说到这儿呢，数据模型和操作接口我们就构造完成了，这是我们的基础工作。接下来呢，我们就要更进一步，考虑一个非常重要的设计问题：**键值对保存在内存还是外存?**

保存在内存的好处是读写很快，毕竟内存的访问速度一般都在百ns级别。但是，潜在的风险是一旦掉电，所

有的数据都会丢失。

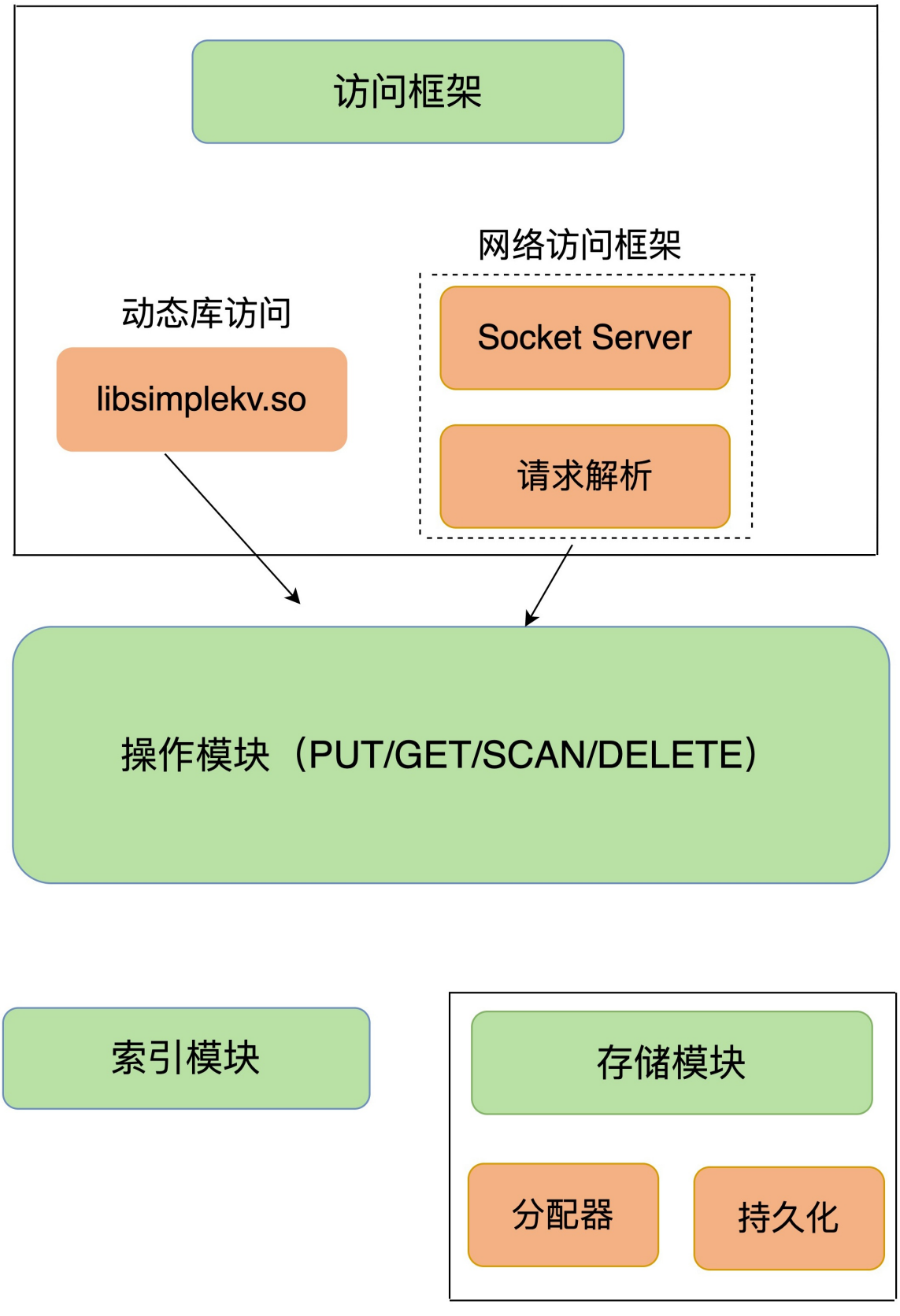
保存在外存，虽然可以避免数据丢失，但是受限于磁盘的慢速读写（通常在几ms级别），键值数据库的整体性能会被拉低。

因此，**如何进行设计选择，我们通常需要考虑键值数据库的主要应用场景**。比如，缓存场景下的数据需要能快速访问但允许丢失，那么，用于此场景的键值数据库通常采用内存保存键值数据。Memcached和Redis都是属于内存键值数据库。对于Redis而言，缓存是非常重要的一个应用场景。后面我会重点介绍Redis作为缓存使用的关键机制、优势，以及常见的优化方法。

为了和Redis保持一致，我们的SimpleKV就采用内存保存键值数据。接下来，我们来了解下SimpleKV的基本组件。

大体来说，一个键值数据库包括了**访问框架、索引模块、操作模块和存储模块**四部分（见下图）。接下来，我们就从这四个部分入手，继续构建我们的SimpleKV。

SimpleKV的基本内部架构



极客时间

采用什么访问模式？

访问模式通常有两种：一种是**通过函数库调用的方式供外部应用使用**，比如，上图中的libsimplekv.so，就是以动态链接库的形式链接到我们自己的程序中，提供键值存储功能；另一种是**通过网络框架以Socket通信的形式对外提供键值对操作**，这种形式可以提供广泛的键值存储服务。在上图中，我们可以看到，网络框架中包括Socket Server和协议解析。

不同的键值数据库服务器和客户端交互的协议并不相同，我们在对键值数据库进行二次开发、新增功能时，必须要了解和掌握键值数据库的通信协议，这样才能开发出兼容的客户端。

实际的键值数据库也基本采用上述两种方式，例如，RocksDB以动态链接库的形式使用，而Memcached和Redis则是通过网络框架访问。后面我还会给你介绍Redis现有的客户端和通信协议。

通过网络框架提供键值存储服务，一方面扩大了键值数据库的受用面，但另一方面，也给键值数据库的性能、运行模型提供了不同的设计选择，带来了一些潜在的问题。

举个例子，当客户端发送一个如下的命令后，该命令会被封装在网络包中发送给键值数据库：

```
PUT “hello” “world”
```

键值数据库网络框架接收到网络包，并按照相应的协议进行解析之后，就可以知道，客户端想写入一个键值对，并开始实际的写入流程。此时，我们会遇到一个系统设计上的问题，简单来说，就是网络连接的处理、网络请求的解析，以及数据存取的处理，是用一个线程、多个线程，还是多个进程来交互处理呢？该如何进行设计和取舍呢？我们一般把这个问题称为I/O模型设计。不同的I/O模型对键值数据库的性能和可扩展性会有不同的影响。

举个例子，如果一个线程既要处理网络连接、解析请求，又要完成数据存取，一旦某一步操作发生阻塞，整个线程就会阻塞住，这就降低了系统响应速度。如果我们采用不同线程处理不同操作，那么，某个线程被阻塞时，其他线程还能正常运行。但是，不同线程间如果需要访问共享资源，那又会产生线程竞争，也会影响系统效率，这又该怎么办呢？所以，这的确是个“两难”选择，需要我们进行精心的设计。

你可能经常听说Redis是单线程，那么，Redis又是如何做到“单线程，高性能”的呢？后面我再和你好好聊一聊。

如何定位键值对的位置？

当SimpleKV解析了客户端发来的请求，知道了要进行的键值对操作，此时，SimpleKV需要查找所要操作的键值对是否存在，这依赖于键值数据库的索引模块。**索引的作用是让键值数据库根据key找到相应value的存储位置，进而执行操作**。

索引的类型有很多，常见的有哈希表、B+树、字典树等。不同的索引结构在性能、空间消耗、并发控制等方面具有不同的特征。如果你看过其他键值数据库，就会发现，不同键值数据库采用的索引并不相同，例如，Memcached和Redis采用哈希表作为key-value索引，而RocksDB则采用跳表作为内存中key-value的索引。

一般而言，内存键值数据库（例如Redis）采用哈希表作为索引，很大一部分原因在于，其键值数据基本都

是保存在内存中的，而内存的高性能随机访问特性可以很好地与哈希表O(1)的操作复杂度相匹配。

SimpleKV的索引根据key找到value的存储位置即可。但是，和SimpleKV不同，对于Redis而言，很有意思的一点是，它的value支持多种类型，当我们通过索引找到一个key所对应的value后，仍然需要从value的复杂结构（例如集合和列表）中进一步找到我们实际需要的数据，这个操作的效率本身就依赖于它们的实现结构。

Redis采用一些常见的高效索引结构作为某些value类型的底层数据结构，这一技术路线为Redis实现高性能访问提供了良好的支撑。

不同操作的具体逻辑是怎样的？

SimpleKV的索引模块负责根据key找到相应的value的存储位置。对于不同的操作来说，找到存储位置之后，需要进一步执行的操作的具体逻辑会有所差异。SimpleKV的操作模块就实现了不同操作的具体逻辑：

- 对于GET/SCAN操作而言，此时根据value的存储位置返回value值即可；
- 对于PUT一个新的键值对数据而言，SimpleKV需要为该键值对分配内存空间；
- 对于DELETE操作，SimpleKV需要删除键值对，并释放相应的内存空间，这个过程由分配器完成。

不知道你注意到没有，对于PUT和DELETE两种操作来说，除了新写入和删除键值对，还需要分配和释放内存。这就不得不提SimpleKV的存储模块了。

如何实现重启后快速提供服务？

SimpleKV采用了常用的内存分配器glibc的malloc和free，因此，SimpleKV并不需要特别考虑内存空间的管理问题。但是，键值数据库的键值对通常大小不一，glibc的分配器在处理随机的大小内存块分配时，表现并不好。一旦保存的键值对数据规模过大，就可能会造成较严重的内存碎片问题。

因此，分配器是键值数据库中的一个关键因素。对于以内存存储为主的Redis而言，这点尤为重要。Redis的内存分配器提供了多种选择，分配效率也不一样，后面我会具体讲一讲这个问题。

SimpleKV虽然依赖于内存保存数据，提供快速访问，但是，我也希望SimpleKV重启后能快速重新提供服务，所以，我在SimpleKV的存储模块中增加了持久化功能。

不过，鉴于磁盘管理要比内存管理复杂，SimpleKV就直接采用了文件形式，将键值数据通过调用本地文件系统的操作接口保存在磁盘上。此时，SimpleKV只需要考虑何时将内存中的键值数据保存到文件中，就可以了。

一种方式是，对于每一个键值对，SimpleKV都对其进行落盘保存，这虽然让SimpleKV的数据更加可靠，但是，因为每次都要写盘，SimpleKV的性能会受到很大影响。

另一种方式是，SimpleKV只是周期性地把内存中的键值数据保存到文件中，这样可以避免频繁写盘操作的性能影响。但是，一个潜在的代价是SimpleKV的数据仍然有丢失的风险。

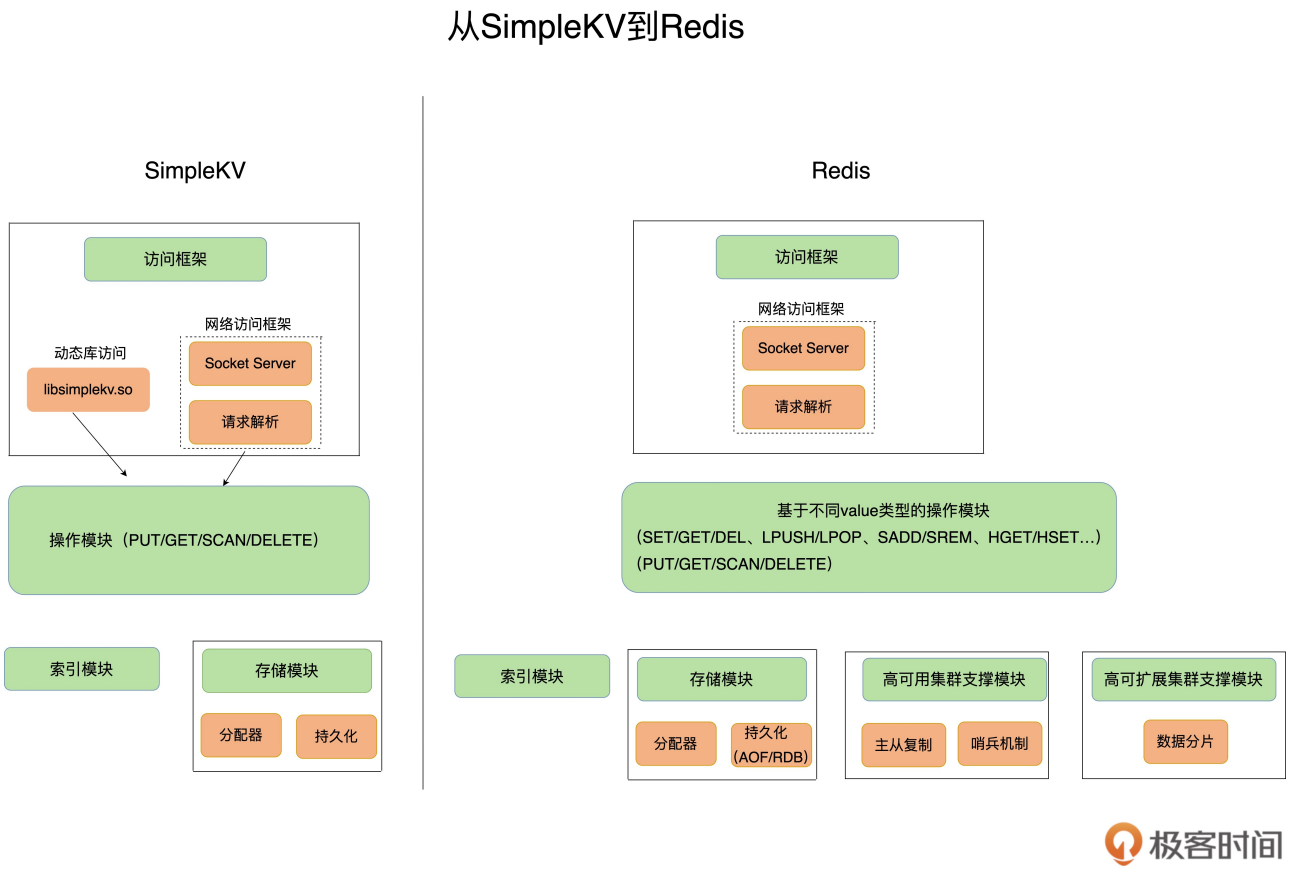
和SimpleKV一样，Redis也提供了持久化功能。不过，为了适应不同的业务场景，Redis为持久化提供了诸多的执行机制和优化改进，后面我会和你逐一介绍Redis在持久化机制中的关键设计考虑。

小结

至此，我们构造了一个简单的键值数据库SimpleKV。可以看到，前面两步我们是从应用的角度进行设计的，也就是应用视角；后面四步其实就是SimpleKV完整的内部构造，可谓是麻雀虽小，五脏俱全。

SimpleKV包含了一个键值数据库的基本组件，对这些组件有了了解之后，后面在学习Redis这个丰富版的SimpleKV时，就会轻松很多。

为了支持更加丰富的业务场景，Redis对这些组件或者功能进行了扩展，或者说是进行了精细优化，从而满足了功能和性能等方面的要求。



从这张对比图中，我们可以看到，从SimpleKV演进到Redis，有以下几个重要变化：

- Redis主要通过网络框架进行访问，而不再是动态库了，这也使得Redis可以作为一个基础性的网络服务进行访问，扩大了Redis的应用范围。
- Redis数据模型中的value类型很丰富，因此也带来了更多的操作接口，例如面向列表的LPUSH/LPOP，面向集合的SADD/SREM等。在下节课，我将和你聊聊这些value模型背后的数据结构和操作效率，以及它们对Redis性能的影响。
- Redis的持久化模块能支持两种方式：日志（AOF）和快照（RDB），这两种持久化方式具有不同的优劣势，影响到Redis的访问性能和可靠性。
- SimpleKV是个简单的单机键值数据库，但是，Redis支持高可靠集群和高可扩展集群，因此，Redis中包含了相应的集群功能支撑模块。

通过这节课SimpleKV的构建，我相信你已经对键值数据库的基本结构和重要模块有了整体认知和深刻理解，这其实也是Redis单机版的核心基础。针对刚刚提到的几点Redis的重大演进，在接下来的课程中，我会依次进行重点讲解。与此同时，我还会结合实战场景，让你不仅能够理解原理，还能真正学以致用，提升实

战能力。

每课一问

给你留个小问题：和你了解的Redis相比，你觉得，SimpleKV里面还缺少什么功能组件或模块吗？

欢迎在留言区写下你的思考和答案，我们一起交流讨论，也欢迎你把今天的内容分享给你的朋友。

精选留言：

- 樱花落花 2020-08-03 17:50:18
  依据高性能，高可用和可扩展的架构模式，SimpleKV还是主要缺乏高可用和可扩展的设计吧，单机高性能可以通过IO线程模型，数据结构内存模型等实现，其他两种没有😐