## 加餐（七）-从微博的Redis实践中，我们可以学到哪些经验？

你好，我是蒋德钧。

我们知道，微博内部的业务场景中广泛使用了Redis，积累了大量的应用和优化经验。微博有位专家曾有过一个分享，介绍了Redis在微博的优化之路，其中有很多的优秀经验。

俗话说“他山之石，可以攻玉”，学习掌握这些经验，可以帮助我们在自己的业务场景中更好地应用Redis。今天这节课，我就结合微博技术专家的分享，以及我和他们内部专家的交流，和你聊聊微博对Redis的优化以及我总结的经验。

首先，我们来看下微博业务场景对Redis的需求。这些业务需求也就是微博优化和改进Redis的出发点。

微博的业务有很多，例如让红包飞活动，粉丝数、用户数、阅读数统计，信息流聚合，音乐榜单等，同时，这些业务面临的用户体量非常大，业务使用Redis存取的数据量经常会达到TB级别。

作为直接面向终端用户的应用，微博用户的业务体验至关重要，这些都需要技术的支持。我们来总结下微博对Redis的技术需求：

- 能够提供高性能、高并发的读写访问，保证读写延迟低；
- 能够支持大容量存储；
- 可以灵活扩展，对于不同业务能进行快速扩容。

为了满足这些需求，微博对Redis做了大量的改进优化，概括来说，既有对Redis本身数据结构、工作机制的改进，也基于Redis自行研发了新功能组件，包括支持大容量存储的RedRock和实现服务化的RedisService。

接下来，我们就来具体了解下微博对Redis自身做的一些改进。

### 微博对Redis的基本改进

根据微博技术专家的分享，我们可以发现，微博对Redis的基本改进可以分成两类：避免阻塞和节省内存。

首先，针对持久化需求，他们使用了全量RDB加增量AOF复制结合的机制，这就避免了数据可靠性或性能降低的问题。当然，Redis在官方4.0版本之后，也增加了混合使用RDB和AOF的机制。

其次，在AOF日志写入刷盘时，用额外的BIO线程负责实际的刷盘工作，这可以避免AOF日志慢速刷盘阻塞主线程的问题。

再次，增加了aofnumber配置项。这个配置项可以用来设置AOF文件的数量，控制AOF写盘时的总文件量，避免了写入过多的AOF日志文件导致的磁盘写满问题。

最后，在主从库复制机制上，使用独立的复制线程进行主从库同步，避免对主线程的阻塞影响。

在节省内存方面，微博有一个典型的优化，就是定制化数据结构。

在使用Redis缓存用户的关注列表时，针对关注列表的存储，他们定制化设计了LongSet数据类型。这个数据类型是一个存储Long类型元素的集合，它的底层数据结构是一个Hash数组。在设计LongSet类型之前，微博是用Hash集合类型来保存用户关注列表，但是，Hash集合类型在保存大量数据时，内存空间消耗较大。

而且，当缓存的关注列表被从Redis中淘汰时，缓存实例需要从后台数据库中读取用户关注列表，再用HMSET写入Hash集合，在并发请求压力大的场景下，这个过程会降低缓存性能。跟Hash集合相比，LongSet类型底层使用Hash数组保存数据，既避免了Hash表较多的指针开销，节省内存空间，也可以实现快速存取。

从刚才介绍的改进工作，你可以看到，微博对Redis进行优化的出发点，和我们在前面课程中反复强调的Redis优化目标是一致的。我自己也总结了两个经验。

第一个经验是：高性能和省内存始终都是应用Redis要关注的重点，这和Redis在整个业务系统中的位置是密切相关的。

Redis通常是作为缓存在数据库层前端部署，就需要能够快速地返回结果。另外，Redis使用内存保存数据，一方面带来了访问速度快的优势，另一方面，也让我们在运维时需要特别关注内存优化。我在前面的课程里介绍了很多和性能优化、节省内存相关的内容（比如说第18～20讲），你可以重点回顾下，并且真正地在实践中应用起来。

第二个经验是，在实际应用中需要基于Redis做定制化工作或二次开发，来满足一些特殊场景的需求，就像微博定制化数据结构。不过，如果要进行定制化或二次开发，就需要了解和掌握Redis源码。所以，我建议你在掌握了Redis的基本原理和关键技术后，把阅读Redis源码作为下一个目标。这样一来，你既可以结合原理来加强对源码的理解，还可以在掌握源码后，开展新增功能或数据类型的开发工作。对于如何在Redis中新增数据类型，我在第13讲中向你介绍过，你可以再复习下。

除了这些改进工作，为了满足大容量存储需求，微博专家还在技术分享中提到，他们把RocksDB和硬盘结合使用，以扩大单实例的容量，我们来了解下。

### 微博如何应对大容量数据存储需求？

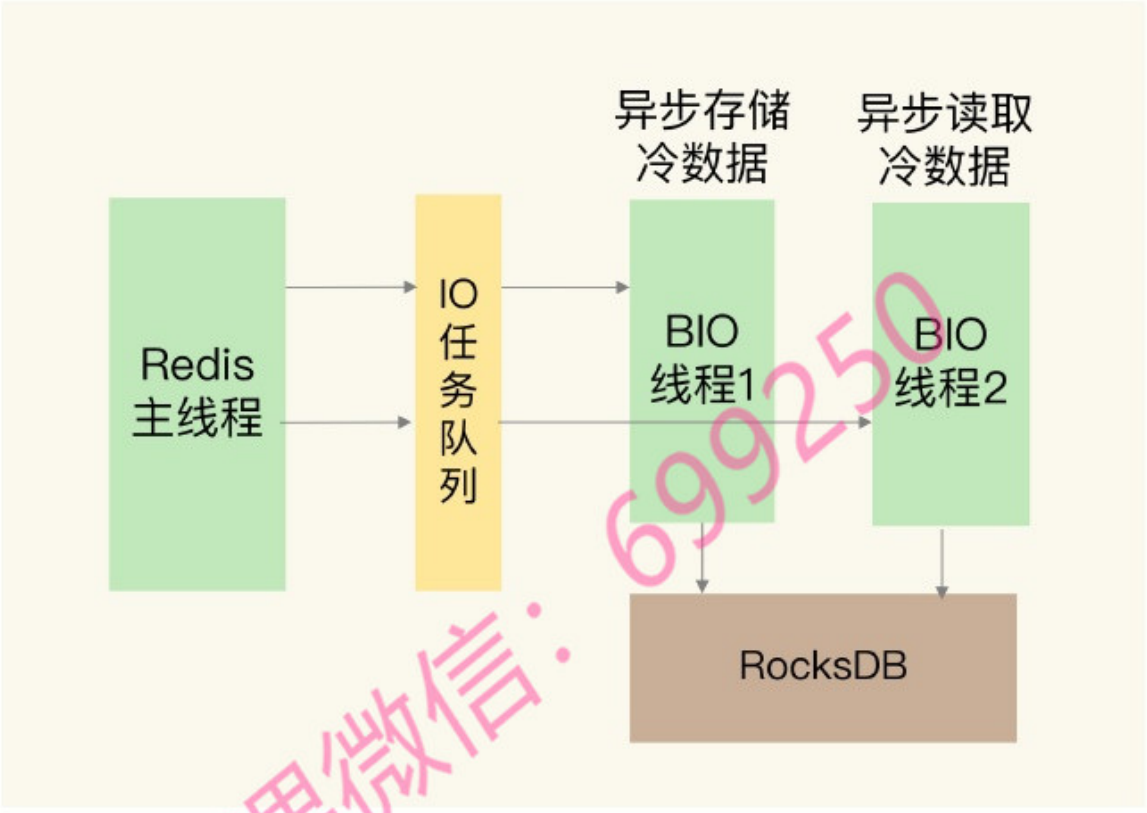
微博业务层要保存的数据经常会达到TB级别，这就需要扩大Redis实例的存储容量了。

针对这个需求，微博对数据区分冷热度，把热数据保留在Redis中，而把冷数据通过RocksDB写入底层的硬盘。

在微博的业务场景中，冷热数据是比较常见的。比如说，有些微博话题刚发生时，热度非常高，会有海量的用户访问这些话题，使用Redis服务用户请求就非常有必要。

但是，等到话题热度过了之后，访问人数就会急剧下降，这些数据就变为冷数据了。这个时候，冷数据就可以从Redis迁移到RocksDB，保存在硬盘中。这样一来，Redis实例的内存就可以节省下来保存热数据，同时，单个实例能保存的数据量就由整个硬盘的大小来决定了。

根据微博的技术分享，我画了一张他们使用RocksDB辅助Redis实现扩容的架构图：






从图中可以看到，Redis是用异步线程在RocksDB中读写数据。

读写RocksDB的延迟毕竟比不上Redis的内存访问延迟，这样做也是为了避免读写冷数据时，阻塞Redis主线程。至于冷数据在SSD上的布局和管理，都交给RocksDB负责。RocksDB目前已经比较成熟和稳定了，可以胜任Redis冷数据管理这个工作。

关于微博使用RocksDB和SSD进行扩容的优化工作，我也总结了两条经验，想和你分享一下。

首先，**实现大容量的单实例在某些业务场景下还是有需求的。**虽然我们可以使用切片集群的多实例分散保存数据，但是这种方式也会带来集群运维的开销，涉及到分布式系统的管理和维护。而且，切片集群的规模会受限，如果能增加单个实例的存储容量，那么，即使在使用较小规模的集群时，集群也能保存更多的数据。

第二个经验是，如果想实现大容量的Redis实例，**借助于SSD和RocksDB来实现是一个不错的方案**。我们在第28讲中学习的360开源的Pika，还有微博的做法，都是非常好的参考。

RocksDB可以实现快速写入数据，同时使用内存缓存部分数据，也可以提供万级别的数据读取性能。而且，当前SSD的性能提升很快，单块SSD的盘级IOPS可以达到几十万级别。这些技术结合起来，Redis就能够在提供大容量数据存储的同时，保持一定的读写性能。当你有相同的需求时，也可以把基于SSD的RocksDB应用起来保存大容量数据。

## 面向多业务线，微博如何将Redis服务化？

微博的不同业务对Redis容量的需求不一样，而且可能会随着业务的变化出现扩容和缩容的需求。

为了能够灵活地支持这些业务需求，微博对Redis进行了服务化改造（RedisService）。所谓服务化，就是指，使用Redis集群来服务不同的业务场景需求，每一个业务拥有独立的资源，相互不干扰。

同时，所有的Redis实例形成一个资源池，资源池本身也能轻松地扩容。如果有新业务上线或是旧业务下线，就可以从资源池中申请资源，或者是把不用的资源归还到资源池中。

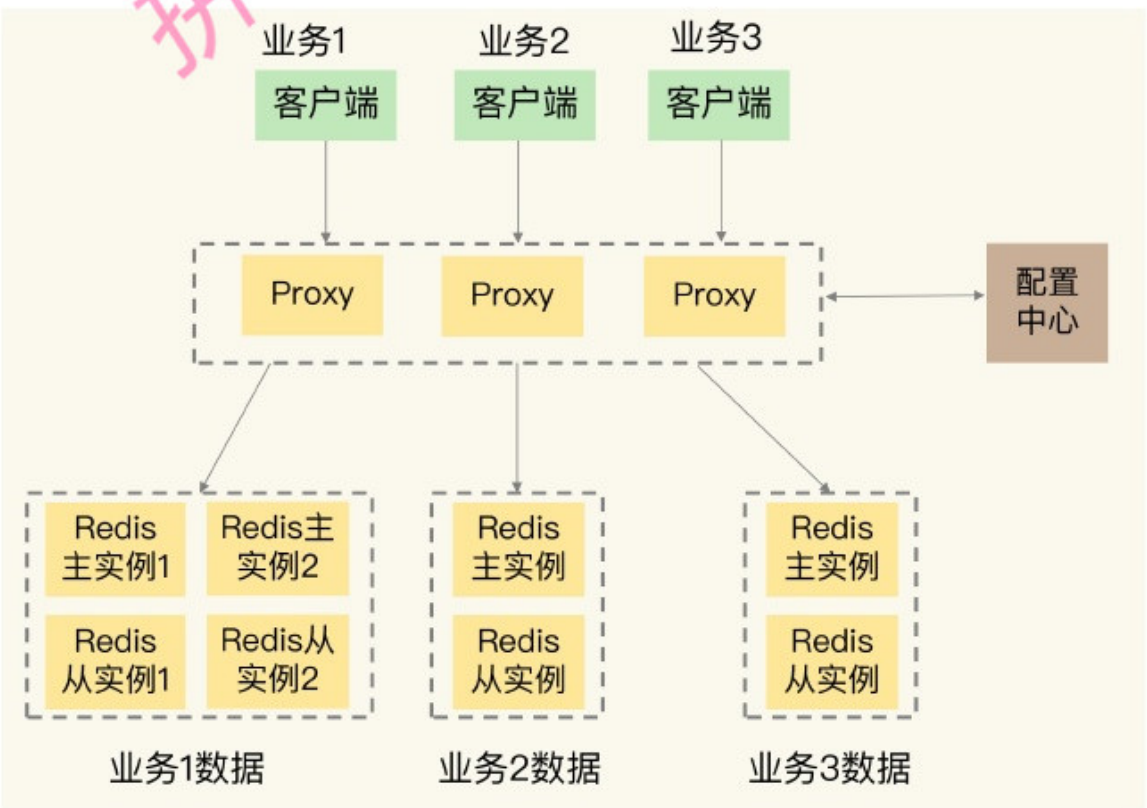
形成了Redis服务之后，不同业务线在使用Redis时就非常方便了。不用业务部门再去独立部署和运维，只要让业务应用客户端访问Redis服务集群就可以。即使业务应用的数据量增加了，也不用担心实例容量问题，服务集群本身可以自动在线扩容，来支撑业务的发展。

在Redis服务化的过程中，微博采用了类似Codis的方案，通过集群代理层来连接客户端和服务器端。从微博的公开技术资料中，可以看到，他们在代理层中实现了丰富的服务化功能支持。

- 客户端连接监听和端口自动增删。
- Redis协议解析：确定需要路由的请求，如果是非法和不支持的请求，直接返回错误。
- 请求路由：根据数据和后端实例间的映射规则，将请求路由到对应的后端实例进行处理，并将结果返回给客户端。
- 指标采集监控：采集集群运行的状态，并发送到专门的可视化组件，由这些组件进行监控处理。

此外，在服务化集群中，还有一个配置中心，它用来管理整个集群的元数据。同时，实例会按照主从模式运行，保证数据的可靠性。不同业务的数据部署到不同的实例上，相互之间保持隔离。

按照我的理解，画了一张示意图，显示了微博Redis服务化集群的架构，你可以看下。




从Redis服务化的实践中，我们可以知道，当多个业务线有共同的Redis使用需求时，提供平台级服务是一种通用做法，也就是服务化。

当把一个通用功能做成平台服务时，我们需要重点考虑的问题，包括**平台平滑扩容、多租户支持和业务数据**

**隔离、灵活的路由规则、丰富的监控功能**等。

如果要进行平台扩容，我们可以借助Codis或是Redis Cluster的方法来实现。多租户支持和业务隔离的需求是一致，我们需要通过资源隔离来实现这两个需求，也就是把不同租户或不同业务的数据分开部署，避免混用资源。对于路由规则和监控功能来说，微博目前的方案是不错的，也就是在代理层proxy中来完成这两个功能。

只有很好地实现了这些功能，一个平台服务才能高效地支撑不同业务线的需求。

## 小结

今天这节课，我们学习了微博的Redis实践，从中总结了许多经验。总结来说，微博对Redis的技术需求可以概括为3点，分别是高性能、大容量和易扩展。

为了满足这些需求，除了对Redis进行优化，微博也在自研扩展系统，包括基于RocksDB的容量扩展机制，以及服务化的RedisService集群。

最后，我还想再跟你分享一下我自己的两个感受。

第一个是关于微博做的RedisService集群，这个优化方向是大厂平台部门同学的主要工作方向。

业务纵切、平台横切是当前构建大规模系统的基本思路。所谓业务纵切，是指把不同的业务数据单独部署，这样可以避免相互之间的干扰。而平台横切是指，当不同业务线对运行平台具有相同需求时，可以统一起来，通过构建平台级集群服务来进行支撑。Redis就是典型的多个业务线都需要的基础性服务，所以将其以集群方式服务化，有助于提升业务的整体效率。

第二个是代码实践在我们成长为Redis高手过程中的重要作用。

我发现，对Redis的二次改造或开发，是大厂的一个必经之路，这和大厂业务多、需求广有密切关系。

微博做的定制化数据结构、RedRock和RedisService都是非常典型的例子。所以，如果我们想要成为Redis高手，成为大厂中的一员，那么，先原理后代码，边学习边实践，就是一个不错的方法。原理用来指导代码阅读的聚焦点，而动手实践至关重要，需要我们同时开展部署操作实践和阅读代码实践。纸上得来终觉浅，绝知此事要躬行，希望你不仅重视学习原理，还要真正地用原理来指导实践，提升自己的实战能力。

## 每课一问

按照惯例，我给你提个小问题，你在实际应用Redis时，有没有一些经典的优化改进或二次开发经验？

欢迎你在留言区聊一聊你的经验，我们一起交流讨论。如果你觉得今天的内容对你有所帮助，也欢迎你把今天的内容分享给你的朋友或同事。

## 精选留言：

- Kaito 2020-11-30 00:20:51
  在实际应用 Redis 时，你有哪些优化和二次开发的经验？

  分享一下我们对 Redis 的二次开发经验

由于我们采用的 Redis 集群方案是 Codis，我们主要对 Codis 下的 Redis Server 进行了二次开发。

我们在做服务跨机房容灾和多活时，需要在两个机房各自部署 Codis/Redis 实例，并且两个机房的实例数据需要实时同步，以支持任意机房故障时，可随时切换应用流量到另一个机房继续提供服务。

但由于两个机房之间的网络是通过专线连通的，而专线的质量和稳定性是不如同机房内的，如果使用原生的 Redis 主从复制方案，当专线长时间故障再恢复时，原生 Redis 主从复制会进行全量的数据同步。全量同步不仅对 master 节点有阻塞风险，同时也会对机房之间的专线的带宽产生巨大的压力，还会影响应用的机房流量切换。

所以我们对 Codis 下的 Redis 做了二次开发，对其增加了类似于 MySQL 的 binlog 模块，把实时的写命令通过异步线程写入到 binlog 文件中，然后开发了数据同步中间件去读取 Redis 的 binlog，向另一个机房实时同步数据，并且支持增量同步、断点续传，这样就可以兼容专线任意时间的故障，故障恢复后我们的同步中间件会从断点处继续增量同步 Redis 数据到另一个机房，避免了全量复制的风险。

同时，我们还对 Codis 进行了二次开发，在集成数据同步中间件时，兼容了 Codis 的日常运维和故障容错，例如集群内节点主从切换、故障转移、数据迁移等操作，保证了机房之间数据同步的最终一致性。

最后，我想说的是，对 Redis 进行改造，优点是可以更加适配自己的业务场景，但缺点是维护和升级成本较高，改造 Redis 相当于自己开辟了一个新的分支，公司内部需要投入人力去持续维护，需要平衡投入产出比。如果没有特别复杂的需求，一般使用官方版本即可，好处是可以第一时间使用最新版本的特性。[17赞]



- 那时刻 2020-11-30 10:22:32
  请问老师，在冷热数据处理的图例中，异步读取冷数据，是直接从IO队列读取，这是异步处理线程发送读取命令后，然后从IO队列读取么？
  另外，如果需要读冷数据，是把冷数据再放回redis变成热数据么？ [1赞]

- yyl 2020-11-30 18:41:49
  一个机房作为另一个机房的热备？

- yyl 2020-11-30 18:38:23
  问题：微博采用的集群部署方案，不是Redis cluster？是自研的吗？

- yyl 2020-11-30 18:30:32
  问题：冷热数据的迁移，是怎么操作的？由后台根据访问量自行迁移吗？