06 | 如何应对热点数据的查询?

在“04 讲和 05 讲”里，我们介绍了基于 Binlog 实现的全量缓存的读服务，以及如何实现一个低延迟、可扩展的同步架构。通过这两讲的学习，你可以构建出一个无毛刺且平均性能在 100ms 以内的读接口。对缓存进行分布式部署后，抗住秒级百万的 QPS 毫无压力。不管是在面试还是在实战中，关于“如何架构一个高性能的读服务”，我相信你都能够轻松应对。

但上述的“百万 QPS”有一个非常重要的限制条件，即这百万的 QPS 都是分属于不同用户的。我们先不讨论是否可能，试想一下如果这百万 QPS 都属于同一个用户，系统还扛得住吗?

如果采用前两讲的架构，必然抗扛不住的！因此本讲将站在一个全新的视角，带你分析此架构待改善的方向，并探寻新的架构优化方案来应对百万 QPS的流量。

为什么扛不住相同用户百万的流量

当百万的 QPS 属于不同用户时，因缓存是集群化的，所有到达业务后台的请求会根据一定路由规则（如 Hash），分散到请求缓存集群中的某一个节点，具体架构如下图 1 所示：

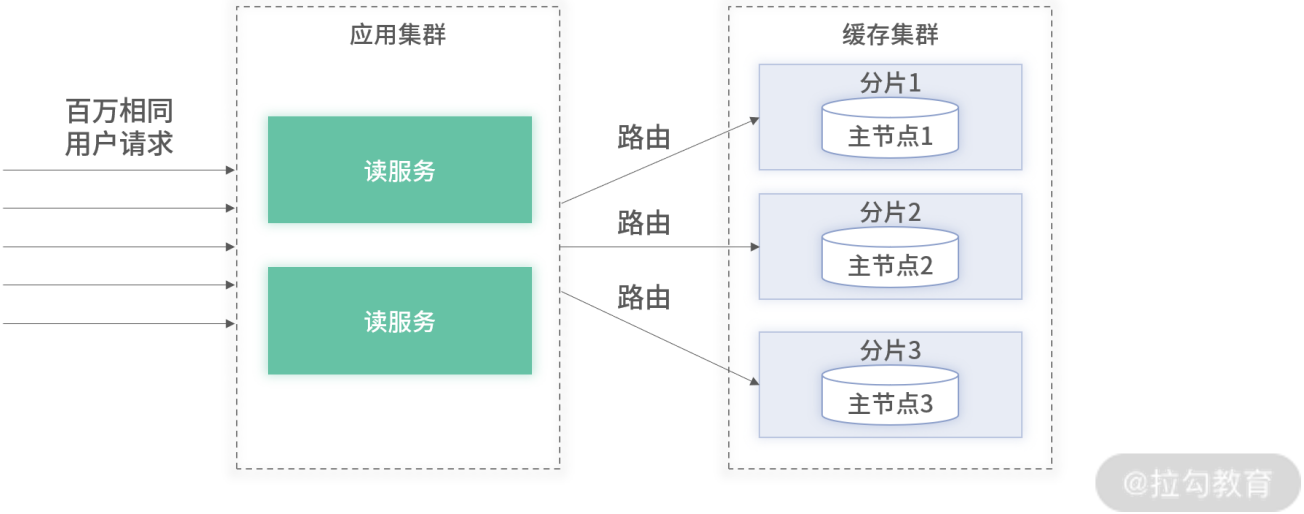

图 1：百万请求属于不同用户的架构图

假设一个节点最大能够支撑 10W QPS，我们只需要在集群中部署 10 台节点即可支持百万流量。但当百万 QPS 都属于同一用户时，即使缓存是集群化的，同一个用户的请求都会被路由至集群中的某一个节点，整体架构如图 2 所示：

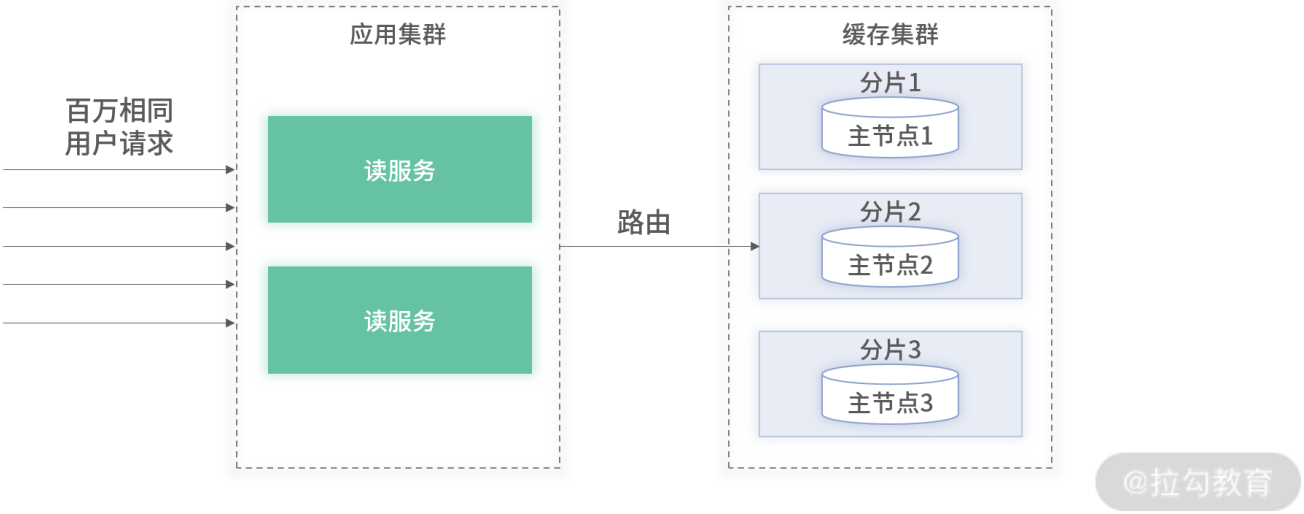

图 2：百万请求属于相同用户的架构图

即使此节点的机器配置非常好，当前能够支持住百万 QPS。但随着流量上涨，它也无法满足未来的流量诉求。原因有 2 点：

1. 单台机器无法无限升级；
2. 缓存程序本身也是有性能上限的。

此类并发次数非常大、数据完全相同的请求称为热点查询。类似的场景还有热点写，区别在于它在请求后会对数据进行写入或变更，如何应对热点写，我将在“**16 讲**”进行详细讲解。

有哪些热点查询场景

除了我在本讲开始介绍的，一个用户并发百万次请求产生热查询的案例外，在实际的日常生活中，会导致热查询的案例也非常多。

1. 微博热点吃瓜事件，百万用户同一时间查询某条微博内容。对此条微博的查询就是热查询。
2. 电商里的秒杀或者低价薅羊毛活动，为了第一时间抢到心仪的商品，成百上千万的用户会不断地刷新商品页面，等待秒杀倒计时。对此商品的查看就是热查询。

主从复制进行垂直扩容

虽然单机的机器配置和程序的性能是有上限的，但我们可以利用节点间的主从复制功能来进行节点间的扩容。主从复制开启后，一个主节点可以挂一至多个从。升级后的架构如下图 3 所示（方案解读在架构图中）：

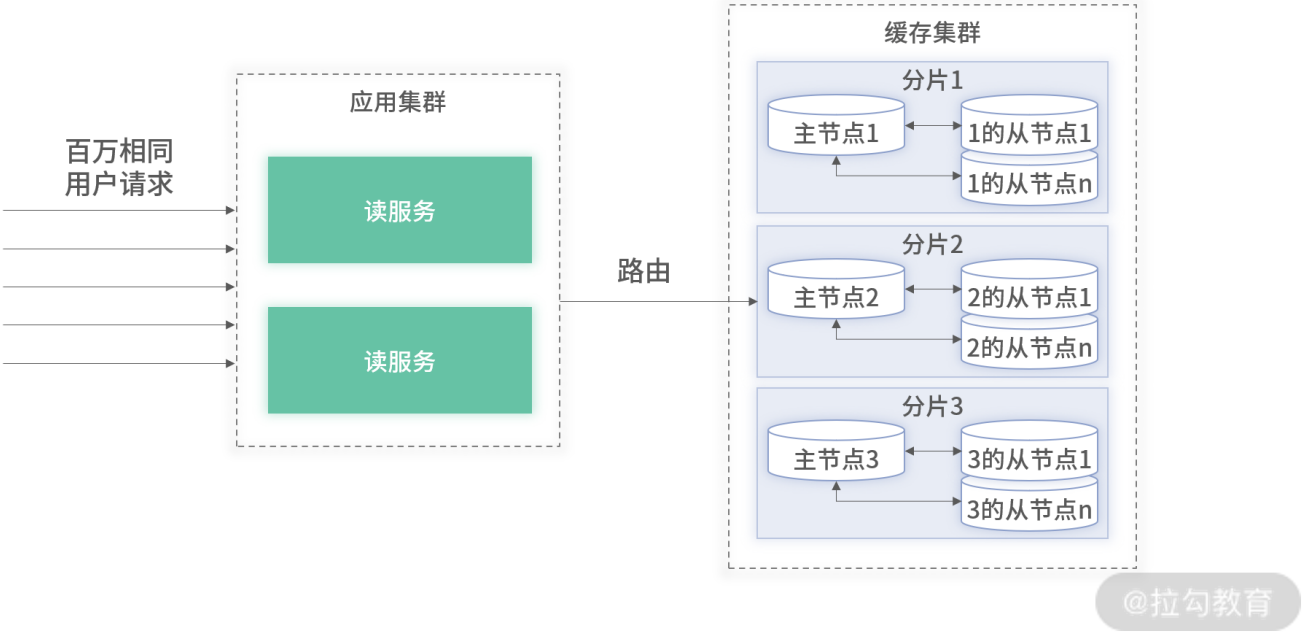

图 3：主从复制应对热点的架构图

在查询时，将应用内的缓存客户端开启主从随机读。此时，包含一个从的分片的并发能力，可以提升至原来的一倍。**随着从节点的增加，单分片的并发性能会不断翻倍**。这对于所有请求只会命中某一个固定单分片的热点查询能够很好地应对。

但此方案存在一个较大的问题，就是浪费资源。

主从复制除了有应对热点的功能，另外一个主要作用是为了高可用。当集群中的某一个主节点发生故障后，集群高可用模块会自动对该节点进行故障迁移，从该节点所属分片里选举一个从节点为主节点。为了高可用模块在故障转移时的逻辑能够简单清晰并做到统一，会将集群的从节点数量设置为相同数量。

相同从节点数量也带来了较大的资源浪费。为了应对热点查询，需要不断扩容从分片。但热点查询只会命中其中一个分片，这就导致所有其他分片的从节点全部浪费了。为了节约资源，可以对高可用模块进行改造，不强制所有分片的从节点必须相同，但这个代价也是非常高昂的。另外，热点查询很多时候是随时出现的，并不能提前预测，所以提前扩容某一个分片意义并不大。

总的来说，主从复制能够解决一定流量的热点查询且实施起来较简单。但不具备扩展性，在应对更大流量的热点时会有些吃力。

利用应用内的前置缓存

热点查询是对相同的数据进行不断重复查询的一种场景。特点是次数多，但需要存储的数据少，因为数据都是相同的。

针对此类业务特性，我们可以将热点数据前置缓存在应用程序内来应对热点查询，并解决前一小节里主从复制方案的扩展性问题。使用了前置缓存的架构如下图 4 所示：

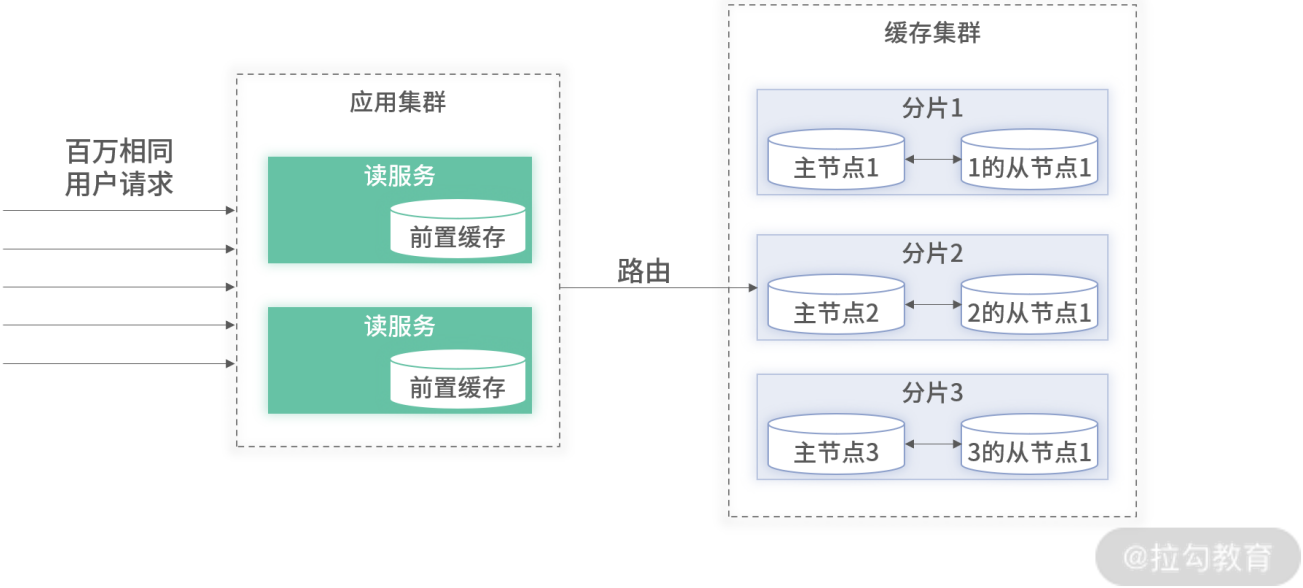

图 4：前置缓存的架构图

应用内的缓存存储的均是热点数据。当应用扩容后，热点缓存的数量也随之增加。在采用了前置缓存后，在面对热查询时只需扩容应用即可。因为所有应用内均存储了所有的热点数据，且前端负载均衡器（如 Nginx 等）会将所有请求平均地分发到各应用中去。

使用应用内的前置缓存应对热点查询时，仍有以下几个问题需要重点关注。

首先是应用内缓存需要设置上限

应用所属宿主机的内存是有限的，且其内存还要支持业务应用使用。固在使用应用内的前置缓存时，必须设置容量的上限且设置容量满时的逐出策略。逐出策略可以是 LRU，将最少使用的缓存在容量满时清理掉，因为热点缓存需要存储的是访问次数多的数据。

此外，前置缓存也需要设置过期时间，毕竟太久无访问的缓存也肯定是非热点数据，所以可以及时清理掉，提前释放内存空间。

其次是根据业务对待延迟的问题

前置缓存的延迟问题的解决方案和“**04 讲**”“**05 讲**”的思路基本类似。要么采用定期刷新，要么采用主动刷新。

如果业务上可以容忍一定时间的延迟，可以在缓存数据上设置一个刷新时间即可。实现起来非常简单。

如果想要实时感知变化，可以采用 Binlog 的方式，在变更时主动刷新。但前置缓存的主动感知不能在前置缓存的应用里实现，因为应用代码也运行在此机器上，通过 MQ 感知变更会消耗非常多的 CPU 和内存资源。另外，前置缓存里数据很少，很多变更消息都会因不在前置缓存中而被忽略掉。**为了实现前置缓存的更新，可以将前置缓存的数据异构一份出来用作判断**，升级的方案如下图 5 所示：

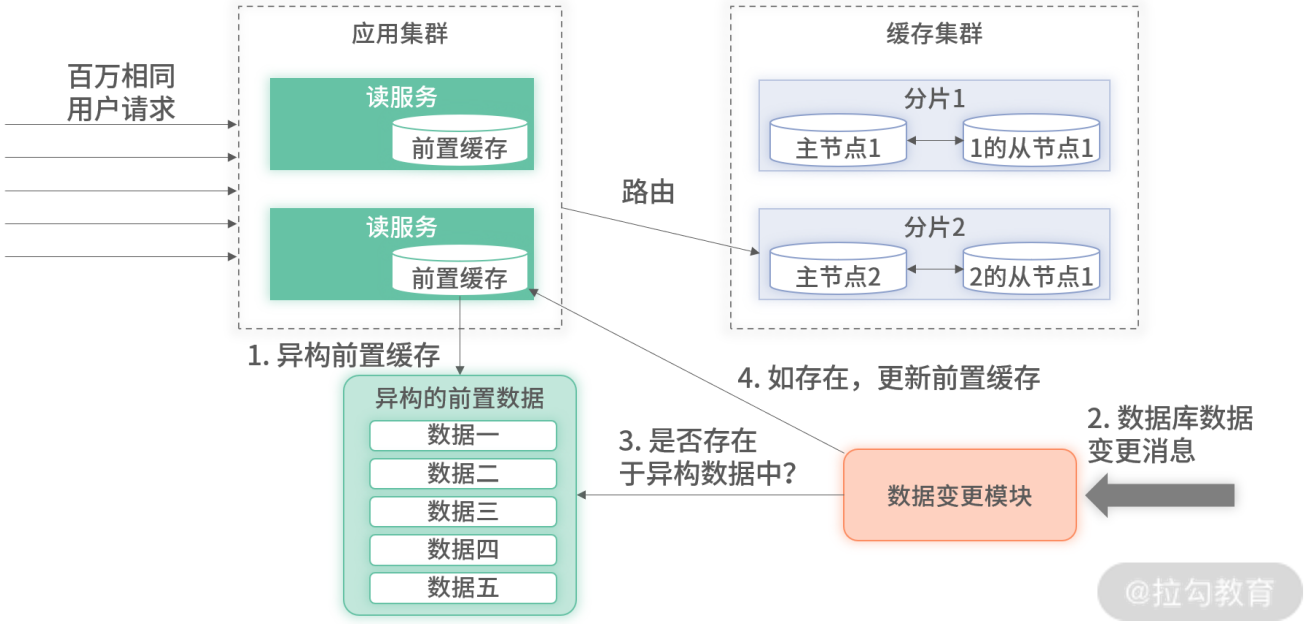

图 5：前置缓存实时更新方案图

通过异构前置缓存用作判断，可以过滤出需要处理的数据，并实时调用对应机器更新即可。此方案实现起来较复杂且异构本来也导致了延迟，实际上大部分场景设置刷新时间即可满足。

再者要把控好瞬间的逃逸流量

应用初始化时，前置缓存是空的。假设在初始化时，瞬间出现热点查询，所有的热点请求都会逃逸到后端缓存里。可能这个瞬间热点就会把后端缓存打挂。

其次，如果前置缓存采用定期过期，在过期时若将数据清理掉，那么所有的请求都会逃逸至后端加载最新的缓存，也有可能把后端缓存打挂。这两种情况对应的流程图如下图 6 所示：

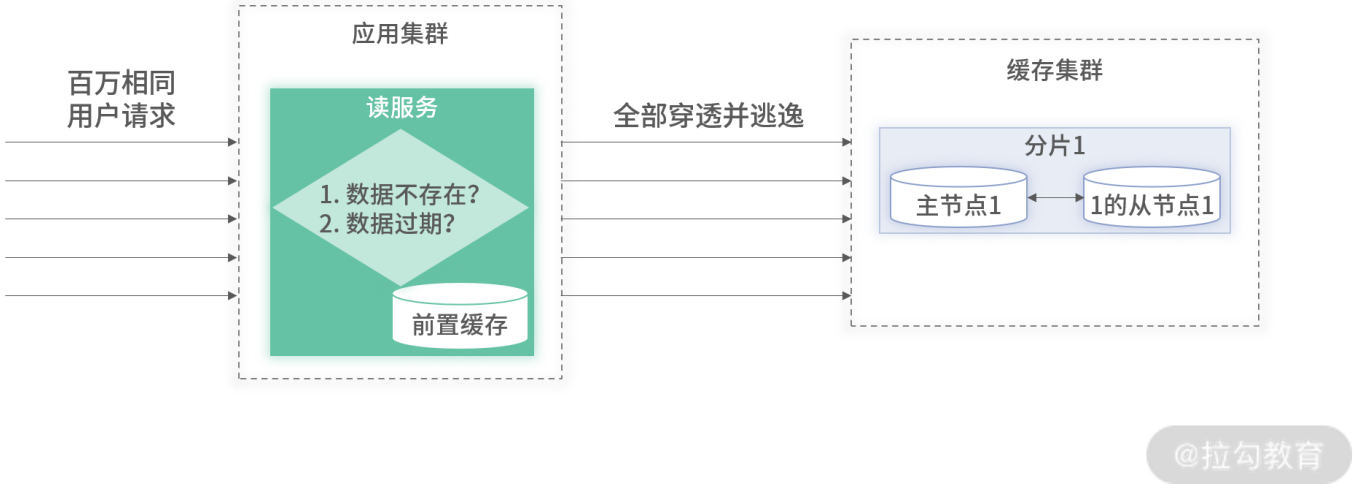

图 6：逃逸流量的架构图

对于这两种情况，可以对逃逸流量进行前置等待或使用历史数据的方案。不管是初始化还是数据过期，在从后端加载数据时，只允许一个请求逃逸。这样最大的逃逸流量为部署的应用总数，量级可控。架构如下图 7 所示：

图 7：逃逸流量控制的架构图

对于数据初始化为空时，其他非逃逸的请求可以等待前置缓存的数据并设置一个超时时间。对于数据过期需要更新时，并不主动清理掉数据。其他非逃逸请求使用历史脏数据，而逃逸的那一个请求负责把数据取回来并刷新前置缓存。

最后如何发现热点缓存并前置

除了需要应对热点缓存，另外一个重点就是如何发现热点缓存。对于发现热点有两个方式，一种是被动发现，另外一种是主动发现。

被动发现是借助前置缓存有容量上限实现的。在被动发现的方案里，读服务接受到的所有请求都会默认从前置缓存中获取数据，如不存在，则从缓存服务器进行加载。因为前置缓存的容量淘汰策略是 LRU，如果数据是热点，它的访问次数一定非常高，因此它一定会在前置缓存中。借助前置缓存的容量上限和淘汰策略，即实现了热点发现。

但此方式也存在一个问题——所有的请求都优先从前置缓存获取数据，并在未查询到时加载服务端数据到本地的前置缓存里，此方式也会把非热点数据存储至前置缓存里，导致非热点数据产生非必要的延迟性。

主动发现则需要借助一些外部计数工具来实现热点的发现。外部计数工具的思路大体比较类似，都是在一个集中的位置对于请求进行计数，并根据配置的阈值判断某请求是否会命中数据。对于判定为热点的数据，主动的推送至应用内的前置缓存即可。下图 8 为在缓存服务器进行计数的架构方案：

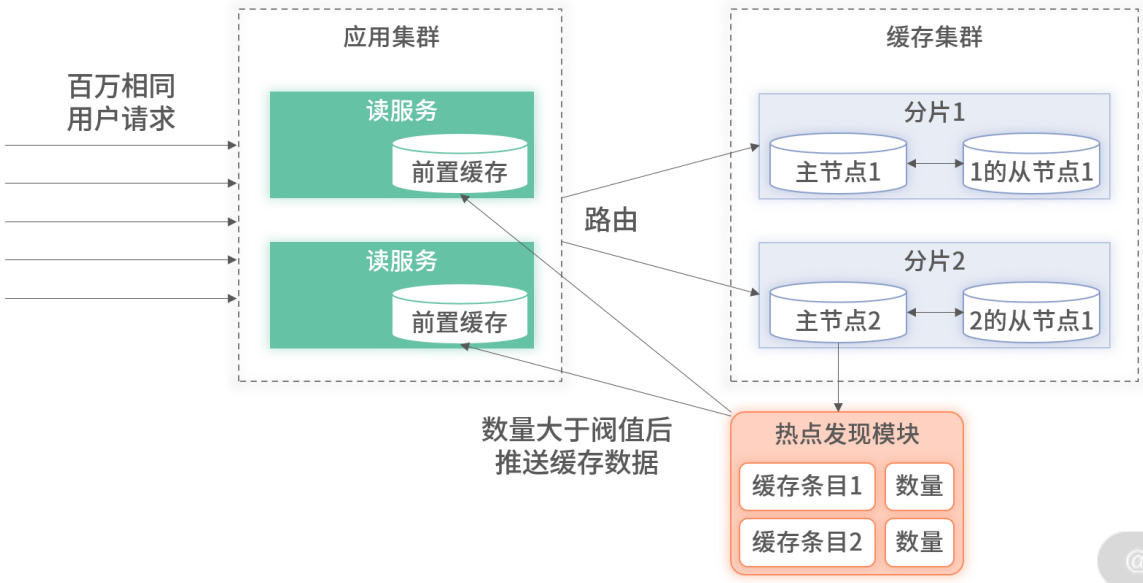

图 8：主动发现热点缓存架构图

采用主动发现的架构后，读服务接受到请求后仍然会默认的从前置缓存获取数据，如获取到即直接返回。如未获取到，会穿透去查询后端缓存的数据并直接返回。但穿透获取到的数据并不会写入本地前置缓存。数据是否为热点且是否要写入前置缓存，均由计数工具

来决定。此方案很好地解决了因误判断带来的延迟问题。

降级兜底不可少

在采用了前置缓存并解决了上述四大类问题之后，当你再次遇到百万级并发时，基本没什么疑难问题了。但这里还存在一个前置条件，即当热点查询发生时，你所部署的容器数量所能支撑的 QPS 要大于热点查询的 QPS。

但实际情况并非如此，你所部署的机器能够支持的 QPS 未必都能够大于当次的热点查询。对于可能出现的超预期流量，可以使用前置限流的策略进行应对。在系统上线前，对于开启了前置缓存的应用进行压测，得到单机最大的 QPS。根据压测值设置单机的限流阈值，阈值可以设置为压测值的一半或者更低。设置为压测阈值的一半或更低，是因为压测时应用 CPU 基本已达到 100%，为了保证线上应用能够正常运转，是不能让 CPU 达到 100% 的。架构如下图 9 所示：

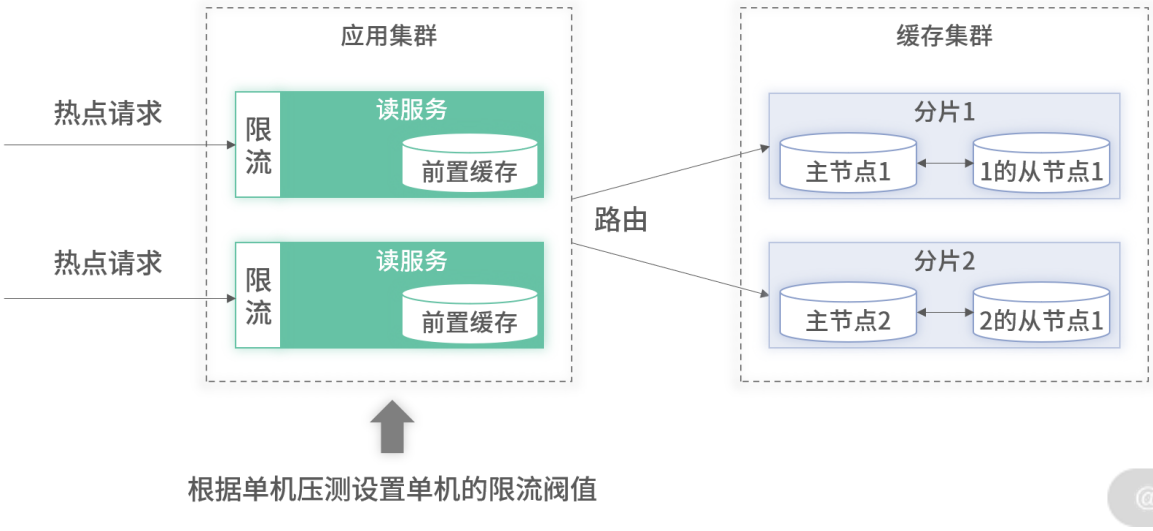

图 9：前置限流的架构图

根据此方案你可以看到，在做架构设计时，即使已经做了非常多的应对方案，最后的兜底降级还是必不可少，因为超出预期的事情说来就来。

其他前置策略

本专栏讨论的是业务后端架构中的各类套路和各项问题的应对方案。但其实在应对热点查询时，除了采用后端应用内的前置缓存进行应对外，在前端的架构里也有一些应对手段。比如在接入层（如 Nginx）进行前置缓存、数据前置至离用户更近的 CDN 及开启浏览器缓存。因专栏定位，此处就不展开讨论，如果你感兴趣，可以订阅拉勾教育里其他关于 CDN 和前端架构的专栏。

总结

本讲介绍了什么是热点查询，以及为什么前两讲的架构支持不了热点查询的并发流量。

在本讲里，介绍了如何使用主从复制，以及前置缓存应对热点查询。同时，对于前置缓存中存在的四大类问题，缓存上限、延迟问题、逃逸流量应对，以及热点数据发现逐一讲解了应对方案。最后，还介绍了在流量超出预期之后，如何进行降低兜底，以及在使用了后端的各种应对策略之后，在前端和接入层有哪些架构方案可供选择。相信你在学习了本讲的内容之后，后续在面对热点查询的场景时，能够轻松应对。

留一个讨论题，你所负责的业务里有热点读的场景吗？你是如何解决的？欢迎留言，我们一起讨论。