35-CodisVSRedisCluster：我该选择哪一个集群方案？

你好，我是蒋德钧。

Redis的切片集群使用多个实例保存数据，能够很好地应对大数据量的场景。在第8讲中，我们学习了Redis官方提供的切片集群方案Redis Cluster，这为你掌握切片集群打下了基础。今天，我再来带你进阶一下，我们来学习下Redis Cluster方案正式发布前，业界已经广泛使用的Codis。

我会具体讲解Codis的关键技术实现原理，同时将Codis和Redis Cluster进行对比，帮你选出最佳的集群方案。

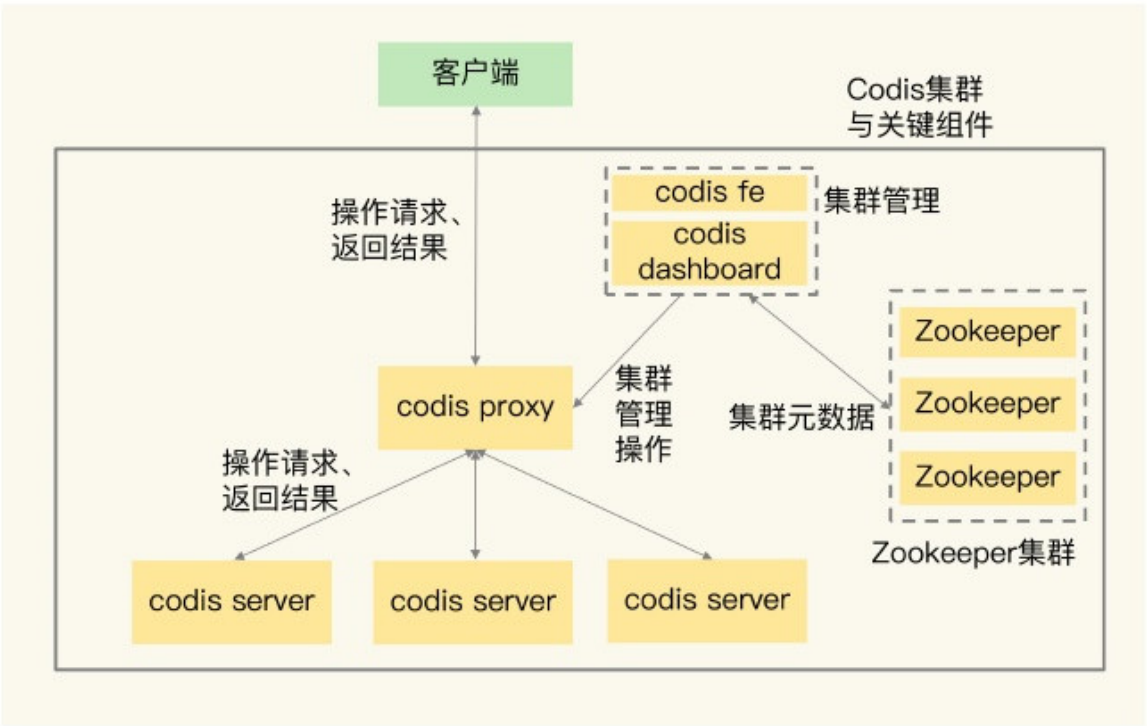
好了，话不多说，我们先来学习下Codis的整体架构和流程。

Codis的整体架构和基本流程

Codis集群中包含了4类关键组件。

- codis server：这是进行了二次开发的Redis实例，其中增加了额外的数据结构，支持数据迁移操作，主要负责处理具体的数据读写请求。
- codis proxy：接收客户端请求，并把请求转发给codis server。
- Zookeeper集群：保存集群元数据，例如数据位置信息和codis proxy信息。
- codis dashboard和codis fe：共同组成了集群管理工具。其中，codis dashboard负责执行集群管理工作，包括增删codis server、codis proxy和进行数据迁移。而codis fe负责提供dashboard的Web操作界面，便于我们直接在Web界面上进行集群管理。

我用一张图来展示下Codis集群的架构和关键组件。

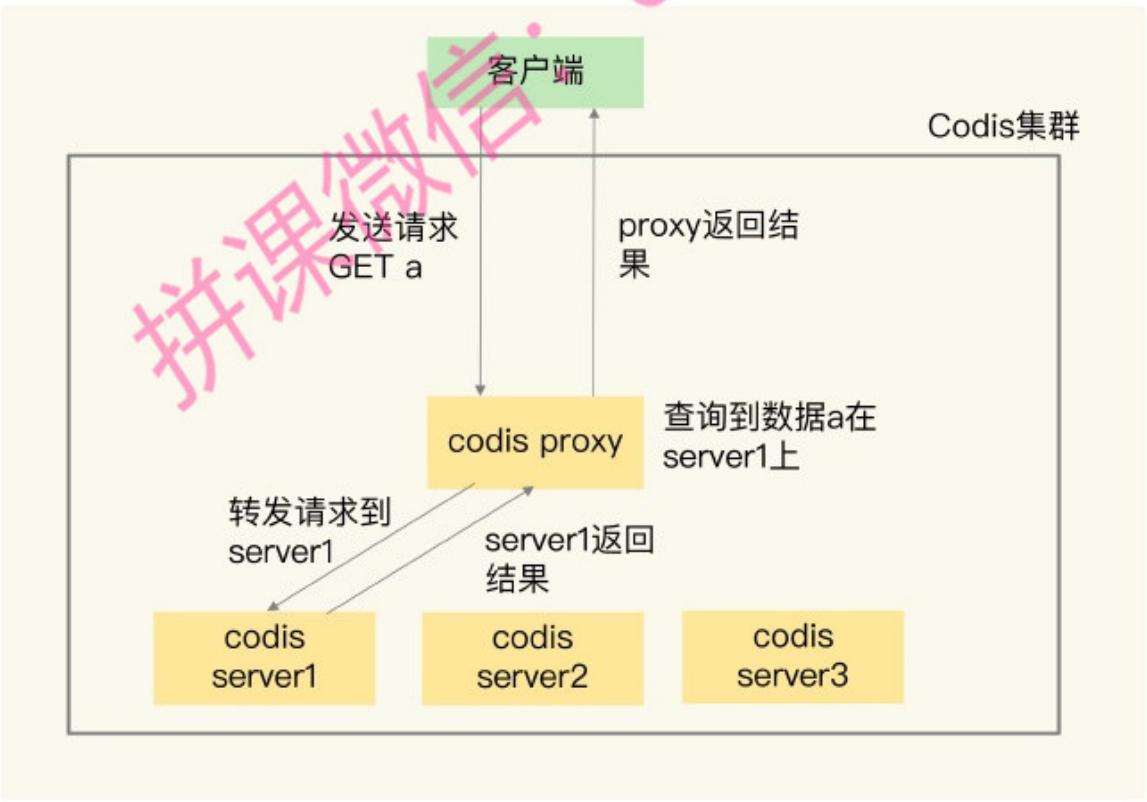


首先，为了让集群能接收并处理请求，我们要先使用codis dashboard 设置codis server和codis proxy的访问地址，完成设置后，codis server和codis proxy才会开始接收连接。

然后，当客户端要读写数据时，客户端直接和codis proxy建立连接。你可能会担心，既然客户端连接的是proxy，是不是需要修改客户端，才能访问proxy？其实，你不用担心，codis proxy本身支持Redis的RESP交互协议，所以，客户端访问codis proxy时，和访问原生的Redis实例没有什么区别，这样一来，原本连接单实例的客户端就可以轻松地和Codis集群建立起连接了。

最后，codis proxy接收到请求，就会查询请求数据和codis server的映射关系，并把请求转发给相应的codis server进行处理。当codis server处理完请求后，会把结果返回给codis proxy，proxy再把数据返回给客户端。

我来用一张图展示这个处理流程：



好了，了解了Codis集群架构和基本流程后，接下来，我就围绕影响切片集群使用效果的4方面技术因素：数据分布、集群扩容和数据迁移、客户端兼容性、可靠性保证，来和你聊聊它们的具体设计选择和原理，帮你掌握Codis的具体用法。

Codis的关键技术原理

一旦我们使用了切片集群，面临的第一个问题就是，**数据是怎么在多个实例上分布的**。

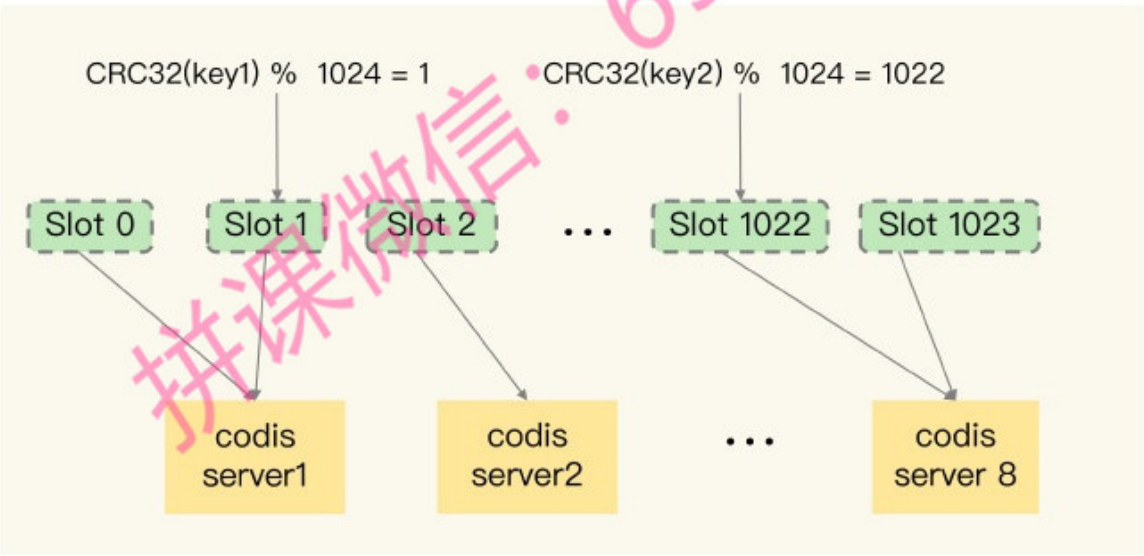
数据如何在集群里分布？

在Codis集群中，一个数据应该保存在哪个codis server上，这是通过逻辑槽（Slot）映射来完成的，具体来说，总共分成两步。

第一步，Codis集群一共有1024个Slot，编号依次是0到1023。我们可以把这些Slot手动分配给codis server，每个server上包含一部分Slot。当然，我们也可以让codis dashboard进行自动分配，例如，dashboard把1024个Slot在所有server上均分。

第二步，当客户端要读写数据时，会使用CRC32算法计算数据key的哈希值，并把这个哈希值对1024取模。而取模后的值，则对应Slot的编号。此时，根据第一步分配的Slot和server对应关系，我们就可以知道数据保存在哪个server上了。

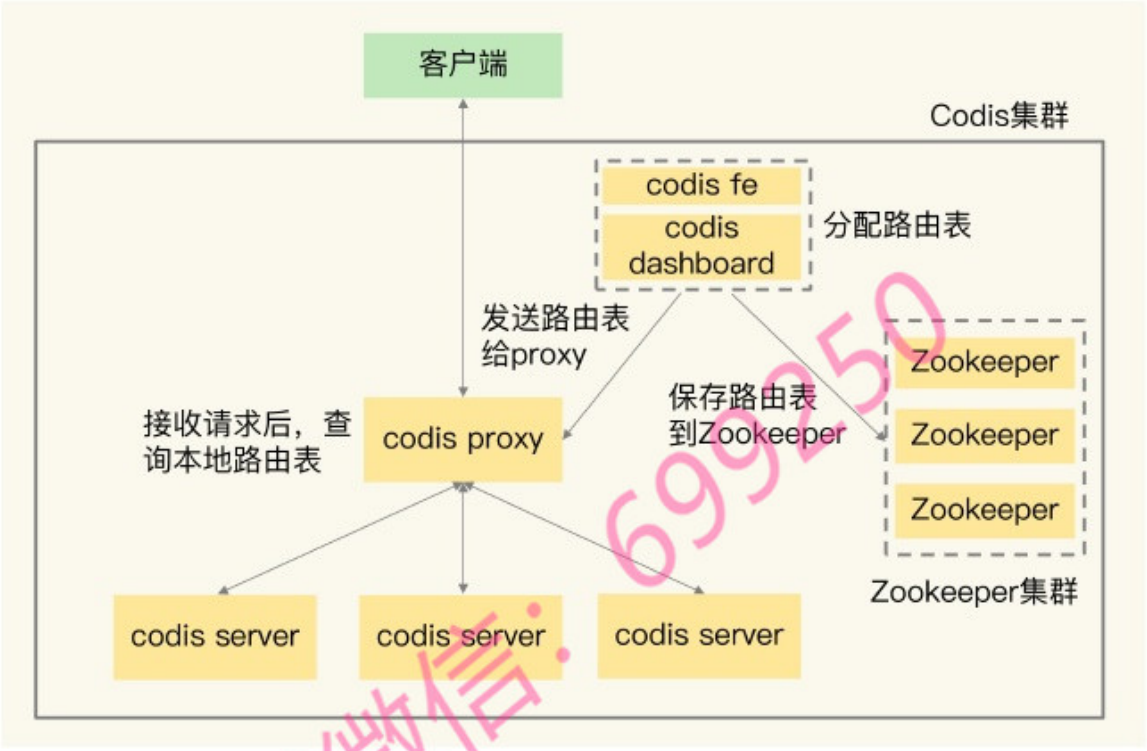
我来举个例子。下图显示的就是数据、Slot和codis server的映射保存关系。其中，Slot 0和1被分配到了server1，Slot 2分配到server2，Slot 1022和1023被分配到server8。当客户端访问key 1和key 2时，这两个数据的CRC32值对1024取模后，分别是1和1022。因此，它们会被保存在Slot 1和Slot 1022上，而Slot 1和Slot 1022已经被分配到codis server 1和8上了。这样一来，key 1和key 2的保存位置就很清楚了。



数据key和Slot的映射关系是客户端在读写数据前直接通过CRC32计算得到的，而Slot和codis server的映射关系是通过分配完成的，所以就需要用一个存储系统保存下来，否则，如果集群有故障了，映射关系就会丢失。

我们把Slot和codis server的映射关系称为数据路由表（简称路由表）。我们在codis dashboard上分配好路由表后，dashboard会把路由表发送给codis proxy，同时，dashboard也会把路由表保存在Zookeeper中。codis-proxy会把路由表缓存在本地，当它接收到客户端请求后，直接查询本地的路由表，就可以完成正确的请求转发了。

你可以看下这张图，它显示了路由表的分配和使用过程。



在数据分布的实现方法上，Codis和Redis Cluster很相似，都采用了key映射到Slot、Slot再分配到实例上的机制。

但是，这里有一个明显的区别，我来解释一下。

Codis中的路由表是我们通过codis dashboard分配和修改的，并被保存在Zookeeper集群中。一旦数据位置发生变化（例如有实例增减），路由表被修改了，codis dashbaord就会把修改后的路由表发送给codis proxy，proxy就可以根据最新的路由信息转发请求了。

在Redis Cluster中，数据路由表是通过每个实例相互间的通信传递的，最后会在每个实例上保存一份。当数据路由信息发生变化时，就需要在所有实例间通过网络消息进行传递。所以，如果实例数量较多的话，就会消耗较多的集群网络资源。

数据分布解决了新数据写入时该保存在哪个server的问题，但是，当业务数据增加后，如果集群中的现有实例不足以保存所有数据，我们就需要对集群进行扩容。接下来，我们再来学习下Codis针对集群扩容的关键技术设计。

集群扩容和数据迁移如何进行？

Codis集群扩容包括了两方面：增加codis server和增加codis proxy。

我们先来看增加codis server，这个过程主要涉及到两步操作：

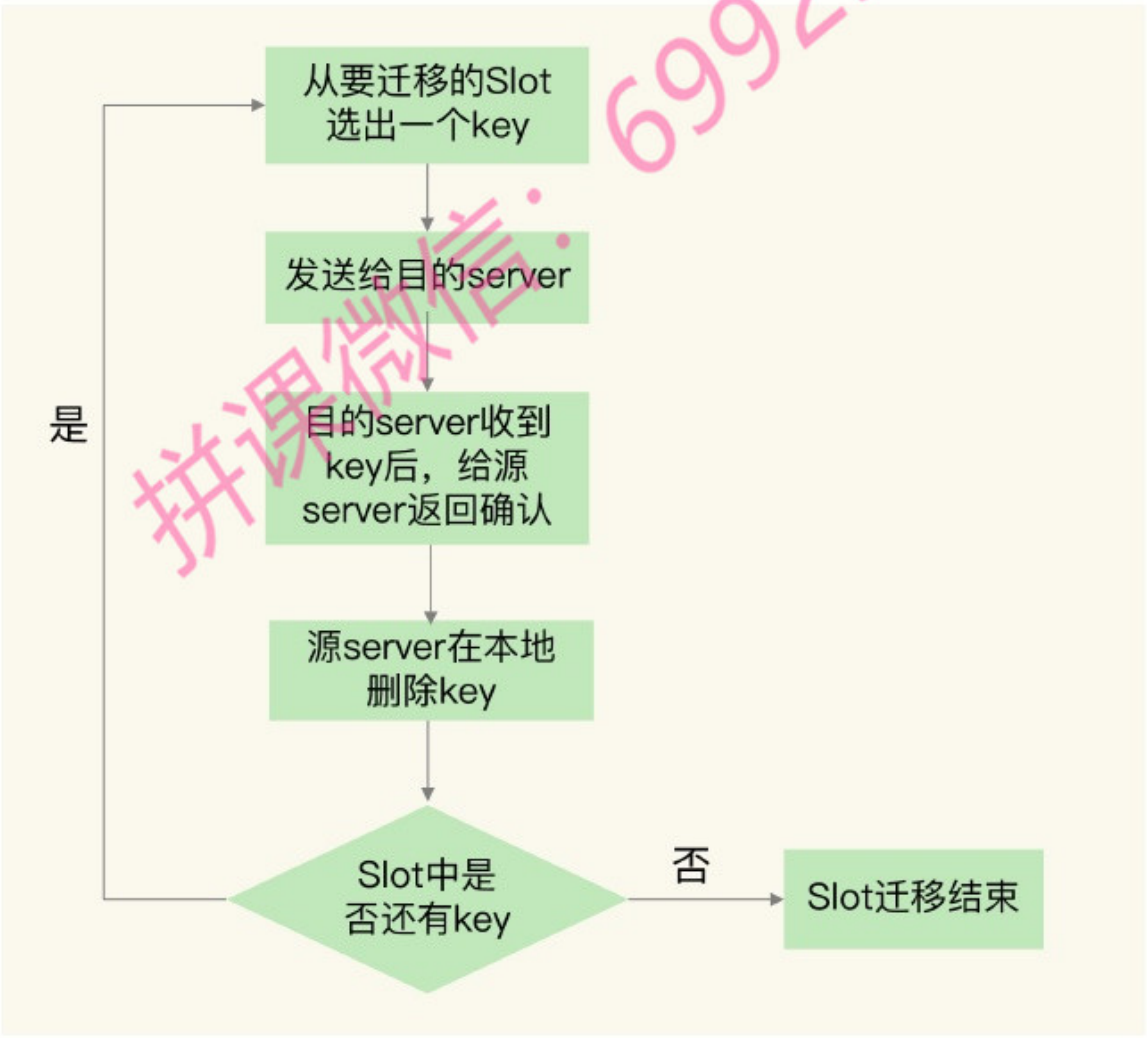
1. 启动新的codis server，将它加入集群；
2. 把部分数据迁移到新的server。

需要注意的是，这里的数据迁移是一个重要的机制，接下来我来重点介绍下。

Codis集群按照Slot的粒度进行数据迁移，我们来看下迁移的基本流程。

1. 在源server上，Codis从要迁移的Slot中随机选择一个数据，发送给目的server。
2. 目的server确认收到数据后，会给源server返回确认消息。这时，源server会在本地将刚才迁移的数据删除。
3. 第一步和第二步就是单个数据的迁移过程。Codis会不断重复这个迁移过程，直到要迁移的Slot中的数据全部迁移完成。

我画了下面这张图，显示了数据迁移的流程，你可以看下加深理解。



针对刚才介绍的单个数据的迁移过程，Codis实现了两种迁移模式，分别是同步迁移和异步迁移，我们来具体看下。

同步迁移是指，在数据从源server发送给目的server的过程中，源server是阻塞的，无法处理新的请求操作。这种模式很容易实现，但是迁移过程中会涉及多个操作（包括数据在源server序列化、网络传输、在目的server反序列化，以及在源server删除），如果迁移的数据是一个bigkey，源server就会阻塞较长时间，无法及时处理用户请求。

为了避免数据迁移阻塞源server，Codis实现的第二种迁移模式就是异步迁移。异步迁移的关键特点有两个。

第一个特点是，当源server把数据发送给目的server后，就可以处理其他请求操作了，不用等到目的server的命令执行完。而目的server会在收到数据并反序列化保存到本地后，给源server发送一个ACK消息，表明迁移完成。此时，源server在本地把刚才迁移的数据删除。

在这个过程中，迁移的数据会被设置为只读，所以，源server上的数据步会被修改，自然也就不会出现“和目的server上的数据不一致”问题了。

第二个特点是，对于bigkey，异步迁移采用了拆分指令的方式进行迁移。具体来说就是，对bigkey中每个元素，用一条指令进行迁移，而不是把整个bigkey进行序列化后再整体传输。这种化整为零的方式，就避免了bigkey迁移时，因为要序列化大量数据而阻塞源server的问题。

此外，当bigkey迁移了一部分数据后，如果Codis发生故障，就会导致bigkey的一部分元素在源server，而另一部分元素在目的server，这就破坏了迁移的原子性。

所以，Codis会在目标server上，给bigkey的元素设置一个临时过期时间。如果迁移过程中发生故障，那么，目标server上的key会在过期后被删除，不会影响迁移的原子性。当正常完成迁移后，bigkey元素的临时过期时间会被删除。

我给你举个例子，假如我们要迁移一个有1万个元素的List类型数据，当使用异步迁移时，源server就会给目的server传输1万条RPUSH命令，每条命令对应了List中一个元素的插入。在目的server上，这1万条命令再被依次执行，就可以完成数据迁移。

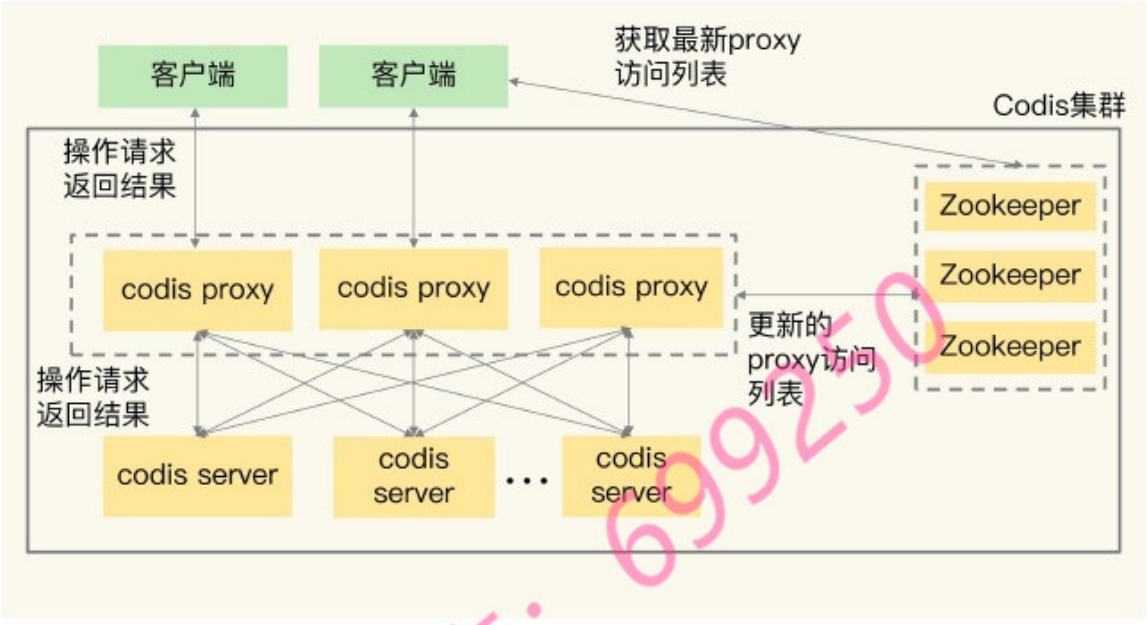
这里，有个地方需要你注意下，为了提升迁移的效率，Codis在异步迁移Slot时，允许每次迁移多个key。**你可以通过异步迁移命令SLOTSMGRTTAGSLOT-ASYNC的参数numkeys设置每次迁移的key数量**。

刚刚我们学习的是codis server的扩容和数据迁移机制，其实，在Codis集群中，除了增加codis server，有时还需要增加codis proxy。

因为在Codis集群中，客户端是和codis proxy直接连接的，所以，当客户端增加时，一个proxy无法支撑大量的请求操作，此时，我们就需要增加proxy。

增加proxy比较容易，我们直接启动proxy，再通过codis dashboard把proxy加入集群就行。

此时，codis proxy的访问连接信息都会保存在Zookeeper上。所以，当新增了proxy后，Zookeeper上会有最新的访问列表，客户端也就可以从Zookeeper上读取proxy访问列表，把请求发送给新增的proxy。这样一来，客户端的访问压力就可以在多个proxy上分担处理了，如下图所示：



好了，到这里，我们就了解了Codis集群中的数据分布、集群扩容和数据迁移的方法，这都是切片集群中的关键机制。

不过，因为集群提供的功能和单实例提供的功能不同，所以，我们在应用集群时，不仅要关注切片集群中的关键机制，还需要关注客户端的使用。这里就有一个问题了：业务应用采用的客户端能否直接和集群交互呢？接下来，我们就来聊下这个问题。

集群客户端需要重新开发吗？

使用Redis单实例时，客户端只要符合RESP协议，就可以和实例进行交互和读写数据。但是，在使用切片集群时，有些功能是和单实例不一样的，比如集群中的数据迁移操作，在单实例上是没有的，而且迁移过程中，数据访问请求可能要被重定向（例如Redis Cluster中的MOVE命令）。

所以，客户端需要增加和集群功能相关的命令操作的支持。如果原来使用单实例客户端，想要扩容使用集群，就需要使用新客户端，这对于业务应用的兼容性来说，并不是特别友好。

Codis集群在设计时，就充分考虑了对现有单实例客户端的兼容性。

Codis使用codis proxy直接和客户端连接，codis proxy是和单实例客户端兼容的。而和集群相关的管理工作（例如请求转发、数据迁移等），都由codis proxy、codis dashboard这些组件来完成，不需要客户端参与。

这样一来，业务应用使用Codis集群时，就不用修改客户端了，可以复用和单实例连接的客户端，既能利用集群读写大容量数据，又避免了修改客户端增加复杂的操作逻辑，保证了业务代码的稳定性和兼容性。

最后，我们再来看下集群可靠性的问题。可靠性是实际业务应用的一个核心要求。**对于一个分布式系统来说，它的可靠性和系统中的组件个数有关：组件越多，潜在的风险点也就越多**。和Redis Cluster只包含Redis实例不一样，Codis集群包含的组件有4类。那你就会问了，这么多组件会降低Codis集群的可靠性吗？

怎么保证集群可靠性？

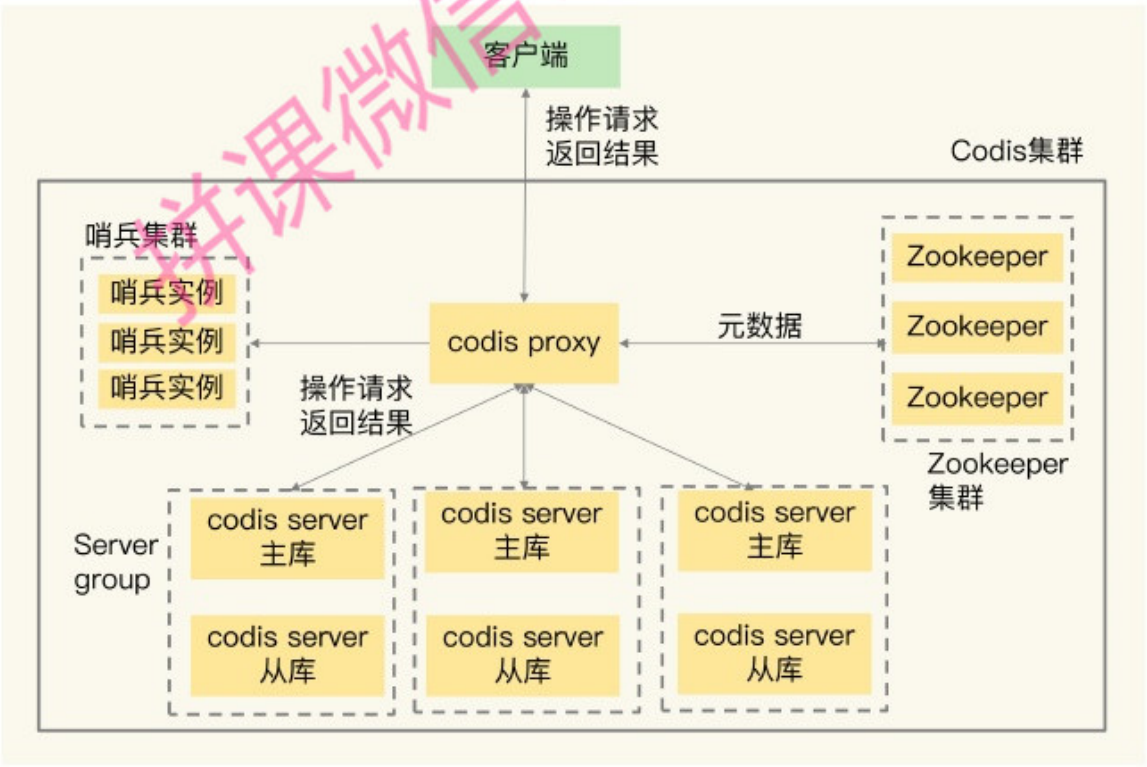
我们来分别看下Codis不同组件的可靠性保证方法。

首先是codis server。

codis server其实就是Redis实例，只不过增加了和集群操作相关的命令。Redis的主从复制机制和哨兵机制在codis server上都是可以使用的，所以，Codis就使用主从集群来保证codis server的可靠性。简单来说就是，Codis给每个server配置从库，并使用哨兵机制进行监控，当发生故障时，主从库可以进行切换，从而保证了server的可靠性。

在这种配置情况下，每个server就成为了一个server group，每个group中是一主多从的server。数据分布使用的Slot，也是按照group的粒度进行分配的。同时，codis proxy在转发请求时，也是按照数据所在的Slot和group的对应关系，把写请求发到相应group的主库，读请求发到group中的主库或从库上。

下图展示的是配置了server group的Codis集群架构。在Codis集群中，我们通过部署server group和哨兵集群，实现codis server的主从切换，提升集群可靠性。



因为codis proxy和Zookeeper这两个组件是搭配在一起使用的，所以，接下来，我们再来看下这两个组件的可靠性。

在Codis集群设计时，proxy上的信息源头都是来自Zookeeper（例如路由表）。而Zookeeper集群使用多个实例来保存数据，只要有超过半数的Zookeeper实例可以正常工作， Zookeeper集群就可以提供服务，也可以保证这些数据的可靠性。

所以，codis proxy使用Zookeeper集群保存路由表，可以充分利用Zookeeper的高可靠性保证来确保codis proxy的可靠性，不用再做额外的工作了。当codis proxy发生故障后，直接重启proxy就行。重启后的proxy，可以通过codis dashboard从Zookeeper集群上获取路由表，然后，就可以接收客户端请求进行转发了。这样的设计，也降低了Codis集群本身的开发复杂度。

对于codis dashboard和codis fe来说，它们主要提供配置管理和管理员手工操作，负载压力不大，所以，它们的可靠性可以不用额外进行保证了。

切片集群方案选择建议

到这里，Codis和Redis Cluster这两种切片集群方案我们就学完了，我把它们的区别总结在了一张表里，你可以对比看下。

对比维度	Codis	Redis Cluster
数据路由信息	中心化保存在Zookeeper，proxy在本地缓存	每个实例都保存一份
集群扩容	增加codis server和codis proxy	增加Redis实例
数据迁移	支持同步迁移、异步迁移	支持同步迁移
客户端兼容性	兼容单实例客户端	需要开发支持Cluster功能的客户端
可靠性	codis server: 主从集群机制保证可靠性 codis proxy: 无状态设计，故障后重启就行 Zookeeper：可靠性高，只要有半数以上节点存在就能继续服务	实例使用主从集群保证可靠性

最后，在实际应用的时候，对于这两种方案，我们该怎么选择呢？我再给你提4条建议。

1. 从稳定性和成熟度来看，Codis应用得比较早，在业界已经有了成熟的生产部署。虽然Codis引入了proxy和Zookeeper，增加了集群复杂度，但是，proxy的无状态设计和Zookeeper自身的稳定性，也给Codis的稳定使用提供了保证。而Redis Cluster的推出时间晚于Codis，相对来说，成熟度要弱于Codis，如果你想选择一个成熟稳定的方案，Codis更加合适些。
2. 从业务应用客户端兼容性来看，连接单实例的客户端可以直接连接codis proxy，而原本连接单实例的客户端要想连接Redis Cluster的话，就需要开发新功能。所以，如果你的业务应用中大量使用了单实例的客户端，而现在想应用切片集群的话，建议你选择Codis，这样可以避免修改业务应用中的客户端。
3. 从使用Redis新命令和新特性来看，Codis server是基于开源的Redis 3.2.8开发的，所以，Codis并不支持Redis后续的开源版本中的新增命令和数据类型。另外，Codis并没有实现开源Redis版本的所有命令，比如BITOP、BLPOP、BRPOP，以及和与事务相关的MUTLI、EXEC等命令。Codis官网上列出了不被支持的命令列表，你在使用时记得去核查一下。所以，如果你想使用开源Redis 版本的新特性，Redis Cluster是一个合适的选择。
4. 从数据迁移性能维度来看，Codis能支持异步迁移，异步迁移对集群处理正常请求的性能影响要比使用同步迁移的小。所以，如果你在应用集群时，数据迁移比较频繁的话，Codis是个更合适的选择。

小结

这节课，我们学习了Redis切片集群的Codis方案。Codis集群包含codis server、codis proxy、Zookeeper、codis dashboard和codis fe这四大类组件。我们再来回顾下它们的主要功能。

- codis proxy和codis server负责处理数据读写请求，其中，codis proxy和客户端连接，接收请求，并转发请求给codis server，而codis server负责具体处理请求。
- codis dashboard和codis fe负责集群管理，其中，codis dashboard执行管理操作，而codis fe提供Web管理界面。

- Zookeeper集群负责保存集群的所有元数据信息，包括路由表、proxy实例信息等。这里，有个地方需要你注意，除了使用Zookeeper，Codis还可以使用etcd或本地文件系统保存元数据信息。

关于Codis和Redis Cluster的选型考虑，我从稳定性成熟度、客户端兼容性、Redis新特性使用以及数据迁移性能四个方面给你提供了建议，希望能帮助到你。

最后，我再给你提供一个Codis使用上的小建议：当你有多条业务线要使用Codis时，可以启动多个codis dashboard，每个dashboard管理一部分codis server，同时，再用一个dashboard对应负责一个业务线的集群管理，这样，就可以做到用一个Codis集群实现多条业务线的隔离管理了。

每课一问

按照惯例，我会给你提个小问题。假设Codis集群中保存的80%的键值对都是Hash类型，每个Hash集合的元素数量在10万～20万个，每个集合元素的大小是2KB。你觉得，迁移一个这样的Hash集合数据，会对Codis的性能造成影响吗？

欢迎在留言区写下你的思考和答案，我们一起交流讨论。如果你觉得今天的内容对你有所帮助，也欢迎你分享给你的朋友或同事。我们下节课见。

精选留言：

- Kaito 2020-11-11 00:21:09
  å‡ è®¾Codis é›†ç¾¤ä¸ä¿å˜çš„80%çš„é”®å¯¹éƒ½æ˜¯Hashç±»åž‹ï¼Œæ¯ä¸ªHashé›†åˆçš„å…ƒç´ æ•°é‡åœ¨10ä¸‡ï½ž20ä¸‡ä¸ªï¼Œæ¯ä¸ªå…ƒç´ çš„å¤§å°æ˜¯2KBã€‚è¿ç§»ä¸€ä¸ªè¿™æ ·çš„Hashé›†åˆæ•°æ®ï¼Œä¼šå¯¹Codisçš„æ€§èƒ½é€ æˆå½±å“å—ï¼Ÿ

  ä¸€èˆ¬æ€§èƒ½æ˜¯æ²¡æœ‰å½±å“çš„ã€‚

  Codis åœ¨è¿ç§»æ•°æ®çš„æ—¶å€™æ˜¯ä»¥é”®ä¸ºç²’åº¦çš„ï¼Œæ•°æ®è¿ç§»è¿‡ç¨‹æ˜¯ä¸ä¼šé˜»å¡žå…¶ä»–è¯·æ±‚çš„ã€‚

  1ã€‚å¼€æ¿€è¿ç§»çš„é”®ï¼Œæ˜¯æŒ‰é¡ºåºä¸€ä¸ªä¸€ä¸ªè¿ç§»çš„ï¼Œå¯¹äºŽä¸€ä¸ªé”®çš„è¿ç§»è¿‡ç¨‹æ˜¯åŽŸåçš„ï¼Œä¸ä¼šå‡ºçŽ°ä¸é—´çŠ¶æ€ï¼Œæ‰€ä»¥ä¸ä¼šå½±å“æ•°æ®çš„æ£ç¡®æ€§ã€‚

  2ã€‚æ¯ä¸ªé”®è¿ç§»çš„æ—¶å€™ï¼Œæ¯ä¸ªé”®éƒ½æ˜¯åŒæ¥è¿ç§»çš„ï¼Œç›®æ ‡Codisä¼šç«‹å³è¿”å›žOKç»™æºCodisï¼Œè¡¨ç¤ºè¿ç§»å®Œæˆäº†ã€‚

  3ã€‚å¦‚æžœé”®æ˜¯ä¸€ä¸ªå¾ˆå¤§çš„é›†åˆï¼Œé‚£ä¹ˆåœ¨è¿ç§»è¿™ä¸ªé”®çš„è¿‡ç¨‹ä¸ï¼Œè¿™ä¸ªé›†åˆå¯èƒ½ä¼šå‡ºçŽ°é˜»å¡žçš„æƒ…å†µã€‚

  4ã€‚bigkey å¯¹ä¸Žæ¯”è¾ƒçš„bigkeyæ‹¥æœ‰å¯¹ä¸Žæ¯”è¾ƒçš„å…ƒç´ æ•°é‡ä¸Žæ¯”è¾ƒçš„å…ƒç´ å¤§å°éƒ½æ˜¯ä¸€æ ·çš„ï¼Œæ‰€ä»¥è¿ç§»bigkeyçš„æ—¶å€™ä¸Žæ¯”è¾ƒçš„é”®ä¼šæœ‰æ˜Žæ˜¾çš„åœé¡¿ã€‚

  5ã€‚ä¸€æ¬¡è¿ç§»å¤šä¸ªkeyï¼Œä¸€æ¬¡è¿ç§»çš„é”®æ•°é‡è¶Š

6ã€è¿ç§»æµé‡æ§åˆ¶ï¼šè¿ç§»æ—¶ä¼šæ§åˆ¶ç¼“å†²åŒºå§°ï¼Œé¿å……å¤æ»¡ç½‘ç»œå¸¦å®½ã€‚

7ã€bigkey è¿ç§»å¯¹æ€§èƒ½çš„å…¼å®¹è¯´æ˜ï¼šè¿ç§»å¤±è´¥åŽŸå› ,ç‰æ€§ä¼šæ‰«æ‰€æœ‰å…ƒç´ ï¼Œæ‰¾åˆ°bigkey æ‰§è¡Œåˆ é™¤æ“ä½œã€‚è¿ç§»å®Œæˆåï¼Œæ¸…ç†æ—§èŠ‚ç‚¹ä¸Šçš„bigkeyã€‚

Codis åœ¨æ · æ®è¿ç§»æ–¹å¼ä¸Ž Redis Cluster å®šä¹‰æ¯”ç§°ï¼Œ Codis è¿˜å¸¦æ ä¸€é©±å¸¸ä½¿ç›‘ç»´æŠ¤ DBA æ‰§è¡Œå¢žåˆ èŠ‚ç‚¹ã€»ä»»åˆ‡æ¢ã€‚

æˆ‘ä½ “æ—¶åœ¨å¯¹ Codis å¼€å‘ æ–¹æ—¶ï¼Œè¿˜æ²¡æœ‰ Redis 3.2.8 å®šçš„ä¸Œæ¬¡å¼€ï¼Œæ‰€ä»¥å¼€å‘ Redis Server æ¯” è¾ƒå›°éš¾ï¼Œæ–°ç‰æ€§ä¸Šä¸Žæ—§ç‰ˆ Redis 3.2.8 ä¸Šæ¯”è¾ƒã€‚

ç°åœ¨ Codis å·²ç»ä¸Šçº¿å†…éƒ¨æ¿ä½¿ç”¨ï¼Œæ¯ä½¿ç¨³å®šå›žæ¡£ Redis é› ç¾¤è§£å†³æ–¹æ¡ˆï¼Œæ€§èƒ½æ¯”å…¶ä»– Redis Server æ¡†æ¶é«˜ã€‚ç›®å‰ Codis æ¯ä¸ªé›†ç¾¤æ¯”è¾ƒç¨³å®šåœ¨ 50G å·¦å³ï¼Œæ¯ä¸ªé›†ç¾¤éƒ½æ˜¯ 32 ä¸ª Group æ„æˆï¼Œæ¯ä¸ª Group éƒ½æ˜¯ 1 master 1 slave çš„é«˜å¯ç”¨ã€‚è¿™ä¸ªè§„æ¨¡ä¹‹å‰æ˜¯ç»è¿‡å¤šæ¬¡å‹æµ‹çš„ï¼Œæ¯ä¸ª Redis å®žä¾‹å†…å˜å ç”¨éƒ½æ§åˆ¶åœ¨ 5GB 以内。

- æ¨¨é€¨æ—2020-11-11 23:24:18
æˆ‘ç®€å• ç”¨ Google æœœäº†ä¸‹ï¼Œæ¿çš„ Redis é› ç¾¤æ–¹æ¡ˆä¸ä¸€æ ·ä¸ä¸€æ ·çš„ï¼š
1. Redis Cluster
2. å¤§å®¶å°é¡¹ç›®ç„¶å¼€å§‹ä¸Šâ€” â€” Twitter å¼€æºçš„ Twemproxyã€Codis
3. ä¹°ä¸“æœ‰çš„ Redis æœåŠ¡â€” â€” Aliyun AparaCacheï¼ˆè¿™ä¸ªå¼€æºçš„æ²¡æœ‰ slot çš„åŠŸèƒ½ï¼‰AWS ElasticCache

ç¬¬äºŒç§ æ–¹çš„è¯´ï¼Œçœ‹äº†ä¸‹Github å¯¹çš„ Repositoryï¼Œä¸€æ¬¡æ›´æ–°æ˜¯å¹´å‰çš„äº‹äº†ï¼Œæ²¡æœ‰è¿è¡Œ Redis çš„ä¸€ä¸ªæ¡ˆä¾‹ï¼Œæ²¡æœ‰ä¸€ä¸ªæ¯”è¾ƒæˆåŠŸç

çš„æ–¹å¼�è¿›è¡Œè¿�ç § »ã€‚å···ä½ “æ�¥è¯´å°±æ˜¯ï¼Œå¯¹ bigkey ä¸æ¯�ä¸ªå···ƒç´ ï¼Œç ”究ä¸€æ�¡æŒ‡ä»¤è¿›è¡Œè¿�ç § »ï¼Œè€Œä¸,æ¯ æŠŠæ · ´ä¸ª bigkey è¿›è¡Œå°�å^—åŒ–å��å†�æ · ´ä½ “ä¼ è¾“ã€‚â€

æ±,é—®è€�å¸ˆï¼Œæ‹ †è¿�ç § »bigKeyçš„æ— ¶â€™ï¼Œæº�serverä¸Šçš„è¿™ä¸ªkeyæ˜¯ä¸,æ˜¯ä¹Ÿæ˜¯å�ªè¯»ï¼Œä¸å···è®¸ä¸ä¸Ÿæ˜¯å¯¹è¿™ä¸ªkeyçš„ä¿®æ”¹ï¼Ÿ
å¦,æ�œæ˜¯çš„è¯�ï¼Œé,£è¿�ç § »çš„è¡ ç¨«ä¸,è¿™ä¸ªkeyçš„ä¿®æ”¹è¯·æ±,ä¼šè¢«é˜»å¡ï¼Œæ€ § èƒ½ä¼šå¯¼è‡´å¯¹å “ï¼Ÿ
å¦,æ�œä¸,æ˜¯çš„è¯ï¼Œæ€�ä¹ˆä¿è¯æ · °æ®ä¸€è ‡ ´æ€ § å ‘¢ï¼Ÿ

- snailshen 2020-11-11 14:22:39
  è€�å¸ˆï¼Œcodisçš„keyå’ Œslotçš„æ˜ å°„æ˜¯crc16,å¹¶ä¸ zookeeperä¿¿å˜çš„å···ƒæ · °æ®ä¿¿æ®ï¼Œå¯¹æ¯ proxyçš„ä¿¿æ®å¹¶ä¸,å˜å,¨slotä¿¿æ®å§ §

- æ¯ å¤©æ™ ç™½ç‰ 2020-11-11 13:30:43
  é€šè¿ ‡å¦®è¿™ç¯ ‡æ– ‡ç« ï¼Œä¸ºäº†æ„Ÿè § ‰é› † ç¾¤æ› ´â€¾å¤ ‘ä° Codis

- å,² 2020-11-11 10:14:02
  è€�å¸ˆå¥½ ç›®å‰…å···¬å,åœ¨ç¯ ¨twemproxyç» ´æŠ¤Redisé› † ç¾¤ï¼Œè€ƒè½ç®å · å¯¹æ¯ ä¸‹codisâ’ Œtwemproxyçš„ä¼˜åŠ£å—ï¼Ÿ