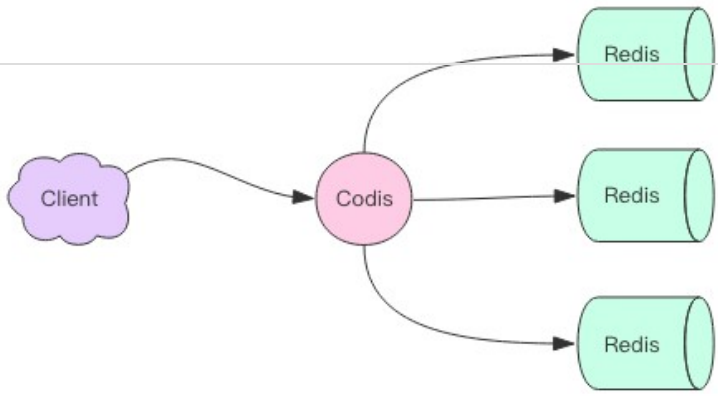
集群 2：分而治之 —— Codis

在大数据高并发场景下，单个 Redis 实例往往会显得捉襟见肘。首先体现在内存上，单个 Redis 的内存不宜过大，内存太大会导致 rdb 文件过大，进一步导致主从同步时全量同步时间过长，在实例重启恢复时也会消耗很长的数据加载时间，特别是在云环境下，单个实例内存往往都是受限的。其次体现在 CPU 的利用率上，单个 Redis 实例只能利用单个核心，这单个核心要完成海量数据的存取和管理工作压力会非常大。

正是在这样的大数据高并发的需求之下，Redis 集群方案应运而生。它可以将众多小内存的 Redis 实例综合起来，将分布在多台机器上的众多 CPU 核心的计算能力聚集到一起，完成海量数据存储和高并发读写操作。

Codis 是 Redis 集群方案之一，令我们感到骄傲的是，它是中国人开发并开源的，来自前豌豆荚中间件团队。绝大多数国内的开源项目都不怎么靠谱，但是 Codis 非常靠谱。有了 Codis 技术积累之后，项目「突头人」刘奇又开发出来中国人自己的开源分布式数据库 —— TiDB，可以说 6 到飞起。👍

从 Redis 的广泛流行到 RedisCluster 的广泛使用之间相隔了好多年，Codis 就是在这样的市场空缺的机遇下发展出来的。大型公司有明确的 Redis 在线扩容需求，但是市面上没有特别好的中间件可以做到这一点。

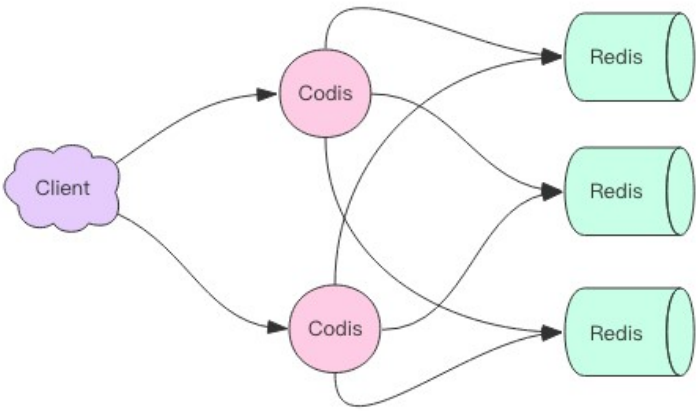


Codis 使用 Go 语言开发，它是一个代理中间件，它和 Redis 一样也使用 Redis 协议对外提供服务，当客户端向 Codis 发送指令时，Codis 负责将指令转发到后面的 Redis 实例来执行，并将返回结果再转回给客户端。

Codis 上挂接的所有 Redis 实例构成一个 Redis 集群，当集群空间不足时，可以通过动态增加 Redis 实例来实现扩容需求。

客户端操纵 Codis 同操纵 Redis 几乎没有区别，还是可以使用相同的客户端 SDK，不需要任何变化。

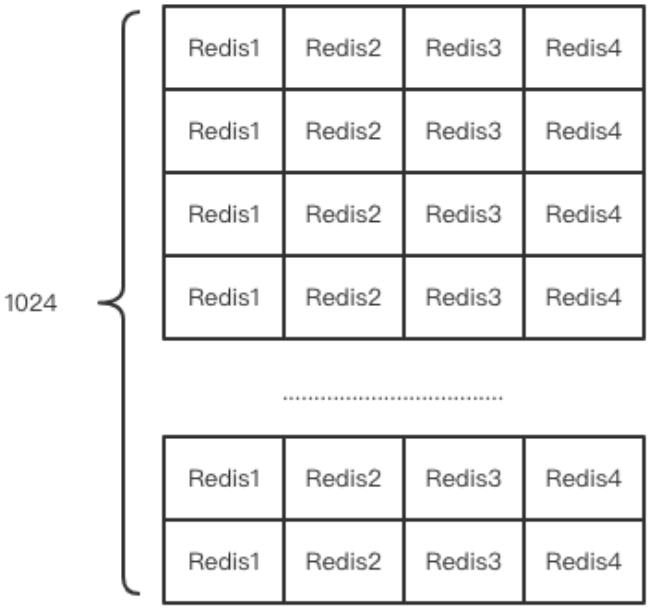
因为 Codis 是无状态的，它只是一个转发代理中间件，这意味着我们可以启动多个 Codis 实例，供客户端使用，每个 Codis 节点都是对等的。因为单个 Codis 代理能支撑的 QPS 比较有限，通过启动多个 Codis 代理可以显著增加整体的 QPS 需求，还能起到容灾功能，挂掉一个 Codis 代理没关系，还有很多 Codis 代理可以继续服务。



Codis 分片原理

Codis 要负责将特定的 key 转发到特定的 Redis 实例，那么这种对应关系 Codis 是如何管理的呢?

Codis 将所有的 key 默认划分为 1024 个槽位(slot)，它首先对客户端传过来的 key 进行 crc32 运算计算哈希值，再将 hash 后的整数值对 1024 这个整数进行取模得到一个余数，这个余数就是对应 key 的槽位。



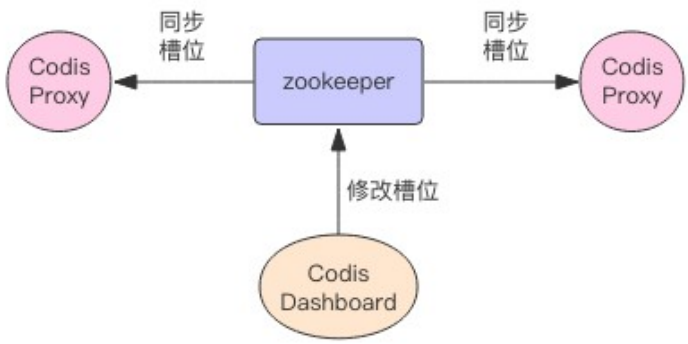
每个槽位都会唯一映射到后面的多个 Redis 实例之一，Codis 会在内存维护槽位和 Redis 实例的映射关系。这样有了上面 key 对应的槽位，那么它应该转发到哪个 Redis 实例就很明确了。

```py
hash = crc32(command.key)
slot_index = hash % 1024
redis = slots[slot_index].redis
redis.do(command)
```

槽位数量默认是1024，它是可以配置的，如果集群节点比较多，建议将这个数值配置大一些，比如2048、4096。

不同的 Codis 实例之间槽位关系如何同步?

如果 Codis 的槽位映射关系只存储在内存里，那么不同的 Codis 实例之间的槽位关系就无法得到同步。所以 Codis 还需要一个分布式配置存储数据库专门用来持久化槽位关系。Codis 开始使用 ZooKeeper，后来连 etcd 也一块支持了。



Codis 将槽位关系存储在 zk 中，并且提供了一个 Dashboard 可以用来观察和修改槽位关系，当槽位关系变化时，Codis Proxy 会监听到变化并重新同步槽位关系，从而实现多个 Codis Proxy 之间共享相同的槽位关系配置。

扩容

刚开始 Codis 后端只有一个 Redis 实例，1024 个槽位全部指向同一个 Redis。然后一个 Redis 实例内存不够了，所以又加了一个 Redis 实例。这时候需要对槽位关系进行调整，将一半的槽位划分到新的节点。这意味着需要对这一半的槽位对应的所有 key 进行迁移，迁移到新的 Redis 实例。

那 Codis 如果找到槽位对应的所有 key 呢?

Codis 对 Redis 进行了改造，增加了 SLOTSSCAN 指令，可以遍历指定 slot 下所有的 key。Codis 通过 SLOTSSCAN 扫描出待迁移槽位的所有的 key，然后挨个迁移每个 key 到新的 Redis 节点。

在迁移过程中，Codis 还是会接收到新的请求打在当前正在迁移的槽位上，因为当前槽位的数据同时存在于新旧两个槽位中，Codis 如何判断该将请求转发到后面的哪个具体实例呢?

Codis 无法判定迁移过程中的 key 究竟在哪个实例中，所以它采用了另一种完全不同的思路。当 Codis 接收到位于正在迁移槽位中的 key 后，会立即强制对当前的单个 key 进行迁移，迁移完成后，再将请求转发到新的 Redis 实例。

```
slot_index = crc32(command.key) % 1024
if slot_index in migrating_slots:
        do_migrate_key(command.key)  # 强制执行迁移
        redis = slots[slot_index].new_redis
else:
        redis = slots[slot_index].redis
redis.do(command)
```

我们知道 Redis 支持的所有 Scan 指令都是无法避免重复的，同样 Codis 自定义的 SLOTSSCAN 也是一样，但是这并不会影响迁移。因为单个 key 被迁移一次后，在旧实例中它就彻底被删除了，也就不可能会再次被扫描出来了。

自动均衡

Redis 新增实例，手工均衡slots太繁琐，所以 Codis 提供了自动均衡功能。自动均衡会在系统比较空闲的时候观察每个 Redis 实例对应的 Slots 数量，如果不平衡，就会自动进行迁移。

Codis 的代价

Codis 给 Redis 带来了扩容的同时，也损失了其它一些特性。因为 Codis 中所有的 key 分散在不同的 Redis 实例中，所以事务就不能再支持了，事务只能在单个 Redis 实例中完成。同样 rename 操作也很危险，它的参数是两个 key，如果这两个 key 在不同的 Redis 实例中，rename 操作是无法正确完成的。Codis 的官方文档中给出了一系列不支持的命令列表。

同样为了支持扩容，单个 key 对应的 value 不宜过大，因为集群的迁移的最小单位是 key，对于一个 hash 结构，它会一次性使用 hgetall 拉取所有的内容，然后使用 hmset 放置到另一个节点。如果 hash 内部的 kv 太多，可能会带来迁移卡顿。官方建议单个集合结构的总字节容量不要超过 1M。如果我们要放置社交关系数据，例如粉丝列表这种，就需要注意了，可以考虑分桶存储，在业务上作折中。

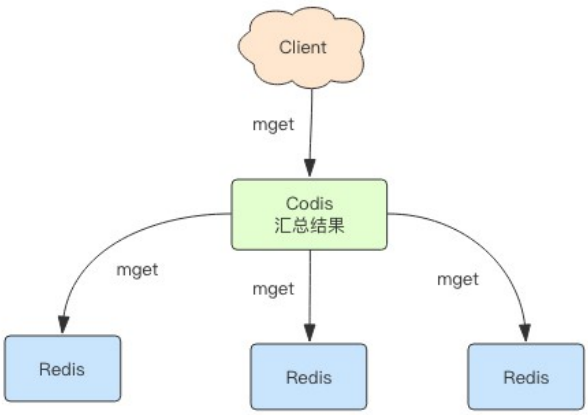
Codis 因为增加了 Proxy 作为中转层，所有在网络开销上要比单个 Redis 大，毕竟数据包多走了一个网络节点，整体在性能上要比单个 Redis 的性能有所下降。但是这部分性能损耗不是太明显，可以通过增加 Proxy 的数量来弥补性能上的不足。

Codis 的集群配置中心使用 zk 来实现，意味着在部署上增加了 zk 运维的代价，不过大部分互联网企业内部都有 zk 集群，可以使用现有的 zk 集群使用即可。

Codis 的优点

Codis 在设计上相比 Redis Cluster 官方集群方案要简单很多，因为它将分布式的问题交给了第三方 zk/etcd 去负责，自己就省去了复杂的分布式一致性代码的编写维护工作。而 Redis Cluster 的内部实现非常复杂，它为了实现去中心化，混合使用了复杂的 Raft 和 Gossip 协议，还有大量的需要调优的配置参数，当集群出现故障时，维护人员往往不知道从何处着手。

MGET 指令的操作过程



mget 指令用于批量获取多个 key 的值，这些 key 可能会分布在多个 Redis 实例中。Codis 的策略是将 key 按照所分配的实例打散分组，然后依次对每个实例调用 mget 方法，最后将结果汇总为一个，再返回给客户端。

架构变迁

Codis 作为非官方 Redis 集群方案，近几年来它的结构一直在不断变化，一方面当官方的 Redis 有变化的时候它要实时去跟进，另一方面它作为 Redis Cluster 的竞争方案之一，它还得持续提高自己的竞争力，给自己增加更多的官方集群所没有的便捷功能。

比如 Codis 有个特色的地方在于强大的 Dashboard 功能，能够便捷地对 Redis 集群进行管理。这是 Redis 官方所欠缺的。另外 Codis 还开发了一个 Codis-fe (federation 联邦) 工具，可以同时对多个 Codis 集群进行管理。在大型企业，Codis 集群往往会有几十个，有这样一个便捷的联邦工具可以降低不少运维成本。

Codis 的尴尬

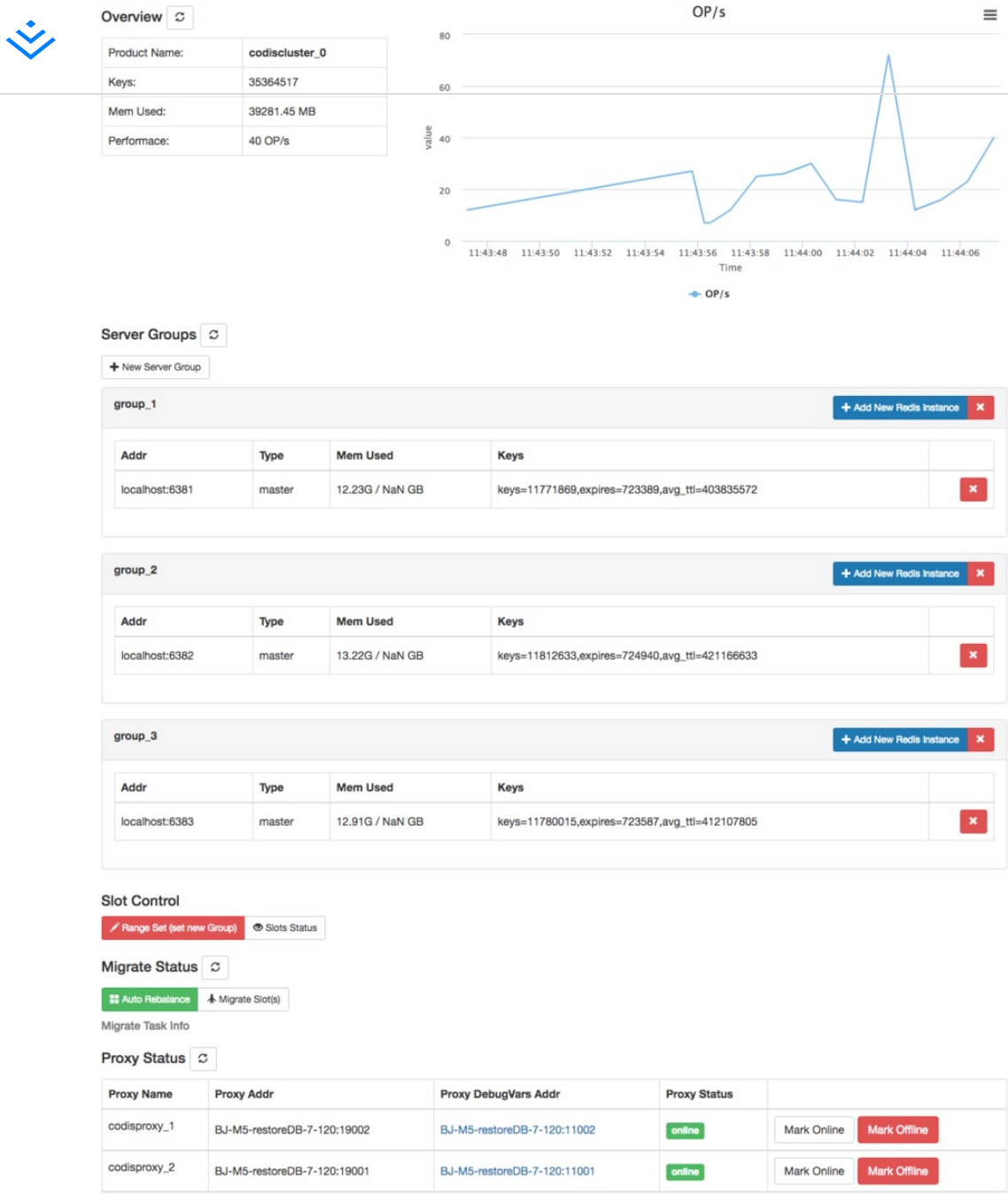
Codis 不是 Redis 官方项目，这意味着它的命运会无比曲折，它总是要被官方 Redis 牵着牛鼻子走。当 Redis 官方提供了什么功能它欠缺时，Codis 就会感到恐惧，害怕自己被市场甩掉，所以必须实时保持跟进。

同时因为 Codis 总是要比 Redis 官方慢一拍，Redis 官方提供的最新功能，Codis 往往要等很久才能同步。比如现在 Redis 已经进入到 4.0 阶段，提供了插件化 Redis-Module 支持，目前 Codis 还没有提供解决方案。

现在 Redis-Cluster 在业界已经逐渐流行起来，Codis 能否持续保持竞争力是个问题，我们看到 Codis 在不断的差异化竞争，竞争的方法就体现在工具上，而不是内核，这个和官方的路线真是相反的，官方对工具无暇顾及，只提供基本的工具，其它完全交给第三方去开发。

Codis 的后台管理

后台管理的界面非常友好，使用了最新的 BootStrap 前端框架。比较酷炫的是可以看到实时的 QPS 波动曲线。



同时还支持服务器集群管理功能，可以增加分组、增加节点、执行自动均衡等指令，还可以直接查看所有 slot 的状态，每个 slot 被分配到哪个 Redis 实例。

思考 & 作业

1. 请读者自己尝试搭建一个 Codis 集群。
2. 使用 Python 或者 Java 客户端体验一下 Codis 集群的常规 Redis 指令。