42 | 他山之石：etcd架构之美

2023-01-14 郑建勋 来自北京

天下无鱼
https://shikey.com/

《Go进阶 · 分布式爬虫实战》

课程介绍 >

讲述：郑建勋

时长 09:42　大小 8.86M

你好，我是郑建勋。

这节课，我们来聊一聊我们将在分布式项目中使用的重要中间件：etcd。

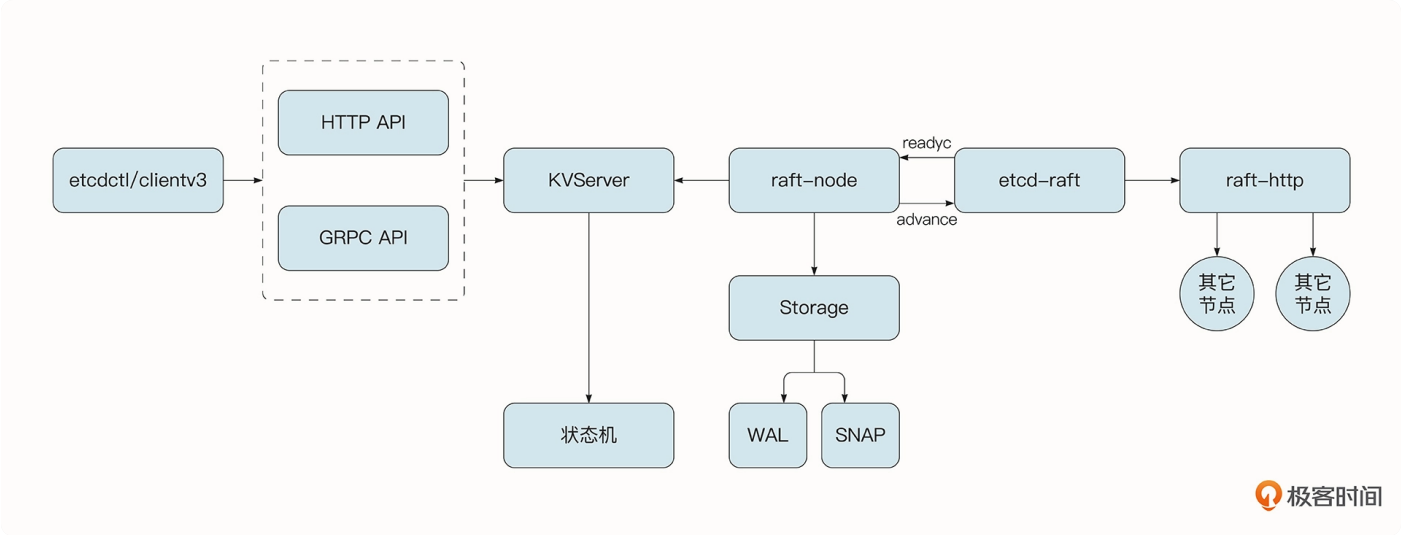
etcd 这个名字是 etc distributed 的缩写。我们知道，在 Linux 中 etc 目录存储了系统的配置文件，所以 etcd 代表了分布式的配置中心系统。然而，它能够实现的功能远不是同步配置文件这么简单。etcd 可以作为分布式协调的组件帮助我们实现分布式系统。

使用 etcd 的重要项目包括了 CoreOS 与 Kubernetes。etcd 使用 Go 书写，底层使用了 Raft 协议，它的架构本身非常优美。这节课就让我们来看一看 etcd 的架构、核心特性和实现机制，这样我们才能利用 etcd 更好地完成分布式协调工作，领会这个优秀的开源组件的设计哲学。同时，掌握 etcd 也有助于我们更深入地了解 Kubernetes 的运行机制。

etcd 全局架构

etcd 的第一个版本 v0.1 于 2013 年发布，现在已经更新到了 v3，在这个过程中，etcd 的稳定性、扩展性、性能都在不断提升。我们这节课主要讨论的是当前最新的 v3 版本。话不多说，我们先来从整体上看一看 etcd 的架构。

天下无鱼 https://shikey.com/



etcd 从大的方面可以分为几个部分，让我们结合图片从右往左说起。

首先 etcd 抽象出了 raft-http 模块，由于 etcd 通常为分布式集群部署方式，该层用于处理和其他 etcd 节点的网络通信。etcd 内部使用了 HTTP 协议来进行通信，由于 etcd 中的消息类型很多，心跳探活的数据量较小，快照信息较大（可达 GB 级别），所以 etcd 有两种处理消息的通道，分别是 Pipeline 消息通道与 Stream 消息通道。

Stream 消息通道是 etcd 中节点与节点之间维护的长连接，它可以处理频繁的小消息，还能复用 HTTP 底层的连接，不必每次都建立新的连接。而 Pipeline 消息通道用于处理快照这样数据量大的信息，处理完毕后连接会关闭。

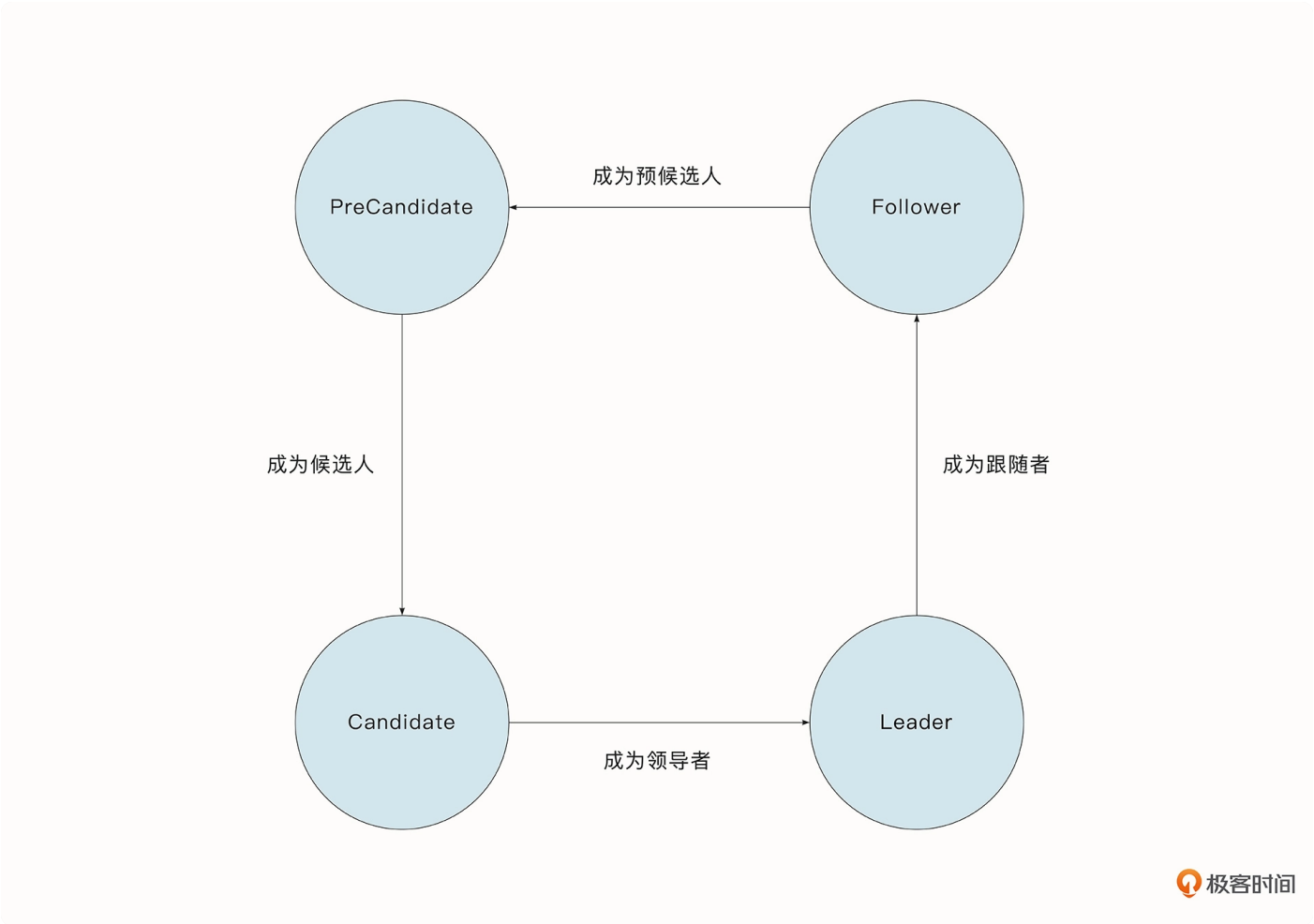
再往左是 etcd-raft 模块，它是 etcd 的核心。该层实现了 Raft 协议，可以完成节点状态的转移、节点的选举、数据处理等重要功能，确保分布式系统的一致性与故障容错性。之前我们也介绍过，Raft 中的节点有 3 种状态，分别是 Leader（领导者），Candidates（候选人）和 Follower（跟随者）。在此基础上，etcd 为 Raft 节点新增了一个 PreCandidate（预候选人）。

我们在讲解 Raft 协议时介绍过，如果节点收不到来自 Leader 的心跳检测，就会变为 Candidates 开始新的选举。如果当前节点位于不足半数的网络分区中，短期内不会影响集群的使用，但是当前节点在不断发起选举的过程中，当前选举周期的 Term 号会不断增长，当网

络分区消失后，由于该节点的 Term 号高于当前集群中 Leader 节点的 Term 号，Raft 协议就会迫使当前的 Leader 切换状态并开始新一轮的选举。

天下无鱼
https://shikey.com/

但是，这种选举是没有意义的。为了解决这样的问题，etcd 在选举之前会有一个新的阶段叫做 PreVote，当前节点会先尝试连接集群中的其他节点，如果能够成功连接到半数以上的节点，才开始新一轮的选举。



Raft节点状态转移

在 etcd-raft 模块基础之上还封装出了 raft-node 模块。这个模块主要是上层模块和下层 Raft 模块沟通的桥梁，它同时还有一个重要任务，就是调用 storage 模块，将记录（Record）存储到 WAL 日志文件中落盘。WAL 日志文件可以存储多种类型的记录，包括下面几种。

- WAL 文件的元数据，记录节点 ID、集群 ID 信息。
- Entry 记录，即客户端发送给服务器处理的数据。
- 集群的状态信息，包含集群的任期号、节点投票信息。
- 数据校验信息，它可以校验文件数据的完整性与正确性。

- 快照信息，包含快照的相关信息，但不包含实际的快照数据。它可以校验快照数据的完整性。

天下无鱼
https://shikey.com/

复制代码

```
type Record struct {
  Type                 int64    `protobuf:"varint,1,opt,name=type" json:"type"`
  Crc                  uint32   `protobuf:"varint,2,opt,name=crc" json:"crc"`
  Data                 []byte   `protobuf:"bytes,3,opt,name=data" json:"data,om
}
```

WAL 日志文件非常重要，它能保证我们的消息在大部分节点达成一致且应用到状态机之前，让记录持久化。这样，在节点崩溃并重启后，就能够从 WAL 中恢复数据了。

WAL 日志的数量与大小随着时间不断增加，可能超过可容纳的磁盘容量。同时，在节点宕机后，如果要恢复数据就必须从头到尾读取全部的 WAL 日志文件，耗时也会非常久。为了解决这一问题，etcd 会定期地创建快照并保存到文件中，在恢复节点时会先加载快照数据，并从快照所在的位置之后读取 WAL 文件，这就加快了节点的恢复速度。快照的数据也有单独的 snap 模块进行管理。

在 raft-node 模块之上是 etcd-server 模块，它最核心的任务是执行 Entry 对应的操作，在这个过程中包含了限流操作与权限控制的能力。所有操作的集合最终会使状态机到达最新的状态。etcd-server 同时还会维护当前 etcd 集群的状态信息，并提供了线性读的能力。

etcd-server 提供了一些供外部访问的 GRPC API 接口，同时 etcd 也使用了 GRPC-gateway 作为反向代理，使服务器也有能力对外提供 HTTP 协议。

最后，etcd 还提供了客户端工具 etcdctl 和 clientv3 代码库，使用 GRPC 协议与 etcd 服务器交互。客户端支持负载均衡、节点间故障自动转移等机制，极大降低了业务使用 etcd 的难度，提升了开发的效率。

此外，etcd 框架中还有一些辅助的功能，例如权限管理、限流管理、重试、GRPC 拦截器等。由于不是核心点，图中并没有一一列举出来。

etcd 架构的优点

etcd 自身的架构与实现有许多值得借鉴的地方。**例如它的高内聚、低耦合、高性能，还有它优雅的数据同步。**

天下无鱼
https://shikey.com/

- **高内聚**

从 etcd 整体的架构图上可以看出，etcd 将相关的核心功能（例如鉴权、网络、Raft 协议）都聚合起来形成了一个单独的模块。功能之间联系紧密，并且只提供核心的接口与外部进行交互。这非常便于理解与开发，也便于后期对功能进行组合。

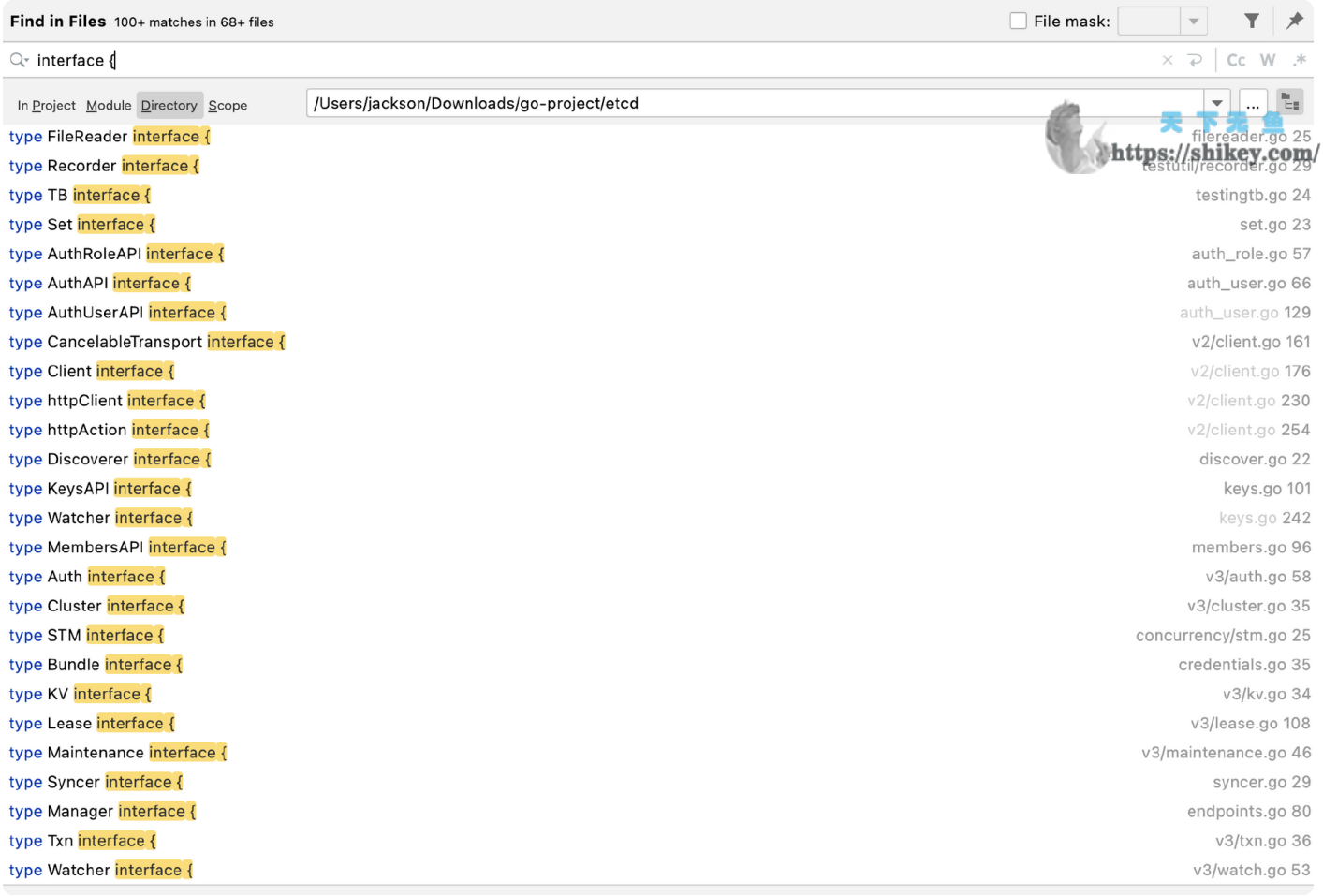
- **低耦合**

各个模块之间边界清晰，用接口来进行交流与组合的设计给了程序极大的扩展性。举一个例子，数据存储的 store 就是一个 interface。

复制代码

```
type Storage interface {
  InitialState() (pb.HardState, pb.ConfState, error)
  Entries(lo, hi, maxSize uint64) ([]pb.Entry, error)
  Term(i uint64) (uint64, error)
  LastIndex() (uint64, error)
  FirstIndex() (uint64, error)
  Snapshot() (pb.Snapshot, error)
}
```

简单地搜索一下，可以发现 etcd 内部使用了大量接口。



天下无鲨 https://shikey.com/

这样做的好处是，etcd 抽象出了 etcd-raft 这个模块，实现了 Raft 协议，开发者可以在这个模块之上构建实现了 Raft 协议的分布式系统，这就减轻了开发者的心理和技术负担。因为分布式协议是非常难实现的，它潜在的问题也很难被发现，发现问题之后又很难定位原因，有了 etcd-raft 模块，这一系列问题都迎刃而解了。

在 etcd 代码库中有一个示例代码，该示例代码基于 etcd-raft 模块实现了一个最简单的分布式内存 KV 数据库。在示例代码中实现了上游的 KVServer 服务器与 raft-node 节点，并与 etcd-raft 模块进行交互，去掉了 etcd 实现的日志落盘等逻辑，将键值对存储到了内存中。如果你有志于深入地学习 etcd，从这个实例入手是非常不错的选择。

- **优雅的数据同步**

在 etcd 中，我们极少看到使用互斥锁的场景。更多的时候，它是借助协程与通道的相互配合来传递信息的，这就既完成了通信又优雅地解决了并发安全问题。

- **更快的读取性能**

etcd 在 etcd-raft 模块中实现了 Raft 协议。我们知道 Raft 并不能够保证读取的线性一致性，也就是说，它有可能读取到过时的数据。

天下无鱼
https://shikey.com/

怎么解决呢？办法有很多。例如，Follower 可以将读请求直接转发给 Leader，不过这样的话 Leader 的压力会很大，并且 Leader 可能已经不是最新的 Leader 了。

第二种解决方案是 etcdv3.2 之前的处理方式。也就是将该请求记录为一个 Entry，从而借助 Raft 机制来保证读到的数据是最新的。

还有一种更轻量级的方法。在 v3.2 之后，etcd 实现了 ReadIndex 机制，这也是在 Raft 论文当中提到过的。Follower 向 Leader 读取当前最新的 Commit Index，同时 Leader 需要确保自己没有被未知的新 Leader 取代。它会发出新一轮的心跳，并等待集群中大多数节点的确认。一旦收到确认信息，Leader 就知道在发送心跳信息的那一刻，不可能存在更新的 Leader 了。也就是说，在那一刻，ReadIndex 是集群中所有节点见过的最大的 Commit Index。Follower 会在自己的状态机上将日志至少执行到该 Commit Index 之后，然后查询当前状态机生成的结果，并返回结果给客户端。

- **可靠的 Watch 机制与高性能的并发处理**

相对于 etcdv2，etcdv3 版本将所有键值对的历史版本都存储了起来，这就让 Watch 机制的可靠性更高了，它实现的 MVCC 机制也提高了系统处理并发请求的数量。

总结

etcd 是用 Go 语言编写的分布式键值存储系统，它的应用广泛而且架构优美。etcd 拥有高内聚和低耦合的特点，它的内部分为了清晰的 etcd-raft 模块、etcd-http 模块，etcd-node 模块、etcd-server 模块等。并且，模块之间可以用清晰的接口进行交流，这就保证了系统的扩展性与可组合性，我们可以在 etcd-raft 模块的基础上快速构建起自己的分布式系统。

etcd 实现了 Raft 协议，并且在此基础上实现了 PreVote 机制与线性读机制，再加上 WAL 日志文件的落盘与快照，最大程度保证了服务的一致性与出现故障时的容错性。

etcd 中的代码实践了 CSP 的编程模式，大量使用了协程与通道的机制来进行通信，对超时的处理、资源的释放、并发的处理都比较优雅，是我们学习 Raft 协议和 Go 语言程序设计比较好的资料。

课后题

学完这节课，还是照例给你留一道思考题。

和 MySQL 一样，etcd 也使用了 MVCC 的机制，请你查阅资料，说明什么是 MVCC，他能解决什么复杂的问题？

欢迎你在留言区与我交流讨论，我们下节课见。

分享给需要的人，Ta购买本课程，你将得 20 元

生成海报并分享

赞 2　提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　41 | 线上综合案例：节约线上千台容器的性能分析实战

下一篇　43 | 分布式协调：etcd读写、MVCC原理与监听机制

精选留言 (1)

写留言

Realm

2023-01-14 来自浙江

思考题：

1. MVCC（Multi-Version Concurrency Control），即多版本并发控制。MVCC 是一种并发控制的方法，可以实现对数据库的并发访问。

2. MySQL的MVCC工作在RC(读提交)和RR(重复读)的隔离级别。
表的行记录逻辑上是一个链表，既保留业务数据本身，还有两个隐藏字段：
- trx_id（最近修改的事务ID）
- roll_ptr(指向上一个版本数据的指针,通过undo log可以实现从高版本到低版本的迁跃)

3. ETCD的MVCC同样可以维护一个数据(key对应的值)的多个历史版本，且使得读写操作没有冲突,不使用锁，增加系统吞吐。

4. 窥探etcd对同一个key进行修改，内部版本的变化

```
> docker exec etcd-gcr-v3.5.5 /bin/sh -c "/usr/local/bin/etcdctl put a 1 "
OK

> docker exec etcd-gcr-v3.5.5 /bin/sh -c "/usr/local/bin/etcdctl get a -w=json"
{"header":{"cluster_id":18011104697467366872,"member_id":6460912315094810421,"revision":22,"raft_term":3},"kvs":[{"key":"YQ==","create_revision":22,"mod_revision":22,"version":1,"value":"MQ=="}],"count":1}

> docker exec etcd-gcr-v3.5.5 /bin/sh -c "/usr/local/bin/etcdctl put a 2 "
OK

> docker exec etcd-gcr-v3.5.5 /bin/sh -c "/usr/local/bin/etcdctl get a -w=json"
{"header":{"cluster_id":18011104697467366872,"member_id":6460912315094810421,"revision":23,"raft_term":3},"kvs":[{"key":"YQ==","create_revision":22,"mod_revision":23,"version":2,"value":"Mg=="}],"count":1}


> docker exec etcd-gcr-v3.5.5 /bin/sh -c "/usr/local/bin/etcdctl put a 3 "
OK

> docker exec etcd-gcr-v3.5.5 /bin/sh -c "/usr/local/bin/etcdctl get a -w=json"
{"header":{"cluster_id":18011104697467366872,"member_id":6460912315094810421,"revision":24,"raft_term":3},"kvs":[{"key":"YQ==","create_revision":22,"mod_revision":24,"version":3,"value":"Mw=="}],"count":1}
```

天下无鱼 https://shikey.com/