



23 | 选型: etcd/ZooKeeper/Consul等我们该如何选择?

2021-03-15 唐聪

etcd实战课

进入课程 >



讲述: 王超凡

时长 16:35 大小 15.20M



你好，我是唐聪。

在软件开发过程中，当我们需要解决配置、服务发现、分布式锁等业务痛点，在面对 [etcd](#)、[ZooKeeper](#)、[Consul](#)、[Nacos](#) 等一系列候选开源项目时，我们应该如何结合自己的业务场景，选择合适的分布式协调服务呢？

今天，我就和你聊聊主要分布式协调服务的对比。我将从基本架构、共识算法、数据模型、重点特性、容灾能力等维度出发，带你了解主要分布式协调服务的基本原理和彼此之间的差异性。



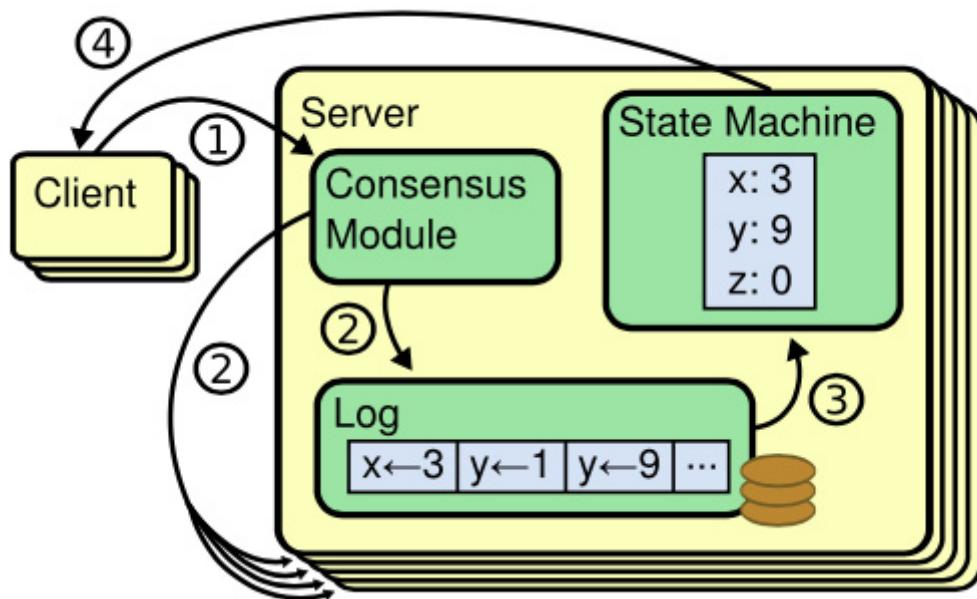
希望通过这节课，让你对 etcd、ZooKeeper、Consul 原理和特性有一定的理解，帮助你选型适合业务场景的配置系统、服务发现组件。

基本架构及原理

在详细和你介绍对比 etcd、ZooKeeper、Consul 特性之前，我们先从整体架构上来了解一下各开源项目的核心架构及原理。

etcd 架构及原理

首先是 etcd，etcd 我们知道它是基于复制状态机实现的分布式协调服务。如下图所示，由 Raft 共识模块、日志模块、基于 boltdb 持久化存储的状态机组成。



以下是 etcd 基于复制状态机模型的写请求流程：

client 发起一个写请求 (put x = 3)；

etcdserver 模块向 Raft 共识模块提交请求，共识模块生成一个写提案日志条目。若 server 是 Leader，则把日志条目广播给其他节点，并持久化日志条目到 WAL 中；

当一半以上节点持久化日志条目后，Leader 的共识模块将此日志条目标记为已提交 (committed)，并通知其他节点提交；

etcdserver 模块从 Raft 共识模块获取已经提交的日志条目，异步应用到 boltdb 状态机存储中，然后返回给 client。

更详细的原理我就不再重复描述，你可以参考 [02读](#) 和 [03写](#) 两节原理介绍。

ZooKeeper 架构及原理

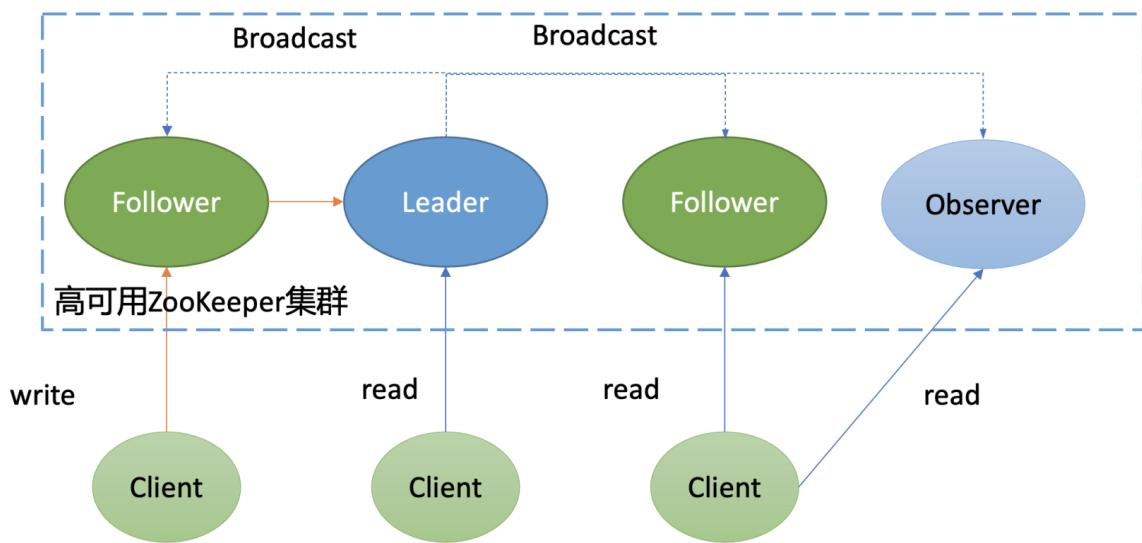
接下来我和你简要介绍下  ZooKeeper 原理，下图是它的架构图。

如下面架构图所示，你可以看到 ZooKeeper 中的节点与 etcd 类似，也划分为 Leader 节点、Follower 节点、Observer 节点（对应的 Raft 协议的 Learner 节点）。同时，写请求统一由 Leader 处理，读请求各个节点都能处理。

不一样的是它们的读行为和共识算法。

在读行为上，ZooKeeper 默认读可能会返回 stale data，而 etcd 使用的线性读，能确保读取到反应集群共识的最新数据。

共识算法上，etcd 使用的是 Raft，ZooKeeper 使用的是 Zab。



那什么是 Zab 协议呢？

Zab 协议可以分为以下阶段：

Phase 0, Leader 选举 (Leader Election)。一个节点只要求获得半数以上投票，就可以当选为准 Leader；

Phase 1, 发现 (Discovery)。准 Leader 收集其他节点的数据信息，并将最新的数据复制到自身；

Phase 2, 同步 (Synchronization)。准 Leader 将自身最新数据复制给其他落后的节点，并告知其他节点自己正式当选为 Leader;

Phase 3, 广播 (Broadcast)。Leader 正式对外服务，处理客户端写请求，对消息进行广播。当收到一个写请求后，它会生成 Proposal 广播给各个 Follower 节点，一半以上 Follower 节点应答之后，Leader 再发送 Commit 命令给各个 Follower，告知它们提交相关提案；

ZooKeeper 是如何实现的 Zab 协议的呢？

ZooKeeper 在实现中并未严格按 [Zab 协议论文](#) 定义的分阶段实现，而是对部分阶段进行了整合，分别如下：

Fast Leader Election。首先 ZooKeeper 使用了一个名为 Fast Leader Election 的选举算法，通过 Leader 选举安全规则限制，确保选举出来的 Leader 就含有最新数据，避免了 Zab 协议的 Phase 1 阶段准 Leader 收集各个节点数据信息并复制到自身，也就是将 Phase 0 和 Phase 1 进行了合并。

Recovery Phase。各个 Follower 发送自己的最新数据信息给 Leader，Leader 根据差异情况，选择发送 SNAP、DIFF 差异数据、Truncate 指令删除冲突数据等，确保 Follower 追赶上 Leader 数据进度并保持一致。

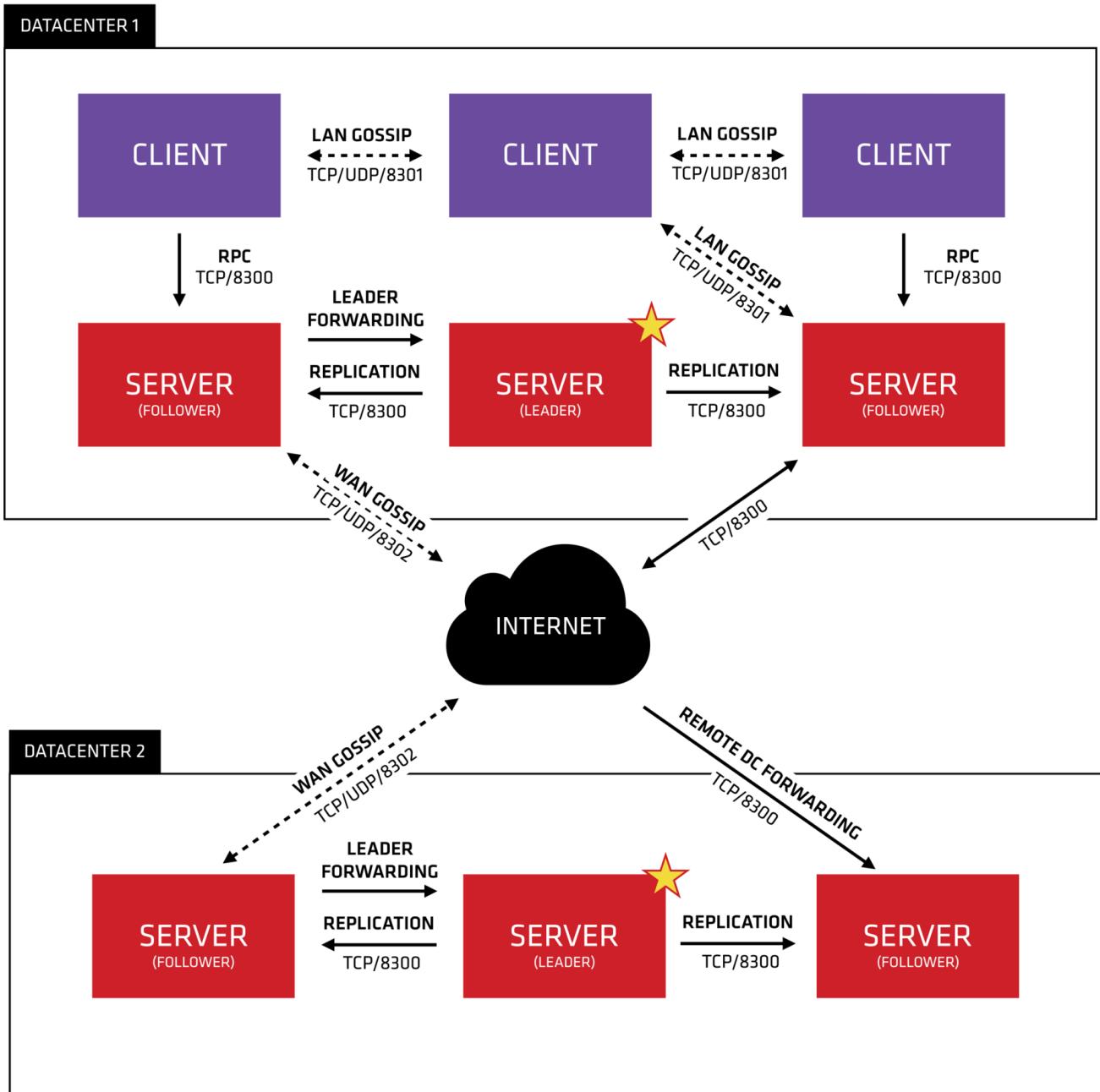
Broadcast Phase。与 Zab 论文 Broadcast Phase 一致。

总体而言，从分布式系统 CAP 维度来看，ZooKeeper 与 etcd 类似的是，它也是一个 CP 系统，在出现网络分区等错误时，它优先保障的数据一致性，牺牲的是 A 可用性。

Consul 架构及原理

了解完 ZooKeeper 架构及原理后，我们再看看 Consul，它的架构和原理是怎样的呢？

下图是 [Consul 架构图](#)（引用自 HashiCorp 官方文档）。



从图中你可以看到，它由 Client、Server、Gossip 协议、Raft 共识算法、两个数据中心组成。每个数据中心内的 Server 基于 Raft 共识算法复制日志，Server 节点分为 Leader、Follower 等角色。Client 通过 Gossip 协议发现 Server 地址、分布式探测节点健康状态等。

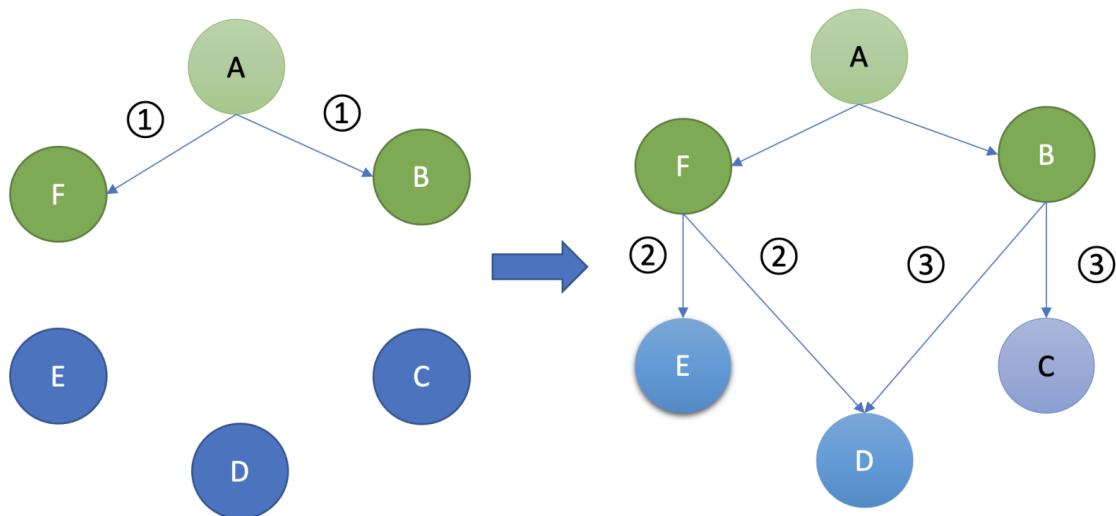
那什么是 Gossip 协议呢？

Gossip 中文名称叫流言协议，它是一种消息传播协议。它的核心思想其实源自我们生活中的八卦、闲聊。我们在日常生活中所看到的劲爆消息其实源于两类，一类是权威机构如国

家新闻媒体发布的消息，另一类则是大家通过微信等社交聊天软件相互八卦，一传十，十传百的结果。

Gossip 协议的基本工作原理与我们八卦类似，在 Gossip 协议中，如下图所示，各个节点会周期性地选择一定数量节点，然后将消息同步给这些节点。收到消息后的节点同样做出类似的动作，随机的选择节点，继续扩散给其他节点。

最终经过一定次数的扩散、传播，整个集群的各个节点都能感知到此消息，各个节点的数据趋于一致。Gossip 协议被广泛应用在多个知名项目中，比如 Redis Cluster 集群版，Apache Cassandra，AWS Dynamo。



了解完 Gossip 协议，我们再看看架构图中的多数据中心，Consul 支持数据跨数据中心自动同步吗？

你需要注意的是，虽然 Consul 天然支持多数据中心，但是多数据中心内的服务数据并不会跨数据中心同步，各个数据中心的 Server 集群是独立的。不过，Consul 提供了 [Prepared Query](#) 功能，它支持根据一定的策略返回多数据中心下的最佳的服务实例地址，使你的服务具备跨数据中心容灾。

比如当你的 API 网关收到用户请求查询 A 服务，API 网关服务优先从缓存中查找 A 服务对应的最佳实例。若无缓存则向 Consul 发起一个 Prepared Query 请求查询 A 服务实例，Consul 收到请求后，优先返回本数据中心下的服务实例。如果本数据中心没有或异常则根

据数据中心间 RTT 由近到远查询其它数据中心数据，最终网关可将用户请求转发给最佳的数据中心下的实例地址。

了解完 Consul 的 Gossip 协议、多数据中心支持，我们再看看 Consul 是如何处理读请求的呢？

Consul 支持以下三种模式的读请求：

默认 (default)。默认是此模式，绝大部分场景下它能保证数据的强一致性。但在老的 Leader 出现网络分区被隔离、新的 Leader 被选举出来的一个极小时间窗口内，可能会导致 stale read。这是因为 Consul 为了提高读性能，使用的是基于 Lease 机制来维持 Leader 身份，避免了与其他节点进行交互确认的开销。

强一致性 (consistent)。强一致性读与 etcd 默认线性读模式一样，每次请求需要集群多数节点确认 Leader 身份，因此相比 default 模式读，性能会有所下降。

弱一致性 (stale)。任何节点都可以读，无论它是否 Leader。可能读取到陈旧的数据，类似 etcd 的串行读。这种读模式不要求集群有 Leader，因此当集群不可用时，只要有节点存活，它依然可以响应读请求。

重点特性比较

初步了解完 etcd、ZooKeeper、Consul 架构及原理后，你可以看到，他们都是基于共识算法实现的强一致的分布式存储系统，并都提供了多种模式的读机制。

除了以上共性，那么它们之间有哪些差异呢？下表是 etcd 开源社区总结的一个 [详细对比项](#)，我们就从并发原语、健康检查及服务发现、数据模型、Watch 特性等功能上详细比较下它们功能和区别。

	etcd	ZooKeeper	Consul	NewSQL (Cloud Spanner, CockroachDB, TiDB)
Concurrency Primitives	Lock RPCs, Election RPCs, command line locks, command line	External cur ator recipes in	Native lock API	Rare, if any

	elections, recipes in go	Java		
Linearizable Reads	Yes	No	Yes	Sometimes
Multi-version Concurrency Control	Yes	No	No	Sometimes
Transactions	Field compares, Read, Write	Version checks, Write	Field compare, Lock, Read, Write	SQL-style
Change Notification	Historical and current key intervals	Current keys and directories	Current keys and prefixes	Triggers (sometimes)
User permissions	Role based	ACLs	ACLs	Varies (per-table GRANT, per-database roles)
HTTP/JSON API	Yes	No	Yes	Rarely
Membership Reconfiguration	Yes	>3.5.0	Yes	Yes
Maximum reliable database size	Several gigabytes	Hundreds of megabytes (sometimes several gigabytes)	Hundreds of MBs	Terabytes+
Minimum read linearization latency	Network RTT	No read linearization	RTT + fsync	Clock barriers (atomic, NTP)

并发原语

etcd 和 ZooKeeper、Consul 的典型应用场景都是分布式锁、Leader 选举，以上场景就涉及到并发原语控制。然而 etcd 和 ZooKeeper 并未提供原生的分布式锁、Leader 选举支持，只提供了核心的基本数据读写、并发控制 API，由应用上层去封装。

为了帮助开发者更加轻松的使用 etcd 去解决分布式锁、Leader 选举等问题，etcd 社区提供了 [concurrency 包](#) 来实现以上功能。同时，在 etcdserver 中内置了 Lock 和 Election 服务，不过其也是基于 concurrency 包做了一层封装而已，clientv3 并未提供 Lock 和 Election 服务 API 给 Client 使用。ZooKeeper 所属的 Apache 社区提供了 [Apache Curator Recipes](#) 库来帮助大家快速使用分布式锁、Leader 选举功能。

相比 etcd、ZooKeeper 依赖应用层基于 API 上层封装，Consul 对分布式锁就提供了 [原生的支持](#)，可直接通过命令行使用。

总体而言，etcd、ZooKeeper、Consul 都能解决分布式锁、Leader 选举的痛点，在选型时，你可能会重点考虑其提供的 API 语言是否与业务服务所使用的语言一致。

健康检查、服务发现

分布式协调服务的另外一个核心应用场景是服务发现、健康检查。

与并发原语类似，etcd 和 ZooKeeper 并未提供原生的服务发现支持。相反，Consul 在服务发现方面做了很多解放用户双手的工作，提供了服务发现的框架，帮助你的业务快速接入，并提供了 HTTP 和 DNS 两种获取服务方式。

比如下面就是通过 DNS 的方式获取服务地址：

 复制代码

```
1 $ dig @127.0.0.1 -p 8600 redis.service.dc1.consul. ANY
```

最重要的是它还集成了分布式的健康检查机制。与 etcd 和 ZooKeeper 健康检查不一样的是，它是一种基于 client、Gossip 协议、分布式的健康检查机制，具备低延时、可扩展的特点。业务可通过 Consul 的健康检查机制，实现 HTTP 接口返回码、内存乃至磁盘空间的检测。

Consul 提供了  多种机制给你注册健康检查，如脚本、HTTP、TCP 等。

脚本是怎么工作的呢？介绍 Consul 架构时，我们提到过的 Agent 角色的任务之一就是执行分布式的健康检查。

比如你将如下脚本放在 Agent 相应目录下，当 Linux 机器内存使用率超过 70% 的时候，它会返回告警状态。

 复制代码

```
1  {
2    "check": {
3      "id": "mem-util"
4      "name": "Memory utilization"
5      "args": [
6        "/bin/sh"
7        "-c"
8        "/usr/bin/free | awk '/Mem/{printf($3/$2*100)}' | awk '{ print($0); if($
9      ]
10     "interval": "10s"
11     "timeout": "1s"
12   }
13 }
```

相比 Consul, etcd、ZooKeeper 它们提供的健康检查机制和能力就非常有限了。

etcd 提供了 Lease 机制来实现活性检测。它是一种中心化的健康检查，依赖用户不断地发送心跳续租、更新 TTL。

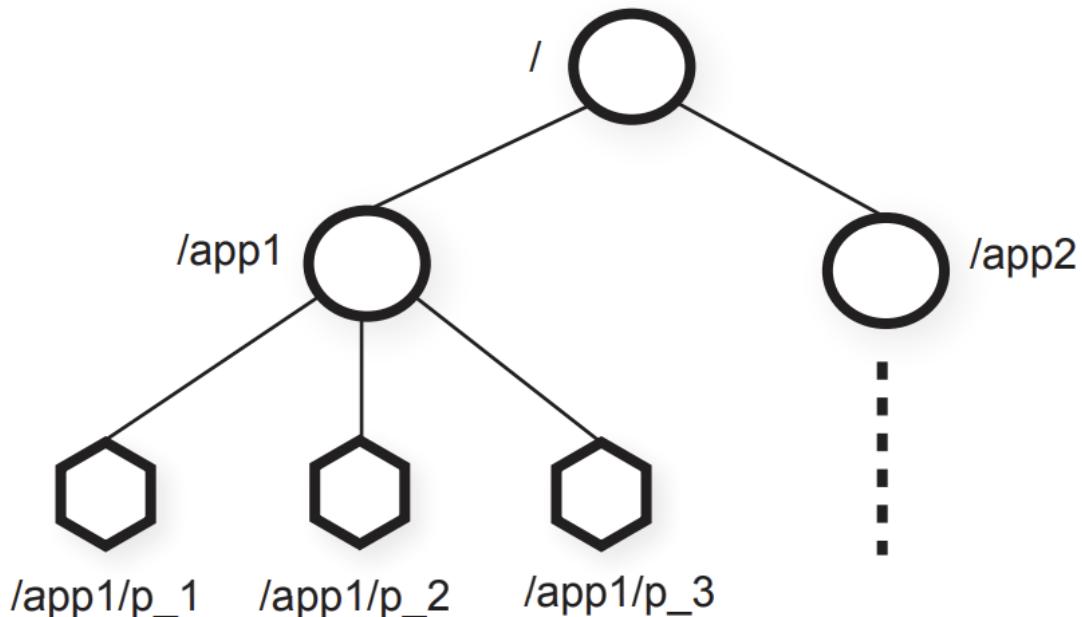
ZooKeeper 使用的是一种名为临时节点的状态来实现健康检查。当 client 与 ZooKeeper 节点连接断掉时，ZooKeeper 就会删除此临时节点的 key-value 数据。它比基于心跳机制更复杂，也给 client 带去了更多的复杂性，所有 client 必须维持与 ZooKeeper server 的活跃连接并保持存活。

数据模型比较

从并发原语、健康检查、服务发现等维度了解完 etcd、ZooKeeper、Consul 的实现区别之后，我们再从数据模型上对比下三者。

首先 etcd 正如我们在 [07节 MVCC](#) 和 [10节 boltdb](#) 所介绍的，它是个扁平的 key-value 模型，内存索引通过 B-tree 实现，数据持久化存储基于 B+ tree 的 boltdb，支持范围查询、适合读多写少，可容纳数 G 的数据。

[ZooKeeper 的数据模型](#)如下。



如上图所示，它是一种层次模型，你可能已经发现，etcd v2 的内存数据模型与它是一样的。ZooKeeper 作为分布式协调服务的祖师爷，早期 etcd v2 的确就是参考它而设计的。

ZooKeeper 的层次模型中的每个节点叫 Znode，它分为持久性和临时型两种。

持久性顾名思义，除非你通过 API 删除它，否则它将永远存在。

临时型是指它与客户端会话绑定，若客户端会话结束或出现异常中断等，它都将被 ZooKeeper server 自动删除，被广泛应用于活性检测。

同时你创建节点的时候，还可以指定一个顺序标识，这样节点名创建出来后就具有顺序性，一般应用于分布式选举等场景中。

那 ZooKeeper 是如何实现以上层次模型的呢？

ZooKeeper 使用的是内存 ConcurrentHashMap 来实现此数据结构，因此具有良好的读性能。但是受限于内存的瓶颈，一般 ZooKeeper 的数据库文件大小是几百 M 左右。

Consul 的数据模型及存储是怎样的呢？

它也提供了常用 key-value 操作，它的存储引擎是基于 [Radix Tree](#) 实现的 [go-memdb](#)，要求 value 大小不能超过 512 个字节，数据库文件大小一般也是几百 M 左右。与 boltdb 类似，它也支持事务、MVCC。

Watch 特性比较

接下来我们再看看 Watch 特性的比较。

正在我在 08 节 Watch 特性中所介绍的，etcd v3 的 Watch 是基于 MVCC 机制实现的，而 Consul 是采用滑动窗口实现的。Consul 存储引擎是基于 [Radix Tree](#) 实现的，因此它不支持范围查询和监听，只支持前缀查询和监听，而 etcd 都支持。

相比 etcd、Consul，ZooKeeper 的 Watch 特性有更多的局限性，它是个一次性触发器。

在 ZooKeeper 中，client 对 Znode 设置了 Watch 时，如果 Znode 内容发生改变，那么 client 就会获得 Watch 事件。然而此 Znode 再次发生变化，那 client 是无法收到 Watch 事件的，除非 client 设置了新的 Watch。

其他比较

最后我们再从其他方面做些比较。

线性读。etcd 和 Consul 都支持线性读，而 ZooKeeper 并不具备。

权限机制比较。etcd 实现了 RBAC 的权限校验，而 ZooKeeper 和 Consul 实现的 ACL。

事务比较。etcd 和 Consul 都提供了简易的事务能力，支持对字段进行比较，而 ZooKeeper 只提供了版本号检查能力，功能较弱。

多数据中心。在多数据中心支持上，只有 Consul 是天然支持的，虽然它本身不支持数据自动跨数据中心同步，但是它提供的服务发现机制、[Prepared Query](#)功能，赋予了业务在一个可用区后端实例故障时，可将请求转发到最近的数据中心实例。而 etcd 和 ZooKeeper 并不支持。

小结

最后我们来小结下今天的内容。首先我和你从顶层视角介绍了 etcd、ZooKeeper、Consul 基本架构及核心原理。

从共识算法角度上看，etcd、Consul 是基于 Raft 算法实现的数据复制，ZooKeeper 则是基于 Zab 算法实现的。Raft 算法由 Leader 选举、日志同步、安全性组成，而 Zab 协议则由 Leader 选举、发现、同步、广播组成。无论 Leader 选举还是日志复制，它们都需要集群多数节点存活、确认才能继续工作。

从 CAP 角度上看，在发生网络分区时，etcd、Consul、ZooKeeper 都是一个 CP 系统，无法写入新数据。同时，etcd、Consul、ZooKeeper 提供了各种模式的读机制，总体上可分为强一致性读、非强一致性读。

其中 etcd 和 Consul 则提供了线性读，ZooKeeper 默认是非强一致性读，不过业务可以通过 sync() 接口，等待 Follower 数据追赶上 Leader 进度，以读取最新值。

接下来我从并发原语、健康检查、服务发现、数据模型、Watch 特性、多数据中心比较等方面和你重点介绍了三者的实现与区别。

其中 Consul 提供了原生的分布式锁、健康检查、服务发现机制支持，让业务可以更省心，不过 etcd 和 ZooKeeper 也都有相应的库，帮助你降低工作量。Consul 最大的亮点则是对多数据中心的支持。

最后如果业务使用 Go 语言编写的，国内一般使用 etcd 较多，文档、书籍、最佳实践案例丰富。Consul 在国外应用比较多，中文文档及实践案例相比 etcd 较少。ZooKeeper 一般是 Java 业务使用较多，广泛应用在大数据领域。另外 Nacos 也是个非常优秀的开源项目，支持服务发现、配置管理等，是 Java 业务的热门选择。

思考题

好了，这节课到这里也就结束了，最后我给你留了一个思考题。

越来越多的业务要求跨可用区乃至地区级的容灾，如果你是核心系统开发者，你会如何选型合适的分布式协调服务，设计跨可用区、地区的容灾方案呢？如果选用 etcd，又该怎么 做呢？

感谢你阅读，也欢迎你把这篇文章分享给更多的朋友一起阅读。

[提建议](#)

更多课程推荐

Redis 核心技术与实战

从原理到实战，彻底吃透 Redis

蒋德钧

中科院计算所副研究员



涨价倒计时

现仅半价 **¥89** 4月17日涨价至 **¥199**

© 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 22 | 配置及服务发现：解析etcd在API Gateway开源项目中应用](#)

[下一篇 24 | 运维：如何构建高可靠的etcd集群运维体系？](#)

精选留言 (2)

 写留言



kkxue

2021-03-15

我觉得pingcap的这个方案可行:

<https://github.com/etcd-io/etcd/issues/11357>

<https://pingcap.com/blog-cn/geographic-data-distribution-traffic-and-latency-halved/>

"这里我们引入了一个新概念 Group, 每一个 Raft 节点都有一个对应的 Group ID, 拥...

展开 ▾



4



kkxue

2021-03-15

还有这个方案就是直接将不同数据中心的延时降低! 如同local datacenter,传闻google做到了



1