这里就平时经常用到的服务发现的产品进行下特性的对比，首先看下结论:

| **Feature** | **Consul** | **zookeeper** | **etcd** | **euerka** |
| --- | --- | --- | --- | --- |
| 服务健康检查 | 服务状态，内存，硬盘等 | (弱)长连接，keepalive | 连接心跳 | 可配支持 |
| 多数据中心 | 支持 | — | — | — |
| kv存储服务 | 支持 | 支持 | 支持 | — |
| 一致性 | raft | paxos | raft | — |
| cap | ca | cp | cp | ap |
| 使用接口(多语言能力) | 支持http和dns | 客户端 | http/grpc | http（sidecar） |
| watch支持 | 全量/支持long polling | 支持 | 支持 long polling | 支持 long polling/大部分增量 |
| 自身监控 | metrics | — | metrics | metrics |
| 安全 | acl /https | acl | https支持（弱） | — |
| spring cloud集成 | 已支持 | 已支持 | 已支持 | 已支持 |

* 服务的健康检查

Euraka 使用时需要显式配置健康检查支持；Zookeeper,Etcd 则在失去了和服务进程的连接情况下任务不健康，而 Consul 相对更为详细点，比如内存是否已使用了90%，文件系统的空间是不是快不足了。

* 多数据中心支持

Consul 通过 WAN 的 Gossip 协议，完成跨数据中心的同步；而且其他的产品则需要额外的开发工作来实现；

* KV 存储服务

除了 Eureka ,其他几款都能够对外支持 k-v 的存储服务，所以后面会讲到这几款产品追求高一致性的重要原因。而提供存储服务，也能够较好的转化为动态配置服务哦。

* 产品设计中 CAP 理论的取舍

Eureka 典型的 AP,作为分布式场景下的服务发现的产品较为合适，服务发现场景的可用性优先级较高，一致性并不是特别致命。其次 CA 类型的场景 Consul,也能提供较高的可用性，并能 k-v store 服务保证一致性。 而Zookeeper,Etcd则是CP类型 牺牲可用性，在服务发现场景并没太大优势；

* 多语言能力与对外提供服务的接入协议

Zookeeper的跨语言支持较弱，其他几款支持 http11 提供接入的可能。Euraka 一般通过 sidecar的方式提供多语言客户端的接入支持。Etcd 还提供了Grpc的支持。 Consul除了标准的Rest服务api,还提供了DNS的支持。

* Watch的支持（客户端观察到服务提供者变化）

Zookeeper 支持服务器端推送变化，Eureka 2.0(正在开发中)也计划支持。 Eureka 1,Consul,Etcd则都通过长轮询的方式来实现变化的感知；

* 自身集群的监控

除了 Zookeeper ,其他几款都默认支持 metrics，运维者可以搜集并报警这些度量信息达到监控目的；

* 安全

Consul,Zookeeper 支持ACL，另外 Consul,Etcd 支持安全通道https.

* Spring Cloud的集成

目前都有相对应的 boot starter，提供了集成能力。

总的来看，目前Consul 自身功能，和 [**spring**](http://lib.csdn.net/base/javaee) cloud 对其集成的支持都相对较为完善，而且运维的复杂度较为简单（没有详细列出讨论），Eureka 设计上比较符合场景，但还需持续的完善。

**著作权声明**

**首次发布于此，转载请保留以上链接**

名词解释：http://www.jdon.com/37625

补充：http://blog.csdn.net/chen77716/article/details/30635543

**分布式领域CAP理论，  
Consistency(一致性), 数据一致更新，所有数据变动都是同步的  
Availability(可用性), 好的响应性能  
Partition tolerance(分区容错性) 可靠性  
  
定理：任何分布式系统只可同时满足二点，没法三者兼顾。  
忠告：架构师不要将精力浪费在如何设计能满足三者的完美分布式系统，而是应该进行取舍。  
  
关系数据库的ACID模型拥有 高一致性 + 可用性 很难进行分区：  
Atomicity原子性：一个事务中所有操作都必须全部完成，要么全部不完成。  
Consistency一致性. 在事务开始或结束时，数据库应该在一致状态。  
Isolation隔离层. 事务将假定只有它自己在操作数据库，彼此不知晓。  
Durability. 一旦事务完成，就不能返回。  
跨数据库事务：2PC (two-phase commit)， 2PC is the anti-scalability pattern (Pat Helland) 是反可伸缩模式的，JavaEE中的JTA事务可以支持2PC。因为2PC是反模式，尽量不要使用2PC，使用BASE来回避。  
  
BASE模型反ACID模型，完全不同ACID模型，牺牲高一致性，获得可用性或可靠性：  
Basically Available基本可用。支持分区失败(e.g. sharding碎片划分数据库)  
Soft state软状态 状态可以有一段时间不同步，异步。  
Eventually consistent最终一致，最终数据是一致的就可以了，而不是时时高一致。  
  
BASE思想的主要实现有  
1.按功能划分数据库  
2.sharding碎片   
  
BASE思想主要强调基本的可用性，如果你需要High 可用性，也就是纯粹的高性能，那么就要以一致性或容错性为牺牲，BASE思想的方案在性能上还是有潜力可挖的。  
  
现在NOSQL运动丰富了拓展了BASE思想，可按照具体情况定制特别方案，比如忽视一致性，获得高可用性等等，NOSQL应该有下面两个流派：  
1. Key-Value存储，如Amaze Dynamo等，可根据CAP三原则灵活选择不同倾向的数据库产品。  
2. 领域模型 + 分布式缓存 + 存储 （**[Qi4j和NoSql运动](http://www.jdon.com/jivejdon/thread/37186)**），可根据CAP三原则结合自己项目定制灵活的分布式方案，难度高。  
  
这两者共同点：都是关系数据库SQL以外的可选方案，逻辑随着数据分布，任何模型都可以自己持久化，将数据处理和数据存储分离，将读和写分离，存储可以是异步或同步，取决于对一致性的要求程度。  
  
不同点：NOSQL之类的Key-Value存储产品是和关系数据库头碰头的产品BOX，可以适合非Java如PHP RUBY等领域，是一种可以拿来就用的产品，而领域模型 + 分布式缓存 + 存储是一种复杂的架构解决方案，不是产品，但这种方式更灵活，更应该是架构师必须掌握的。**