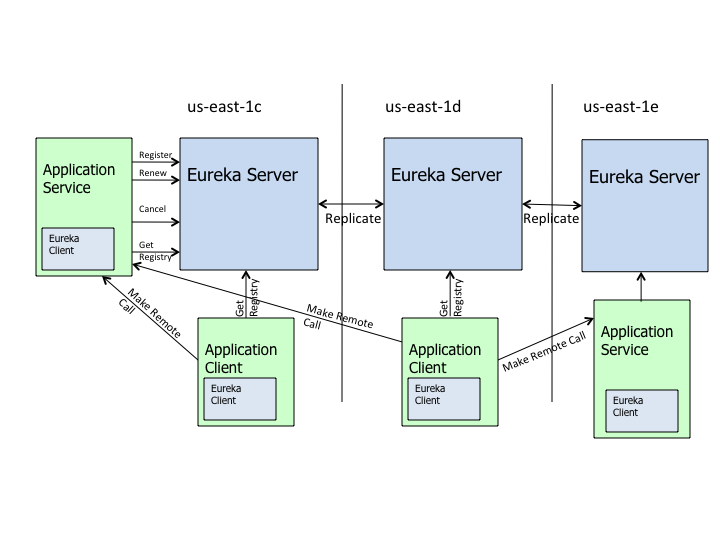
1、Eureka 简介：

Eureka在Spring Cloud全家桶中担任着服务的注册与发现的落地实现。Netflix在设计Eureka时遵循着AP原则，它基于REST的服务，用于定位服务，以实现云端中间层服务发现和故障转移，功能类似于Dubbo的注册中心Zookeeper。其中，Eureka又可细分为Eureka Server和Eureka Client。Eureka包括两个端：

Eureka Server：注册中心服务端，用于维护和管理注册服务列表。

Eureka Client：注册中心客户端，向注册中心注册服务的应用都可以叫做Eureka Client（包括Eureka Server本身）

上图是来自eureka的官方架构图，这是基于集群配置的eureka；

- us-east-xxx：表示集群所在的区域

- Eureka Client：表示Eureka客户端

- 处于不同节点的eureka通过Replicate进行数据同步

- Application Service为服务提供者

- Application Client为服务消费者

- Make Remote Call完成一次服务调用

如图所示，现在有三个区us-east-1c,us-east-1d,us-east-1e，每个区里都有一个Eureka Server集群，以及不定的Application Service和Application Client。

**同区域的服务注册与调用过程：**

us-east-1c区域代表了同区域内的服务注册与调用过程。

- Application Service启动后向Eureka Server注册中心注册服务，包括IP、Port、服务名等信息。

- Application Client启动后从Eureka Server拉取注册列表。

- Application Client发起远程调用的时候优先调用本区域内的Application Service。如果本区内没有可用的Application Service，才会发起对其他区内的Service调用。

**不同区域的服务注册与调用过程：**

三个区域us-east-1c,us-east-1d,us-east-1e结合在一起代表了不同区域内的服务注册与调用过程。

- us-east-1c内的Application Service启动后，向本区内的Eureka Server注册服务信息。并跟本区内的Eureka Server维持心跳续约。

- Eureka Server会将服务信息同步至相邻的us-east-1d的Eureka Server以及us-east-1e的Eureka Server

- us-east-1e内的Application Service启动后，向本区内的Eureka Server注册服务信息。并跟本区内的Eureka Server维持心跳续约。

- Eureka Server会将服务信息同步至相邻的us-east-1d的Eureka Server以及us-east-1c的Eureka Server

- us-east-1d内的Application Client启动后，从本区内的Eureka Server拉取注册列表。

- us-east-1d内的Application Client发起远程调用时，会先检索本区有没有可用的Application Service，如果没有就会通过某种算法调用us-east-1c或us-east-1e中的Application Service服务。

**服务续约：**

Application Service内的Eureka Client后台启动一个定时任务，跟Eureka Server保持一个心跳续约任务，每隔一段时间(默认30S)向Eureka Server发送一次renew请求，进行续约，告诉Eureka Server我还活着，防止被Eureka Server的Evict任务剔除。

**服务下线：**

Application Service应用停止后，向Eureka Server发送一个cancel请求，告诉注册中心我已经退出了，Eureka Server接收到之后会将其移出注册列表，后面再有获取注册服务列表的时候就获取不到了，防止消费端消费不可用的服务。

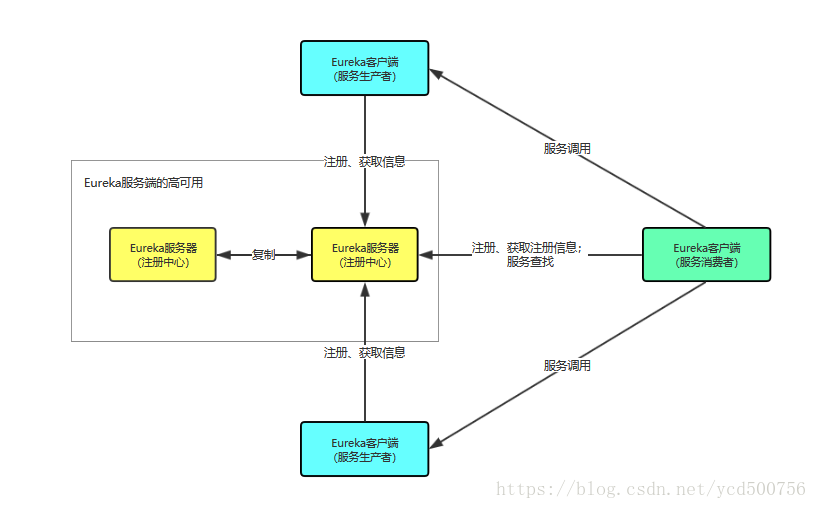
**服务剔除：**

Eureka Server启动后在后台启动一个Evict任务，对一定时间内没有续约的服务进行剔除。值得注意的是，注册、续约、下线的请求默认优先选择本区域内的Eureka Server，只有当本区内的Eureka Server都不可用，才会选择其他区的Eureka Server。

服务启动后向Eureka注册，Eureka Server会将注册信息向其他Eureka Server进行同步，当服务消费者要调用服务提供者，则向服务注册中心获取服务提供者地址，然后会将服务提供者地址缓存在本地，下次再调用时，则直接从本地缓存中取，完成一次调用。

当服务注册中心Eureka Server检测到服务提供者因为宕机、网络原因不可用时，则在服务注册中心将服务置为**DOWN状态**，并把当前服务提供者状态向订阅者发布，订阅过的服务消费者更新本地缓存。

服务提供者在启动后，周期性（默认30秒）向Eureka Server发送心跳，以证明当前服务是可用状态。Eureka Server在一定的时间（默认90秒）未收到客户端的心跳，则认为服务宕机，注销该实例。



Eureka采用C-S的设计架构，即包括了Eureka Server（服务端），Eureka client（客户端）。

1.1.**Eureka Server 提供服务注册**，各个节点启动后，在Eureka server中进行注册；

1.2 Eureka Client是一个Java客户端，用于和服务端进行交互，同时客户端也是一个内置的默认使用**轮询负载均衡算法的负载均衡器**。在应用启动后，会向Eureka Server发送心跳（默认30秒）。如果Eureka Server在多个心跳周期内没有接受到某个节点的心跳，Eureka Server将会从服务注册表中将这个服务移出（默认90秒）。

2.Eureka的自我保护机制

在默认配置中，Eureka Server在默认90s没有得到客户端的心跳，则注销该实例，但是往往因为微服务跨进程调用，网络通信往往会面临着各种问题，比如微服务状态正常，但是因为网络分区故障时，Eureka Server注销服务实例则会让大部分微服务不可用，这很危险，因为服务明明没有问题。为了解决这个问题，Eureka 有自我保护机制，通过在Eureka Server配置如下参数，可启动保护机制:

**eureka.server.enable-self-preservation=true**

它的原理是，当Eureka Server节点在短时间内丢失过多的客户端时（可能发送了网络故障），那么这个节点将进入自我保护模式，不再注销任何微服务，当网络故障回复后，该节点会自动退出自我保护模式.自我保护模式的架构哲学是宁可放过一千，决不可错杀一个。

3. 作为服务注册中心，Eureka比Zookeeper好在哪里

著名的CAP理论指出，一个分布式系统不可能同时满足C(一致性)、A(可用性)和P(分区容错性)。由于分区容错性在是分布式系统中必须要保证的，因此我们只能在A和C之间进行权衡。在此Zookeeper保证的是CP, 而Eureka则是AP。

3.1 Zookeeper保证CP

当向注册中心查询服务列表时，我们可以容忍注册中心返回的是几分钟以前的注册信息，但不能接受服务直接down掉不可用。也就是说，服务注册功能对可用性的要求要高于一致性。但是zk会出现这样一种情况，当master节点因为网络故障与其他节点失去联系时，剩余节点会重新进行leader选举。问题在于，选举leader的时间太长，30 ~ 120s, 且选举期间整个zk集群都是不可用的，这就导致在选举期间注册服务瘫痪。在云部署的环境下，因网络问题使得zk集群失去master节点是较大概率会发生的事，虽然服务能够最终恢复，但是漫长的选举时间导致的注册长期不可用是不能容忍的。

3.2 Eureka保证AP

Eureka看明白了这一点，因此在设计时就优先保证可用性。Eureka各个节点都是平等的，几个节点挂掉不会影响正常节点的工作，剩余的节点依然可以提供注册和查询服务。而Eureka的客户端在向某个Eureka注册或时如果发现连接失败，则会自动切换至其它节点，只要有一台Eureka还在，就能保证注册服务可用(保证可用性)，只不过查到的信息可能不是最新的(不保证强一致性)。除此之外，Eureka还有一种自我保护机制，如果在15分钟内超过85%的节点都没有正常的心跳，那么Eureka就认为客户端与注册中心出现了网络故障，此时会出现以下几种情况：

1. Eureka不再从注册列表中移除因为长时间没收到心跳而应该过期的服务

2. Eureka仍然能够接受新服务的注册和查询请求，但是不会被同步到其它节点上(即保证当前节点依然可用)

3. 当网络稳定时，当前实例新的注册信息会被同步到其它节点中

因此，Eureka可以很好的应对因网络故障导致部分节点失去联系的情况，而不会像zookeeper那样使整个注册服务瘫痪。

4. 总结

Eureka作为单纯的服务注册中心来说要比zookeeper更加“专业”，因为注册服务更重要的是可用性，我们可以接受短期内达不到一致性的状况。不过Eureka目前1.X版本的实现是基于servlet的Java web应用，它的极限性能肯定会受到影响。期待正在开发之中的2.X版本能够从servlet中独立出来成为单独可部署执行的服务