# 初识微服务

微服务架构风格是将一个单体的应用程序开发拆解为一组“小”的服务，这里的“小”是根据业务边界来区分的，而不是根据代码的多少来区分的。每个服务都运行在一个单独的进程中，服务之间通过轻量级的机制进行通信，例如使用HTTP资源接口；每个服务都可以通过全自动化的部署机制来独立部署；微服务中的各个服务可以以多种语言来编写，但是在实际开发中，由于各个公司的技术栈有限，通常会指定一门擅长的技术语言。

## 微服务缺点

1. 微服务架构将一个整体应用拆分成一系列的服务，这样不可避免地就会在不同的服务中存在一些冗余的代码。如，在服务A和服务B中，都使用到了Redis集群，可能我们需要在这两个服务中编写JedisCluser的工具类，而这个工具类的代码是一模一样的。
2. 微服务架构将单一的应用程序拆分为几十个或者上百个服务。维护一个应用程序简单，但是维护几十个或上百个服务就不是那么简单了。所以在微服务运维中，运维人员需要维护的进程数量会大大增加，这就需要运维人员有更多的技术来应对这样的挑战，运维过程需要更多的自动化。
3. 在微服务架构中，服务众多，服务之间的调用也由原来的单体架构中的进程内调用变为了进程之间的调用，

这就需要考虑的很多的问题，如使用什么通信方式，如何处理网络的延迟及服务容错，分布式事物，异步消息等问题。

1. 从零搭建一套微服务架构需要掌握的组件技术会比较多的，从而开发的难度较大，开发周期会比较长。

## 微服务中的组件和技术类型

### 服务注册和服务发现

服务注册：形象的来讲就是将服务的IP和Port注册到注册中心，我们可以使用Map来形象比喻服务注册，其中的key是服务的唯一标识，而value是一个包含IpAndPort的结构体的集合。

服务发现：根据服务的唯一标识，从注册中心获取指定服务所在的服务器列表。

好处：实现了服务之间的解耦，如果不适用服务注册中心，那么我们就需要将被调用服务的服务器列表直接写在调用服务的配置文件中，这造成了两个服务的直接耦合。之后，在调用服务的时候，通过一定的负载均衡算法，从服务列表中获取一台服务器进行调用，此时，如果被调用服务所在服务器有宕机情况或者新添加了服务器，调用服务也不会发现，除非修改配置文件，之后还需要重启服务。而使用了服务注册和服务发现后，调用服务会及时发现新增的机器或者宕机的机器，而且不需要重启服务。

实现服务注册的技术：Consul，ZooKeeper，Etcd和Eureka等。

### 健康检查

健康检查是检查两个东西是否处于正常状态：一个是服务所在的服务器的运行状态；一个是服务本身的运行状态。健康检查的目的是为了在服务发现和服务路由的时候，可以将服务的调用请求发送到处于健康状态的机器上，不至于使服务调用因为请求被发送到不健康的机器上而失败。

常用技术：Consul，SpringBot的Actuator等。

### 配置管理

配置管理主要做三件事：第一，在一个地方将服务集中管理。

第二，实现服务的配置与代码的分离，这样修改了配置信息后不需要重新编译，打包，部署整个服务，

第三，实现“热配置”，即修改配制文件后，不需要重启服务就可以自动获取修改后的配置信息。

常用技术：Consul，Archaius等

### 服务通信

服务通信是指服务之间相互调用。服务之间的调用协议可以使用TCP协议，也可以使用HTTP协议。

实现服务通信的技术有：Netty，Mina，Retrofit，OkHttp和AsyncHttpClient等。其中Netty和Mina是基于

TCP或UDP协议的通信；OkHttp和AsyncHttpClient是基于HTTP协议通信，后者的效率要高于前者。

### 服务路由

服务路由过程：当一个请求过来时，通过服务发现和健康检查选出健康 的服务列表，之后采用一定的负载均衡策略从这些服务器中选出一台，最后将请求发送到这台服务器上去。服务路由器通常会包含一个内建的负载均衡器，而且会包含多种负载均衡策略，这些策略都是可插拔的，并且会在本地缓存一份可用的服务器列表，当然，也会通过一定的策略来更新该服务器列表，例如定时使用服务发现来刷新本地服务器列表缓存，进而达到宕机或添加了新机器的时候，本地服务器列表缓存可以及时更新。

常用技术：Ribbon

### 服务容错