一．几种架构的比较

随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，亟需一个治理系统确保架构有条不紊的演进。

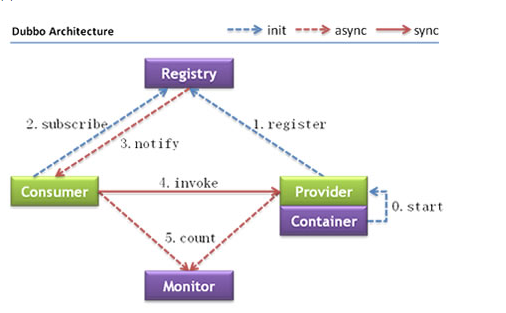


* **单一应用架构**
  + 当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。
  + 此时，用于简化增删改查工作量的 **数据访问框架(ORM)** 是关键。
* **垂直应用架构**
  + 当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。
  + 此时，用于加速前端页面开发的 **Web框架(MVC)** 是关键。

备注：一个服务启动只能调用本服务器的接口。

* **分布式服务架构**
  + 当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。
  + 此时，用于提高业务复用及整合的 **分布式服务框架(RPC)** 是关键。
* **流动计算架构**
  + 当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。
  + 此时，用于提高机器利用率的 **资源调度和治理中心(SOA)** 是关键。

二．原理



**节点角色说明：**

* **Provider:** 暴露服务的服务提供方。
* **Consumer:** 调用远程服务的服务消费方。
* **Registry:** 服务注册与发现的注册中心。如：
* **Monitor:** 统计服务的调用次调和调用时间的监控中心。
* **Container:** 服务运行容器。

**调用关系说明：**

* 0. 服务容器负责启动，加载，运行服务提供者。
* 1. 服务提供者在启动时，向注册中心注册自己提供的服务。
* 2. 服务消费者在启动时，向注册中心订阅自己所需的服务。
* 3. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。
* 4. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。（核心部分，可以一对多进行调用）
* 5. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。



在大规模服务化之前，应用可能只是通过RMI或Hessian等工具，简单的暴露和引用远程服务，通过配置服务的URL地址进行调用，通过F5等硬件进行负载均衡。

**(1) 当服务越来越多时，服务URL配置管理变得非常困难，F5硬件负载均衡器的单点压力也越来越大。（动态注册服务）**

此时需要一个服务注册中心，动态的注册和发现服务，使服务的位置透明。

并通过在消费方获取服务提供方地址列表，实现软负载均衡和Failover，降低对F5硬件负载均衡器的依赖，也能减少部分成本。

**(2) 当进一步发展，服务间依赖关系变得错踪复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系。**

这时，需要自动画出应用间的依赖关系图，以帮助架构师理清理关系。

**(3) 接着，服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？什么时候该加机器？**

为了解决这些问题，第一步，要将服务现在每天的调用量，响应时间，都统计出来，作为容量规划的参考指标。

其次，要可以动态调整权重，在线上，将某台机器的权重一直加大，并在加大的过程中记录响应时间的变化，直到响应时间到达阀值，记录此时的访问量，再以此访问量乘以机器数反推总容量。

以上是Dubbo最基本的几个需求，