**SOA and Duboo**

1. SOA

A **service-oriented architecture** (**SOA**) is an [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern) in computer software design in which application components provide services to other components via a [communications protocol](https://en.wikipedia.org/wiki/Communications_protocol), typically over a network.

面向服务的体系结构（service-oriented architecture）是构造分布式计算的应用程序的方法。它将应用程序功能作为服务发送给最终用户或者其他服务。

Web服务（Web Services）在很多人眼里还是个十分神秘的概念，究其根源，我想主要是由于Web服务被宣传得很多，但实际应用却鲜见，给人一种很复杂和难以理解的感觉。另外，Web服务是基于XML的，不少人对XML本身也缺乏理解，虽然他们可能每天都在写XML格式的配置文件。

提到Web服务的起源就一定要先说一说SOA（面向服务的体系结构），和很多具有划时代意义的软件技术一样，SOA的出现根本上也是为了解决软件危机问题。做过项目的人都有过这种感受，随着项目推进，模块之间关系越来越紧密，任何一个小的修改都可能引起整个系统的不稳定，而客户需求偏偏总是在改变，结果是项目以差不多失败的结果告终。

从（分布式）软件发展的趋势来看，C/S->B/S->SOA，模块之间的耦合度是由紧密到松散的，松散的耦合有利于修改。我们常说的各种设计模式，其中大部分不也是为了降低类之间的耦合度吗。

这里我引用一下IBM网站上对SOA的定义：面向服务的体系结构（service-oriented architecture）是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种这样的系统中的服务可以以一种统一和通用的方式进行交互。（全文）

说得通俗一点就是，系统中分为三种角色：服务提供者、服务使用者和注册中心，提供者发布服务到注册中心，使用者通过注册中心发现所需服务，然后与该服务的提供者绑定，并调用服务。

那么Web服务和SOA是什么关系呢，可以这样说，Web服务是SOA的一种实现，有点像Tomcat和JSP/Servlet规范的关系。SOA是一个比较虚的概念，例如它只提出定义一些接口和协议，那么这些东西具体应该怎样定义呢，Web服务就将它们具体化了：Web服务使用的协议都是基于XML的；SOA只说应该有三种角色，而Web服务里这三种角色都有具体的实现方式。看到这里你应该会问，那么SOA还有哪些实现呢？CORBA、DCOM和J2EE都可以算是，但我认为它们不能算很纯粹，至少它们并不都具有中立的协议。

现在该用一个具体的例子来说明一下Web服务了，假设我们的系统中需要一项功能是查询当地的天气情况（世界时间、货币汇率等等，都一样），显然我们不会自己做一个从气象部门数据库中查找数据的程序，这需要很多手续也没有必要，更要命的是，这样做会增加我们与气象部门的耦合度。试想某一天气象部门的数据库结构改变了，我们将不得不修改自己的代码，如果他们忘记通知我们这一改变，想象一下客户会看到什么？

为了利用Web服务，我们从某一注册中心查找和天气有关的服务，在结果中也许我们会选择收费较低，或者收费稍高但更稳定和准确的服务。从注册中心我们能够得到所选服务的完整描述，其中包含了各种数据类型和调用方式，利用这些信息，可以使用工具生成这些必要的类，以及客户端Stub，利用这个Stub就可以调用远程的Web服务了。在我们的例子中，调用后服务提供者会返回一个含有结果的消息，在我们的系统中可以从这个消息里得到所要的结果，并显示给客户。这样就形成一个完整的Web服务调用。这种调用方式被称为静态调用，因为在Stub里服务提供者的地址（被称为调用端点endpoint）是写定的，还有另外一种方式被称为动态调用，以后会讲到。

那么Web服务和以前的RPC（远程过程调用）有什么分别呢？RPC通常要求调用者和被调用者是同构的，即使用同样的语言编写，而Web服务没有这个要求（诀窍在于使用了XML封装消息），这就大大增加了灵活程度；另外，Web服务的调用除这种类似RPC的方式外，还可以是基于消息的方式，服务使用者可以只接收消息，或是只发送消息，在一些应用中这种方式十分有用。

内容总结一下就是：Web服务是SOA的实现，Web服务不是RPC。

**SOA的原则**

服务封装

服务松耦合(Loosely coupled) - 服务之间的关系最小化，只是互相知道。

服务契约 - 服务按照服务描述文档所定义的服务契约行事。

服务抽象 - 除了服务契约中所描述的内容，服务将对外部隐藏逻辑。

服务的重用性 - 将逻辑分布在不同的服务中，以提高服务的重用性。

服务的可组合性 - 一组服务可以协调工作并组合起来形成一个组合服务。

服务自治 – 服务对所封装的逻辑具有控制权

服务无状态 – 服务将一个活动所需保存的资讯最小化。

服务的可被发现性 – 服务需要对外部提供描述资讯，这样可以通过现有的发现机制发现并访问这些服务。

服务导向的架构通常被定义为通过Web服务协议栈暴露的服务[来源请求]。与SOA相关的Web服务的标准主要有：

XML - 一种标记语言，用于以文档格式描述消息中的数据。

HTTP（或HTTPS） - 客户端和服务端之间用于传送信息而发送请求/回复的协议。

SOAP（Simple Object Access Protocol） - 在计算机网络上交换基于XML的消息的协议，通常是用HTTP。

WSDL（Web Services Description Language，Web服务描述语言） - 基于XML的描述语言，用于描述与服务交互所需的服务的公共接口，协议绑定，消息格式。

UDDI（Universal Description, Discovery, and Integration，是统一描述、发现和集成） - 基于XML的注册协议，用于发布WSDL并允许第三方发现这些服务。

注意，一个系统要成为服务导向的系统并不需要这些协议，比如一些服务导向的系统可以通过CORBA实现。

1. Dubbo

**背景**

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。
* **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。

**需求**



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。**

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，更多服务治理问题参见：

[http://code.alibabatech.com/blog/experience\_1402/service-governance-process.html](javascript:if(confirm(%27http://code.alibabatech.com/blog/experience_1402/service-governance-process.html%20%20\n\nThis%20file%20was%20not%20retrieved%20by%20Teleport%20Ultra,%20because%20it%20is%20addressed%20on%20a%20domain%20or%20path%20outside%20the%20boundaries%20set%20for%20its%20Starting%20Address.%20%20\n\nDo%20you%20want%20to%20open%20it%20from%20the%20server?%27))window.location=%27http://code.alibabatech.com/blog/experience_1402/service-governance-process.html%27)

**架构**



**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。
* **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。
* 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。

**(1) 连通性：**

* 注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，压力较小
* 监控中心负责统计各服务调用次数，调用时间等，统计先在内存汇总后每分钟一次发送到监控中心服务器，并以报表展示
* 服务提供者向注册中心注册其提供的服务，并汇报调用时间到监控中心，此时间不包含网络开销
* 服务消费者向注册中心获取服务提供者地址列表，并根据负载算法直接调用提供者，同时汇报调用时间到监控中心，此时间包含网络开销
* 注册中心，服务提供者，服务消费者三者之间均为长连接，监控中心除外
* 注册中心通过长连接感知服务提供者的存在，服务提供者宕机，注册中心将立即推送事件通知消费者
* 注册中心和监控中心全部宕机，不影响已运行的提供者和消费者，消费者在本地缓存了提供者列表
* 注册中心和监控中心都是可选的，服务消费者可以直连服务提供者

**(2) 健状性：**

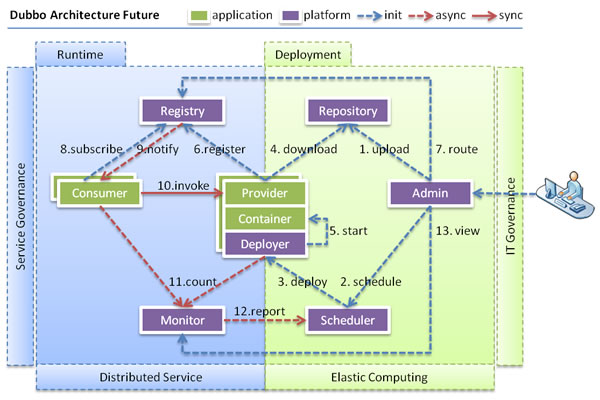
* 监控中心宕掉不影响使用，只是丢失部分采样数据
* 数据库宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务
* 注册中心对等集群，任意一台宕掉后，将自动切换到另一台
* 注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯
* 服务提供者无状态，任意一台宕掉后，不影响使用
* 服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复

**(3) 伸缩性：**

* 注册中心为对等集群，可动态增加机器部署实例，所有客户端将自动发现新的注册中心
* 服务提供者无状态，可动态增加机器部署实例，注册中心将推送新的服务提供者信息给消费者

**(4) 升级性：**

* 当服务集群规模进一步扩大，带动IT治理结构进一步升级，需要实现动态部署，进行流动计算，现有分布式服务架构不会带来阻力：



* **Deployer:**
* **Repository:**
* **Scheduler:**
* **Admin:**