SOA/软件架构设计---面向服务的架构（SOA详细解释）

文章比较多，但干货慢慢，请耐心阅读

目录

面向服务的架构

迄今为止，对于面向服务的架构（Service-Oriented Architecture，SOA）还没有一个公认的定义。许多组织从不同的角度和不同的侧面对 SOA 进行了描述，较为典型的有以下三个：

（1）W3C 的定义：SOA 是一种应用程序架构，在这种架构中，所有功能都定义为独立的服务，这些服务带有定义明确的可调用接口，能够以定义好的顺序调用这些服务来形成业务流程。

（2）Service-architecture.com 的定义：服务是精确定义、封装完善、独立于其他服务所处环境和状态的函数。SOA 本质上是服务的集合，服务之间彼此通信，这种通信可能是简单的数据传送，也可能是两个或更多的服务协调进行某些活动。服务之间需要某些方法进行连接。

（3）Gartner 的定义：SOA 是一种 C/S 架构的软件设计方法，应用由服务和服务使用者组成，SOA 与大多数通用的 C/S 架构模型不同之处，在于它着重强调构件的松散耦合，并使用独立的标准接口。

1 SOA 概述

SOA 是一种在计算环境中设计、开发、部署和管理离散逻辑单元（服务）模型的方法。 SOA 并不是一个新鲜事物，而只是面向对象模型的一种替代。虽然基于 SOA 的系统并不排除使用 OOD 来构建单个服务，但是其整体设计却是面向服务的。由于 SOA 考虑到了系统内的对象，所以虽然SOA 是基于对象的，但是作为一个整体，它却不是面向对象的。

SOA 系统原型的一个典型例子是 CORBA，它已经出现很长时间，其定义的概念与 SOA 相似。SOA 建立在 XML 等新技术的基础上，通过使用基于 XML 的语言来描述接口，服务已经转到更动态且更灵活的接口系统中，CORBA 中的 IDL 无法与之相比。图 9-13 描述了一个完整的 SOA 模型。

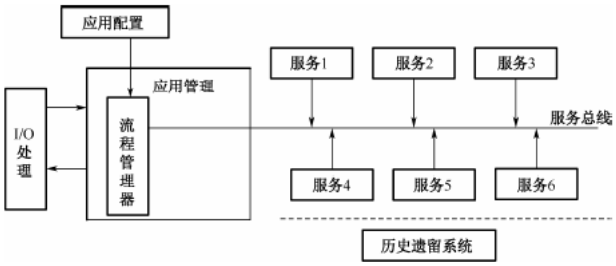

图 9-13　SOA 模型示例

在 SOA 模型中，所有的功能都定义成了独立的服务。服务之间通过交互和协调完成业务的整体逻辑。所有的服务通过服务总线或流程管理器来连接。这种松散耦合的架构使得各服务在交互过程中无需考虑双方的内部实现细节，以及部署在什么平台上。

1. 服务的基本结构

一个独立的服务基本结构如图 9-14 所示。

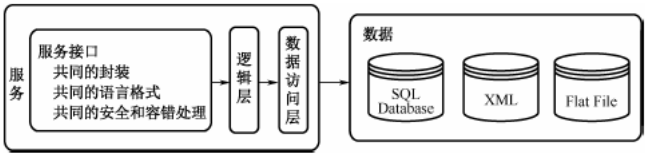

图 9-14　单个服务内部结构

由图 9-14 可以看出，服务模型的表示层从逻辑层分离出来，中间增加了服务对外的接口层。通过服务接口的标准化描述，使得服务可以提供给在任何异构平台和任何用户接口使用。这允许并支持基于服务的系统成为松散耦合、面向构件和跨技术实现，服务请求者很可能根本不知道服务在哪里运行、是由哪种语言编写的，以及消息的传输路径，而是只需要提出服务请求，然后就会得到答案。

2.SOA 设计原则

在 SOA 架构中，继承了来自对象和构件设计的各种原则，例如，封装和自我包含等。那些保证服务的灵活性、松散耦合和复用能力的设计原则，对 SOA 架构来说同样是非常重要的。关于服务，一些常见的设计原则如下：

（1）明确定义的接口。服务请求者依赖于服务规约来调用服务，因此，服务定义必须长时间稳定，一旦公布，不能随意更改；服务的定义应尽可能明确，减少请求者的不适当使用；不要让请求者看到服务内部的私有数据。

（2）自包含和模块化。服务封装了那些在业务上稳定、重复出现的活动和构件，实现服务的功能实体是完全独立自主的，独立进行部署、版本控制、自我管理和恢复。

（3）粗粒度。服务数量不应该太多，依靠消息交互而不是远程过程调用，通常消息量比较大，但是服务之间的交互频度较低。

（4）松耦合。服务请求者可见的是服务的接口，其位置、实现技术、当前状态和私有数据等，对服务请求者而言是不可见的。

（5）互操作性、兼容和策略声明。为了确保服务规约的全面和明确，策略成为一个越来越重要的方面。这可以是技术相关的内容，例如，一个服务对安全性方面的要求；也可以是与业务有关的语义方面的内容，例如，需要满足的费用或者服务级别方面的要求，这些策略对于服务在交互时是非常重要的。

3. 服务构件与传统构件

服务构件架构（Service Component Architecture，SCA）是基于 SOA 的思想描述服务之间组合和协作的规范，它描述用于使用 SOA 构建应用程序和系统的模型。它可简化使用 SOA 进行的应用程序开发和实现工作。SCA 提供了构建粗粒度构件的机制，这些粗粒度构件由细粒度构件组装而成。SCA 将传统中间件编程从业务逻辑分离出来，从而使程序员免受其复杂性的困扰。它允许开发人员集中精力编写业务逻辑，而不必将大量的时间花费在更为底层的技术实现上。

SCA 服务构件与传统构件的主要区别在于，服务构件往往是粗粒度的，而传统构件以细粒度居多；服务构件的接口是标准的，主要是服务描述语言接口，而传统构件常以具体 API 形式出现；服务构件的实现与语言是无关的，而传统构件常绑定某种特定的语言；服务构件可以通过构件容器提供 QoS 的服务，而传统构件完全由程序代码直接控制。

2 SOA 的关键技术

SOA 伴随着无处不在的标准，为企业的现有资产或投资带来了更好的复用性。SOA 能够在最新的和现有的系统之上创建应用，借助现有的应用产生新的服务，为企业提供更好的灵活性来构建系统和业务流程。SOA 是一种全新的架构，为了支持其各种特性，相关的技术规范不断推出。与 SOA 紧密相关的技术主要有 UDDI、WSDL、SOAP 和 REST 等，而这些技术都是以 XML 为基础而发展起来的。

1. UDDI

UDDI（Universal DescriptionDiscovery and Integration，统一描述、发现和集成）提供了一种服务发布、查找和定位的方法，是服务的信息注册规范，以便被需要该服务的用户发现和使用它。UDDI 规范描述了服务的概念，同时也定义了一种编程接口。通过 UDDI 提供的标准接口，企业可以发布自己的服务供其他企业查询和调用，也可以查询特定服务的描述信息，并动态绑定到该服务上。

在 UDDI 技术规范中，主要包含以下三个部分的内容：

（1）数据模型。UDDI 数据模型是一个用于描述业务组织和服务的 XML Schema。

（2）API。UDDI API 是一组用于查找或发布 UDDI 数据的方法，UDDI API 基于 SOAP。

（3）注册服务。UDDI 注册服务是 SOA 中的一种基础设施，对应着服务注册中心的角色。

2.WSDL

WSDL（Web ServiceDescription Language，Web 服务描述语言）是对服务进行描述的语言，它有一套基于 XML 的语法定义。WSDL 描述的重点是服务，它包含服务实现定义和服务接口定义，如图 9-15 所示。

图 9-15　基本服务描述

采用抽象接口定义对于提高系统的扩展性很有帮助。服务接口定义就是一种抽象的、可重用的定义，行业标准组织可以使用这种抽象的定义来规定一些标准的服务类型，服务实现者可以根据这些标准定义来实现具体的服务。

服务实现定义描述了给定服务提供者如何实现特定的服务接口。服务实现定义中包含服务和端口描述。一个服务往往会包含多个服务访问入口，而每个访问入口都会使用一个端口元素来描述，端口描述的是一个服务访问入口的部署细节，例如，通过哪个地址来访问，应当使用怎样的消息调用模式来访问等。

3.SOAP

SOAP（Simple ObjectAccess Protocol，简单对象访问协议）定义了服务请求者和服务提供者之间的消息传输规范。SOAP 用 XML 来格式化消息，用 HTTP 来承载消息。通过 SOAP，应用程序可以在网络中进行数据交换和远程过程调用（Remote Procedure Call， RPC）。SOAP 主要包括以下四个部分：

（1）封装。SOAP 封装定义了一个整体框架，用来表示消息中包含什么内容，谁来处理这些内容，以及这些内容是可选的还是必需的。

（2）编码规则。SOAP 编码规则定义了一种序列化的机制，用于交换系统所定义的数据类型的实例。

（3）RPC 表示。SOAP RPC 表示定义了一个用来表示远程过程调用和应答的协议。

（4）绑定。SOAP 绑定定义了一个使用底层传输协议来完成在节点之间交换 SOAP 封装的约定。

SOAP 消息基本上是从发送端到接收端的单向传输，但它们常常结合起来执行类似于请求/应答的模式。所有的 SOAP 消息都使用 XML 进行编码。SOAP 消息包括以下三个部分：

（1）封装（信封）。封装的元素名是 Envelope，在表示消息的 XML 文档中，封装是顶层元素，在 SOAP 消息中必须出现。

（2）SOAP 头。SOAP 头的元素名是 Header，提供了向 SOAP 消息中添加关于这条 SOAP 消息的某些要素的机制。SOAP 定义了少量的属性用来表明这项要素是否可选以及由谁来处理。SOAP 头在 SOAP 消息中可能出现，也可能不出现。如果出现的话，必须是 SOAP 封装元素的第一个直接子元素。

（3）SOAP 体。SOAP 体的元素名是 Body，是包含消息的最终接收者想要的信息的容器。SOAP 体在 SOAP 消息中必须出现且必须是 SOAP 封装元素的直接子元素。如果有头元素，则SOAP 体必须直接跟在 SOAP 头元素之后；如果没有头元素，则 SOAP 体必须是 SOAP 封装元素的第一个直接子元素。

4.REST

REST（RepresentationalState Transfer，表述性状态转移）是一种只使用 HTTP 和 XML 进行基于 Web 通信的技术，可以降低开发的复杂性，提高系统的可伸缩性。它的简单性和缺少严格配置文件的特性，使它与 SOAP 很好地隔离开来，REST 从根本上来说只支持几个操作（POST、GET、PUT 和 DELETE），这些操作适用于所有的消息。REST 提出了如下一些设计概念和准则：

（1）网络上的所有事物都被抽象为资源。

（2）每个资源对应一个唯一的资源标识。

（3）通过通用的连接件接口对资源进行操作。

（4）对资源的各种操作不会改变资源标识。

（5）所有的操作都是无状态的。

3 SOA 的实现方法

SOA 只是一种概念和思想，需要借助于具体的技术和方法来实现它。从本质上来看， SOA 是用本地计算模型来实现一个分布式的计算应用，也有人称这种方法为“本地化设计，分布式工作”模型。CORBA、DCOM 和 EJB 等都属于这种解决方式，也就是说，SOA 最终可以基于这些标准来实现。有关这些标准的知识，已经在 13.1.1 节中详细介绍。另外，这些标准分别使用的 ORB、RPC 和 RMI（Remote Method Invocation，远程方法调用）等技术，将在 17.1.2 节中介绍，此处不再赘述。

从逻辑上和高层抽象来看，目前，实现 SOA 的方法也比较多，其中主流方式有 Web Service、企业服务总线和服务注册表。

1.Web Service

在 Web Service（Web 服务）的解决方案中，一共有三种工作角色，其中服务提供者和服务请求者是必需的，服务注册中心是一个可选的角色。它们之间的交互和操作构成了 SOA 的一种实现架构，如图 9-16 所示。

（1）服务提供者。服务提供者是服务的所有者，该角色负责定义并实现服务，使用 WSDL 对服务进行详细、准确、规范地描述，并将该描述发布到服务注册中心，供服务请求者查找并绑定使用。

（2）服务请求者。服务请求者是服务的使用者，虽然服务面向的是程序，但程序的最终使用者仍然是用户。从架构的角度看，服务请求者是查找、绑定并调用服务，或与服务进行交互的应用程序。服务请求者角色可以由浏览器来担当，由人或程序（例如，另外一个服务）来控制。

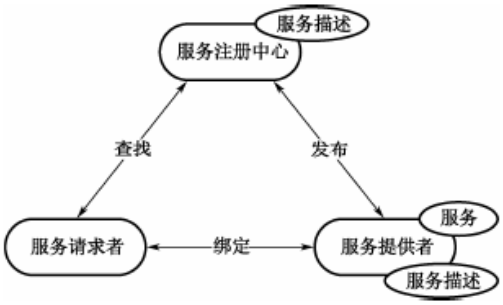

图 9-16　Web Service 模型

（3）服务注册中心。服务注册中心是连接服务提供者和服务请求者的纽带，服务提供者在此发布他们的服务描述，而服务请求者在服务注册中心查找他们需要的服务。不过，在某些情况下，服务注册中心是整个模型中的可选角色。例如，如果使用静态绑定的服务，服务提供者则可以把描述直接发送给服务请求者。

Web Service 模型中的操作包括发布、查找和绑定，这些操作可以单次或反复出现。

（1）发布。为了使用户能够访问服务，服务提供者需要发布服务描述，以便服务请求者可以查找它。

（2）查找。在查找操作中，服务请求者直接检索服务描述或在服务注册中心查询所要求的服务类型。对服务请求者而言，可能会在生命周期的两个不同阶段中涉及查找操作，首先是在设计阶段，为了程序开发而查找服务的接口描述；其次是在运行阶段，为了调用而查找服务的位置描述。

（3）绑定。在绑定操作中，服务请求者使用服务描述中的绑定细节来定位、联系并调用服务，从而在运行时与服务进行交互。绑定可以分为动态绑定和静态绑定。在动态绑定中，服务请求者通过服务注册中心查找服务描述，并动态地与服务交互；在静态绑定中，服务请求者已经与服务提供者达成默契，通过本地文件或其他方式直接与服务进行绑定。

在采用 Web Service 作为 SOA 的实现技术时，应用系统大致可以分为六个层次，分别是底层传输层、服务通信协议层、服务描述层、 服务层、业务流程层和服务注册层。

（1）底层传输层。底层传输层主要负责消息的传输机制，HTTP、JMS（Java Messaging Service，Java 消息服务）和 SMTP 都可以作为服务的消息传输协议，其中 HTTP 使用最广。

（2）服务通信协议层。服务通信协议层的主要功能是描述并定义服务之间进行消息传递所需的技术标准，常用的标准是 SOAP 和 REST 协议。

（3）服务描述层。服务描述层主要以一种统一的方式描述服务的接口与消息交换方式，相关的标准是 WSDL。

（4）服务层。服务层的主要功能是将遗留系统进行包装，并通过发布的 WSDL 接口描述被定位和调用。

（5）业务流程层。业务流程层的主要功能是支持服务发现，服务调用和点到点的服务调用，并将业务流程从服务的底层调用抽象出来。

（6）服务注册层的主要功能是使服务提供者能够通过 WSDL 发布服务定义，并支持服务请求者查找所需的服务信息。相关的标准是 UDDI。

2. 服务注册表

服务注册表（service registry）虽然也具有运行时的功能，但主要在 SOA设计时使用。它提供一个策略执行点（Policy Enforcement Point，PEP），在这个点上，服务可以在 SOA 中注册，从而可以被发现和使用。服务注册表可以包括有关服务和相关构件的配置、依从性和约束文件。从理论上来说，任何帮助服务注册、发现和查找服务合约、元数据和策略的信息库、数据库、目录或其他节点都可以被认为是一个注册表。大多数商用服务注册产品支持服务注册、服务位置和服务绑定功能。

（1）服务注册。服务注册是指服务提供者向服务注册表发布服务的功能（服务合约），包括服务身份、位置、方法、绑定、配置、方案和策略等描述性属性。使用服务注册表实现 SOA 时，要限制哪些新服务可以向注册表发布、由谁发布以及谁批准和根据什么条件批准等，以便使服务能够有序的注册。

（2）服务位置。服务位置是指服务使用者，帮助它们查询已注册的服务，寻找符合自身要求的服务。这种查找主要是通过检索服务合约来实现的，在使用服务注册表实现 SOA 时，需要规定哪些用户可以访问服务注册表，以及哪些服务属性可以通过服务注册表进行暴露等，以便服务能得到有效的、经过授权的使用。

（3）服务绑定。服务使用者利用查找到的服务合约来开发代码，开发的代码将与注册的服务进行绑定，调用注册的服务，以及与它们实现互动。可以利用集成的开发环境自动将新开发的服务与不同的新协议、方案和程序间通信所需的其他接口绑定在一起。

3. 企业服务总线

ESB 的概念是从 SOA 发展而来的，它是一种为进行连接服务提供的标准化的通信基础结构，基于开放的标准，为应用提供了一个可靠的、可度量的和高度安全的环境，并可帮助企业对业务流程进行设计和模拟，对每个业务流程实施控制和跟踪、分析并改进流程和性能。

在一个复杂的企业计算环境中，如果服务提供者和服务请求者之间采用直接的端到端的交互，那么随着企业信息系统的增加和复杂度的提高，系统之间的关联会逐渐变得非常复杂，形成一个网状结构，这将带来昂贵的系统维护费用，同时也使得 IT 基础设施的复用变得困难重重。ESB 提供了一种基础设施，消除了服务请求者与服务提供者之间的直接连接，使得服务请求者与服务提供者之间进一步解耦。

ESB 是由中间件技术实现并支持 SOA的一组基础架构，是传统中间件技术与 XML、 Web Service 等技术结合的产物，是在整个企业集成架构下的面向服务的企业应用集成机制。具体来说，ESB 具有以下功能：

（1）支持异构环境中的服务、消息和基于事件的交互，并且具有适当的服务级别和可管理性。

（2）通过使用 ESB，可以在几乎不更改代码的情况下，以一种无缝的非侵入方式使现有系统具有全新的服务接口，并能够在部署环境中支持任何标准。

（3）充当缓冲器的 ESB（负责在诸多服务之间转换业务逻辑和数据格式）与服务逻辑相分离，从而使不同的系统可以同时使用同一个服务，不用在系统或数据发生变化时，改动服务代码。

（4）在更高的层次，ESB 还提供诸如服务代理和协议转换等功能。允许在多种形式下通过像HTTP、SOAP 和 JMS 总线的多种传输方式，主要是以网络服务的形式，为发表、注册、发现和使用企业服务或界面提供基础设施。

（5）提供可配置的消息转换翻译机制和基于消息内容的消息路由服务，传输消息到不同的目的地。

（6）提供安全和拥有者机制，以保证消息和服务使用的认证、授权和完整性。

在企业应用集成方面，与现存的、专有的集成解决方案相比，ESB 具有以下优势：

（1）扩展的、基于标准的连接。ESB 形成一个基于标准的信息骨架，使得在系统内部和整个价值链中可以容易地进行异步或同步数据交换。ESB 通过使用 XML、SOAP 和其他标准，提供了更强大的系统连接性。

（2）灵活的、服务导向的应用组合。基于 SOA，ESB 使复杂的分布式系统（包括跨多个应用、系统和防火墙的集成方案）能够由以前开发测试过的服务组合而成，使系统具有高度可扩展性。

（3）提高复用率，降低成本。按照 SOA 方法构建应用，提高了复用率，简化了维护工作，进而减少了系统总体成本。

（4）减少市场反应时间，提高生产率。ESB 通过构件和服务复用，按照 SOA 的思想简化应用组合，基于标准的通信、转换和连接来实现这些优点。

4 微服务

微服务顾名思义，就是很小的服务，所以它属于面向服务架构的一种。通俗一点来说，微服务类似于古代著名的发明，活字印刷术，每个服务都是一个组件，通过编排组合的方式来使用，从而真正做到了独立、解耦、组件化、易维护、可复用、可替换、高可用、最终达到提高交付质量、缩短交付周期的效果。

从专业的角度来看，微服务架构是一种架构模式，它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相沟通（通常是基于 HTTP 协议的 RESTful API）。每个服务都围绕着具体业务进行构建，并且能够被独立的部署到生产环境、类生产环境等。另外，应当尽量避免统一的、集中式的服务管理机制，对具体的一个服务而言，应根据业务上下文，选择合适的语言、工具对其进行构建。

所以总结起来，微服务的核心特点为：小, 且专注于做一件事情、轻量级的通信机制、松耦合、独立部署。

1.微服务的优势

微服务之所以能盛行，必然是有它独特优势的，下面我们来分析微服务有哪些方面的优势。

（1）技术异构性

在微服务架构中，每个服务都是一个相对独立的个体，每个服务都可以选择适合于自身的技术来实现。如，要开发一个社交平台，此时，我们可能使用文档型数据库来存储帖 子的内容，使用图数据来存储朋友圈的这些关系等，这样可以把每一块的性能都充分发挥 出来。

同时，在应用新技术时，微服务架构也提供了更好的试验场。因为对于单块的系统而 言，采用一个新的语言、数据库或者框架都会对整个系统产生巨大的影响，这样导致我们 想尝试新技术时，望而却步。但微服务不同，我们完全可以只在一个微服务中采用新技术， 待技术使用熟练之后，再推广到其他服务。

（2）弹性

弹性主要讲的是系统中一部分出现故障会引起多大问题。在单块系统中，一个部分出现问题，可能导致整体系统的问题。而微服务架构中，每个服务可以内置可用性的解决方 案与功能降级方案，所以比单块系统强。

（3）扩展

单块系统中，我们要做扩展，往往是整体进行扩展。而在微服务架构中，可以针对单个服务进行扩展。

（4）简化部署

在大型单块系统中，即使修改一行代码，也需要重新部署整个应用系统。这种部署的影响很大、风险很高，因此不敢轻易地重新部署。而微服务架构中，每个服务的部署都是 独立的，这样就可以更快地对特定部分的代码进行部署。

（5）与结织结构相匹配

我们都知道，团队越大越难管理，同时团队越大也代表系统规模越大代码库越大，这样容易引起一系列的问题。且当团队是分布式的时候，问题更严重。微服务架构就能很好地解决这个问题，微服务架构可以将架构与组织结构相匹配，避免出现过大的代码库，从而获得理想的团队大小及生产力。服务的所有权也可以在团队之 间迁移，从而避免异地团队的出现。

（6）可组合性

在微服务架构中，系统会开放很多接口供外部使用。当情况发生改变时，可以使用不同的方式构建应用，而整体化应用程序只能提供一个非常粗粒度的接口供外部使用。

（7）对可替代性的优化

在单块系统中如果删除系统中的上百行代码，也许不知道会发生什么，引起什么样的问题，因为单块系统中关联性很强。但在微服务架构中，我们可以在需要时轻易地重写服务，或者删除不再使用的服务。

2. 微服务面临的挑战

软件开发业内有一句名言“软件开发没有银弹”，虽然前面介绍了微服务很多方面的优势，但微服务并不能解决所有问题。下面我们来分析在使用微服务架构时可能面临的一些挑战。

（1）分布式系统的复杂度

使用微服务实现分布式系统的复杂度要比单块系统高。这表现在多个方面，如：性能方面微服务是拆分成多个服务进行部署，服务间的通信都是通过网络，此时的性能会受影响。同时可靠性也会受影响。数据一致性也需要严格控制，其成本也比单块系统高。

（2）运维成本

相比传统的单块架构应用，微服务将系统分成多个独立的部分，每个部分都是可以独立部署的业务单元。这就意味着，原来适用于单块架构的集中式的部署、配置、监控或者日志收集等方式，在微服务架构下，随着服务数量的增多，每个服务都需要独立的配置、部署、监控、日志收集等，因此成本呈指数级增长。

（3）部署自动化

传统单块系统部署往往是以月、周为单位，部署频度很低，在这种情况下，手动部署是可以满足需求的。而对于微服务架构而言，每个服务都是一个独立可部署的业务单元，每个服务的修改都需要独立部署。这样部署的成本是比较高的，如亚马逊，每天都要执行数十次、甚至上百次的部署，此时仍用人工部署是行不通的，要使用自动化部署。如何有效地构建自动化部署流水线，降低部署成本、提高部署频率，是微服务架构下需要面临的一个挑战。

（4）DevOps 与组织结构

传统单块架构中，团队通常是按技能划分，如开发部、测试部、运维部，并通过项目的方式协作，完成系统交付。而在微服务架构的实施过程中，除了如上所述的交付、运维上存在的挑战，在组织或者团队层面，如何传递 DevOps 文化的价值，让团队理解 DevOps 文化的价值，并构建全功能团队，也是一个不小的挑战。

微服务不仅表现出一种架构模型，同样也表现出一种组织模型。这种新型的组织模型意味着开发人员和运维的角色发生了变化，开发者将承担起服务整个生命周期的责任，包括部署和监控，而运维也越来越多地表现出一种顾问式的角色，尽早考虑服务如何部署。因此，如何在微服务的实施中，按需调整组织架构，构建全功能的团队，是一个不小的挑战。

（5）服务间依赖测试

由于微服务架构是把系统拆分为若干个可独立部署的服务，所以需要进行服务间的依赖测试。在服务数量较多的情况下，如何有效地保证服务之间能有效按照接口的约定正常工作，成为微服务实施过程中必须面临的巨大挑战。

（6）服务间依赖管理

传统的单块系统，功能实现比较集中，大部分功能都运行在同一个应用中，同其他系统依赖较少。而微服务架构则不同，在将系统功能拆分成相互协作的独立服务之后，随着微服务个数的增多，如何清晰有效地展示服务之间的依赖关系，成为了一个挑战。

3.微服务与 SOA

微服务可以讲是 SOA 的一种，但仔细分析与推敲，我们又能发现他们的一些差异。这种差异表现在多个方面，具体如表 9-8 所示。

表 9-8　微服务与 SOA

微服务	SOA
能拆分的就拆分	是整体的，服务能放一起的都放一起
纵向业务划分	是水平分多层
由单一组织负责	按层级划分不同部门的组织负责
细粒度	粗粒度
两句话可以解释明白	几百字只相当于 SOA 的目录
独立的子公司	类似大公司里面划分了一些业务单元（BU）
组件小	存在较复杂的组件
业务逻辑存在于每一个服务中	业务逻辑横跨多个业务领域
使用轻量级的通信方式，如 HTTP	企业服务产总线（ESB）充当了服务之间通信的角色

这些差异自然影响到其实现，在实现方面的主要差异如表 9-9 所示。

表 9-9　微服务与 SOA 实现对比

微服务架构实现	SOA 实现
团队级，自底向上开展实施	企业级，自顶向下开展实施
一个系统被拆分成多个服务，粒度细	服务由多个子系统组成，粒度大
无集中式总线，松散的服务架构	企业服务总线，集中式的服务架构
集成方式简单（HTTP/REST/JSON）	集成方式复杂（ESB/WS/SOAP）
服务能独立部署	单块架构系统，相互依赖，部署复杂
