借助微服务、容器和运行时实现灵活性

FORRESTER 开展的特色调查

如何获得微服务设计的好处

借助微服务、容器和运行时实现灵活性

借助微服务、容器和运行时实现灵活性

Forrester 分析师报告简介：如何获得微服务设计的好处

应用程序和运行它的平台及服务之间始终存在关系。很长一段时间，这种关系非常紧密，涵盖了许多与应用程序设计有关的领域，从应用程序的编写语言到管理行为（如监视和记录），甚至涵盖事务管理、前端 UI 开发或整合方法等领域。

本文档中的内容

从某种意义上说，应用程序平台 只是应用程序的另一个设计考虑事项。几年前，随着滚滚来袭的云计算、容器和虚拟机等平台技术的重大变革，CTO 和应用程序架构师的首要关注点是如何迁移到这些平台。平台选择是应用程序的主要部分。重点是 — 以这种或那种方式 — 过渡到“云”。

在过去的几年中，客户的重点已经转移。由 Red Hat 开展的 2016 年 IT 专业人士调查发现，只有不到 25% 的受访者将注意力实际集中在将应用程序迁移到云方面 — 大部分人都专注于编写新应用程序或维护现有应用程序。重点正从平台转向应用程序本身。

这甚至改变了他们如何看待应用程序本身。虽然目前有 64% 的受访者正在运行传统的 Java EE 应用程序，但是大约三分之一的人希望在下一年能够混合运行传统的 Java EE 和云本机应用程序，另外还有 38% 的人希望能够在云中以独占方式运行应用程序。平台的变化不是目标，它是编写新的云本机应用程序的结果。

重点关注应用程序，而不仅仅是基础设施

必须更快速地开发、测试和发布应用程序。像敏捷和 DevOps 这样的方法可以分解已交付代码的大小，从而帮助加快团队循环访问功能和新服务的速度。这已经帮助开发人员和架构师重新确定了他们的应用程序设计 — 这自然加快了微服务的快速引入。

数十年来，传统的单片应用程序并没有真正成为单片应用程序。某些关键功能早已抽象为单独的应用程序，例如 SQL 数据库或为 Java 应用程序提供服务的 JavaScript UI。

开始打破新服务的界线，而不是尝试向现有的单片应用程序中添加功能已成为一个自然的举动，而使用任何语言和框架对于特定服务都最有意义。Forrester 称这是一种实用方法 — 使用精确的工具来完成精确的工作。新的平台和部署选项使这变得更加容易，因为云实例和容器是为小型、轻量级、易于部署的应用程序而设计的。在不浪费计算资源的情况下，可以轻松部署、缩放和废弃相关服务。

应用程序设计影响平台要求，反之则不然

整个 IT 部门过去都是由他们运行的应用程序平台（如 Java 商店，C/C ++ 或 COBOL）来定义的。将应用程序放在第一位，为开发人员带来了新的自主权和创造力。这意味着 IT 组织需要改变他们在应用程序平台上的评估方式以及了解什么是重要的。IT 经理应该考虑提供允许开发人员独立工作的工具，而不是尝试找到符合其现有基础设施的开发人员或应用程序体系结构。

适应性战略应用程序平台有三个主要特征：

- **混合基础设施。** 随着服务的变化，它们运行的环境也在变化。应用程序应该在物理硬件、虚拟机、公有云（如 Amazon 和 Azure）或私有云（如 OpenStack）中具有相同的功能。这意味着开发环境应该创建可移植的应用程序。
- **多个运行时和语言。**虽然 Java 仍然是企业应用程序的第一编程语言，但它不再可能是企业中的唯一语言。应用程序环境需要同时支持多种语言，以便既可以创建新的独立服务，也可以帮助向后移植和扩展现有的应用程序。支持多种语言，使开发人员可以选择最佳的服务技术。
- **容器映像目录。**分布式应用程序的主要问题之一是增加了复杂性。保持一致性，甚至协调相关的服务，可能是非常困难的。容器提供了不变的映像，可以在开发和生产环境中保持一致性和良好质量。容器管理工具（例如映像目录，编排服务和监视功能）使开发人员和运营团队能够更轻松地管理应用程序并控制分布式体系结构的固有复杂性。

Red Hat 提供了采用 Red Hat OpenShift Application Runtimes 的完整应用程序平台，它具有 Red Hat OpenShift Container Platform 和不同运行时的容器平台及管理功能，包括 Red Hat JBoss Enterprise Application Platform、Wildfly Swarm、Vert.x、Node.js，甚至 Spring。 JBoss EAP 是一个完整的 Java EE 认证平台 ；Wildfly Swarm 是 Eclipse Microprofile 的实现，它是一个面向微服务的规范，包含 Java EE 库的一个子集。这些产品提供了从更传统的 Java 应用程序到微服务的整个 Java 系列开发环境。另外，还有一套完整的开发人员工具和应用程序服务，例如集成和业务逻辑。这套完整的开发人员和应用程序工具为开发人员提供了更多自由，并帮助组织使现有应用程序实现云就绪，以及使用云本机应用程序进行创新。

如何获得微服务设计的好处

采用的三个阶段从今天的编程模型和平台开始

作者：Jeffrey S. Hammond 和 John R. Rymer

2016 年 5 月 26 日

为何阅读本报告

使用微服务设计开发软件是热门的新方法。原因何在？微服务承诺实施的灵活性、丰富的扩展性、更快的交付和更高的弹性。但是，对于大多数企业 AD & D 团队而言，全面的微服务体系结构过于复杂，因此开发人员正在寻找实用且渐进的方式，通过新的编程模型和平台框架来获得微服务设计的好处。阅读本报告可了解这些新兴方法及其影响。

要点

开发人员寻找实用的微服务

基于微服务体系结构的纯粹、全面的开发方法对于大多数企业而言很难快速采用。开发人员转而通过新的、微服务启发的编程框架和云平台服务，以采用新方法的各个方面。

微服务正在重塑应用平台

平台供应商正在添加支持部署微服务的容器服务。与此同时，提供微服务设计和部署本地支持的新一代应用程序平台正以新的平台即服务选项的形式出现，或者作为团队可从中构建自己的平台的组件。

如何获得微服务设计的好处

采用的三个阶段从今天的编程模型和平台开始

作者：Jeffrey S. Hammond 和 John R. Rymer
共同作者：Christopher Mines、Randy Heffner、Dave Bartoletti、Claudia Tajima 和 Rachel Birrell
2016 年 5 月 26 日

目录

注释和资源

Forrester 采访了 10 位使用微服务的 AD&D 专业人员，并与 Cloudsoft、Google、Mesosphere、Microsoft、Oracle、Pivotal Labs、Red Hat、Salesforce 和 Weaveworks 进行了交谈。

相关研究文献

Boost Application Delivery Speed And Quality With Agile DevOps Practices

The Dawn Of Enterprise JavaScript

Microservices Have An Important Role In The Future Of Solution Architecture

Vendor Landscape:Public Cloud Platforms Consolidate, But New Disruptions On The Way

FORRESTER®

Forrester Research, Inc., 60 Acorn Park Drive, Cambridge, MA 02140 USA
+1 617-613-6000 | 传真：+1 617-613-5000 | forrester.com

微服务拥有广阔的前景，但是复杂性阻碍了采用

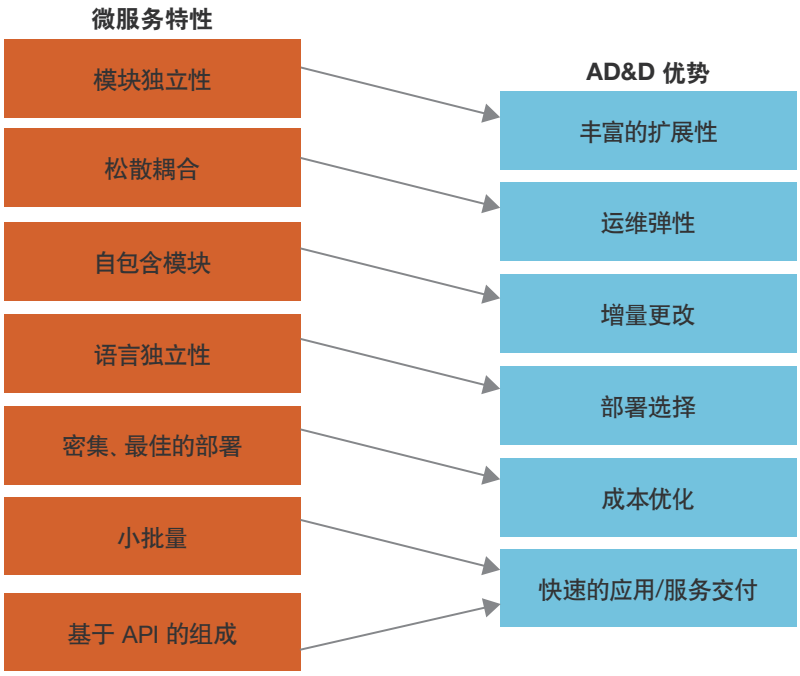
随着开发人员在客户时代面对构建应用程序的挑战，微服务设计在他们的思维中显得非常突出。[1] *微服务*是一个软件组件，它只做好一件事（而且只有一件事），是独立的，而且通过独立于特定编程语言的 API 或消息与其他微服务进行通信。[2]

微服务设计的潜在好处

开发人员使用术语*微服务*来表示旨在进行快速应用程序交付以实现业务创新和响应性的实践和技术 — 特别是（见图 1）：

- **加快应用交付。**现代应用程序交付的一个关键原则是更快地交付软件，并减小代码模块的大小，以遏制错误和 / 或故障风险。微服务设计源于这种开发实践。这种策略的连锁反应是通过 API 交互的代码模块之间的松散耦合。因此，重复使用 API 以加速开发现在是获得这种好处的另一种方式。
- **支持快速、增量的应用更改。**小软件组件的松散耦合还可提高开发人员将能更改一项服务而不中断依赖于它的其他服务的可能性。在规模上，这允许组织每天多次发布小的更改。[3] Amazon 使用以微服务为中心的交付管道，仅在 2014 年就发布了超过 5000 万次更改，平均每秒两次。[4]
- **确保规模的弹性。**独立的微服务不仅可以实现增量、独立的更改，而且允许系统的其他部分在微服务出现故障时继续运行。Netflix 使用一系列以 simian 为主题的测试工具，通过随机查杀基础架构组件来测试生产中的微服务弹性，以便快速隔离和消除硬编码的依赖关系。[5]
- **优化生产成本。**微服务设计通过将更多资源打包到虚拟机或容器中来提高软件部署的密度。此外，松散耦合的独立组件更容易根据需要进行安装和关闭，仅在软件运行时才会产生成本。
- **支持丰富的扩展性。**作为独立服务构建的应用（特别是那些使用无状态设计的应用），只需添加新实例即可在商品硬件上实现一致的性能扩展。开发人员喜欢从小型工作负载轻松扩展到大规模工作负载，而无需重写其应用或使用昂贵的基础架构。
- **让开发人员可以使用最好的工具来工作。**通过 API 交互的独立微服务还允许开发人员为任何特定服务选择最佳语言和框架。

图 1 开发人员对微服务的特性和好处的看法



四个主要限制增加了复杂性并阻止了微服务设计的采用

为实现微服务的这些潜在好处，开发人员必须克服四个主要限制。这些都增加了当前应用程序开发环境和实践的复杂性，使得过渡具有挑战性。同时克服所有四个限制是一项艰巨的任务。

- **微服务带来了一个依赖性管理挑战。**大多数开发人员至少在理论上同意微服务的定义。但是，在微服务设计支持的广泛分布的应用程序中，对于管理微服务和更改管理之间的设计和运行时依赖关系的体系结构和实践并没有什么共识。

> “我们正试图打破一些习惯。您必须将每个依赖关系视为第三方集成，即使您拥有它亦如此。另外，还期望能够涉及向前和向后兼容性。”（大型金融服务公司云工程总监）

- **持续集成和交付难以扩展。**简单地加快现有的发布管理流程会导致灾难和不稳定。能够以最小的风险对应用进行增量更改的能力，取决于允许独立发布的良好整体设计、体系结构和交付流程。成功的团队使用彻底的自动化，简化分段，控制流量，使用功能和版本标志，并组织围绕各微服务的并行交付管道。不要期望这些能力在一夜之间成熟。

- **微服务组合需要新的 API 和消息传递技术。**为使其专注于在更大的应用程序中工作，微服务必须能够向其他微服务请求帮助，并反过来为其他微服务的请求提供服务。开发人员实现此目标最常用的方法是什么？使用 RESTful API 和 JSON 有效负载，或将消息从队列或事件总线中拉出。例如，Netflix 开发人员使用 Ribbon 库对微服务间的进程间请求进行服务和负载平衡。[6]

› **公有云平台上的 Internet 级应用程序需要新的平台。**AWS、Google 和 Netflix 等微服务领导者在全球大规模范围内运营，同样需要软件和基础体系结构。公有云平台可提供大量的计算和存储池，但在保持可靠运营和最优成本的同时利用这一容量，需要新的应用程序结构、服务运行时和能够管理大规模分布式应用程序的平台服务。单独采用无状态应用程序体系结构是不够的。

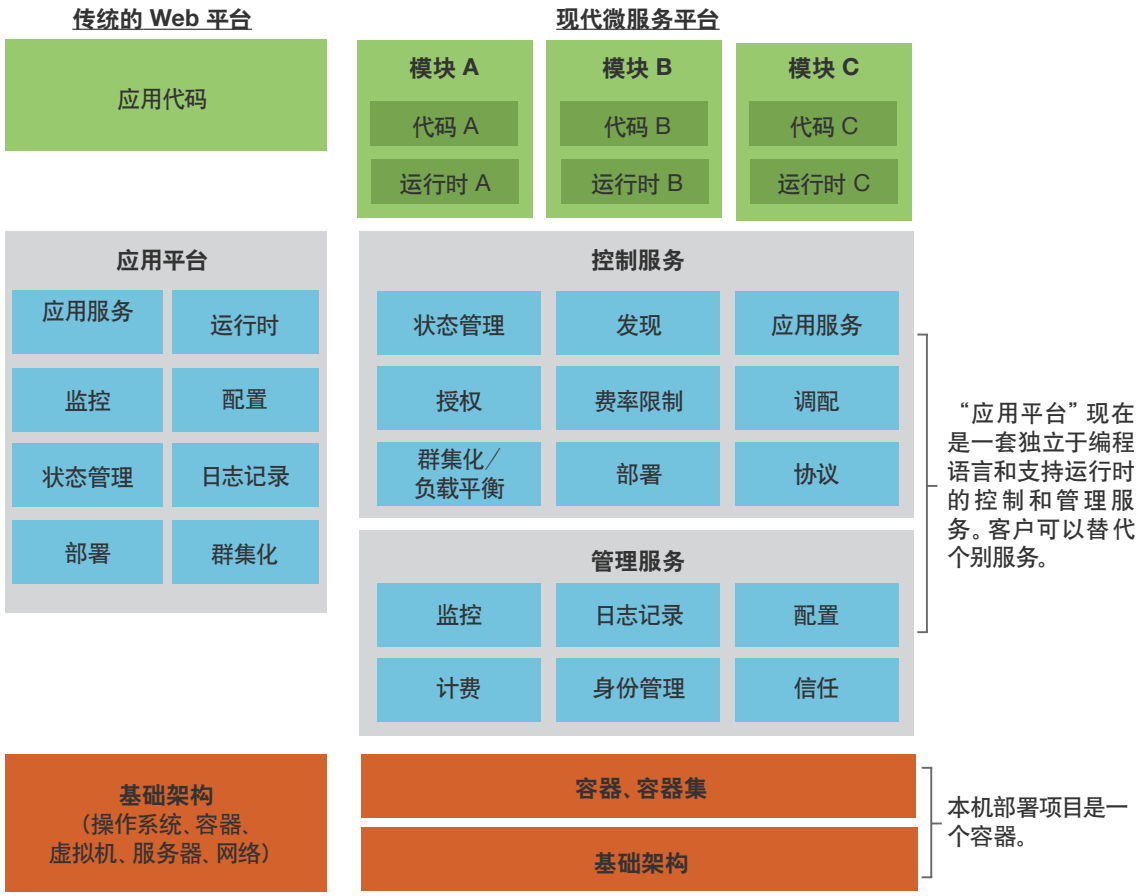
克服微服务复杂性的三种方法

积极的开发人员总会找到前进的方法。为交付他们的即时产品，而不是从头开发全新的企业架构，开发人员正在通过以下途径寻找实用、增量的方法来获得与微服务设计相关的好处：

1. 使今天的编程模型和平台适应微服务设计。
2. 在今天的平台上采用新的编程模型。
3. 采用新的编程模型和新的微服务平台。

这与 Google 和 Netflix 等微服务创新者的发展是相同的。最早的推动者开始使用传统的编程模型和平台。随着他们对微服务设计的承诺加深，他们采用了容器（主要是 Docker），而不是虚拟机 (VM) 进行部署。[7] 他们还发明或采用了新的平台服务来托管、连接、组合、扩展和管理其微服务（见图 2）。企业 AD & D 团队得益于追随先行者的足迹。

图 2 Web 平台与微服务平台



使今天的编程和平台适应微服务

今天的典型应用程序平台（包括它们支持的 Web 编程模型）均适应微服务的发展。例如，开发人员可以选择限制 Java 模块的设计以便只做好一件事，但他们仍然必须将该 Java 代码及其配置作为归档文件部署到传统的 Java 应用服务器中。

“您自然会将您的代码库分解为更小的服务。”（大型金融服务公司云工程总监）

开发人员还可以选择在虚拟机上部署他们的应用程序（无论是传统的 C #、Java 等，还是新的微服务），而不使用容器（如同使用云基础架构服务时那样）。主要的云平台供应商现在也提供独立的容器服务，开发人员可以使用这些服务来代替虚拟机。尽管实用，但这些微服务设计和部署在现有平台上限制了开发人员只提供快速交付和增量更改的好处。原因何在？

- **与语言无关的运行时很少。**应用程序服务器倾向于至少支持一组有限的编程语言。[8] Java 应用程序服务器支持多种语言，但对于微服务方法来说过于庞大。
- **庞大的运行时会增加成本。**除非它们是虚拟化的，否则 Web 应用服务器会假定它们拥有自己的“服务器”（VM、核心或实际服务器）以运行，并且拥有充足的计算和存储资源。[9] 如果应用程序需要数千个微服务实例，则此运行时配置（一项服务；一个应用服务器实例；一个 VM、核心或服务器）将生成昂贵的账单。更好的是只在实际需要时运行实例，而传统的应用程序服务器则无法实现。对于我们交谈过的一家 Azure 公共部门客户，微服务方法将其峰值需求从 8,000 台 VM 减少到大约 1,000 台。
- **运行时托管相互依赖的服务，降低了弹性。**典型的 Web 应用程序包含许多相互依赖的服务，这种设计需要运行所有服务（包括应用程序服务器运行时）以支持弹性。这与由松散耦合的独立服务组成的应用程序相反，这些独立服务可以绕过服务故障而继续运行。微服务支持更加紧凑的运行时，这些运行时可以快速可靠地启动、停止、重定位和更改。
- **存在对无状态服务的偏见。**无状态服务非常适合云平台的弹性扩展，因此类似各种 Cloud Foundry 实现的应用程序平台几乎总是用于无状态服务和应用。开发人员将状态存储在外部服务中，这与 MVC 应用程序常见的共享状态设计有很大的不同；这增加了应用程序在扩展时的复杂性。[10]

在今天的平台上采用新的编程模型

随着开发人员对微服务的采用范围不断扩大，他们对新的微服务平台、框架以及设计和交付方法的实验也越来越多。早期的宠儿已经出现。这些新的编程模型是微服务革命的先锋。

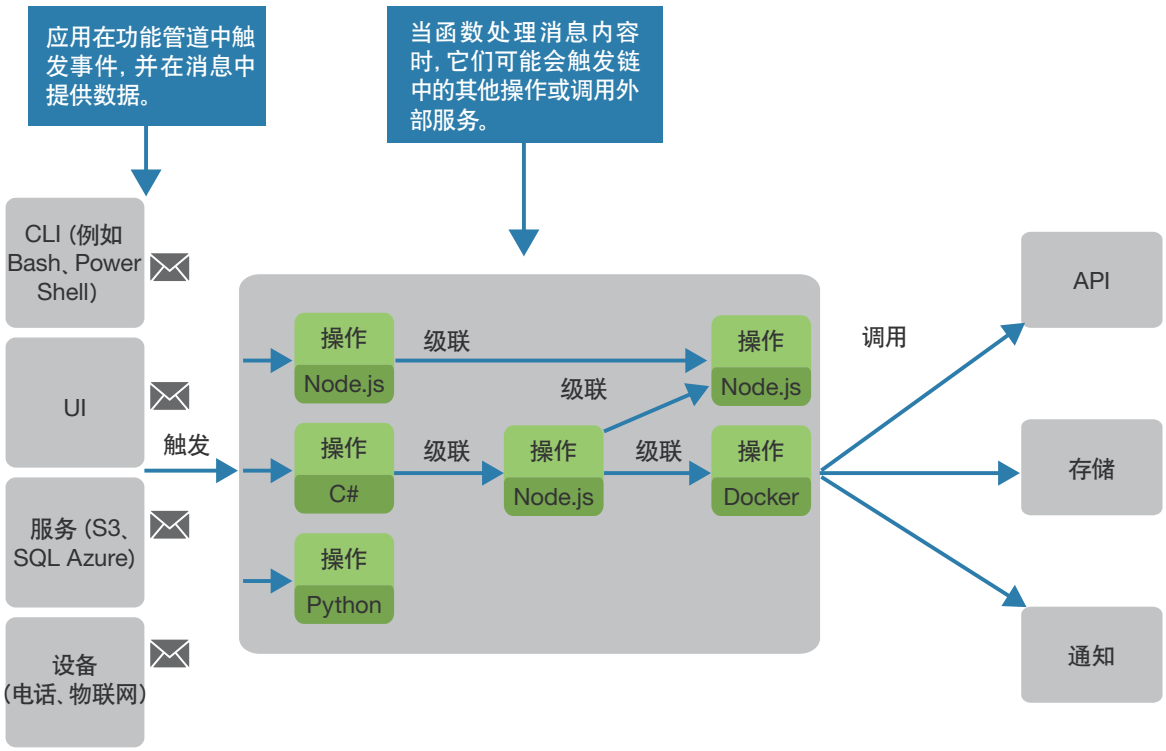
功能管道可轻松处理大规模数据

2013 年，我们观察到现代（移动）应用程序由编程模式提供支持，与传统的 .NET 和 Java 应用程序中使用的*模型 - 视图 - 控制器* (MVC) 实现模式有很大的不同。[11] 快速发展到今天，我们发现 Amazon、Google、IBM 和 Microsoft 等公有云提供商通过自动化*管道和过滤器*模式（具有通过服务、HTTP 端点或应用内活动按需触发的功能代码片段）来支持开发人员。开发人员使用这些功能代码片段过滤传入的数据，然后触发其他过滤器（见图 3）。在该过程中，他们构建了复杂的事件管道，处理可变规模和时间的不可预测的数据流，而没有考虑底层的服务器基础架构。开发团队喜欢这个*功能管道*编程风格，因为代码：

- **很容易创建。**当 Amazon 在 2013 年推出 AWS Lambda 时，其 CTO Werner Vogels 使用了 Microsoft Excel 宏的概念作为隐喻，以传达其编程模型的简单性。[12] 编写一个宏或正则表达式的等效内容的工作比设置应用服务器、规则引擎或业务流程服务器要少得多，因此开发人员可以执行几十行代码。
- **快速部署。**开发人员部署代码片段，而不考虑虚拟机、容器或服务配置。通过成千上万次的努力来创建一个过滤器，很容易理解为什么我们说过的早期采用者已经迅速行动起来。开发人员越来越多地将底层基础架构的这种抽象称为“无服务器”，这意味着他们不必关心其过滤器下的所有基础架构和代码，但是当环境事件发生时，会正确执行这些过滤器。

› **使用便宜。**数字机构和初创公司的开发人员也喜欢功能管道，因为他们只有在过滤器实际运行时才会付费，而不管是否使用过基础架构。因此，其应用程序基础架构的执行成本，比在基于 VM 的基础架构内部署和管理复杂事件处理运行时或业务规则服务器要低一个数量级。功能管道还允许开发人员对事件只是偶尔触发的应用程序进行实验，因为他们仅为执行时间，而不是用于侦听消息的时间付费。

图 3 功能管道事件流程



随着云并发挑战的增加，角色模型看到新的兴趣

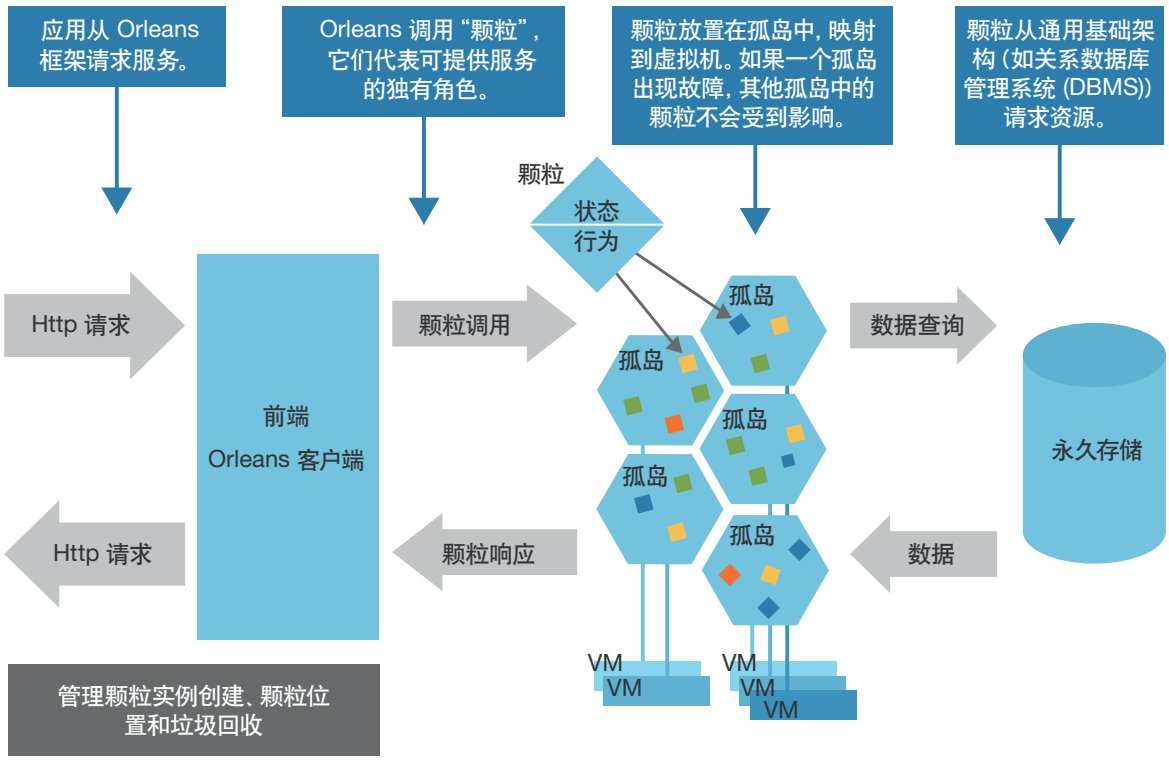
*角色*消息传递模型可追溯到 1973 年的一篇论文。[13] 它设想了“高度并行计算机的前景，由几十个、几百个甚至几千个独立的微处理器组成，每个微处理器都有自己的本地内存和通信处理器，通过高性能的通信网络进行通信”。[14] 在某些方面，主流开发和横向扩展公有云已经赶上了昔日的系统设计理论。随着横向扩展的复杂性变得更加痛苦，小部分知识型开发人员利用角色模型寻求帮助。Microsoft 的 Halo 4 云服务团队在 2011 年转向 Microsoft Research 的“Orleans”项目时就是这样做的（见图 4）。[15] 为什么？角色模型：

- **使复杂的并行处理任务更简单。**使用多线程进行并发处理的传统共享状态对象模型几乎不可能让普通开发人员实现大规模管理。角色不共享状态，并且通过异步消息传递实现并发。由于没有共享状态，因此不需要锁定角色来对其进行更新或同步处理线程。底线：与在面向对象的框架中编写多线程代码相比，更大比例的开发人员发现模型是可接近的。[16]
- **提供实现弹性的构建块。**虽然角色实现很好地处理了并发性，但是如果系统中的单个角色关闭或者网络阻碍了虚拟机之间的消息流动，它仍然可能被证明是不可靠的。新平台通过为开发人员添加冗余和故障转移功能提供帮助 Azure Service Fabric 创建“虚拟角色”，抽象不同 VM 上的多个实际角色并管理它们之间的一致性；如果一个发生故障，则集合中的其他角色会处理这种情况。Lightbend 的 Apache 授权的 Akka 框架使用可靠的代理模式，以确保跨 VM 发送的消息具有与在 VM 内的角色之间发送的消息相同的服务质量。
- **便于快速更新各个角色。**在默认的五 VM 配置下实施时，Azure Service Fabric 允许开发人员一次对一台 VM 应用更改，而不会中断当前的服务交付。每台 VM 上的角色实例使用最终的一致性模型来协调更改，同时完全将开发人员从复杂性中抽离出来。

“我们从 Service Fabric 中看到几个好处，包括滚动更新和按需横向扩展各项服务的功能”。（Mesh Systems 首席软件架构师 TJ Butler）

- **为许多当前应用程序提供了一个更简单的迁移路径。**角色与经典对象的不同之处在于它们不支持继承或允许通过函数进行通信；而是使用消息。也就是说，可以通过相对直接的方式将实体模型映射到角色，然后用消息替换函数调用。例如，Microsoft 在 Service Fabric 中实现了其 Reliable Collections，以紧密映射到许多 .NET 开发人员今天使用的传统 Collections 类。

图 4 Orleans 项目将角色映射到多个 VM 并为开发人员管理并发



在新的微服务平台上采用新的编程

应用程序平台未来会将新的编程模型与构建以支持微服务的平台结合起来。现在，这个未来正在发展，而不仅仅是 Amazon、Google、Microsoft 和 Netflix 等独特的数字企业。AD & D 团队已经通过使用 Netflix（通过其开源工具）、Kubernetes 和 Mesos（客户最常提到的微服务编排工具）提供的技术以及其他服务构建自己的平台，或通过采用 CoreOS、Docker、Engine Yard、Google、Jelastic、Microsoft (Service Fabric) 和 Red Hat (OpenShift v3) 提供的微服务平台即服务，将本地微服务平台投入测试（见图 5）。[17]

实质上，新平台使用容器作为其本地服务部署单元，并从应用程序服务器中删除控制和管理功能，以便它们可以适用于所有微服务，而无论使用哪种语言或运行时。这些平台：

› **支持自包含的、独立的、与语言无关的模块。**对于本地平台，微服务是独立、自包含的单元，能够使用平台的控制和管理服务。因此，平台与语言无关。容器图像不是语言无关的；每个都包含其所需的运行时服务。

› **支持 Internet 级应用扩展的弹性和最佳成本。**微服务的控制服务有助于实现语言独立性。这些服务提供了动态部署和调配服务，这些对于 Internet 扩展和即付即用经济至关重要，包括用于实现弹性的群集和多重部署。控制服务为同一节点或群集内的微服务交互，以及使用支持组合、管理访问和使用限制的 API 与外部服务的交互提供通信协议。最后，控制服务提供状态管理和其他应用服务，还收集平台管理平面使用的信号和数据。

› **支持弹性操作和信任边界。**在微服务平台中，监视、日志记录和配置是适用于所有服务的独立服务，有助于确保弹性和良好的性能，以及帮助解决问题。管理服务包括身份解析和通过微服务和租户的网络配置建立信任边界。使用情况跟踪和计费（如适用）也是管理平面的一部分。

图 5 新一代微服务平台

平台	编程模型	发布状态	编程语言支持	计费模型
Akka	角色模型	GA	Java、Scala	开源：商业支持可从 Lightbend 中的分层每服务器模型获得。
Amazon AWS Lambda	功能管道	GA	Node.is (JavaScript)、Java 和 Python	每次执行付费 (增量为 100 毫秒)
Google Cloud Functions	功能管道	Alpha	Node.is (JavaScript)	即将公布
IBM Bluemix OpenWhisk	功能管道	实验版	Node.is (JavaScript) 和 Swift	根据执行付费
IronWorker Iron.io	功能管道	GA	Ruby、Python、PHP、.NET、Node.is (JavaScript)	根据消耗的资源对每个客户进行定制报价
Microsoft Azure Service Fabric	角色模型	GA	C++、C#、F#	每 VM 计算实例 + 存储 + 网络 (增量为 1 分钟)
Microsoft Azure Functions	功能管道	预览版	JavaScript、C#、Python 和 PHP	免费增值层级后根据执行付费 (增量为 100 毫秒)
Pivotal Cloud Foundry Spring Cloud Services	功能管道 + 注释	GA	Java、Groovy、Kotlin	开源：商业支持可在每个容器的基础上从 Pivotal 获得。

建议

认识您的微服务之旅的目的地

您的开发人员可能已经开始了微服务设计之旅，并最终开始构建新的应用程序体系结构。这些开始工作可能采取持续集成和交付实践以及新工具链的形式。它们可以在公有云平台和服务所启用的新设计中找到，或者在使用 API 的增加的软件组合中找到。所有这些道路都通向微服务。

AD & D 领导者现在最重要的举措是认清目的地，并帮助渠道开发人员为实现安全有效的战略而努力。短期待办事项包括：

- **缩小部署单元。**小型部署单元的任何进展都是实现微服务的好处的进展。
- **调查您的开发人员，以了解当前应用程序适应情况如何。**期待向微服务迈进，以推动应用程序现代化决策的新浪潮。具有良好结构的实体模型的应用程序将适合使用角色模型框架的现代化，而紧密耦合的单一“烂泥团”将成为新的微服务背后的替代、弃用或封装候选内容。
- **找到新的体系结构平衡。**开发人员今天做出的选择将会长期影响企业架构，因此对于 AD & D 领导者而言，了解微服务如何已进入他们的应用程序组合很重要。为了迈向微服务的全面体系结构，将良好的设计实践作为准则和惯例，并确定最有效的工具和方法，以应对微服务设计的风险。
- **使新的工作负载适应最佳的编程模式。**使用功能管道进行实时流处理；ETL；文件处理；具有不可预测、可变或较低基础架构需求的实验应用；以及作为移动应用的基础架构服务层。将角色模型框架用于高扩展性事务系统，或作为连接的 IoT 设备的服务器端表示。
- **明确地决定平台服务的构建与购买。**构建其自己的微服务和 / 或容器管理平台的 AD & D 团队会将中间件工程技能用于该任务。大多数企业都无法实现这样的平台工程承诺，最好采用一个集成平台。

这意味着什么

为无服务器、无应用的未来做好准备

无服务器计算是服务的聚合，无需了解用于支持它们的基础架构或平台服务。微服务可创造无服务器、无应用的未来。原因何在？

- **今天的 PaaS 投资可实现无服务器计算。**支持微服务的编程模型和平台将这一愿景视为第一代平台即服务的自然演进。具有讽刺意味的是，本地微服务平台本身提供微服务交互以及微服务环境运行所需的服务，但是这些服务大多隐藏于开发人员的编程模型、清单和策略定义背后。
- **微服务可实现对应用的协作控制。**在做好一件事（而且只有一件事）的交互服务的世界中，什么是应用程序？在微服务世界中，应用程序定义为其：a) 业务目标；b) 与其他微服务的交互，以在任何特定时刻实现此目标，而不是由包含相关项目的部署存档实现。因此，微服务设计会将对实施一致性较高的单片应用程序服务器中应用程序项目和服务的控制转移到实施一致性较低的服务网络。即使在单一企业内，多个服务的所有者也将分担应用程序性能、扩展性、可靠性、安全性和更改管理的责任。

与分析师接洽

通过与 Forrester 思想领袖合作，将我们的研究成果运用于您的特定业务和技术计划，让您对自己的决策更有信心。

分析师垂询

提出与我们的研究相关的问题；Forrester 分析师会帮助您将其付诸实践并采取后续步骤。安排与分析师进行 30 分钟通话，或者选择通过电子邮件接收回复。

详细了解垂询，包括从讨论中获得最多信息的提示。

分析师咨询

利用对您的特定业务和技术挑战的深入分析，将研究成果付诸实践。接洽包括定制咨询电话、战略日、座谈会、演讲和网络研讨会。

详细了解互动咨询活动以及我们如何能够支持您的计划。

补充材料

为撰写本报告而采访的公司

Cloudsoft

Google

Mesosphere

Microsoft

Oracle

Pivotal Labs

Red Hat

Salesforce

Weaveworks

尾注

[1] 有关微服务和应用程序体系结构的讨论，请参阅“Microservices Have An Important Role In The Future Of Solution Architecture”Forrester 报告。

[2] “一件事”做得好是从服务消费者的角度而言，它可以是一种小型服务或大型而强大的服务。大型服务的实施将进一步细分为更多的微服务。资料来源：Chris Richardson，“Introduction to Microservices”，NGINX，2015 年 5 月 19 日 (https://www.nginx.com/blog/introduction-to-microservices/)。

[3] 在客户需求和竞争压力的推动下，应用程序开发和交付 (AD&D) 领导者有必要更快地交付应用程序，但许多人仍纠结于从何处开始，以及接下来会发生什么。他们感到这种明显的巨大变化令人不知所措。好消息是什么？道路现在已经很好理解了，旅程是一个渐进的过程，进展是逐步、逐个实践以及逐个团队取得的。要了解更多信息，请参阅“Boost Application Delivery Speed And Quality With Agile DevOps Practices”Forrester 报告。

[4] 有关更多信息，请观看 AWS 2015 会议上讨论其发布管理实践的以下视频。资料来源：“AWS re:Invent 2015 | (DVO202) DevOps at Amazon:A Look at Our Tools and Processes”，YouTube，2015 年 10 月 15 日 (https://www.youtube.com/watch?v=esEFaY0FDKc)。

[5] 有关 Netflix Simian Army 及其工作原理的更多信息，请查看以下博客。资料来源：“The Netflix Simian Army”，Netflix 技术博客，2011 年 7 月 19 日 (http://techblog.netflix.com/2011/07/netflix-simian-army.html)。

[6] 有关更多信息，请阅读以下博客。资料来源：Allen Wang 和 Sudhir Tonse，“Announcing Ribbon:Tying the Netflix Mid-Tier Services Together”，Netflix 技术博客，2013 年 1 月 28 日 (http://techblog.netflix.com/2013/01/announcing-ribbon-tying-netflix-mid.html)。

[7] 为什么是容器而不是（或优先于）VM？ Forrester 在下面简要总结了原因。请参阅“Brief:Why Docker Is All the Rage”Forrester 报告。

[8] Microsoft 的 .NET 运行时用于 C # 和 Visual Basic。Zend Technologies 的运行时仅支持 PHP。除 Java 之外，Java 运行时还支持许多语言。

[9] IBM 的 WebSphere Application Server 和 Oracle 的 WebLogic 均允许开发人员在单独的安装中配置虚拟应用服务器。其结果可能是将服务更紧密地打包到单个物理应用程序服务器实例中。AD & D 团队可以使用应用服务器虚拟化在 WebSphere 或 WebLogic 中创建“微服务环境”。

[10] MVC：模型 - 视图 - 控制器。

[11] 有关移动开发和 MVC 的更多信息，请参阅“The Future Of Mobile Application Development”Forrester 报告。

[12] 资料来源：Goran (Kima) Kimovski，“Is It Time For Cloud 2.0?”TriNimbus 博客，2014 年 11 月 14 日 (http://www.trinimbus.com/blog/is-it-time-for-cloud-2-0/)。

[13] 资 料 来 源：Carl Hewitt、Peter Bishop 和 Richard Steiger，“IJCAI’73 Proceedings of the 3rd international joint conference on Artificial intelligence”，ACM 数字图书馆，1973 年 8 月 20 日 (http://dl.acm.org/citation.cfm?id=1624804)。

[14] 资料来源：William Douglas Clinger，“Foundations of Actor Semantics”，DSpace@MIT，1981 年 5 月 1 日 (http://hdl.handle.net/1721.1/6935)。

[15] 有关 Halo 4 团队经验的更多信息，请观看以下视频。资料来源：Sergey Bykov 和 Hoop Somuah，“Using Orleans to Build Halo 4’s Distributed Cloud Services in Azure”，Channel19，2014 年 4 月 2 日 (https://channel9.msdn.com/Events/Build/2014/3-641)。

[16] 有关 Actor（角色）模型方法对并发性的好处的更多信息，请访问以下演示文稿。资料来源：Dror Bereznitsky，“The Actor Model - Towards Better Concurrency”，SlideShare，2010 年 1 月 17 日 (http://www.slideshare.net/drorbr/the-actor-model-towards-better-concurrency)。

[17] 要了解公有云平台市场的关键部分，以及新的需求和技术将如何推动下一阶段的增长，请参阅“Vendor Landscape:Public Cloud Platforms Consolidate, But New Disruptions On The Way”Forrester 报告。

FORRESTER®
CHALLENGE THINKING. LEAD CHANGE.

我们与业务和技术负责人合作，以制定能够推动企业发展的以客户为中心的战略。

产品和服务

› 核心研究和工具
› 数据和分析
› 同行协作
› 分析师参与
› 咨询
› 活动

Forrester 的研究和见解专为您的角色和重要业务方案量身定制。

我们担任的角色

营销和战略专业人员
CMO
B2B 营销
B2C 营销
客户体验
客户洞察
电子商务和渠道战略

技术管理专业人员
CIO
› 应用程序开发和交付
企业架构
基础设施和运营
安全和风险
采购和供应商管理

技术行业专业人员
分析师关系

客户支持

有关硬拷贝或电子副本的信息，请与客户支持部联系，电话为 +1 866-367-7378 或 +1 617-613-5730，电子邮件地址为 clientsupport@forrester.com。我们为学术性和非营利性机构提供批量折扣和特价。

技术不创造文化；它支持文化。Red Hat 是开源软件的领先供应商，正在利用其社区和工程师的协作能力和创造力来开发技术。Red Hat 一直是 Linux 容器和编排等技术的创新者和贡献者，这些技术是其 Red Hat OpenShift Container Platform 的基础。Red Hat 还有一套完整的中间件产品，从 Java EE、响应式编程和 Eclipse Microprofile、API 管理到业务流程引擎，应有尽有。这些中间件服务为开发人员提供了真正独立于平台的工具，使他们能够在本地物理硬件、公有云 / 私有云和容器中工作。Red Hat 产品提供的不仅仅是技术，还提供开放的文化和协作氛围，可以让开发人员和技术人员创建更多内容。有关现代化应用程序开发的更多信息，请访问 red.ht/AppMod；若要学习如何使用多运行时构建新的云本机应用程序，请访问 red.ht/rhoar。