# 解密微服务

微服务是一种架构风格和软件开发方法以满足现代业务需求。微服务不是一种发明，它更多的是从之前架构风格的一种演进。

我们将通过仔细观察微服务架构从传统单机架构的演进来开启本章。我们也会考察微服务的定义、概念和特性。最后，我们会分析微服务的经典实用案例，并且建立微服务和其它架构方法的相同点和关系，像**面向服务架构（SOA）**和**Twelve-Factor应用**。Twelve-Factor应用定义了一套在云上开发应用程序的软件设计原则。

在这一章，你会学到：

* 微服务的演进
* 微服务架构定义和示例
* 微服务架构的概念和特性
* 微服务架构的经典实用案例
* 微服务和SOA及Twelve-Factor应用的联系

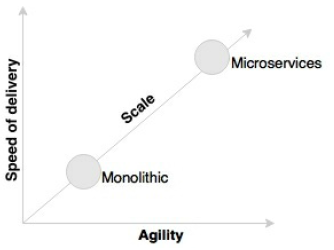
## 微服务的演进

微服务是SOA之后日益流行的架构模式之一，由DevOps和云提供补充。微服务的演进受到了近几年现代企业中颠覆性的数字化创新趋势和技术演进的极大影响。我们将在这一段中考察这两个因素。

## 业务需求作为微服务演进的催化剂

在这个向数字化转型的时代，企业更多的采用技术来作为从根本上提升收益和客户基础的关键因素之一。企业主要使用社交媒体、手机、云、大数据和物联网作为达到颠覆性创新的工具。利用这些技术，企业找到了快速渗透市场的新途径，这个传统IT交付机制带来了严重挑战。

下图展示了传统开发和微服务对比像敏捷性、交付速度和扩展性等新的企业挑战的状态。



**提示**

现代架构期望最大化替换它们部件的能力，并最小化替换它们部件的成本。微服务方法就是实现这一点的途径。

## 技术作为微服务演进的催化剂

新兴技术也使我们重新思考构建软件系统的方式。比如，几十年前，我们甚至不能想象一个没有二阶段提交的分布式应用。之后，NoSQL数据库使我们以不同的方式来思考。

类似的，这些技术上的范式转移重塑了软件架构的所有层面。

HTML5和CSS3的出现以及移动应用程序的进步重新定位了用户界面。客户端JavaScript框架，如Angular, Ember, React, BackBone等等，由于它们的客户端渲染和响应式设计而变得极其流行。

与云采取蒸汽进入主流，**平台即服务（PaaS）**供应商，如Pivotal CF, AWS, Salesforce.com, IBMs Bluemix, RedHat OpenShift等等使我们重新思考构建中间组建的方式。由Docker创建的容器革命极大影响了基础设施空间。现在，基础设施被视为商品服务。

集成格式也随着**集成平台即服务（iPaaS）**的出现而发生改变。如Dell Boomi, Informatica, MuleSoft等这些平台是iPaas的例证。这些工具帮助组织扩展了传统企业之外的集成边界。

NoSQLs彻底改变了数据库空间。几年前，我们只有几个流行的数据库，都是基于传统数据建模原则。我们如今有一个很长的数据库列表：Hadoop, Cassandra, CouchDB和Neo 4j等等。这些数据库每个都解决了某些特定架构问题。

## 命令式架构演进

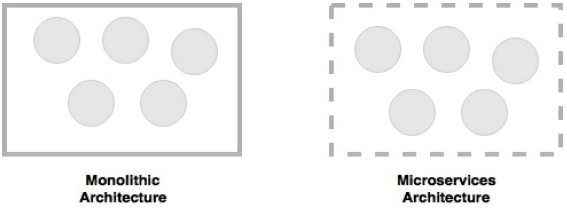
应用程序架构一直在满足苛刻的业务需求和技术的演变。架构经历了从古老的大型机系统的发展到完全抽象的云服务，如AWS Lambda。

**提示**

使用AWS Lambda，研发人员可以立即去掉他们的“功能”并放到一个完全托管的计算服务中。

阅读更多关于Lambda：<https://aws.amazon.com/documentation/lambda/>

不同的架构方法和风格，如大型主机、客户服务器、N-层、面向服务，在不同时期都很流行。无论选择哪一种架构风格，我们总是用来构建一种或另外一种组成形式的单机架构。微服务架构是现代需求演变的结果，如敏捷、快速交付、新兴技术和从前几代架构的学习。



微服务帮助我们打破单机应用的边界，并构建一个系统中逻辑独立的更小型系统，如上图所示。

**提示**

如果我们把单机应用看做视为一组包含物理边界的逻辑子系统，那么微服务是一组没有封闭物理边界的独立子系统。

## 什么是微服务

微服务是当今许多组织使用的一种架构风格，作为一种游戏规则来实现高度敏捷、快速交付以及可扩展性。微服务给我们提供了一种方式来开发更加物理隔离的模块化应用。

微服务不是一个新的发明。许多组织，如Netflix，Amazon和eBay，成功地用分治技术从功能上将它们的应用划分成更小的原子单元，每个执行单一的功能。这些组织解决了许多他们在单机应用上碰到的普遍问题。

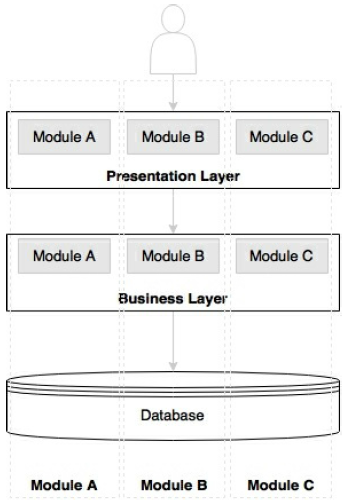
随着这些组织的成功，许多其他组织开始采取这作为一种常规模型来重构他们的单机应用。之后，evangelist把这个模式叫做微服务架构。

微服务起源于Alistair Cockburn创造的六边形架构的想法。六边形架构也是为人所知的端口和适配模式。

**提示**

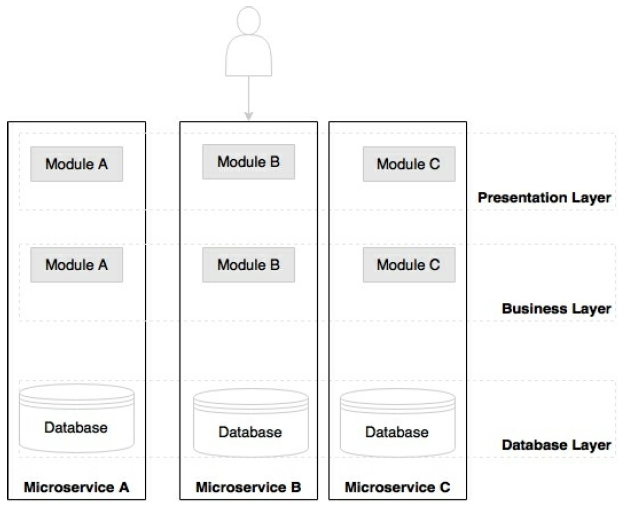
阅读更多关于六边形架构：<http://alistair.cockburn.us/Hexagonal+architecture>

微服务是一种架构风格或者一种构建一套具备业务能力的IT系统的方法，这些业务能力包括自治、自包含和松耦合。



上图描述了一个传统的N-层应用架构，有展示层、业务层和数据库层。模块A,B和C代表3种不同的业务能力。图中的层表示架构关注点的分离。每一层有拥有与该层相关的所有三个业务功能。展示层有所有3个模块的Web组件，业务层有所有3个模块的业务组件，数据库主机有所有3个模块的表。大多数情况下，层是物理可扩展的，然而层里面的模块是硬连线的。

现在让我们考察一个基于微服务的架构。



我们可以在上图中注意到，在微服务中边界是反向的。每一个垂直分片代表一个微服务。每一个微服务有属于它自己的展示层、业务层、数据库层。微服务与业务能力保持一致。通过这样做，改变一个微服务并不影响其它的。微服务并没有一个标准的通讯和传输机制。通常，微服务之间的互相通信广泛采用轻量级协议，如HTTP和REST，或者消息协议，如JMS和AMQP。特殊情况下，可能会选择更加优化的通信协议，如Thrift, ZeroMQ, Procotol Buffers或者Avro。

由于微服务更符合业务功能，并且有可独立管理的生命周期，因此它是企业开展DevOps和云的理想选择。DevOps和云是微服务的两个方面。

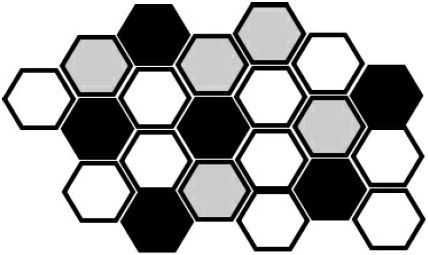
**提示**

DevOps是一种IT重组来缩小传统IT开发和运营之间的差距，为获取更好的效率。

获取更多关于DevOps：<http://dev2ops.org/2010/02/what-is-devops/>

## 微服务-蜂窝类比

蜂窝是一个理想的类比来表现微服务架构的演进。



在现实世界中，蜜蜂通过标准六边形蜡蜂窝来构建蜂巢。他们开始很小，用不同的材料来构建蜂窝。建筑是基于建筑时可用的。重复的蜂窝形成一个样品并最终组织成一个牢固的组织结构。蜂巢中的每一个蜂窝都是独立的，但也与其它蜂窝集成在一起。通过新增蜂窝，蜂巢有组织的成长为一个大型牢固架构。每个蜂窝里面的内容是抽象的，而且对外不可见。损坏其中一个蜂窝不会影响其它的蜂窝，而且蜜蜂可以在不影响整体蜂巢的情况下重新构建这些蜂窝。

## 微服务原则

在这一节，我们会考察为服务架构的一些原则。这些原则是设计和开发微服务时的“必需品”。

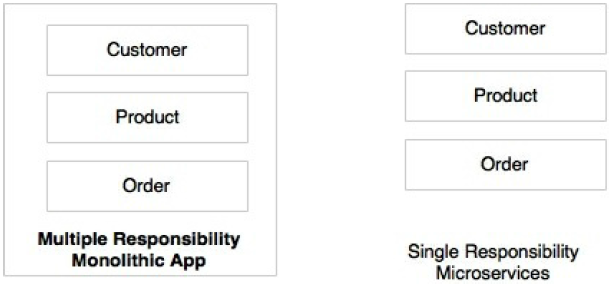
## 一服务一职责

单职责原则是SOLID设计模式定义的原则中的一部分。它表述的是一个单元模块只应该有一个职责。

**提示**

获取更多关于SOLID设计模式：<http://wiki.c2.com/?PrinciplesOfObjectOrientedDesign>

这表明，一个单元，也可以是一个类，一个方法，或者是一个服务，应该只有一个职责。任何时候不应该两个单元模块分担一个职责或者一个单元模块有不止一个职责。一个单元模块有不止一个职责时表明是高耦合。



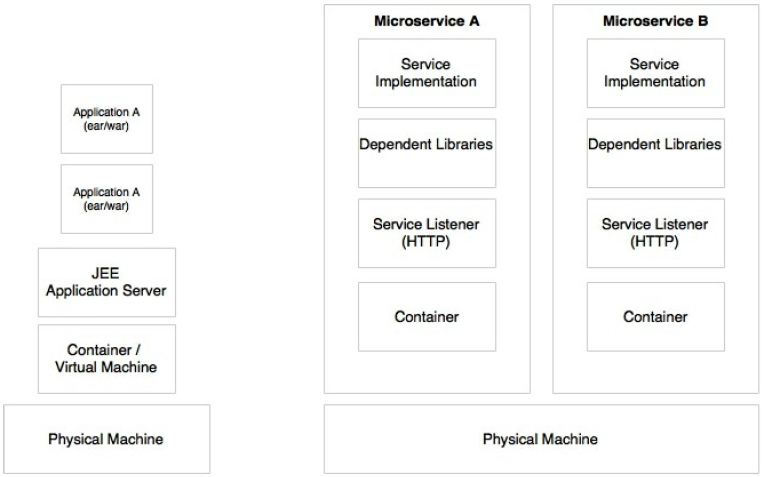
如上图所示，**Customer**, **Product**和**Order**在电子商务应用中是不同的功能。与其将它们构建在一个应用中，不如用3个不同的服务，每个恰好负责一个业务功能，所以更改某个功能职责时不会影响其它的。在上图情景中，**Customer, Product**和**Order**被处理为3个独立的微服务。

## 微服务是自治的

微服务是自包含的、可独立部署的和自治的服务，它全面负责业务的能力和执行。它们绑定所有的依赖，包括包依赖和执行环境，如Web服务器和容器或者虚拟机之类的抽象物理资源。

微服务和SOA之间的一个主要不同点在于它们的自治层级。大多数SOA应用提供服务级的抽象，微服务更进一步抽象了实现和执行环境。

在传统应用开发中，我们构建一个WAR或者EAR，然后将它部署到一个JEE应用服务器，如Jboss, WebLogic, WebSphere等等。我们可能在同一个JEE容器中部署多个应用。在微服务中，每个微服务会被构建成一个肥胖的Jar，嵌入所有的依赖，以一个独立的Java进程来运行。



微服务也可能由它自己的容器来执行，如上图所示。容器是便携、独立管理和轻量级的运行环境。容器技术，如Docker，是部署微服务的理想选择。

## 微服务的特性

这一章前面随意讨论了微服务的定义。Evangelists和从业者对微服务有很强但有时不一样的意见。微服务还没有一个单一的、具体的和普遍被接受的定义。然而，所有成功的微服务实现展示出许多共同的特性。因此，理解这些特性比执着于理论上的定义更重要。这些共同特性中的一部分有在这一节中详细说明。

## 服务是一流的公民

在微服务的世界中，服务是一流公民。微服务以APIs的形式暴露服务端口，并抽象所有的实现细节。内部实现逻辑、架构和技术（包括编程语言、数据库和服务机制的质量等等）是完全隐藏在服务API后面的。

而且，在微服务架构中没有更多的应用开发，与之替代的是组织者关注于服务开发。在大多数企业中，这需要在构建应用程序的方式上发生重大思想转变。

在一个**用户画像**微服务中，内部的数据结构、技术、业务逻辑等等都是隐藏的，他们不会暴露或对外部实体可见。访问被限制在服务端和APIs上。举一个实例，用户画像微服务可能会暴露**注册用户**和**获取用户**两个APIs供他人调用。

## 微服务中服务的特性

就像微服务或多或少有一点SOA的风味，许多在SOA中定义的服务特性也适用于微服务。

下面是一些也适用于微服务的服务特性：

* **服务约定**：类似于SOA，微服务是通过定义好的服务约定来描述的。在微服务的世界中，JOSN和REST是被普遍接受用于服务通信的。在JSON/REST的用例中，有许多技术用于定义服务约定。JSON Schema, WADL, Swagger和RAML是少数几个示例。
* **松散耦合**：微服务是独立和松耦合的。在大多数用例中，微服务接收一个事件作为输入并响应另外一个事件。Messaging, HTTP和REST在微服务交互中被广泛使用。基于消息的端提供更高层次的解耦合。
* **服务抽象**：在微服务中，服务抽象不仅仅是服务实现的抽象，也给所有库和环境明细提供一个全面的抽象，就如之前讨论的。
* **服务重用**：微服务是粗粒度的可重用业务服务。可以被移动设备、桌面通道、其它微服务甚至其它系统访问。
* **无状态**：好的微服务设计是无状态的，而且没有任何共享状态或服务维持的会话状态。这里的用例需要维持状态，它们被维持在一个数据库，也可能是内存中。
* **服务是显露的**：微服务是显露的。在一个经典微服务环境中，微服务自我告知它们的存在并是它们自己可被发现。当服务停了，它们自动将自己从微服务体系中脱离出来。
* **服务互通性**：服务是互通的，因为它们使用标准协议和消息交互标准。Messaging, HTTP等这些用作传输机制。在微服务世界中，REST/JSON是开发能彼此协作的服务的最流行方法。如果需要在通信上进一步优化，可以用其它的协议，如Protocol Buffers, Thrift, Avro或者Zero MQ。不过，使用这些协议可能会限制服务的整体互通性。
* **服务可组合性**：微服务是可以组合的。服务可组合性是通过服务匹配或服务编排来完成的。

**提示**

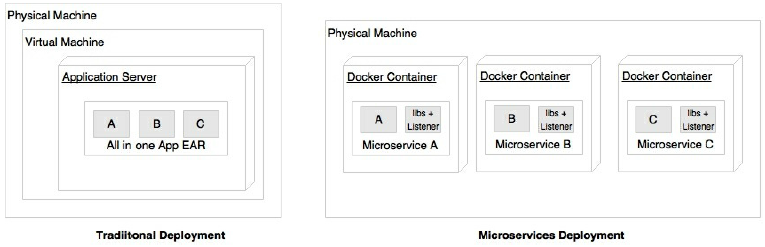
获取更详细的SOA原则可见：<http://serviceorientation.com/serviceorientation/index>

## 微服务是轻量级的

良好设计的微服务趋向于单个业务能力的实现，所以它们只执行一个功能。结果，我们在大多数实现中看到的一个共同特征是微服务的规模变小。

当选择支持的技术，如Web容器，我们会保证它也是轻量级的以至于整体规模依旧易于管理。例如，Jetty或者Tomcat是作为应用容器更好的选择，相比于更加复杂和传统的应用容器，如WebLogic或者WebSphere。

相比于管理程序，如VMWare或Hyper-V，容器技术也帮助我们保持基础规模尽可能小，如Docker。



如上图所示，微服务通常部署在Docker容器中，它封装了业务逻辑和所需要的库。这帮助我们快速复制整体配置到一个新的机器或者一个完全不同的主机环境，甚至移动到一个不同的云环境。由于没有物理基础设施依赖，容器化的微服务易于移植。

## 多语言架构的微服务

由于微服务是自治的，抽象了服务API背后的一切，使不同的微服务有不同的架构成为可能。我们在微服务实现中看到的一些共同特性是：

* 不同服务使用相同技术的不同版本。一个微服务可能在Java1.7上编写的，另一个可以在Java1.8。
* 用不同语言开发不同微服务。比如一个微服务用Java开发另一个用Scala。
* 使用不同架构，比如一个微服务用Redis缓存服务数据，而另外一个微服务用MySQL作为持久化数据存储。

