解析微服务架构(一)：什么是微服务

dW_Vikki | *Apr 20 2016* |
Visits (58402) 2

解析微服务架构系列文章将分几篇描述微服务的定义、特点、应用场景、企业集成架构的演进以及微服务转型思路和技术决策考虑等内容，并以IBM技术为例介绍如何实现微服务架构转型。

- **为什么需要微服务架构**

“微服务”架构是近期软件应用领域非常热门的概念。让我们先来看看传统IT架构面临的一些问题：

- 使用传统的整体式架构(Monolithic Architecture)应用开发系统，如CRM、ERP等大型应用，随着新需求的不断增加，企业更新和修复大型整体式应用变得越来越困难；
- 随着移动互联网的发展，企业被迫将其应用迁移至现代化UI界面架构以便能兼容移动设备，这要求企业能实现应用功能的快速上线；
- 许多企业在SOA投资中得到的回报有限，SOA可以通过标准化服务接口实现能力的重用，但对于快速变化的需求，受到整体式应用的限制，有时候显得力不从心；
- 随着应用云化的日益普及，生于云端的应用具有与传统IT不同的技术基因和开发运维模式。

此外，从技术方面看，云计算及互联网公司大量开源轻量级技术不停涌现并日渐成熟：

- 互联网/内联网/网络更加成熟；
- 轻量级运行时技术的出现(node.js, WAS Liberty等)；
- 新的方法与工具(Agile, DevOps, TDD, CI, XP, Puppet, Chef…)；
- 新的轻量级协议(RESTful API接口, 轻量级消息机制)；

- 简化的基础设施：操作系统虚拟化(hypervisors), 容器化(e.g. Docker), 基础设施即服务 (IaaS), 工作负载虚拟化(Kubernetes,Spark…)等；
- 服务平台化(PaaS)：云服务平台上具有自动缩放、工作负载管理、SLA 管理、消息机制、缓存、构建管理等各种按需使用的服务；
- 新的可替代数据持久化模型：如NoSQL, MapReduce, BASE, CQRS等；
- 标准化代码管理：如Github等。

这一切都催生了新的架构设计风格 — 微服务架构的出现。

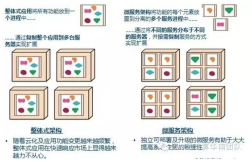
- **什么是微服务**

微服务是一种架构风格，一个大型复杂软件应用由一个或多个微服务组成。系统中的各个微服务可被独立部署，各个微服务之间是松耦合的。每个微服务仅关注于完成一件任务并很好地完成该任务。在所有情况下，每个任务代表着一个小的业务能力。

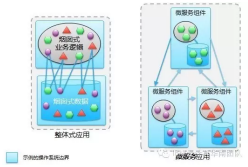
微服务的概念源于2014年3月Martin Fowler所写的一篇文章“Microservices”(http://martinfowler.com/articles/microservices.html)。

尽管“微服务”这种架构风格没有精确的定义，但其具有一些共同的特性，如围绕业务能力组织服务、自动化部署、智能端点、对语言及数据的“去集中化”控制等等。

微服务架构的思考是从与整体应用对比而产生的。



其中，对应用组件封装的方式是整体架构与微服务架构的主要差异，微服务架构将相关联的业务逻辑及数据放在一起形成独立的边界，其目的是能在不影响其他应用组件(微服务)的情况下更快地交付并推出市场。



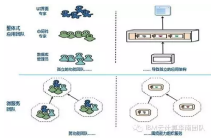
- **微服务架构的一些通用特性**

根据MartinFowler的分析，微服务架构有以下的一些通用特性，但并非所有微服务架构应用都必须具备所有这些特性：

1. 通过服务实现应用的组件化(Componentization via Services)：微服务架构中将组件

定义为可被独立替换和升级的软件单元，在应用架构设计中通过将整体应用切分成可独立部署及升级的微服务方式进行组件化设计。

2. 围绕业务能力组织服务(Organizedaround Business Capabilities)：微服务架构采取以业务能力为出发点组织服务的策略，因此微服务团队的组织结构必须是跨功能的（如：既管应用，也管数据库）、强搭配的DevOps开发运维一体化团队，通常这些团队不会太大（如：亚马逊的“Two pizzateam”– 不超过12人）。

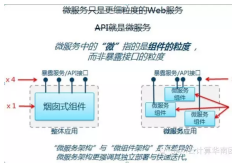


3. 产品而非项目模式(Productsnot Projects)：传统的应用模式是一个团队以项目模式开发完整的应用，开发完成后就交付给运维团队负责维护；微服务架构则倡导一个团队应该如开发产品般负责一个“微服务”完整的生命周期，倡导“谁开发，谁运营”的开发运维一体化方法。
4. 智能端点与管道扁平化(Smartendpoints and dumb pipes)：微服务架构主张将组件间通讯的相关业务逻辑/智能放在组件端点侧而非放在通讯组件中，通讯机制或组件应该尽量简单及松耦合。RESTful HTTP协议和仅提供消息路由功能的轻量级异步机制是微服务架构中最常用的通讯机制。
5. “去中心化”治理(DecentralizedGovernance)：整体式应用往往倾向于采用单一技术平台，微服务架构则鼓励使用合适的工具完成各自的任务，每个微服务可以考虑选用最佳工具完成(如不同的编程语言)。微服务的技术标准倾向于寻找其他开发者已成功验证解决类似问题的技术。
6. “去中心化”数据管理(DecentralizedData Management)：微服务架构倡导采用多样性持久化(Polyglot Persistence)的方法，让每个微服务管理其自有数据库，并允许不同微服务采用不同的数据持久化技术。
7. 基础设施自动化(InfrastructureAutoma

tion)：云化及自动化部署等技术极大地降低了微服务构建、部署和运维的难度，通过应用持续集成和持续交付等方法有助于达到加速推出市场的目的。

8. 故障处理设计(Designfor failure)：微服务架构所带来的一个后果是必须考虑每个服务的失败容错机制。因此，微服务非常重视建立架构及业务相关指标的实时监控和日志机制。
9. 演进式的设计(EvolutionaryDesign)：微服务应用更注重快速更新，因此系统的计会随时间不断变化及演进。微服务的设计受业务功能的生命周期等因素影响。如某应用是整体式应用，但逐渐朝微应用架构方向演进，整体式应用仍是核心，但新功能将使用应用所提供的API构建。再如在某微服务应用中，可替代性模块化设计的基本原则，在实施后发现某两个微服务经常必须同时更新，则这很可能意味着应将其合并为一个微服务。

- **微服务的一些常见误解**



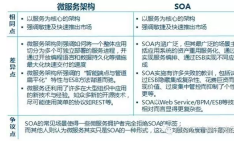
关于一些比较概念的澄清：

1. 在同一范畴内比较才有意义：
   - **微服务架构 vs. SOA** — 两者都是架构风格范畴，但其关注领域与涉及范围不同。SOA更关注企业规模范围，微服务架构则更关注应用规模范围。
   - **微服务组件 vs. 服务组件** — 两者都是描述业务功能的具体实现，其区别在于粒度不同，此外还有在可管理性、灵活性上的差异。
2. 概念混淆的不恰当比较
   - **微服务 vs. SOA — 不恰当的比较**。微服务是组件范畴，而SOA是一种架构设计风格。因此应该比较的是微服务架构与SOA。
   - **微服务 vs. API — 不恰当的比较**。API是接口，是业务功能暴露的一种机制。微服务架构是用于实施

业务功能的组件架构。因此直接比较它们是没有意义的。

- **微服务 vs. 服务—不恰当的比较**。“服务”在不同的场景下有不同的含义，需要进一步澄清其描述的语境，是指服务实施、服务暴露、服务定义还是其他？微服务亦是如此，需要有特定语境才可判断比较是否有意义。

- **微服务架构与SOA架构的比较**



- **一个简单的微服务应用例子：航班预订应用**

将航班预订应用划分为预订航班、时间表查询、计算票价、分配座位、管理奖励、更新客户、调整库存七个微服务实施。

- **哪些应用会从微服务收益？**

1. **记录型系统(System of Record)**将从微服务方法中获益最多。例如可将大型应用按相对独立的业务功能分解成若干个微服务实现。
2. **交互型系统(System of Engagement)**也将受益于微服务方法，例如渠道应用可以应用“后端服务前端”的模式实现。
3. **分析型系统(System of Insight)**则可能对微服务受益不多。其他架构模式如管道及过滤模式可能更适用于分析型系统。

- **微服务架构的优点：**

1. 每个服务都比较简单，只关注于一个业务功能。
2. 微服务架构方式是松耦合的，可以提供更高的灵活性。
3. 微服务可通过最佳及最合适的不同的编程语言与工具进行开发，能够做到有的放矢地解决针对性问题。
4. 每个微服务可由不同团队独立开发，互不影响，加快推出市场的速度。
5. 微服务架构是持续交付(CD)的巨大推动力，允许在频繁发布不同服务的同时保持系统其他部分的可用性和稳定性。

- **微服务架构的缺点：**

微服务的一些想法在实践上是好的，但当整体实现时也会呈现出其复杂性。

1. **运维开销及成本增加**：整体应用可能只需部署至一小片应用服务区集群，而微服务架构可能变成需要构建/测试/部署/运行数十个独立的服务，并可能需要支持多种语言和环境。这导致一个整体式系统如果由20个微服务组成，可能需要40~60个进程。
2. **必须有坚实的DevOps开发运维一体化技能**：开发人员需要熟知运维与投产环境，开发人员也需要掌握必要的数据存储技术如NoSQL，具有较强DevOps技能的人员比较稀缺，会带来招聘人才方面的挑战。
3. **隐式接口及接口匹配问题**：把系统分为多个协作组件后会产生新的接口，这意味着简单的交叉变化可能需要改变许多组件，并需协调一起发布。在实际环境中，一个新品发布可能被迫同时发布大量服务，由于集成点的大量增加，微服务架构会有更高的发布风险。
4. **代码重复**：某些底层功能需要被多个服务所用，为了避免将“同步耦合引入到系统中”，有时需要向不同服务添加一些代码，这就会导致代码重复。
5. **分布式系统的复杂性**：作为一种分布式系统，微服务引入了复杂性和其他若干问题，例如网络延迟、容错性、消息序列化、不可靠的网络、异步机制、版本化、差异化的工作负载等，开发人员需要考虑以上的分布式系统问题。
6. **异步机制**：微服务往往使用异步编程、消息与并行机制，如果应用存在跨微服务的事务性处理，其实现机制会变得复杂化。
7. **可测性的挑战**：在动态环境下服务间的交互会产生非常微妙的行为，难以可视化及全面测试。经典微服务往往不太重视测试，更多的是通过监控发现生产环境的异常，进而快速回滚或采取其他必要的行动。但对于特别在意风险规避监管或投产环境错误会产生显著影响的场景下需要特别注意。

- **关于微服务架构的取舍**

1. 在合适的项目，合适的团队，采用微服务架构收益会大于成本。
2. 微服务架构有很多吸引人的地方，但在拥抱微服务之前，也需要认清它所带来的挑战。
3. 需要避免为了“微服务”而“微服务”。
4. 微服务架构引入策略 — 对传统企业而言，开始时可以考虑引入部分合适的微服务架构原则对已有系统进行改造或新建微服务应用，逐步探索及积累微服务架构经验，而非全盘实施微服务架构。

下一篇微服务文章将介绍融入微服务的企业集成架构的演进，并介绍交互式系统的微服务模式及技术决策的例子。

——本文转载自 IBM 云计算华南团队（微信号：IBMCloud_SC）

相关链接：

- 解析微服务架构(三)：微服务重构应用及IBM解决方案
- 解析微服务架构(二)：融入微服务的企业集成架构