链滴

微服务架构解析

作者：wanliang

原文链接：https://ld246.com/article/1467348892182

来源网站：链滴

    近日，Spring 4.0 GA版发布这是时隔几年后Spring发布的又一个重大版本，提供了诸多的新特性。Spring 4.0是首个完全支持Java 8特性的框架，还提供了对云、大数据及微服务架构的支持。此外，Spring 4还提供了对Java EE 6和、WebSocket、SockJS以及STOMP及动态语言Groovy的支持。在新增的众多特性中，微服务架构个很有趣的概念，它的主要作用是将功能分解到离散的各个服务当中，从而降低系统的耦合性，并提更加灵活的服务支持。软件咨询师James Hughes专门撰文详细介绍了微服务的概念、作用、适用性、应用场景以测试相关的话题。
微服务架构（MSA）是一种架构概念，旨在通过将功能分解到各个离散的服务中以实现对解决案的解耦。你可以将其看作是在架构层次而非获取服务的类上应用很多SOLID原则。
从概念上来说，MSA并不难理解，不过从实践上来说，它会引发很多问题。这些服务之间是如通信的呢？服务之间的延迟是怎样的？如何测试服务呢？如何检测失败并对其作出响应呢？如果存在量互相依赖的情况该如何管理部署呢？接下来的内容就将回答这些问题，我们一起来看看MSA是否值学习和使用。
<h2>解析微服务</h2>
首先我们来看看到底什么是微服务。实际上微服务本身并没有一个严格的定义，不过从很多人的馈来看，大家都达成了这样一个共识：微服务是一种简单的应用，大概有10到100行代码。我知道使代码行数来比较实现其实很不靠谱，因此你能理解这个意思就行，不必过分拘泥于细节。不过有一点要注意，那就是微服务通常都是很小的，甚至是微型的。这意味着你不会在大型框架上看到很多小服，这是不切实际的。简单与轻量级是当今的主流。诸如Sinat a、Webbit、Finagle与<a href="http://www.senchalabs.org/connect/">Connect</a等小型框架在将你的代码包装到一个薄薄的通信层这方面做得刚刚好。
从物理角度来说，这些服务都很小，你可以在同一台机器上运行大量服务，不必担心内存或是资等问题。重申一遍，基于大型框架的简单库将会取得最后的胜利，你会发现对第三方库的依赖越来越。
这种服务层上的解耦还提供了另外一个有趣儿的选择。我们将很多老旧应用的复杂性推到了基础施层，不再受限于单个技术栈或是语言了。我们现在可以发挥出任何技术栈或是语言的优势。我们完可以通过多种语言和库来构建系统，稍后我将会对此进行介绍，其实它是个双刃剑。
你不会看到任何基于微服务的架构是托管在应用服务器上的，这是关键。本质上，微服务是自我管的，他们获取一个端口然后监听。这意味着你将失去典型的企业应用服务器所带来的很多好处，服需要提供这些必要的功能（性能度量、监控等等）。
<h2>通信</h2>
这是个有趣的话题。服务之间该如何通信呢？这个问题是无法通过一个简单的答案解决的，甚至单个解决方案中也是难以做到的。最基本的答案就是通过HTTP公开所有服务，然后以JSON作为数据换格式。服务探测（一个服务是如何找到另一个服务的）可以很简单，只需将端点细节信息放到配置件中即可（硬编码也行）。
你可能会发现在某些情况下，一个完整事务中对JSON负载的序列化与反序列化的代价会造成系瓶颈。也许JSON并不适合，你可能需要使用别的协议，如Protobuf等。
不过硬编码的URL会导致耦合。在分层应用中，这是有意义的，不过对于基于服务的架构来说，意味着你不能再被这种潜在的限制所约束。服务之间的某些通信可以是完全解耦的，事实上，有些服可以随意发布事件或是数据。他们只是将其扔出去（比如说消息总线），也许有一天出现了某个服务然后开始监听了。此外，也许系统的某些部分会以批量/离线的流程进行运作，他们可能会在几小时才从队列中取出消息。微服务架构可以实现这种灵活性而无需修改整个架构。
<h2>监控与度量</h2>
分层解决方案的组件出现问题时不会销声匿迹，要么是编译失败，要么是遇到问题时抛出异常（非你将抛出的异常隐藏掉了）。在基于服务的方式中，有的服务可能出现了问题，而其他服务则会很易发现问题（特别是在pub/sub模型中）。这意味着我们必须要能对服务进行监控和编排。事实上，知道服务还能用是不够的，服务是不是还能提供业务价值？还能否继续使用？它是可靠交易的瓶颈么
监控总是非常重要的事情，对于基于服务的架构来说更是如此，因为这时出现的失败并不容易被

现。JVM世界中的Metrics与Ostrich等库不仅可以收集度量信息，还提供了与Nagios和Ganglia等服的集成，可以将数据发送给他们。
<h2>测试</h2>
对于基于微服务架构的系统来说，测试服务并没有什么特殊之处，不过我这里要强调的是你不必对每个服务使用完整的测试套件了。因为一个服务只做一件事，因此引入系统Bug的几率明显降低了这要归功于基于服务的系统的天生的行为。我的意思并不是说不用做测试，而是建议你在使用过去的试前多思考一下。
<h2>微服务架构的应用</h2>
微服务还能在大型遗留系统中大显身手。处理遗留代码是非常有风险的。对于运行了多年的系统说，人们很有可能缺乏必要的知识来理解系统内部的运行方式。处理这种代码就像纸牌堆成的房子一，一处出现了问题就会影响到其他地方。这些系统通常都是关键任务的系统，因此错误的代价是非常的。你可以通过微服务方式解决这个难题。相对于直接深入到遗留代码基的做法来说，你可以编写一小服务，它能完成你的想法（没错，你可以复制遗留代码），并且可以通过它代理服务（比如说通过ninx/apache或是纯代码方式）。