用行为驱动开发和面向接口的设计做微服务开发

作者：Ken Pugh 译者：冬雨

阅读数：350 2019 年 2 月 11 日 话题：云计算 微服务 最佳实践

本文关键点

- 针对微服务的行为驱动开发主要侧重于消费者开发人员、生产者开发人员和测试人员三方的协作
- 使用面向接口的设计为微服务接口创建良好定义的契约
- 微服务通常需要测试替身以提升其他微服务测试的速度。
- 测试应该独立于实现
- 创建测试，检查失败是否得到适当的处理

微服务被其他微服务和整个应用程序所依赖。这就要求服务需经过良好地定义和充分地测试。通过测试对行为和接口的契约进行详细的规格说明可以实现这些目标。通过行为驱动开发 (BDD)，服务的功能由专注于待执行操作的测试来描述，而不是以这些操作的语法 (如 JSON 或 XML) 来描述。对于以自己的 BDD 测试对行为进行详细规格说明的其他微服务，这些测试的自动化通常需要测试替身。面向接口设计 (Interface Oriented Design, IOD) 包括微服务的其他契约义务，例如资源使用、吞吐量和错误报告方面的限制。行动驱动开发和面向接口设计一起使用有助于描述服务的行为，以便消费者能够更容易理解和依赖它。

上下文

BDD 涉及客户、开发人员和测试人员几方的三个视角。它通常应用于应用程序的外部行为。由于微服务是内部的，所以客户的视角即内部消费者的视角，也就是使用服务的那部分实现（例如其他微服务）。所以它主要侧重于消费者开发人员、微服务开发人员和测试人员三方之间的协作。

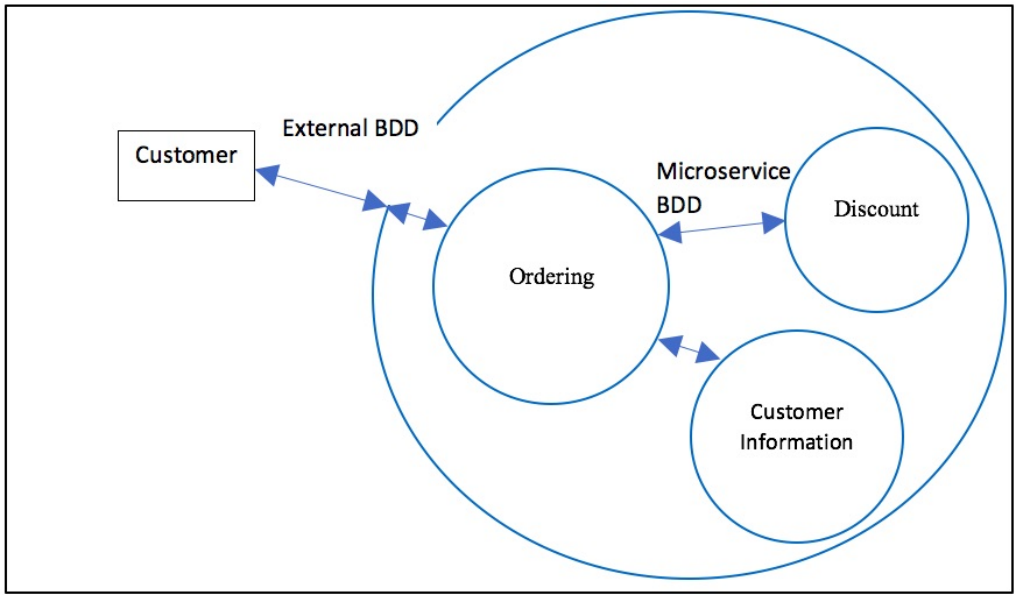
行为通常以一种"给定——当时候——那么"的形式来表达，例如，给定一个特定的状态，当一个动作或事件发生时，该状态就会发生变化和 / 或输出。如业务规则和计算之类的无状态行为，只是展示从输入到输出的转换。

如何响应。领域驱动设计 (DDD) 可以帮助定义行为和接口中涉及的术语。

微服务可以是同步的，即消费者直接调用另一个生产者微服务并等待结果；也可以是异步的，即服务响应消费者放在队列中的消息。本文的示例将基于同步的服务。

示例的上下文

服务提供一组内聚的相关操作。这个示例是订单应用程序中的一个服务，它为下订单的客户计算折扣。



此服务的行为概要可以是这样的：

为客户计算折扣

给定这些输入：

客户类别

订单数量

那么服务输出：

折扣金额

服务可以使用代码实现、本地数据库或调用其他服务来计算结果。我们待会再看。

服务可以使用 JSON 或 XML 作为底层通信协议。但是，独立于实现申明服务行为的方式有助于将操作从语法中分离出来。

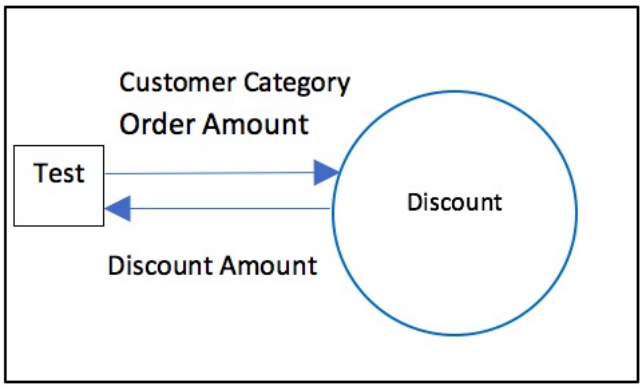
什么是行为?

使用 BDD，可以从样品数据开始了解所需的行为。可能会想到这三方面：

客户类型	订单金额	折扣金额?
良好的	100.00 美元	1.00 美元

优质的	100.00 美元	2.00 美元

前两列是服务的输入，右边的列是服务的输出。



该样品标识了可能需要进一步描述其行为的领域术语，如允许的值。这三方面可能允许的值如下所示。对于这些服务，有个隐含的契约，那就是如果输入的值在允许的范围内，那么它应该返回适当的值。

客户类型
良好的
优质的
超级优质的

货币
美元
欧元
加元

行为，特别是使用微服务时的行为，通常包括表示操作失败的响应。潜在故障的定义可以帮助消费者明确需要处理些什么。消费者可以使用标准库（例如 Netflix 的 Hystrix）来处理其中一些故障。以下是一些可能会出现的故障：

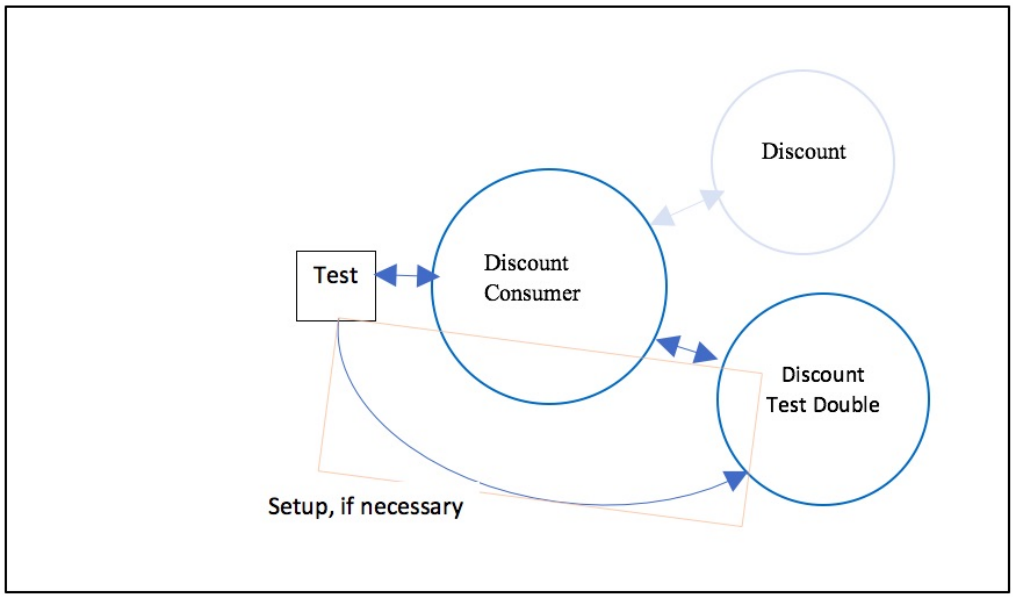
故障
语法不合法
所依赖的服务超时
参数值不合法

在通信协议中，可以以数值或符号值表示故障。在行为驱动开发中使用有意义的名称而不是陈述故障，这有助于更清楚地表达故障。例如，如果客户类别传递的值不在有效值的列表中，那么服务将返回一个对应于"参数值不合法"的失败标志。可能在底层服务中这表示为"400 - Bad Request"的一个响应，负载为"参数值不合法"。

行为驱动开发服务测试可以为组成服务的实体 (例如类) 的单元测试生成上下文。通过设计过程，将通过行为驱动开发测试的责任分配给类和方法。单元测试对这些职责予以详细规格说明。

Test Doubles 测试替身

服务的使用者通常需要一个其调用的服务的测试替身。特别是，速度慢、成本高或随机的服务更需要测试替身。如果折扣服务的行为从未改变，那么得出折扣的服务就可以用于测试消费者。然而，更改是不可避免的，因此通常需要一个测试替身。



测试替身可以是一个总返回相同值的测试，比如：

客户类型	订单金额	折扣金额?
良好的	100.00 美元	1.00 美元
优质的	100.00 美元	2.00 美元

消费者的测试可以依赖于这些值。在这个例子中，不变的行为可能就足够了。但是，对于其他测试来说，最好是测试测试替身设定的响应。

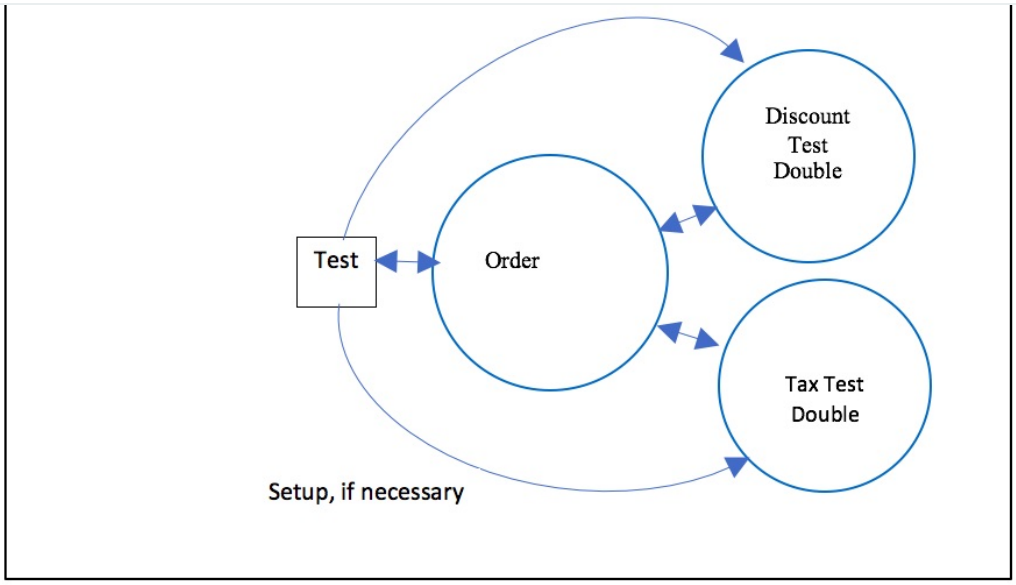
消费者测试将设置折扣测试替身，以便在输入出现时使用给定的值进行响应。例如：

客户类型	订单金额	折扣金额?
良好的	100.00 美元	1.00 美元

或者，折扣测试替身可以以该金额直接予以响应，而不管输入是什么。

我们来看看这个测试替身是如何用于更大的场景的。下图有订单的行为，包括折扣和税收。

税收是由一个类似于折扣的微服务来计算的。



给定一个客户

客户类型	位置
良好的	北卡罗莱纳

然后将折扣设置为：

客户类型	订单金额	折扣金额?
良好的	100.00 美元	1.00 美元

然后将税额设置为：

位置	金额	税额?
北卡罗莱纳	99.00 美元	6.60 美元

当客户下了如下的订单：

订单金额
100.00 美元

那么订单上的相关金额是：

订单金额	折扣额	折后金额	税额	应付款总额
100 美元	1.00 美元	99.00 美元	6.60 美元	105.60 美元

有状态服务

如果折扣服务依赖于从本地数据库获取如何计算折扣的相关信息，那么这些数据库的内容即表示该服务的一个状态。应该记录服务状态在响应数据更新时是如何变化的。例如，假设服务依赖于以下数据：

良好的	100.00 美元	1%
优质的	50.00 美元	2%

一些其他方面的服务将可以更新这些数据。可以安排好这个更新，以便可以更新单个元素，或者一次性更新整个表。下面是一个针对个体更新的行为测试示例：

给定当前的数据：

客户类型	阈值	折扣率
良好的	100.00 美元	1%
优质的	50.00 美元	2%

当一个元素被更新为：

客户类型	阈值	折扣率
优质的	50.00 美元	3.5%

那么更新后的数据为：

客户类型	阈值	折扣率
良好的	100.00 美元	1%
优质的	50.00 美元	3.5%

这一条可以检查用于计算折扣的更新过的数据：

客户类型	阈值	折扣率
优质的	50.00 美元	3.5%

折扣服务可以使用本地持久存储机制来保存前面示例中的数据。但是，它也可以依赖于另一个持久层服务来维护该数据。如果是这种情况，该服务应使用上一节中的测试。每个依赖都会带来另一个问题。如果一个服务的依赖项不可用，那么它应该有什么样的行为？对于折扣服务来说，它应该反馈失败了还是仅仅返回默认值？有时全局失败策略可以告诉你应该怎么做，但通常要基于服务的上下文来做决策。

测试的制定及自动化

一旦对微服务的行为达成共识，就可以将其制定成可自动化的测试。目前，有几个可以使用的微服务测试框架，如 PACT 或 Karate。或者，您也可以使用行为驱动开发框架，如 Cucumber 或 FIT。如 Cucumber 步骤定义等测试实现使用微服务库来执行请求 / 响应。其他环境信息可以作为场景或背景的一部分予以提供。例如，一个 Cucumber 特性文件可以包含以下内容，根据您的测试约定，可能有几种变体：

复制代码

```
1    场景：计算订单金额的折扣
2
```

```
| URL    | http://myrestservice.com  |

当使用以下参数计算折扣时：

| Method | GET                       |
| Path   | discount                  |
| Version| 1                         |

那么，对于每个实例的结果是：

| 客户类型  | 订单金额     | 折扣额?    |
| 良好的    | 100.00 美元| 1.00 美元|
| 优质的    | 100.00 美元| 2.00 美元|
```

可以将前两列中的值转换为某种形式的调用约定，例如转换为查询参数。数据体中的结果应该与第三列相匹配。如果查询名称和值是列的名称和值，那么测试和实现之间将更加平滑。

为了可重用性，可以针对所有计算或确定业务规则结果的服务编写步骤定义，以便使用公共解析库。在上面的示例中，类似于"?"的约定 (如上面提到的折扣金额) 有助于解析器区分什么是输入，什么是输出。

测试还应包括针对故障模式的测试，比如:

那么，针对每个实例的结果是:

复制代码

```
| 客户类型    | 订单金额     | 折扣额?    |结果                  |
| 良好的      | 100.00 美元| 1.00 美元| 通过                 |
| 没那么好的| 100.00 美元| 2.00 美元| 参数值不合法 |
| 优质的      | 100.00 中币| 2.00 美元| 参数值不合法 |
```

总结

使用行为驱动开发来设计微服务的重点是操作的语义，而不是其实现的底层语法。按照面向接口设计的指导方针，服务负责执行其操作，并将问题通知某方（使用者或日志记录器），这有助于划分对故障作出反应的职责。有了定义良好的服务，就更容易让它们一起协作以提供所需的外部行为。

关于作者

Ken Pugh 通过培训和指导帮助公司发展成为精益敏捷的技术型组织。他的特殊兴趣是用验收测试驱动开发 / 行为驱动开发创建高质量的系统，通过协作加速 DevOps，以及使用精益原则快速交付业务价值。他写过几本软件开发书籍，包括 2006 年赢得"卓越奖"（Jolt Award）的《预构（Prefactoring）》和他的最新力作：《Lean-Agile Acceptance Test-Driven development: Better software Through Collaboration》。Ken 帮助过的客户遍布全球，从伦敦到波士顿，从悉尼到北京再到海得拉巴。他是 SAFe®敏捷软件工程课程的共同创造者。您可以点击查看他提供的所有服务，并通过ken@kenpugh.com与他联系。

查看英文原文：[Developing Microservices with Behavior Driven Development and Interface Oriented Design](

0 人喜欢

收藏 评论 微信 微博

写下你的想法，一起交流

发表评论

注册/登录 InfoQ 发表评论

注册/登录

0

喜欢

收藏

评论

微信

微博