专访 Eric Evans：领域驱动设计最新进展

作者 霍泰稳 发布于 2007年5月2日 下午8时34分

社区 .NET Architecture Agile Java Ruby SOA 主题 方法论 客户及需求 领域特定语言

InfoQ.com采访了领域驱动设计的创始人Eric Evans，以了解这一设计技术的最新进展。

为什么领域驱动设计一直都很重要？

基本上，领域驱动设计是我们应该专注于用户所关心领域里的重要问题的指导原则。我们的智慧应该用在理解这一领域上，和那个领域的其他专家一起将它抽象成一个概念。这样，我们就可以应用这个抽象出来的概念构造强大而灵活的软件。

它是一个永远不会过时的指导原则。不论我们何时操作一个复杂的领域，它都有用。大趋势是软件会应用于越来越复杂的问题，越来越趋近于业务的核心。对我来说，这一趋势好像中断了很多年，因为Web突然出现在我们面前。人们的注意力被从富于逻辑和艰深的解决方案上移开，因为有太多的数据需要传递到Web上，只需要简单的动作即可。因为太多的数据要传递，但短时间内在Web上做这一简单的事情又是比较困难的，所以消耗了软件开发的所有能力。

但是现在人们大步跨越了这一Web应用的基本层次，又把注意力集中在业务逻辑上了。

最近，Web开发平台逐渐成熟，足以应用领域驱动设计来做Web开发，有很多积极的信号，比如，SOA，如果应用的好，就可以提供给我们一个非常有用的解析领域的方法。

同时，敏捷过程也有了足够的影响力，大多数项目现在多少都意识到了迭代、和业务伙伴亲密协作、应用持续集成和在强沟通环境下工作的重要性。

所以领域驱动设计在未来会越来越重要，目前已经有了些基础。

技术平台，像Java、.NET、Ruby等一直在变化。领域驱动设计如何适应这一情况？

实际上，新的技术和流程应该由它们是否支持团队专注于他们的领域来验证，而不是置领域于不理。领域驱动设计不依附于哪一个特定的平台，但是有些平台为创造业务逻辑提供了更多的好方法，有些平台更加专注。最近几年的发展显示，为创造业务逻辑提供方法的平台是一个有希望的方向，尤其在可怕的20世纪90年代后期。

Java是这几年默认的选择，从表达角度看，它是一种典型的面向对象语言。对于Distracting Clutter来讲，基础语言是个不错的选择。它有垃圾收集功能，实践证明这一功能很有用（这一点是相比于需要非常关注底层细节的C++而言的）。 Java语法有些地方比较乱，但是POJO（Plain Old Java Objects）仍然可以设计的让人理解。Java 5语法的一些创新之处提高了代码的可读性。

但是在J2EE框架初次出来时，完全将那些基本的表达淹没在大量的框架代码之中。根据早期的契约（比如EJB Home，为所有变量写的Get/Set前缀存取等）生产出可怕而糟糕的对象。这一工具是如此笨重，使得开发团队不得不全力以赴才能让它工作。而且改变对象非常困难，一旦产生的大量代码和XML出现问题，人们往往束手无策。这样的平台很难开发出高效的领域模型。

另外就是使用初级的第一代工具勉强开发藉由HTTP和HTML（不是为此目的而设计的）来完成的Web UI。在那个时候，创建和维护一个像样的UI是如此困难，以至很少有人去关注有着复杂内部功能的设计。有意思的是，恰在这时，对象技术出现了，很好地解决了复杂建模和设计的问题。

这一情况在.NET平台上也是如此，有些事情处理的比较好一些，有些还更糟糕。

那是一个让人沮丧的时代，但是在过去的四年里，总算有了些转变。首先来看Java，关于怎样有选择地使用框架社区里有了新的看法，很多新的优秀的框架（多数是开源的）也应运而生。比如Hibernate和Spring这些框架可以用一种更轻量级的方法处理J2EE曾试图做到的特定工作。像AJAX这样试图解决UI问题的方法也更加便捷。现在的项目在选择使用可以提供价值和混合的新J2EE项时也更加聪明。术语POJO就是在这时产生的。

结果是项目的技术贡献逐渐明显地减少，在把业务逻辑和系统的其他部分隔离方面也有了明显的进步，这样逻辑就可以基于POJO来写了。这不会自动产生领域驱动设计，但是它提供了一个可行的机会。

这就是Java世界。然后就是像Ruby这样的新来者。Ruby有表达力很强的语法，在这一基础层面上它会是领域驱动设计的一个很好的语言（尽管我没有听说在那些应用程序中有哪些实际案例）。Rails带给人们很多兴奋的地方，因为它最终好像能够使得开发Web UI像上世纪90年代初期在Web出现之前开发UI时一样简单。很快，这种能力就被大量使用在构建众多没有太多领域背景的Web应用上，因为就是开发这些简单的应用在过去来说也是很困难的。但是我的希望是，当问题里的UI实现部分减少时，人们可以把这看成专注领域的机遇。如果Ruby使用是从这个方向起步的，那么我认为它会为领域驱动设计提供一个很棒的平台。（一些基础设施可能要被添加进来）

更多前沿的话题发生在域描述语言（DSL）领域，我一直深信DSL会是领域驱动设计发展的下一大步。现在，我们还没有一个工具可以真正给我们想要的东西。但是人们在这一领域比过去做了更多的实验，这使我对未来充满了希望。

现在，我所能说的是大多数尝试使用领域驱动设计的人们是基于Java或者.NET平台，也有少部分在Smalltalk上。在有直接效果的Java世界里，这是一个积极的信号。

从你写完这本书，在领域驱动社区里发生了那些值得注意的事情？

一个让我很兴奋的事情是，人们采用我在书中提到的原则然后用一些我所没有预料到的方法实践。比如，在挪威国家石油公司的StatOil项目中对战略设计的使用。那儿的一个架构师根据他的经验写了一篇报道。

在其他项目里，还有人做了上下文映射，并应用到在抉择是自己构建还是购买时到对已有软件的评估当中。值得一提的是，我们中的有些人已经通过开发一些许多项目中要用到的基础领域对象，探索这方面的话题了。

我们一直在探索在仍然用Java实现对象时，究竟能把这种域描述语言推动多远。

已经走了很远。在人们告诉我他们在做什么事情的时候，我一直都很感激。

请您给要学习领域驱动设计的人一些建议？

读我的书！;-)另外，在项目中舍得投入时间和资金。我们最初的目标之一就是提供一个好的案例，让人们可以通过它进行学习。

要谨记一点的是领域驱动设计主要是由团队来操作，所以你也许要成为一个布道者。现实一点讲，可能需要你找到一个大家正在努力完成的项目。

另外还要注意下面几个领域建模时的重要之处（Pitfall）：

注重实践。模型需要代码。

专注于具体场景。抽象思维需要落地于具体案例。

不要试图对任何事情都进行领域驱动设计。画一张范围表，然后决定哪些应该进行领域驱动设计，哪些不用。不要担心边界之外的事情。

不停地实验，期望能产生错误。模型是一个创造性的流程。

作者简介：Eric Evans是《领域驱动设计——软件核心复杂性应对之道》（Addison-Wesley 2004，已由清华大学出版社翻译出版）一书的作者。早 20 世纪 90 年代，他就参与了很多项目，用具有多种不同的方法多种不同的输出的对象开发大型的业务系统。这本书是那些经验的总结。它提供了一个建模和设计技术的系统，成功的团队应用这一系统可以组装有业务需求的复杂软件系统，并使系统在增大时仍然保持敏捷。Eric现在是“Domain Language”的负责人。Domain Language是一个咨询小组，它指导和训练团队实施领域驱动设计，帮助他们使自己的开发工作对业务而言更有生产力和更有价值。