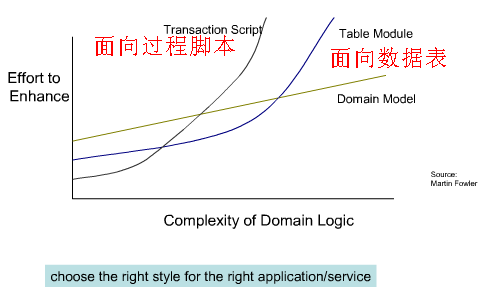
DDD（领域驱动设计）：Domain-driven Design

需求分析的结果无法直接进行设计编程，而能够进行编程运行的代码却扭曲需求，导致客户运行软件后才发现很多功能不是自己想要的，而且软件不能快速跟随需求变化，DDD则打破了这种隔阂，提出了领域模型概念，同意了分析和设计编程，使得软件能够灵活快速跟随需求变化

下图是DDD与传统CRUD或过程脚本或者面向数据表等在开发效率上比较：



服务器后端发展的三个阶段：

1. UI + Data Base 的两层架构，这种面向数据库的架构没有灵活性
2. UI + Service + Data Base 的多层SOA架构，这种服务+表模型的架构易使服务变得臃肿，难于维护拓展，伸缩性能差，

SOA：是一种应用程序架构，在这种架构中，所有功能都定义为独立的服务，这些服务带走定义明确的可调用接口，能够以定义好的顺序调用这些服务来形成业务流程。只是一种概念和思想，需要借助具体的技术和方法来实现它。

SOA设计原则：明确接口的定义，自包含和模块化，粗粒度，松耦合，互操作性/兼容和策略声明

1. DDD+SOA的事件驱动的CQRS读写分离架构，应付复杂业务逻辑，以聚合模型代替数据表模型，以并发的事件驱动替代串联的小区驱动，真正实现以业务实体为核心的灵活拓展。DDD的革命性在于，领域模型准确反映了业务语言，传统jaee或者spring + Hibernate 等事物性编程模型只关心数据，这些数据对象除了简单的set/get 方法外，没有任何业务方法，被比喻成失血模型。

DDD最大好处是：接触到需求第一步就是考虑领域模型（？各个领域模型都不同吗？哪些差别），而不是将其切割成数据和行为，然后数据用数据库实现，行为使用服务实现，最后造成需求的前后分离。DDD 让你首先考虑的是业务语言（？怎么理解，具体在哪些方面？与传统的不同点，没有发现较大差别），而不是数据。

DDD在软件生成流程中定位如下图，DDD落地实现离不开 In-memory缓存，CQRS，DCI，EDA或Event Source 几大相关领域 （？这些都是什么 ？ 每个都是什么意思）

