Atitit 简化设计 表驱动开发 vs 领域驱动设计ddd vs tdd

目录

[1. TDD EDD DDD 1](#_Toc5676)

[1.1. 事件驱动设计 1](#_Toc26782)

[1.2. 表驱动 VS DDD 1](#_Toc4114)

[2. 表驱动开发更加简单 1](#_Toc13898)

[2.1. 事务脚本 + 贫血模型 2](#_Toc17177)

[2.2.   三、进行表设计 2](#_Toc24250)

[2.3. 四、开发页面，设计存储过程？？ 2](#_Toc17495)

[3. 领域驱动特征 3](#_Toc11761)

[3.1.1. 充血模型和领域驱动设计 3](#_Toc5636)

[4. 两种开发模式的区别 3](#_Toc21140)

[5. 数据驱动与域驱动 5](#_Toc21561)

[5.1.1. 数据驱动 5](#_Toc10305)

[5.1.2. 域驱动 5](#_Toc25715)

[6. 领域驱动的缺点 6](#_Toc32110)

[6.1. 复杂繁琐 6](#_Toc9854)

[7. 业务架构设计演进 6](#_Toc31033)

[7.1. 事务脚本 + 贫血模型 6](#_Toc22904)

[7.2. 6](#_Toc10459)

[7.3. 6](#_Toc15870)

[7.4. service + 贫血模型 6](#_Toc14751)

[7.4.1. service + 充血模型 6](#_Toc1741)

[8. View驱动对设计 7](#_Toc19329)

[9. 综合使用 80%对表驱动+20%的领域设计比较好 7](#_Toc29528)

# TDD EDD DDD

## 事件驱动设计

## 表驱动 VS DDD

# 表驱动开发更加简单

易于使用和实现

## 事务脚本 + 贫血模型

* 面向数据建模
* 事务脚本
* 数据驱动设计
* 简单业务开发快

模型中只有数据，没有行为，这时候模型只是数据库表在代码中的映射，在软件设计里面有一个专有名词描述这种现象-贫血模型

## **三、进行表设计**

    这个过程有DBA主导，每个模块的负责人和DBA一道，根据Demo页面上面展现的功能、表单、表单域，来设计这个模块相关的表和表的字段。这个过程也要耗费较长的时间，主要纠结于该用什么英文名称来对应表字段，和太多的表以及表字段的含义、类型

## **四、开发页面，设计存储过程？？**

开发人员按照前期的Demo页面，用ASP.NET来实现一遍，主要的工作就是写好多JS代码，来动态调用后台数据。而DBA就将数据工作全包了，为开发人员编写一个个的SQL函数，输出每个页面用到数据，为了对接方便，输出的数据字段名称用的是中文；程序的功能，也就是所谓的业务逻辑，就由DBA全部写在存储过程中了。DBA专心写存储过程，使用SqlServer 2008上的那些最酷的特性；开发人员专心做ASPX页面，BLL层只是一个传声筒，DAL层已经由PDF.NET的代码生成器自动生成了，不用开发人员操心，只是问问DBA这个功能的SQL该怎么写而已。DBA乐得专心，开发人员乐得简单（虽然是体力活），这样大家都HAPPY

# 领域驱动特征

领域驱动设计（Domain Driven Design，DDD）其实并非新理论，大家可以看看 Eric Evans 编著的《领域驱动设计》原稿首版是2003年，距今已十余年时间。与现在的分布式、微服务相比，绝对是即将步入中年的“老家伙”了。

### **充血模型和领域驱动设计**

而领域驱动设计，也就是DDD，是建立在充血模型之上的，即：

领取驱动设计中的领域模型都是充血模型

# **两种开发模式的区别**

    下面，回过头来看看“领域驱动开发”模式，有什么不一样的地方：

1. 领域驱动注重“领域对象模型”的设计，可以先设计，再测试，最后才开发；
2. 领域驱动能够产生系统的核心价值--“领域对象模型”；
3. 领域驱动使得整个开发过程更容易关注系统的重点功能，使得“有的放矢”；
4. 领域驱动无需重点关注数据问题，使得系统跨数据库移植非常容易；
5. 领域驱动更关注“业务”，而不是“数据本身”，适合业务非常复杂的场景；
6. **领域驱动更关注“业务对象”，从而能够使用各种设计模式，架构模式，使得系统更容易扩展和优化。**

 关于这点，在我们现有系统中深有体会，由于所有业务逻辑的写到了存储过程中，而现在系统运行效率比较低下，在不改变硬件的前提下，想优化的空间都没有。

    当然，表驱动开发模式并发一无是处，它比较适的情况是：

1. **开发团队的整体设计能力欠缺；**
2. 项目的业务不是很复杂，不经常变更业务功能；
3. 以数据为中心，数据在项目中具有核心价值；
4. 有很强的DBA团队

与域驱动相比，数据驱动的优点之一是易于使用和实现。因此，在有必要应用域驱动的情况下（通常会无意识地发生这种情况），使用数据驱动。问题来自于数据驱动几乎与面向对象编程的概念不兼容的事实（当然，如果您确实使用OOP）。在小型应用程序中，这些问题几乎是看不见的。在中型应用程序上，这些问题已经很明显，并开始导致反模式。在大型项目中，问题变得很严重，需要采取适当的措施。

反过来，领域驱动的胜利在大型项目上胜出，但在小型项目上，则使解决方案复杂化，需要更多的开发资源，这对于业务需求而言通常是至关重要的（将项目“尽快”推向市场）预算）。

# 数据驱动与域驱动

### **数据驱动**

#### **优点**

1. 允许您快速开发应用程序或原型
2. 方便设计（代码生成，方案等）
3. 对于中小型项目可能是一个很好的解决方案

#### **缺点**

1. 可能导致反模式并失去OOP
2. 导致大型项目混乱，支持复杂等。

### **域驱动**

#### **优点**

1. 使用OOP的力量
2. 允许您控制范围（域）的复杂性
3. 有许多本文未描述的优点，例如，创建语言域和使用BDD
4. 提供用于开发复杂和大型解决方案的强大工具

#### **缺点**

1. 需要更多的开发资源，从而导致更高的解决方案成本
2. 某些部分变得越来越难以支持（映射器数据等）

# 领域驱动的缺点

## 复杂繁琐

领域驱动开发模式，是先分析需求，得到领域模型，然后和业务一起验证该模型，逐步改善完善模型，第二步是实现业务场景，得到哪些领域对象的属性是需要持久化的，得到哪些组合的属性是需要给前端显示的（ViewModel），第三步才是设计View，使用ViewModel，设计实体类，最后才开始开发用户界面。

# 业务架构设计演进

## **事**务脚本 + 贫血模型

## 

## 

## service + 贫血模型

### **service + 充血模型**

随着需求更加的复杂，业务体量越来越大时，我们需要如此改进：

# View驱动对设计

兼容ddd和tdd ，，解耦业务内部表细节

# 综合使用 80%对表驱动+20%的领域设计比较好

各有优缺点

根据我的经验，我可以说指出了两种方法：关注数据（数据驱动）和关注逻辑（域驱动）。实际上，它们是相互竞争的方法，但在实践中可以与共生相结合，这通常被称为反模式。