好的软件都要有架构模式，软件开发行业发展到今天，估计每个开发人员或多或少都接触过架构模式、设计模式，我们公司也不例外，我们公司的项目主要使用微软的 Asp.net 做基于 B/S 结构 Web 应用开发。
一般的软件我们都采用“事务脚本”架构模式，到具体的面向对象代码结构组织时进行分层，再根据不同需要采用工厂、抽象工厂、原型等设计模式。相信这种模式很多开发人员都有接触过，将 Model、DAL、BLL 各层分开设计，最后在 UI 后台 aspx.cs 或 aspx.vb 中进行调用，这样使得代码结构很清晰，维护和扩展起来非常方便，比如：我现在想把开发好的软件数据库由 MSSQL 换成 Oracle，我直接修改 DAL 层即可，如果采用了工厂设计模式，甚至不需要改任何类代码就可以适配不同数据库。
虽然上述设计已经略显便捷，但整体架构过于繁琐、领域对象非常空洞、业务逻辑变得沉重、且不够面向对象，也因而被国际著名的面向对象专家 Martin Fowler 称之为“贫血模型”，同时也与之对应的有“充血模型”，从字面上看就应该优于“贫血模型”，那么什么是“充血模型”呢？它和“贫血模型”有什么区分呢？
贫血模型：上面的“事务脚本”架构模式的举例就是。在领域对象里只有 get/set 属性，所有的业务逻辑都不包含在内而是放在业务逻辑层，所有的 CRUD 放在数据持久层。优点是系统的层次结构清楚，各层之间单向依赖。缺点是不够面向对象，领域对象只是作为保存状态或者传递状态使用，领域对象中没有行为动作，要增加扩展一些行为动作要一层一层去写，所以被称为“贫血模型”，相信做过几年开发的同胞都能理解。
充血模型： 符合“活动记录”架构模式。领域对象里面包含 get/set、业务逻辑、数据持久化等一切属性和行为动作，业务逻辑层可以没有（如果有需要也可以封装部分控制事务、权限等放在业务逻辑层）。缺点是如何再划分业务逻辑层比较模糊。优点是结构更加轻量化，扩展更加简洁灵活，特别适合 Web 快速开发，且更加面向对象：如同一个人，既有眼睛、鼻子、嘴巴、耳朵、眉毛、四肢……等属性同时也有行、坐、卧、立、跑、跳、蹲……等行为动作，所以被称为“充血模型”。
“充血模型”设计具体要求：
（1）每一个数据库表对应创建一个类，类的每一个对象实例对应于数据库中表的一行记录， 通常表的每个字段在类中都有相应的 Field。
（2）领域对象同事封装了业务逻辑。
（3）领域对象同时负责数据持久化，在其中封装对数据库的访问的 CRUD 行为。这样的设计结构与数据库耦合更加紧密。下面以我们公司开发的一套“龙腾政府投资工程项目管理系统”项目为例做一介绍。
这个项目中有个“项目基本信息管理”模块，需求是：录入项目名称、项目码、项目分类、占地面积、建设地址……等属性,同时有项目添加、修改、删除、查询、导出等动作。MSSQL 数据库中有一张表 XiangMu，结构如下：

Visual Studio 中 App_Code 结构如下：

Entities 目录中放置了所有的实体类（领域对象），每张数据表对应一个类。
Utils 目录下放置了一些公共类，如：数据库公共方法、Excel 导入导出公共方法、短信发送公共方法等。

领域对象 XiangMu 类中的具体代码结构如下：

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Data.Entity;
namespace Cormin
{
    /// <summary>
    /// 项目基本信息
    /// </summary>
    public class XiangMu
    {
        /// <summary>
        /// 唯一标识
        /// </summary>
        public int ID { get; set; }

        /// <summary>
        /// 项目名称
        /// </summary>
        public string MingCheng { get; set; }

        /// <summary>
        /// 项目码
        /// </summary>
        public string BianMa { get; set; }

        /// <summary>
        /// 项目分类
        /// </summary>
        public string FenLei { get; set; }

        /// <summary>
        /// 建设面积
        /// </summary>
        public int MianJi { get; set; }

        /// <summary>
        /// 建设地址
        /// </summary>
        public string DiZhi { get; set; }

        /// <summary>
```

```
        /// 负责人
        /// </summary>
        public string FuZeRen { get; set; }

        /// <summary>
        /// 添加时间
        /// </summary>
        public DateTime ShiJian { get; set; }
        /// <summary>
        /// 添加人
        /// </summary>
        public string TianJiaRen { get; set; }
        /// <summary>
        /// 创建
        /// </summary>
        public void Create()
        {
            if (ShiJian == DateTime.MinValue)
                ShiJian = DateTime.Now;
            string sql = "insert into [XiangMu]([MingCheng], [BianMa], [Fen
Lei], [MianJi], [DiZhi], [FuZeRen], [ShiJian], [TianJiaRen]) values(@MingCh
eng, @BianMa, @FenLei, @MianJi, @DiZhi, @FuZeRen, @ShiJian, @TianJiaRen);";
            Database db = Database.Default;
            DbCommand command = db.CreateCommand(sql);
            db.AddInParameter(command, "MingCheng", DbType.String, 200, Min
gCheng);
            db.AddInParameter(command, "BianMa", DbType.String, 50, BianMa)
;
            db.AddInParameter(command, "FenLei", DbType.String, 50, FenLei)
            db.AddInParameter(command, "MianJi", DbType.Int32, MianJi);
            db.AddInParameter(command, "DiZhi", DbType.String, 200, DiZhi);
            db.AddInParameter(command, "FuZeRen", DbType.String, 50, FuZeRe
n);
            db.AddInParameter(command, "ShiJian", DbType.DateTime, ShiJian
== DateTime.MinValue ? (object)null : ShiJian);
            db.AddInParameter(command, "TianJiaRen", DbType.String, 50, Tia
nJiaRen);
            db.ExecuteNonQuery(command);
        }
        /// <summary>
        /// 更新
        /// </summary>
        public void Update()
        {
```

```
            string sql = "update [XiangMu] set [MingCheng]=@MingCheng, [Bia
nMa]=@BianMa, [FenLei]=@FenLei, [MianJi]=@MianJi, [DiZhi]=@DiZhi, [FuZeRen]
=@FuZeRen, [ShiJian]=@ShiJian, [TianJiaRen]=@TianJiaRen where [ID]=@ID;";
            Database db = Database.Default;
            DbCommand command = db.CreateCommand(sql);
            db.AddInParameter(command, "ID", DbType.Int32, ID);
            db.AddInParameter(command, "MingCheng", DbType.String, 200, Min
gCheng);
            db.AddInParameter(command, "BianMa", DbType.String, 50, BianMa)
;
            db.AddInParameter(command, "FenLei", DbType.String, 50, FenLei)
;
            db.AddInParameter(command, "MianJi", DbType.Int32, MianJi);
            db.AddInParameter(command, "DiZhi", DbType.String, 200, DiZhi);
            db.AddInParameter(command, "FuZeRen", DbType.String, 50, FuZeRe
n);
            db.AddInParameter(command, "ShiJian", DbType.DateTime, ShiJian
== DateTime.MinValue ? (object)null : ShiJian);
            db.AddInParameter(command, "TianJiaRen", DbType.String, 50, Tia
nJiaRen);
            db.ExecuteNonQuery(command);
        }

        /// <summary>
        /// 删除
        /// </summary>
        /// <param name="ID"></param>
        public static void DeleteBy(int ID)
        {
            string sql = "delete [XiangMu] where [ID]=@ID;";
            Database db = Database.Default;
            DbCommand command = db.CreateCommand(sql);
            db.AddInParameter(command, "ID", DbType.Int32, ID);
            db.ExecuteNonQuery(command);
        }

        /// <summary>
        /// 按 Id 获取
        /// </summary>
        /// <param name="ID"></param>
        public static XiangMu GetById(int ID)
        {
            string sql = "select * from [XiangMu] where [ID]=@ID;";
            Database db = Database.Default;
```

```
            DbCommand command = db.CreateCommand(sql);
            db.AddInParameter(command, "ID", DbType.Int32, ID);
            using (IDataReader reader = db.ExecuteReader(command))
            {
                return EntityMapper<XiangMu>.MapToEntity(reader);
            }
        }
        /// <summary>
        /// 获取所有
        /// </summary>
        public static List<XiangMu> GetAll()
        {
            string sql = "select * from [XiangMu];";
            Database db = Database.Default;
            DbCommand command = db.CreateCommand(sql);
            using (IDataReader reader = db.ExecuteReader(command))
            {
                return EntityMapper<XiangMu>.MapToEntities(reader);
            }
        }
    }
}
```

属性、业务、数据持久化都在一起，整个 XiangMu 类就相当于数据表 XiangMu 的完全面向对象的映像，属性、行为动作为一体，形成一个独立的完整对象，同时你也可以把这里的 CRUD 操作再分离出来一层。

下次我如果要扩展“项目基本信息管理”模块，直接修改这一层代码即可，比“贫血模式”的确快捷高效了很多，同时与页面 UI 部分依然是分离的。

现在假设有两个变更，第一个变更是把 MSSQL 数据库换成了 Oracle 数据库，这时我只需要将这个类中的所有 Database 替代为 OracleDatabase 即可(这两种操作类已经在 Utils 目录下的 DataBaseExtension.cs 中进行了封装)；第二个变更是我把 Web 应用改成 Windows 应用了，由于我的领域对象完全和 UI 脱离的，在这个类中我不需要做任何修改，只要在 UI 层重新调用即可。