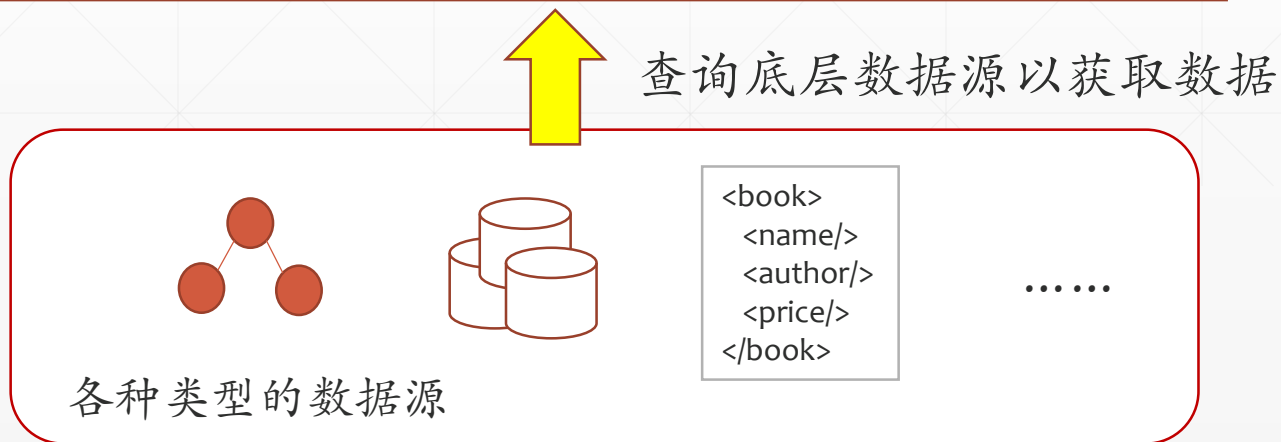
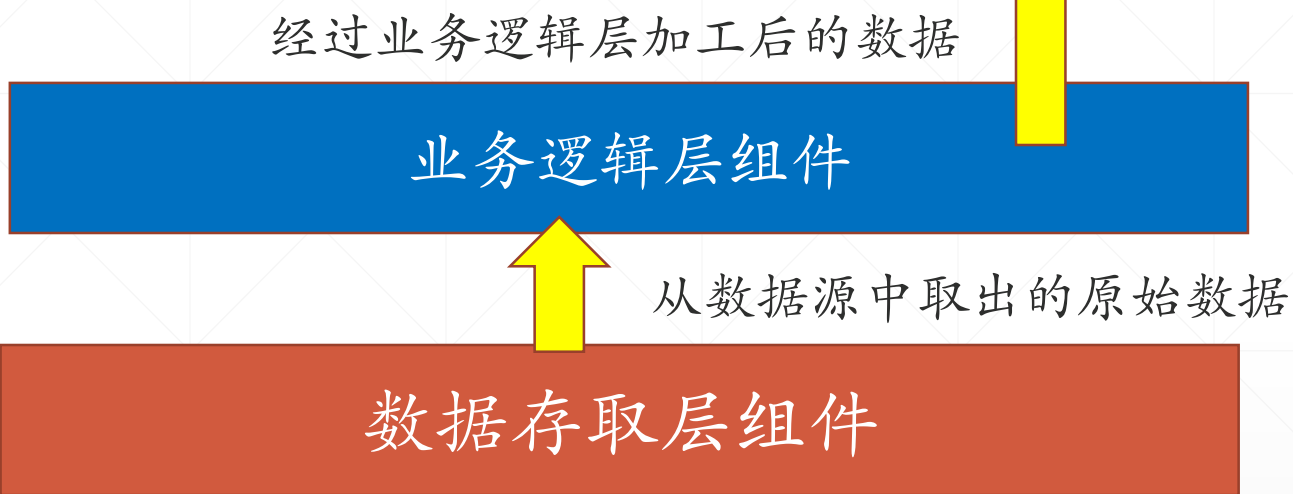


# 数据存取层的设计与实现

---

北京理工大学计算机学院  
金旭亮

我们先来看看典型的软件系统  
分层架构是什么样的.....



从底向上的数据流动



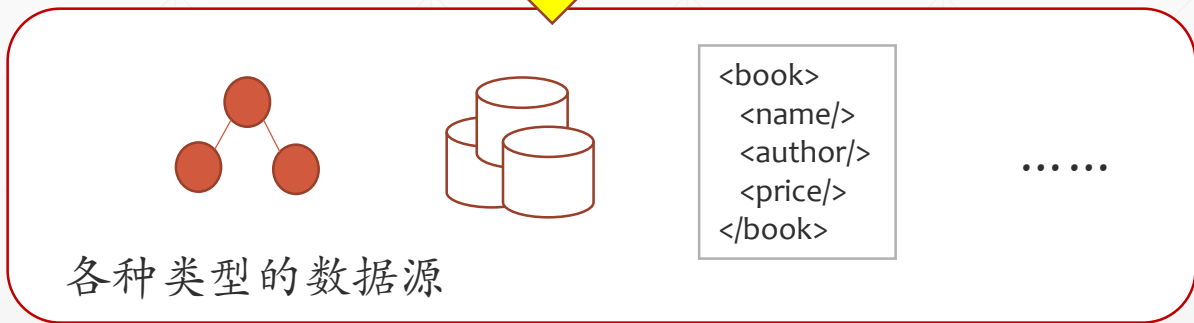
用户对数据所进行的修改



通过业务逻辑校验的数据



Update、Delete、Insert



从上至下的数据流动

# 设计数据存取层

## 设计目标

尽可能地将业务逻辑层与具体数据存取技术相隔离

## 具体实现

在开发中，可以使用Repository设计模式。

为了方便开发分层的数据库应用软件，我们可以将默认情况下“绑”在一起的Entity Framework数据模型中DbContext子类 and 各个数据实体对象分离开来.....



# 抽取数据实体到独立的项目-1

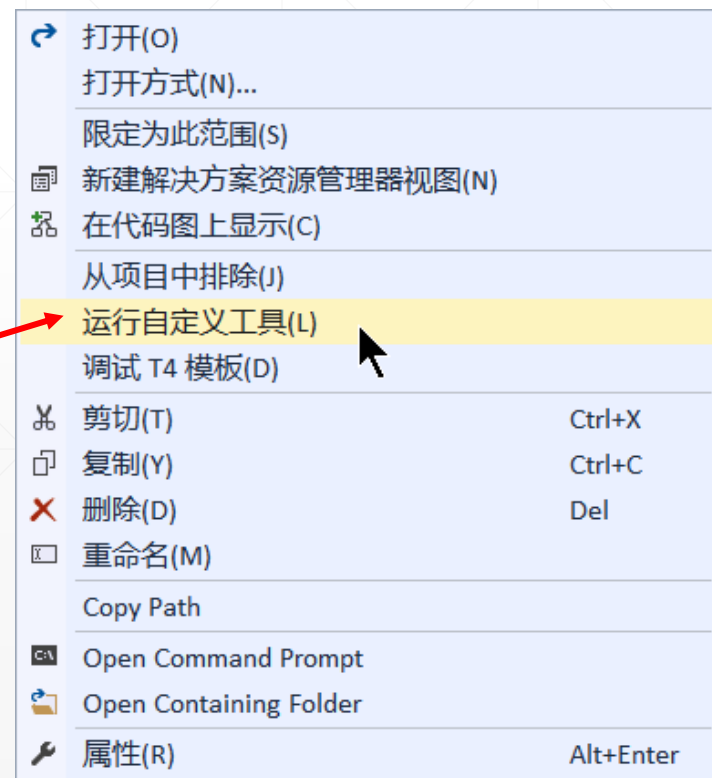
将Entity Framework所使用的“.tt”模板移到独立的项目中.....

```
MyDataModel.tt  [X]
<#@ template language="C#" debug="false" hostspecific="true" #>
<#@ include file="EF6.Utility.CS.ttinclude" #><#@
    output extension=".cs" #><#

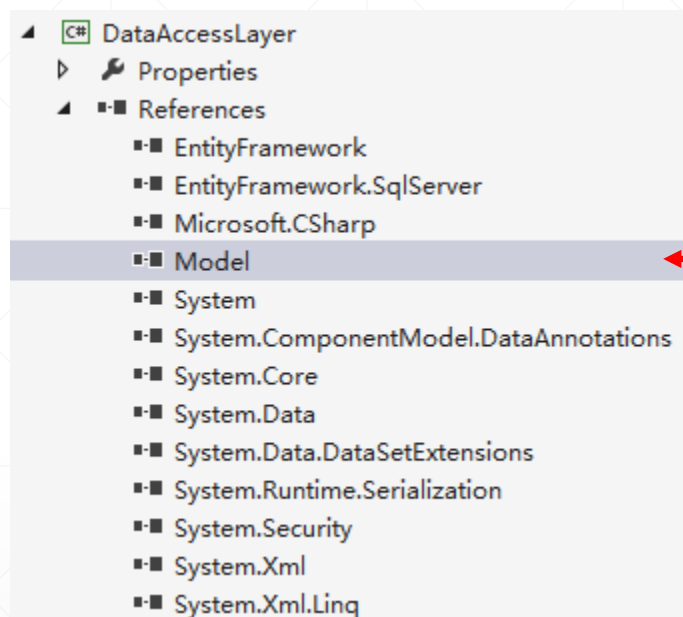
const string inputFile = @"..\DataAccessLayer\MyDataModel.edmx";
var textTransform = DynamicTextTransformation.Create(this);
var code = new CodeGenerationTools(this);
var ef = new MetadataTools(this);
```

修改模板，引用Entity Framework  
数据模型文件（.edmx）

右击.tt文件，选“运行自定义工具”（或者重新编译解决方案），生成数据实体类



# 抽取数据实体到独立的项目-2



设置DataAccessLayer项目引用Model项目

修改MyDataModel.Context.tt模板，  
导入数据实体类所在的命名空间

```
MyDataModel.Context.tt  X
#>
using System;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using Model;
<#
if (container.FunctionImports.Any())
{
#>
```



# 动手手

跟着本讲的视频，把整个过程在自己的电脑上“重做”一遍。

开发数据存取层组件，经常会用到Repository设计模式，它的具体编程实现方法是：

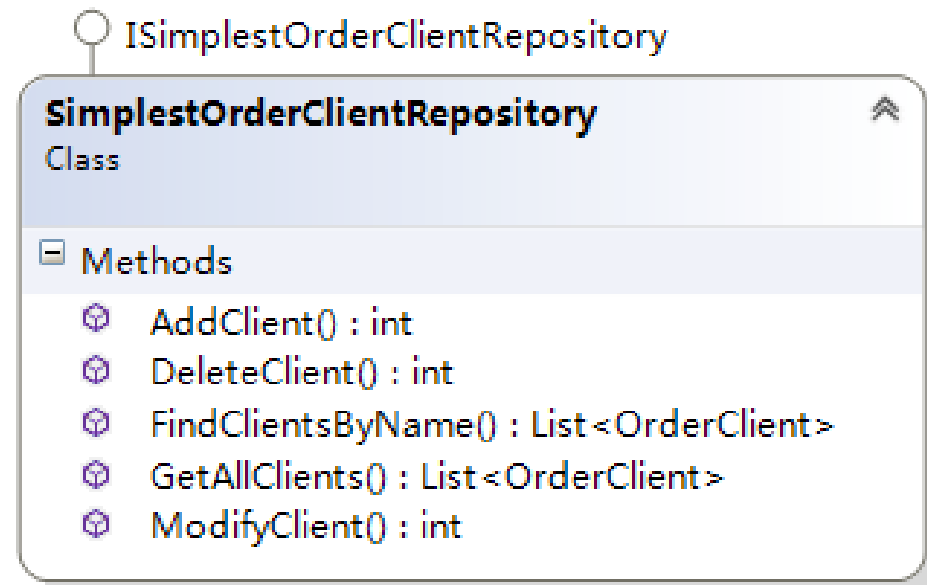
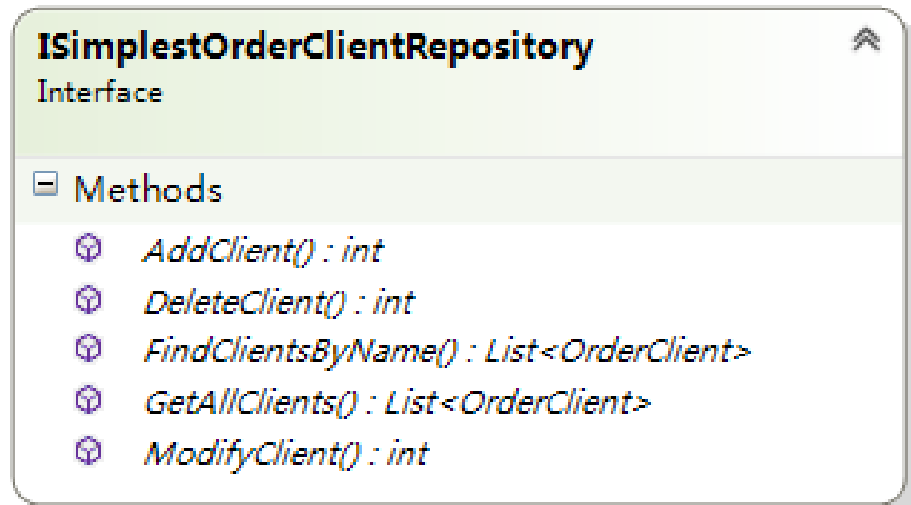
编写一个接口，定义好实现数据CRUD的方法，之后，再编写一个数据仓库（Repository）类实现这个接口，用它来封装数据的CRUD功能。

使用Repository类，可以大大地简化上层组件的数据存取代码。

# Repository 设计模式的典型实现

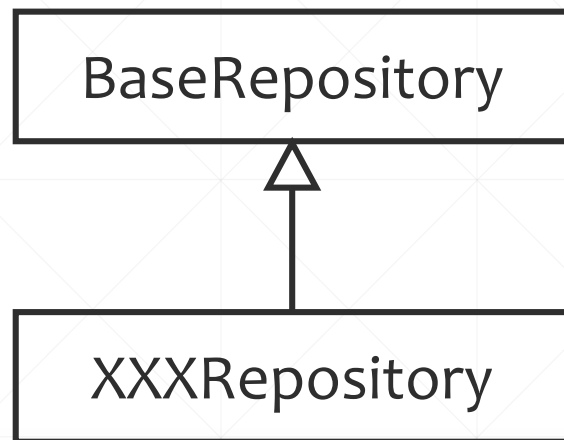
定义一个接口，向外界提供  
CRUD功能.....

然后，让完成CRUD操作的类实现这一接口。



Repository类的主要职责就是实现CRUD!

# 改进版的“Repository”



存放各个Repository共享  
的公用代码

负责各个具体表的CRUD






# 应用了继承特性的Repository-1

使用接口封装CRUD功能

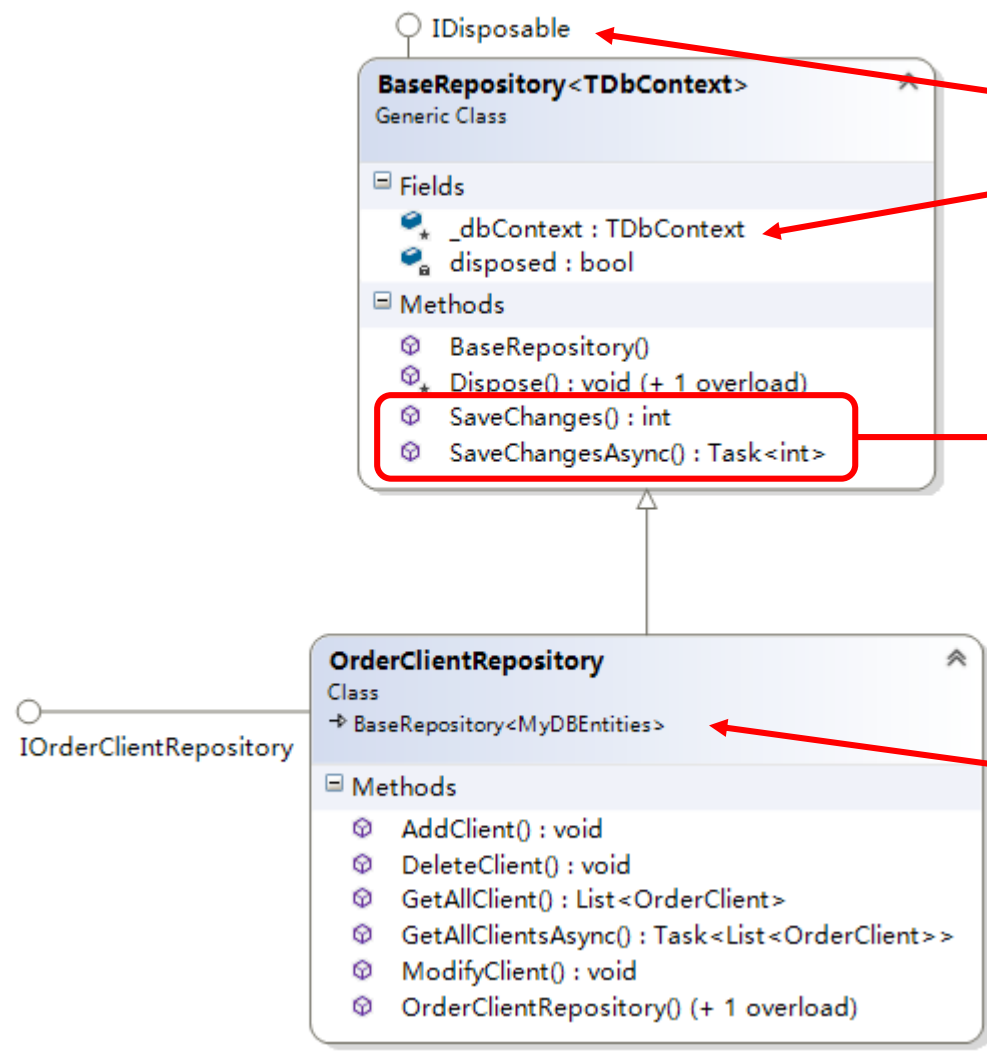
## **IOrderClientRepository**

Interface

### Methods

-  *AddClient() : void*
-  *DeleteClient() : void*
-  *GetAllClient() : List<OrderClient>*
-  *GetAllClientsAsync() : Task<List<OrderClient>>*
-  *ModifyClient() : void*

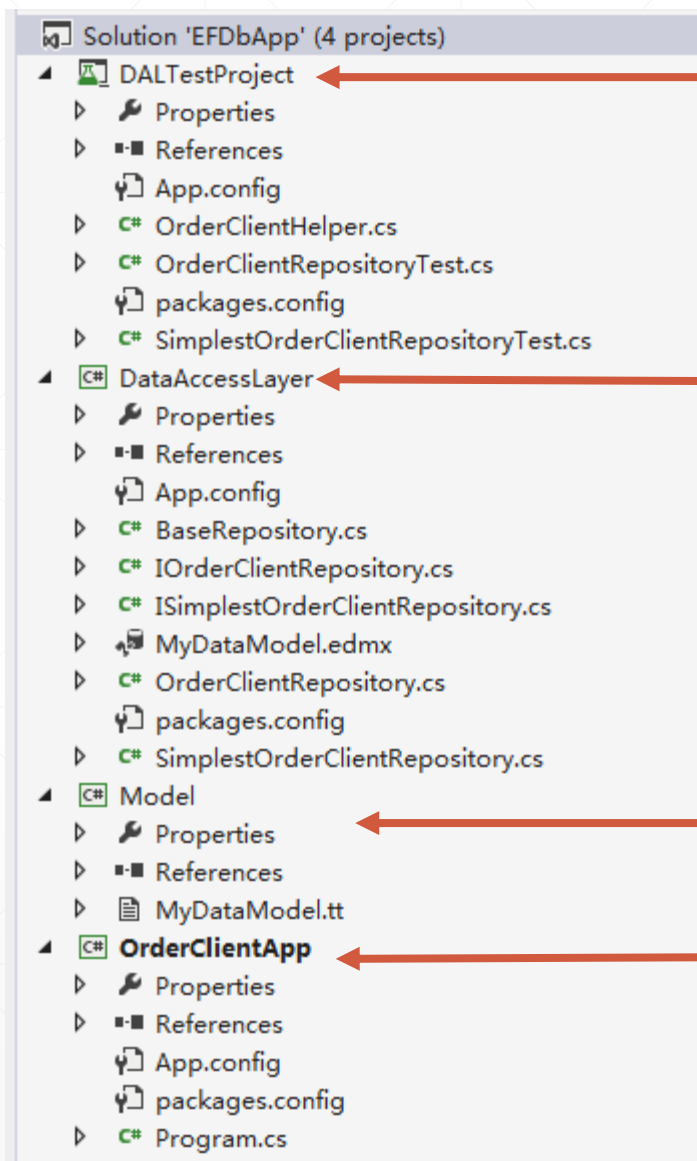
# 应用了继承特性的Repository-2



基类封装DbContext，实现 IDisposable 接口，以实现自动释放 DbContext 的目的

基类可以封装 SaveChanges() 等所有 Repository 都需要调用的功能

具体的 Repository 派生自 BaseRepository，传入具体的 DbContext 子类，同时实现特定接口，最终实现 CRUD

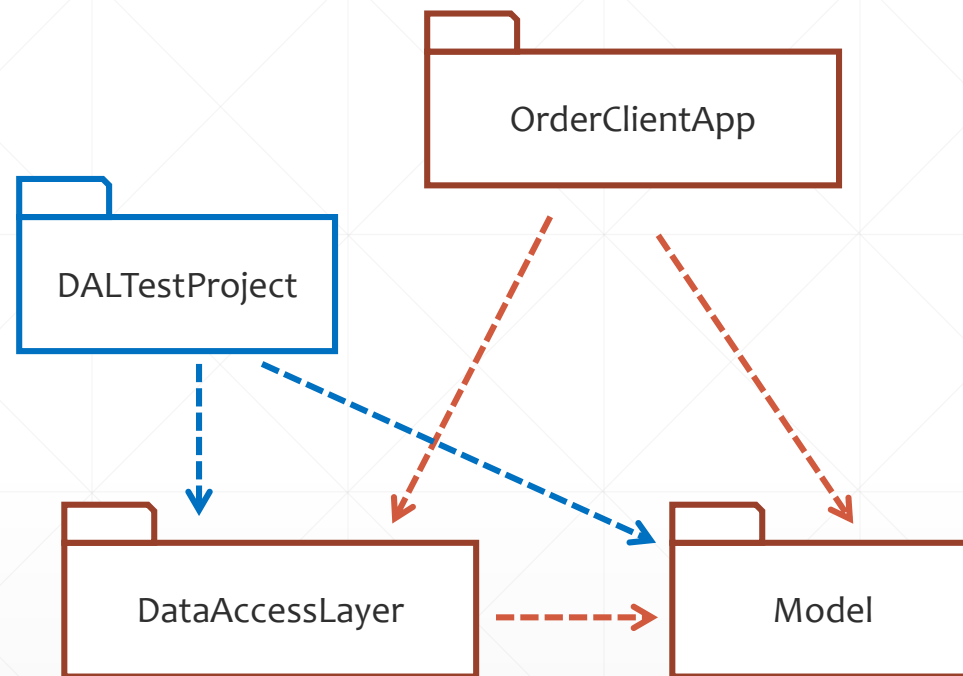


此项目对数据存取层组件进行单元测试

此项目封装Entity Framework，实现CRUD

此项目集中保存所有数据实体类

此项目属UI层，与用户进行交互



注：本示例省去了业务逻辑层。

# 我真的需要一个Repository吗?

只使用**一种**数据源  
数据结构**稳定不变**

你并不需要实现一个Repository，当然，写一个Repository其实也不费事.....

---

只使用**一种**数据源  
数据结构**会变**

你可以不实现一个Repository，但你同样需要一种机制能把底层数据源中的数据转换为业务逻辑层中的实体对象，结果是：你得到了另一个类似于Repository的东西.....

---

使用**多种**数据源  
数据结构**会变**

没有其他选择，必须实现Repository，否则每次底层数据项的增减和变化都将引起业务逻辑层代码的大量修改，迟早让人受不了的.....



# 学习指南

请总结一下本部分所介绍的各种开发套路，理解是应用的前提。

今后再编写以数据为中心的软件，可以参考或模仿本部分所介绍的技术方案，为自己的软件构建一个数据存取层。