

عنوان: معرفی Microsoft.Data.dll یا WebMatrix.Data.dll

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۰۷/۱۵ ۱۴:۲۵:۰۰

آدرس: [www.dotnettips.info](http://www.dotnettips.info)

برچسب‌ها: ADO.NET

مایکروسافت اخیرا علاوه بر تکمیل ORM های خود مانند LINQ to SQL و همچنین Entity framework ، لایه دیگری را نیز بر روی ADO.NET جهت کسانی که به هر دلیلی دوست ندارند با ORMs کار کنند و از نوشتن کوئری‌های مستقیم SQL لذت می‌برند، ارائه داده است که Microsoft.Data library نام دارد و از قابلیت‌های جدید زبان سی شارپ مانند واژه کلیدی dynamic استفاده می‌کند.

در ادامه قصد داریم جهت بررسی توانایی‌های این کتابخانه از بانک اطلاعاتی معروف Northwind استفاده کنیم. این بانک اطلاعاتی را [از اینجا](#) می‌توانید دریافت کنید.

مراحل استفاده از Microsoft.Data library:

الف) این اسمبلی جدید به همراه پروژه WebMatrix ارائه شده است. بنابراین ابتدا باید آن را دریافت کنید: [+](#)  
لازم به ذکر است که این کتابخانه اخیرا به WebMatrix.Data.dll تغییر نام یافته است. (اگر وب را جستجو کنید فقط به Microsoft.Data.dll اشاره شده است)

ب) پس از نصب، ارجاعی را از اسمبلی WebMatrix.Data.dll به پروژه خود اضافه نمایید. این اسمبلی در صفحه‌ی Add References ظاهر نمی‌شود و باید کامپیوتر خود را برای یافتن آن جستجو کنید که عموما در آدرس زیر قرار دارد:

```
C:\Program Files\Microsoft ASP.NET\ASP.NET Web Pages\v1.0\Assemblies\WebMatrix.Data.dll
```

ج) اتصال به بانک اطلاعاتی

پیش فرض اصلی این کتابخانه بانک اطلاعاتی SQL Server CE است. بنابراین اگر قصد استفاده از پروایدرهای دیگری را دارید باید به صورت صریح آن را ذکر نمایید:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<appSettings>
  <add key="systemData:defaultProvider" value="System.Data.SqlClient" />
</appSettings>
<connectionStrings>
  <add name="Northwind"
    connectionString="Data Source=(local);Integrated Security = true;Initial Catalog=Northwind"
    providerName="System.Data.SqlClient" />
</connectionStrings>
</configuration>
```

این تعاریف در فایل web.config و یا app.config برنامه وب یا ویندوزی شما قرار خواهند گرفت.

د) نحوه‌ی تعریف کوئری‌ها و دریافت اطلاعات

```
using System;
using WebMatrix.Data;

namespace TestMicrosoftDataLibrary
{
  class Program
  {
    static void Main(string[] args)
    {
```

```

        getProducts();
        Console.Read();
        Console.WriteLine("Press a key ...");
    }

    private static void getProducts()
    {
        using (var db = Database.Open("Northwind"))
        {
            foreach (var product in db.Query("select * from products where UnitsInStock < @0", 20))
            {
                Console.WriteLine(product.ProductName + " " + product.UnitsInStock);
            }
        }
    }
}

```

پس از افزودن ارجاعی به اسمبلی WebMatrix.Data و مشخص سازی رشته‌ی اتصالی به بانک اطلاعاتی، استفاده از آن جهت دریافت اطلاعات کوئری‌ها همانند چند سطر ساده‌ی فوق خواهد بود که از امکانات dynamic زبان سی شارپ 4 استفاده می‌کند؛ به این معنا که product.ProductName و product.UnitsInStock در زمان اجرا مورد ارزیابی قرار خواهند گرفت. همچنین نکته‌ی مهم دیگر آن نحوه‌ی تعریف پارامتر در آن است (همان @0 ذکر شده) که نسبت به ADO.NET کلاسیک به شدت ساده شده‌است (و نوشتن کوئری‌های امن و SQL Injection safe را تسهیل می‌کند). در اینجا Database.Open کار گشودن name ذکر شده در فایل کانفیگ برنامه را انجام خواهد داد. اگر بخواهید این تعریف را در کدهای خود قرار دهید (که اصلاً توصیه نمی‌شود)، می‌توان از متد Database.OpenConnectionString استفاده نمود.

یا مثالی دیگر: استفاده از LINQ حین تعریف کوئری‌ها:

```

private static void getCustomerFax()
{
    using (var db = Database.Open("Northwind"))
    {
        var product = db.Query("SELECT * FROM [Customers] WHERE City=@0",
"Paris").FirstOrDefault();
        if (product != null)
            Console.WriteLine(product.Fax);
        else
            Console.WriteLine("not found.");
    }
}

```

ه) اجرای کوئری‌ها بر روی بانک اطلاعاتی

```

private static void ExecQuery()
{
    using (var db = Database.Open("Northwind"))
    {
        int affectedRecords = db.Execute("UPDATE [Customers] SET fax = fax + '*' WHERE City = @0",
"Paris");
        Console.WriteLine("Affected records: {0}", affectedRecords);
    }
}

```

با استفاده از متد Execute آن می‌توان کوئری‌های دلخواه خود را بر روی بانک اطلاعاتی اجرا کرد. خروجی آن تعداد رکورد تغییر کرده است.

و) نحوه‌ی اجرای یک رویه ذخیره شده و نمایش خروجی آن

```

private static void ExecSPShowResult()
{

```

```

using (var db = Database.Open("Northwind"))
{
    var customer = db.Query("exec CustOrderHist @0", "ALFKI").FirstOrDefault();
    if (customer != null)
    {
        Console.WriteLine(customer.ProductName);
    }
}

```

در این مثال رویه ذخیره شده CustOrderHist در بانک اطلاعاتی Northwind اجرا گردیده و سپس اولین خروجی آن نمایش داده شده است.

(ز) اجرای یک تابع و نمایش خروجی آن

```

private static void useFuncs()
{
    using (var db = Database.Open("Northwind"))
    {
        var query = db.Query("SELECT dbo.FN_GET_CATEGORY_TREE(@0) as Rec1", 3);
        foreach (var tree in query)
        {
            Console.WriteLine(tree.Rec1);
        }
    }
}

```

در اینجا تابع FN\_GET\_CATEGORY\_TREE موجود در بانک اطلاعاتی Northwind انتخاب گردیده و سپس خروجی آن به کمک یک نام مستعار (برای مثال Rec1) نمایش داده شده است.

سؤال : آیا WebMatrix.Data.dll بهتر است یا استفاده از ORMs ؟

در اینجا چون از قابلیت‌های داینامیک زبان سی شارپ 4 استفاده می‌شود، کامپایلر درکی از اشیاء خروجی و خواص آن‌ها برای مثال tree.Rec1 (در مثال آخر) ندارد و تنها در زمان اجرا است که مشخص می‌شود آیا یک چنین ستونی در خروجی کوئری وجود داشته است یا خیر. اما حین استفاده از ORMs این طور نیست و Schema یک بانک اطلاعاتی پیشتر از طریق نگاشت‌های جداول به اشیاء دات نت، به کامپایلر معرفی می‌شوند و همین امر سبب می‌شود تا اگر ساختار بانک اطلاعاتی تغییر کرد، پیش از اجرای برنامه و در حین کامپایل بتوان مشکلات را دقیقاً مشاهده نمود و سپس برطرف کرد.

ولی در کل استفاده از این کتابخانه نسبت به ADO.NET کلاسیک بسیار ساده‌تر بوده، می‌توان اشیاء و خواص آن‌ها را مطابق نام جداول و فیلدهای بانک اطلاعاتی تعریف کرد و همچنین تعریف پارامترها و برنامه نویسی امن نیز در آن بسیار ساده‌تر شده است.

برای مطالعه بیشتر:

[Introduction to Microsoft.Data.dll](#)

## نظرات خوانندگان

نویسنده: سامان نام نیک  
تاریخ: ۱۳۸۹/۰۷/۱۷ ۱۳:۳۳:۴۵

سلام  
خیلی مفید بود  
من که مدت هاس به دلیل استفاده از linq دیگه از ado.net استفاده نمیکنم  
ولی اگه روزی بخوام استفاده کنم قطعا از کتابخانه فوق استفاده می کنم راستی فک کنم به نسبت ado.net performance بالاتری داشته باشه  
دیگه از دست reader, adapter, datatable, ..... خلاص میشیم  
نظر شما چیه؟

نویسنده: وحید نصیری  
تاریخ: ۱۳۸۹/۰۷/۱۷ ۱۴:۱۵:۴۷

بله. طراحی ADO.NET مربوط به دات نت یک است و از هیچکدام از پیشرفت‌های اخیر بدیهی است که استفاده نمی‌کند. به همین جهت است که در این کتابخانه ترکیبی از LINQ و قابلیت‌های dynamic زبان سی شارپ 4 را مشاهده می‌کنید.

نویسنده: Meysam  
تاریخ: ۱۳۸۹/۰۷/۱۷ ۲۲:۲۶:۴۱

EF یا LINQ خودشون رو ADO.Net کار میکنن

نویسنده: وحید نصیری  
تاریخ: ۱۳۸۹/۰۷/۱۸ ۰۰:۴۳:۵۵

بله. همانطور که در مقدمه بحث عنوان شد، WebMatrix.Data.dll هم لایه‌ای است روی ADO.NET. مابقی هم به همین صورت؛ به این جهت که از پیشرفت‌های زبان‌های دات نت استفاده کنند. زمانیکه ADO.NET ارائه شد نه Generics وجود داشت، نه LINQ نه قابلیت‌های پویای زبان و نه ...

نویسنده: وحید نصیری  
تاریخ: ۱۳۸۹/۰۷/۲۲ ۰۱:۲۹:۰۵

کم کم داره از این دست پروژه‌ها زیاد میشه. دقیقا بر همین اساس و ایده استفاده از قابلیت‌های پویای زبان:

[Kynetic ORM: An ORM without configuration using C# 4.0 Dynamics, Generics and Reflection](#)

عنوان: استفاده از DbProviderFactory

نویسنده: فرهاد فرهمندخواه

تاریخ: ۱۹:۳۲ ۱۳۹۱/۰۵/۱۹

آدرس: [www.dotnettips.info](http://www.dotnettips.info)

برچسب‌ها: ADO.NET

استفاده از DbProviderFactory امکان اتصال به دیتابیس‌های مختلف با یک کد واحد را برای شما فراهم می‌سازد، بطوریکه اگر بخواهید برنامه‌ای بنویسید که قابلیت اتصال به Oracle و SqlServer و دیگر دیتابیس‌ها را داشته باشد، استفاده از DbProviderFactory، کار شما را تسهیل می‌نماید.

DbProviderFactory در [.Net Framework 2.0](#) ارائه شده است. برای درک و چگونگی استفاده از DbProviderFactory مثالی را بررسی می‌نماییم. ابتدا کد زیر را درون یک فرم کپی نمایید:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.Common;

namespace DBFactory
{
    public partial class Form1 : Form
    {
        private string _MySQLProvider = "MySql.Data.MySqlClient";
        private string _SQLProvider = "System.Data.SqlClient";
        private string _OracleProvider = "System.Data.OracleClient";
        private DbProviderFactory _DbProviderFactory;
        private DbConnection _DbConnection = null;
        private DbCommand _DbCommand = null;
        private DbDataAdapter _DbDataAdapter = null;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            try
            {
                string _SQLconnectionstring = "Integrated Security=SSPI;Persist Security
Info=False;Initial Catalog=Test;Data Source=FARHAD-PC";
                string _Oracleconnectionstring = "Data Source=ServiceName;User
Id=Username;Password=Password";

                _DbProviderFactory = DbProviderFactories.GetFactory(_SQLProvider);
                _DbConnection = _DbProviderFactory.CreateConnection();
                _DbConnection.ConnectionString = _SQLconnectionstring;

                _DbConnection.Open();

                if (_DbConnection.State == ConnectionState.Closed)
                {
                    MessageBox.Show("اتصال با دیتابیس برقرار نشده است");
                }
                else
                {
                    MessageBox.Show("اتصال با دیتابیس با موفقیت برقرار شده است");
                }
            }
            catch (System.Exception excep)
            {
                MessageBox.Show(excep.Message.ToString());
            }
        }
    }
}
```

```

    }
}

```

برای استفاد از DbProviderFactory می‌بایست از فضای نامی System.Data.Common استفاده نمایید. بعد از اعلان کلاس فرم تعدادی آبجکت تعریف شده است، که سه آبجکت ابتدایی آن، بیانگر Provider دیتابیس‌های MySQL، SQLSERVER و Oracle می‌باشد:

```

private string _MySQLProvider = "MySql.Data.MySqlClient";
private string _SQLProvider="System.Data.SqlClient";
private string _OracleProvider ="System.Data.OracleClient";

```

Providerهای بیان شده، جهت استفاده DBFactory برای تشخیص نوع Database می‌باشد، تا بتواند آبجکت‌های مربوط به دیتابیس را ایجاد و در اختیار برنامه نویسی قرار دهد. در این مثال ارتباط با دیتابیس SQLSERVER را امتحان می‌کنیم. بنابراین خواهیم داشت:

```

_DbProviderFactory = DbProviderFactories.GetFactory("System.Data.SqlClient");

```

در کد بالا، Provider، دیتابیس SQLSERVER به DbProviderFactory به عنوان ورودی داده شده است، بنابراین آبجکت‌های مربوط به دیتابیس SQL Server ایجاد و در اختیار شما قرار می‌گیرد.

اگر به نام فضای نامی System.Data.Common توجه نمایید، از کلمه Common استفاده شده است و منظور این است که تمامی کلاس‌هایی را که این فضای نامی ارائه می‌دهد، در هر دیتابیس قابل استفاده می‌باشد. برای تشخیص، کلاس‌های مربوط به این فضای نامی نیز در ابتدای نام آنها از دو حرف DB استفاده شده است. تمامی کلاس‌های زیر در فضای نامی System.Data.Common قابل ارائه و استفاده می‌باشد:

```

DbCommand
DbCommandBuilder
DbConnection
DbDataAdapter
DbDataReader
DbException
DbParameter
DbTransaction

```

جهت اطلاع: ممکن است سئوالی در ذهن شما ایجاد شود که دات نت چگونه براساس نام Provider نوع دیتابیس را تشخیص می‌دهد؟

جواب: زمانی که دیتابیس‌های مختلف روی سیستم شما نصب می‌شود، Providerهای مربوط به هر دیتابیس درون فایل Machine.config که مربوط به دات نت میباشد، درج می‌شود. و دات نت براساس اطلاعات مربوط به همین فایل آبجکت‌های دیتابیس را ایجاد می‌نماید.

امیدوارم مطلب فوق مفید واقع شود.

## نظرات خوانندگان

نویسنده:

وحید نصیری

تاریخ:

۱۹:۵۸ ۱۳۹۱/۰۵/۱۹

من توصیه می‌کنم که ADO.NET رو به شکل خام آن فراموش کنید. این نوع روش‌ها هرچند پایه و اساس تمام ORM‌های نوشته شده هستند، اما فقط ابتدای کار را به شما نشان می‌دهند. واقعیت این است که سوئیچ کردن بین بانک‌های اطلاعاتی مختلف نیاز به تولید SQL قابل فهم برای آن موتور خاص را نیز دارد. اینجا است که ORM‌ها در وقت شما صرفه جویی می‌کنند. شما کوئری LINQ می‌نویسید اما در پشت صحنه بر اساس پروایدر مورد استفاده، این کوئری LINQ به معادل SQL قابل فهم برای بانک اطلاعاتی مورد نظر ترجمه می‌شود. خیلی از توابع هستند که در بانک‌های اطلاعاتی مختلف تفاوت می‌کنند و این SQL ایی که مورد بحث است ... در عمل آنچنان استاندارد نیست. توابع تاریخ در SQLite با SQL Server فرق می‌کند. نوع‌های داده‌ای این‌ها عموماً تطابق ندارد و مسایل دیگر. ORM‌ها می‌توانند این مسایل را به خوبی مدیریت کنند بدون اینکه شما آنچنان درگیر این جزئیات شوید.

نویسنده:

فرهاد فرهمندخواه

تاریخ:

۲۱:۱۲ ۱۳۹۱/۰۵/۱۹

سلام

با تشکر از توصیه شما

تا حدودی با نظر شما موافق هستم، اگر بخواهیم با امکانات جدید میکروسافت نرم افزاری ایجاد نماییم. قطعاً، روش بیان شده ضرورتی ندارد، اما برای پروژه‌هایی که با امکانات قدیمی‌تر نوشته شده اند و بدایلی امکان بازنویسی آنها وجود، ندارد، و از طرفی میبایست با دیتابیس‌های مختلف نیز کار کند، روش فوق می‌تواند مفید باشد، در مورد اینکه دیتابیس‌ها با هم متفاوت می‌باشند، نیز با شما موافقم، حتی معتقدم که Provider ی را که میکروسافت برای Oracle ارائه داده است، در مقایسه با Provider شرکت Oracle بسیار ضعیف‌تر عمل می‌نماید، به عنوان مثال در جاهایی که مدت زمان درج اطلاعات زیادی بصورت Batch بسیار اهمیت دارد، Provider، شرکت Oracle برای دیتابیس Oracle سازگارتر و کارتر می‌باشد.

نویسنده:

وحید نصیری

تاریخ:

۲۱:۳۱ ۱۳۹۱/۰۵/۱۹

- اگر به هر دلیلی مجبور هستید که از دات نت 2 استفاده کنید، NHibernate می‌تونه پیشنهاد خوبی باشه و نسخه مخصوص دات نت 2 هم دارد (به [آرشیو قدیمی](#) آن سایت مراجعه کنید). (پایه زبان فعلی جاوا از خیلی از جهات شبیه به دات نت 2 است)

- میکروسافت کلاً توسعه پروایدر ADO.NET مخصوص اوراکل را [رسماً متوقف کرده](#) و خود اوراکل الان داره این کار رو [ادامه می‌ده](#).

. خلاصه از پروایدر میکروسافت برای کار با اوراکل استفاده نکنید.

همانطور که می‌دانید GridView جزء جداناپذیر از اکثر پروژه‌های برنامه نویسان ASP.NET Web forms می‌باشد. اکثراً روشی که در میان برنامه نویسان بیشتر استفاده می‌شود، قرار دادن یک دکمه/لینک در هر ردیف از GridView برای حذف رکورد مورد نظر می‌باشد. در این مقاله قصد دارم روشی را ارائه کنم تا کاربر قادر باشد هر تعداد رکورد را که مدنظر دارد، انتخاب کرده و با فشردن دکمه "حذف" رکوردهای انتخاب شده را حذف کند.

برای درک بهتر، ابتدا جدولی به اسم "Employee" را در SQL Server با مشخصات زیر ساخته :

```
CREATE TABLE [dbo].[Employee] (
    [EmpId] INT NOT NULL,
    [FirstName] VARCHAR (20) NOT NULL,
    [LastName] VARCHAR (20) NOT NULL,
    [City] VARCHAR (20) NOT NULL,
    PRIMARY KEY CLUSTERED ([EmpId] ASC)
);
```

1- یک GridView به صفحه افزوده و خاصیت AutoGenerateColumns آن را برابر False قرار دهید . 2- فیلدهایی را که قصد نمایش آنها در GridView را دارید به صورت زیر به GridView بیفزایید :

```
<asp:BoundField DataField="FirstName" HeaderText="First Name" />
```

3- برای قرار دادن کنترل‌های Asp.net که در اینجا منظور CheckBox می‌باشد می‌بایست از TemplateField و قرار دادن تگ ItemTemplate درون آن، به صورت زیر استفاده نمایید :

```
<asp:TemplateField>
    <ItemTemplate>
        <asp:CheckBox ID="chkDel" runat="server" />
    </ItemTemplate>
</asp:TemplateField>
```

و بعد از تگ GridView دکمه‌ای را برای حذف موارد انتخابی در فرم قرار دهید :

```
<asp:Button ID="btnDeleteRecord" runat="server" OnClick="btnDeleteRecord_Click" Text="Delete" />
```

برای نمایش یک پیغام به کاربر به منظور Confirm کردن دستور حذف در سمت کلاینت، قطعه کد Javascript زیر را قرار می‌دهیم:

```
function DeleteConfirm()
{
    var Ans = confirm("Do you want to Delete Selected Employee Record?");
    if (Ans)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

و در رویداد Page\_Load کدهای زیر را جهت نمایش مقادیر در GridView و افزودن تابع فوق به دکمه، قبل از حذف رکوردها می‌افزاییم :

```
protected void Page_Load(object sender, EventArgs e)
```



```

{
    if(!IsPostBack)
    {
        //Displaying the Data
        showData();
        //Adding an Attribute to Server Control(i.e. btnDeleteRecord)
        btnDeleteRecord.Attributes.Add("onclick", "javascript:return DeleteConfirm()");
    }
}

```

```

//Method for Displaying Data
protected void showData()
{
    DataTable dt = new DataTable();
    SqlConnection con = new SqlConnection(cs);
    SqlDataAdapter adapt = new SqlDataAdapter("select * from Employee",con);
    con.Open();
    adapt.Fill(dt);
    con.Close();
    GridView1.DataSource = dt;
    GridView1.DataBind();
}

```

ابتدا تابع DeleteRecode را به صورت زیر پیاده سازی میکنیم :

که یک پارمتر را از ورودی دریافت میکند که ID رکورد انتخاب شده می باشد و با استفاده از ID، رکورد مورد نظر را حذف میکنیم :

```

protected void DeleteRecord(int empid)
{
    SqlConnection con = new SqlConnection(cs);
    SqlCommand com = new SqlCommand("delete from Employee where EmpId=@ID",con);
    com.Parameters.AddWithValue("@ID",empid);
    con.Open();
    com.ExecuteNonQuery();
    con.Close();
}

```

و اما بخش مهم مربوط به رویداد دکمه می باشد. در هنگام کلیک بر روی دکمه باید تمامی رکوردهای GridView را چک و تمامی رکوردهایی را که CheckBox آنها تیک خورده است گرفته و ID رکورد مورد نظر را به تابع DeleteRecode فرستاد و در پایان برای اعمال تغییرات، متد ShowDate را فراخوانی و GridView را مجدداً Bind می کنیم.

```

protected void btnDeleteRecord_Click(object sender, EventArgs e)
{
    foreach (GridViewRow grow in GridView1.Rows)
    {
        //Searching CheckBox("chkDel") in an individual row of Grid
        CheckBox chkdel = (CheckBox)grow.FindControl("chkDel");
        //If CheckBox is checked then delete the record with particular empid
        if(chkdel.Checked)
        {
            int empid = Convert.ToInt32(grow.Cells[1].Text);
            DeleteRecord(empid);
        }
    }
    //Displaying the Data in GridView
    showData();
}

```

## نظرات خوانندگان

نویسنده: محمد حسین فخرآوری  
تاریخ: ۱۸:۱۹ ۱۳۹۳/۰۷/۰۳

شرط بر اساس

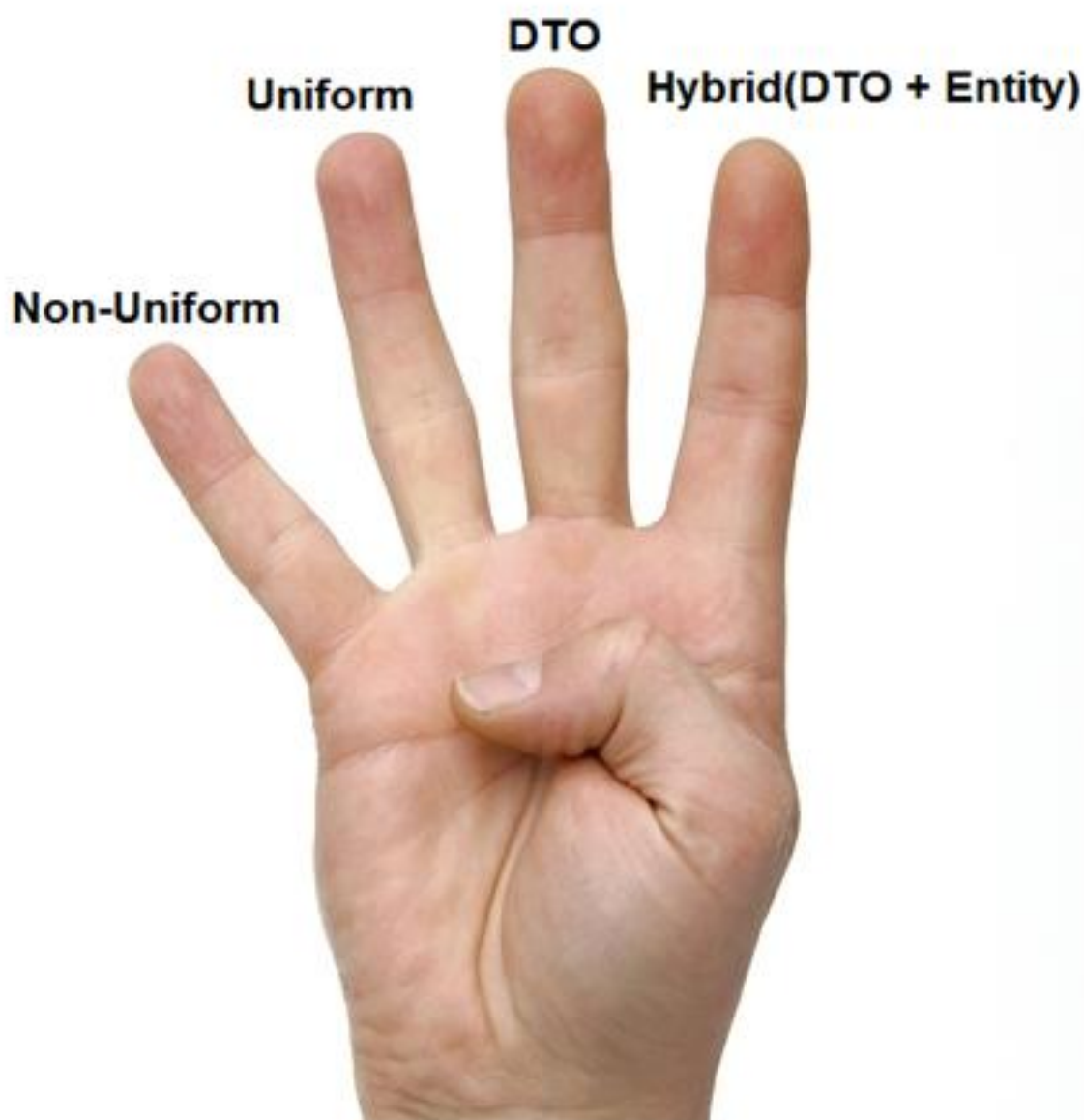
where id in (.....)

در (chkdel.Checked)  
شما ID بگیرید

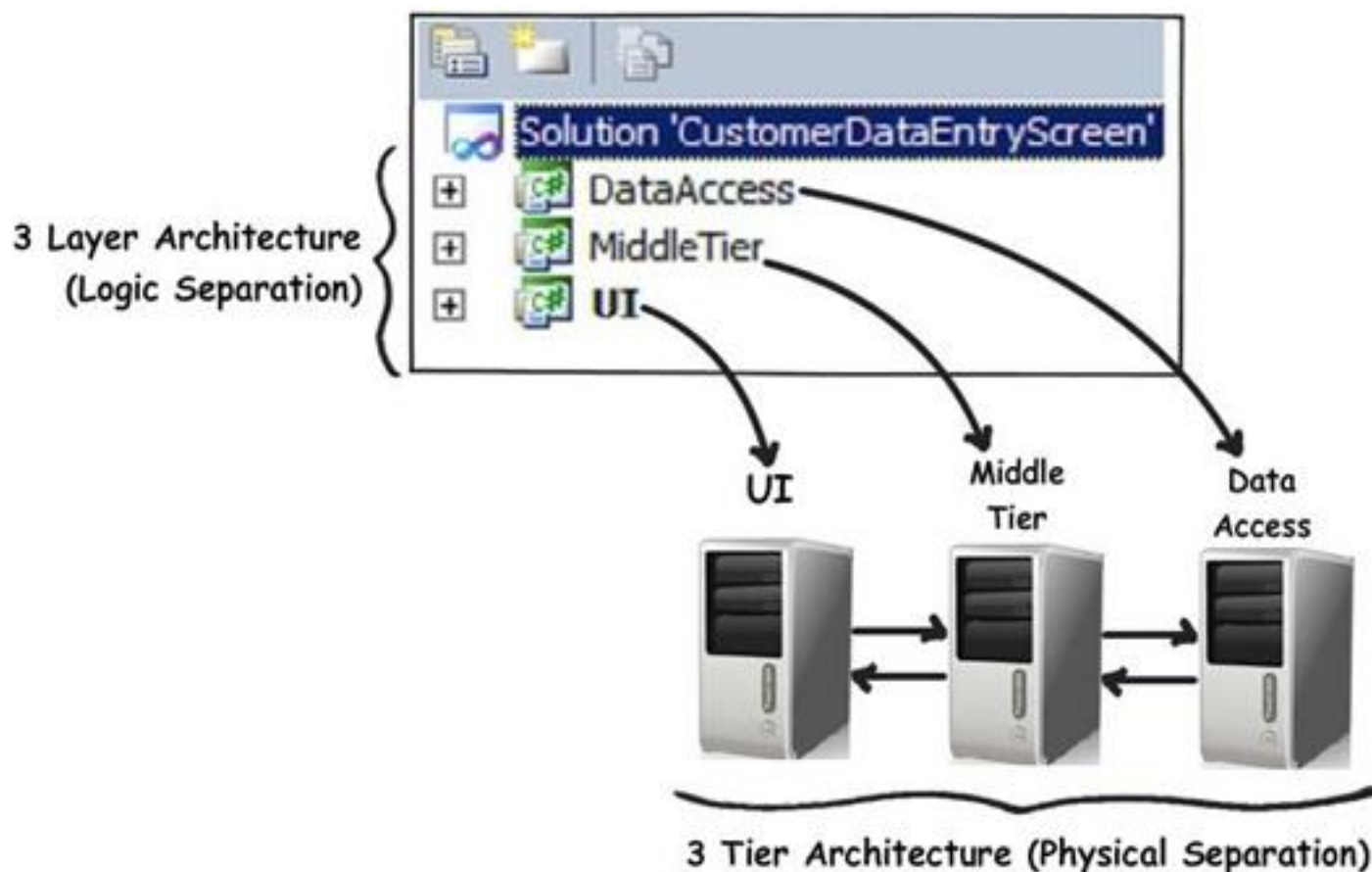
نویسنده: عثمان رحیمی  
تاریخ: ۲۱:۴۱ ۱۳۹۳/۰۷/۰۳

بله حق با شماست می‌توان کدهای فوق را نسبت به چیزی که گذاشته ام بهینه‌تر نوشت . ممنون از شما

معماری لایه بندی شده، یک معماری بسیار همه گیر می باشد. به این خاطر که به راحتی decoupling ، SOC و قدرت درک کد را بسیار بالا می برد. امروزه کمتر برنامه نویس و فعال حوضه ی نرم افزاری است که با لایه های کلی و وظایف آنها آشنا نباشد ( UI layer آنچه که ما می بینیم، middle layer برای مقاصد منطق کاری، data access layer برای هندل کردن دسترسی به داده ها). اما مسئله ای که بیشتر برنامه نویسان و توسعه دهندگان نرم افزار با استانداردهای آن آشنا نیستند، راه های تبادل داده ها مابین layer ها می باشد. در این مقاله سعی داریم راه های تبادل داده ها را مابین لایه ها، تشریح کنیم.

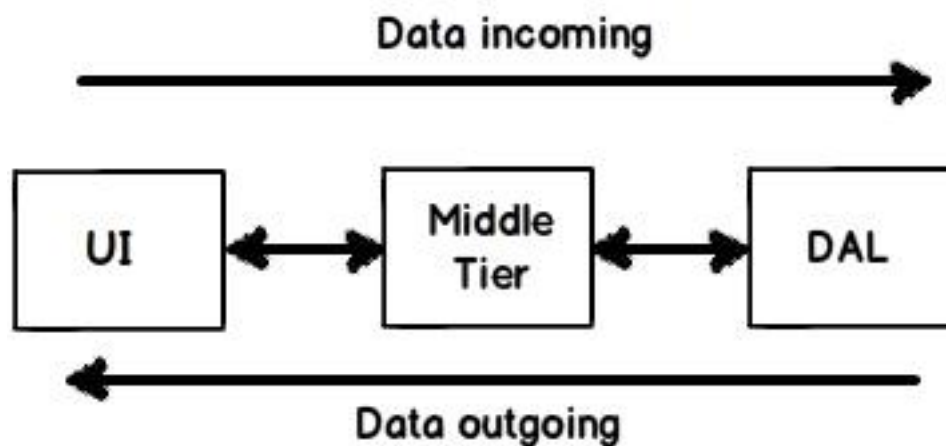


Layer با Tier متفاوت است. هنگامیکه در مورد مفهوم layer و Tier دچار شک شدید، دیاگرام ذیل می تواند به شما بسیار کمک کند. layer به مجزاسازی منطقی کد و Tier هم به مجزا سازی فیزیکی در ماشین های مختلف اطلاق می شود. توجه داشته باشید که این نکته یک شفاف سازی کلی در مورد یک مسئله مهم بود.



داده های وارد شونده (incoming) و خارج شونده (outgoing)

ما باید تبادل داده ها را از دو جنبه مورد بررسی قرار دهیم؛ اول اینکه داده ها چگونه به سمت لایه Data Access می روند، دوم اینکه داده ها چگونه به لایه UI پاس می شوند، در ادامه شما دلیل این مجزا سازی را درک خواهید کرد.



### روش اول: Non-uniform

این روش اولین روش و احتمالاً عمومی‌ترین روش می‌باشد. خوب، اجازه دهید از لایه‌ی UI به لایه DAL شروع کنیم. داده‌ها از لایه UI به Middle با استفاده از getter ها و setter ها ارسال خواهد شد. کد ذیل این مسئله را به روشنی نمایش می‌دهد.

```
Customer objCust = new Customer();
objCust.CustomerCode = "c001";
objCust.CustomerName = "Shivprasad";
```

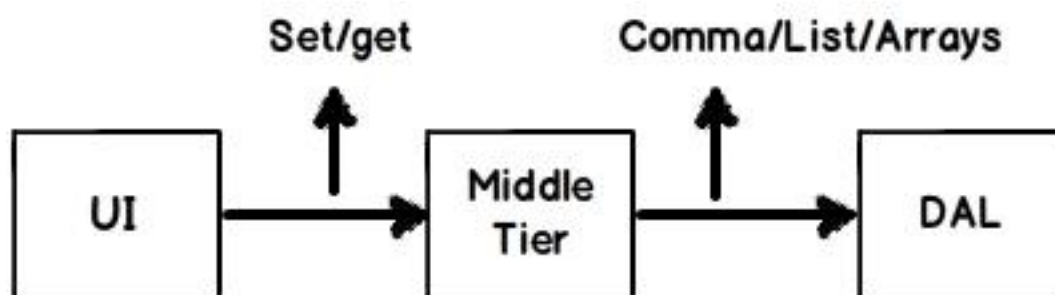
بعد از آن، از لایه Middle به لایه Data Access داده‌ها با استفاده از مجزاسازی به وسیله comma و سایر روش‌های non-uniform پاس داده می‌شوند. در کد ذیل به متد Add دقت کنید که چگونه فراخوانی به لایه Data Access را با استفاده از پارامترهای ورودی انجام می‌دهد.

```
public class Customer
{
    private string _CustomerName = "";
    private string _CustomerCode = "";
    public string CustomerCode
    {
        get { return _CustomerCode; }
        set { _CustomerCode = value; }
    }
    public string CustomerName
    {
        get { return _CustomerName; }
        set { _CustomerName = value; }
    }
    public void Add()
    {
        CustomerDal obj = new CustomerDal();
        obj.Add(_CustomerName,_CustomerCode);
    }
}
```

کد ذیل، متد add در لایه Data Access را با استفاده از دو متد نمایش می‌دهد.

```
public class CustomerDal
{
    public bool Add(string CustomerName,string CustomerCode)
    {
        // Insert data in to DB
    }
}
```

بنابراین اگر بخواهیم به صورت خلاصه نحوه پاس دادن داده ها را در روش non-uniform بیان کنیم، شکل ذیل به زیبایی این مسئله را نشان می دهد.



• از لایه UI به لایه Middle با استفاده از getter و setter

• از لایه Middle به لایه data access با استفاده از comma , input , array

حال نوبت این است بررسی کنیم که چگونه داده ها از DAL به UI در روش non-uniform پاس خواهند شد. بنابراین اجازه دهید که اول از UI شروع کنیم. از لایه UI داده ها با استفاده از object های لایه Middle واکشی می شوند.

```
Customer obj = new Customer();  
List<Customer> oCustomers = obj.getCustomers();
```

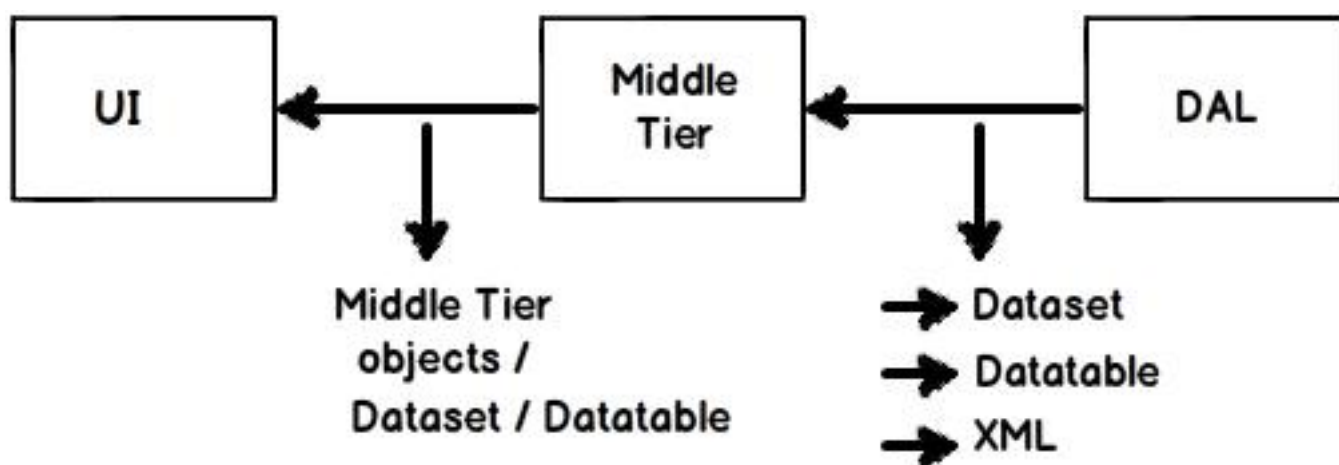
از لایه Middle هم داده ها با استفاده از datatable , dataset و xml پاس خواهند شد. مهمترین مسئله برای لایه loop , middle بر روی dataset و تبدیل آن به strong type object ها می باشد. برای مثال می توانید کد تابع getCustomers که بر روی dataset loop می زند و یک لیست از Customer ها را آماده می کند در ذیل مشاهده کنید. این تبدیل باید انجام شود، به این دلیل که UI به کلاس های strongly typed دسترسی دارد.

```
public class Customer  
{  
    private string _CustomerName = "";  
    private string _CustomerCode = "";  
    public string CustomerCode  
    {  
        get { return _CustomerCode; }  
        set { _CustomerCode = value; }  
    }  
    public string CustomerName  
    {  
        get { return _CustomerName; }  
        set { _CustomerName = value; }  
    }  
    public List<Customer> getCustomers()  
    {  
        CustomerDal obj = new CustomerDal();  
        DataSet ds = obj.getCustomers();  
        List<Customer> oCustomers = new List<Customer>();  
        foreach (DataRow orow in ds.Tables[0].Rows)  
        {  
            // Fill the list  
        }  
        return oCustomers;  
    }  
}
```

با انجام این تبدیل به یکی از بزرگترین اهداف معماری لایه بندی شده می‌رسیم؛ یعنی اینکه « UI نمی‌تواند به طور مستقیم به کامپوننت‌های لایه Data Access مانند OLEDB ، ADO.NET و غیره دستیابی داشته باشد. با این روش اگر ما در ادامه متدولوژی Data Access را تغییر دهیم تاثیری بر روی لایه UI نمی‌گذارد.» آخرین مسئله اینکه کلاس CustomerDal یک Dataset را با استفاده از ADO.NET بر می‌گرداند و Middle از آن استفاده می‌کند.

```
public class CustomerDal
{
    public DataSet getCustomers()
    {
        // fetch customer records
        return new DataSet();
    }
}
```

حال اگر بخواهیم حرکت داده‌ها را به لایه UI ، به صورت خلاصه بیان کنیم، شکل ذیل کامل این مسئله را نشان می‌دهد.



• داده‌ها از لایه DAL به لایه Middle با استفاده از XML ، Datareader ، Dataset ارسال خواهند شد.

• از لایه Middle به UI از strongly typed classes استفاده می‌شود.

### مزایا و معایب روش non-uniform

یکی از مزایای non-uniform

• به راحتی قابل پیاده سازی می‌باشد، در مواردی که روش data access تغییر نمی‌کند این روش کارآیی لازم را دارد.

تعدادی از معایب این روش

• به خاطر اینکه یک ساختار uniform نداریم، بنابراین نیاز داریم که همیشه در هنگام عبور از یک لایه به یک لایه دیگر از یک ساختار به یک ساختار دیگر تبدیل را انجام دهیم.

• برنامه نویسان از روش‌های خودشان برای پاس دیتا استفاده می‌کنند؛ بنابراین این مسئله خود باعث پیچیدگی می‌شود.

• اگر برای مثال شما بخواهید متدولوژی Data Access خود را تغییر دهید، تغییرات بر تمام لایه‌ها تاثیر می‌گذارد.

## نظرات خوانندگان

نویسنده: بابک جهانگیری  
تاریخ: ۱۳:۲۰ ۱۳۹۴/۰۴/۰۴

آیا در این روش می توان به صورت DataView لیست مشتریها رو برگردوند به جای اینکه از `<List<Customer>` استفاده کنیم ؟ باز هم به آن non-uniform می گویند ؟

نویسنده: ریوف مدرسی  
تاریخ: ۱۷:۵۳ ۱۳۹۴/۰۴/۰۵

در این روش مسئله اصلی این نیست که داده ها رو به صورت list یا DataView برگردونید، بلکه مسئله اصلی این است که شما مجبورید در گذر از هر لایه تبدیل ساختار داده ها را انجام دهید، پس نکته این روش این است که تعداد تبدیل ساختار داده ها زیاد است.

نویسنده: محسن اسماعیل پور  
تاریخ: ۸:۲۵ ۱۳۹۴/۰۴/۰۸

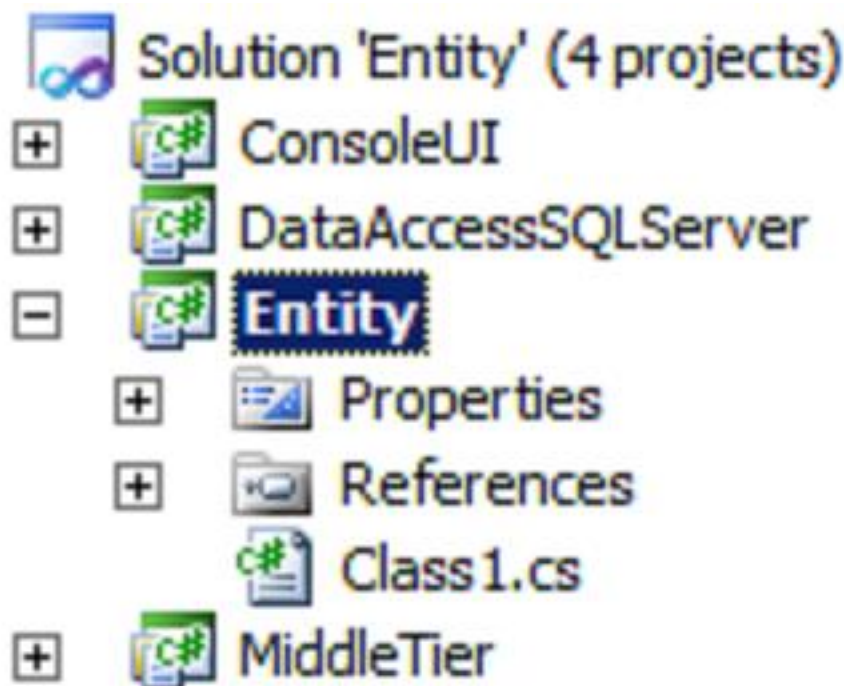
مدل Customer که شما برای مثالهایتان از آن استفاده کرده اید از [Active record pattern](#) تبعیت میکند. از آنجا که Entity یا Model با عملیات CRUD لایه دیتا Couple شده و بعضا ممکن است Business Logic داخل این متدها قرار گیرد، این مسئله با Separation Of Concern منافات دارد.



قسمت اول : تبادل داده ها بین لایه ها- قسمت اول

## روش دوم: (Uniform Entity classes)

روش دیگر پاس دادن داده ها، روش uniform است. در این روش کلاس های Entity ، یک سری کلاس ساده به همراه یکسری Property های Set و Get می باشند. این کلاس ها شامل هیچ منطق کاری نمی باشند. برای مثال کلاس CustomerEntity که دارای دو Property ، Customer Name و Customer Code می باشد. شما می توانید تمام Entity ها را به صورت یک پروژه ی مجزا ایجاد کرده و به تمام لایه ها رفرنس دهید.



```
public class CustomerEntity
{
    protected string _CustomerName = "";
    protected string _CustomerCode = "";
    public string CustomerCode
    {
        get { return _CustomerCode; }
        set { _CustomerCode = value; }
    }
    public string CustomerName
    {
        get { return _CustomerName; }
        set { _CustomerName = value; }
    }
}
```

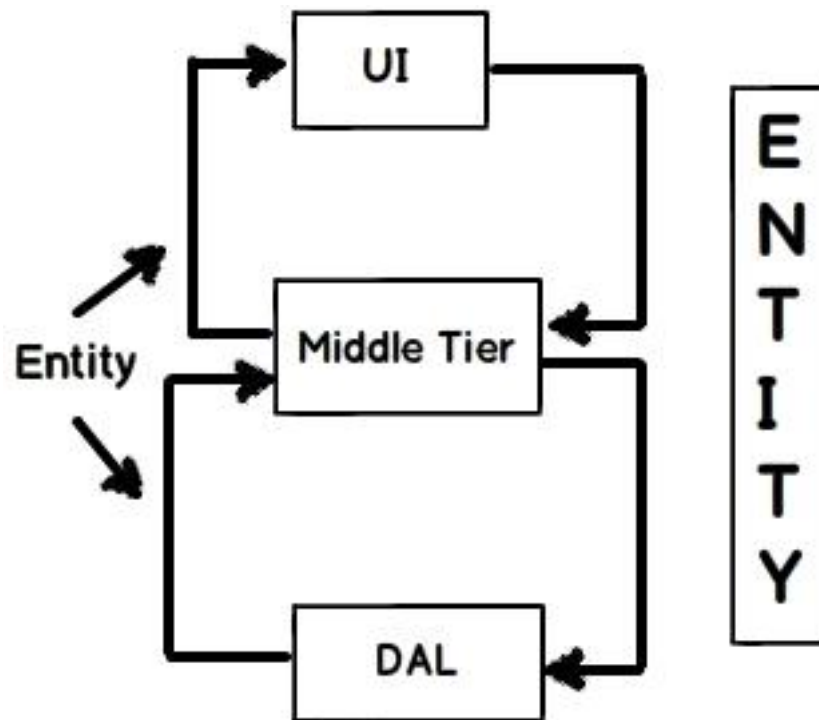
خوب، اجازه دهید تا از CustomerDal شروع کنیم. این کلاس یک Collection از CustomerEntity را بر می گرداند و همچنین یک CustomerEntity را برای اضافه کردن به دیتابیس. توجه داشته باشید که لایه Data Access وظیفه دارد تا دیتای دریافتی از دیتابیس را به CustomerEntity تبدیل کند.

```
public class CustomerDal
{
    public List<CustomerEntity> getCustomers()
    {
        // fetch customer records
        return new List<CustomerEntity>();
    }
    public bool Add(CustomerEntity obj)
    {
        // Insert in to DB
        return true;
    }
}
```

لایه Middle از CustomerEntity ارث بری می کند و یکسری operation را به entity class اضافه خواهد کرد. داده ها در قالب Entity Class به لایه Data Access ارسال می شوند و در همین قالب نیز بازگشت داده می شوند. این مسئله در کد ذیل به روشنی مشاهده می شود.

```
public class Customer : CustomerEntity
{
    public List<CustomerEntity> getCustomers()
    {
        CustomerDal obj = new CustomerDal();
        return obj.getCustomers();
    }
    public void Add()
    {
        CustomerDal obj = new CustomerDal();
        obj.Add(this);
    }
}
```

لایه UI هم با تعریف یک Customer و فراخوانی operation های مربوط به آن، داده ی مد نظر خود را در قالب CustomerEntity بازیابی خواهد کرد. اگر بخواهیم عمکرد روش uniform را خلاصه کنیم باید بگوییم، در این روش دیتای رد و بدل شده ی مابین کلیه لایه ها با یک ساختار استاندارد، یعنی Entity پاس داده می شوند.



مزایا و معایب روش uniform

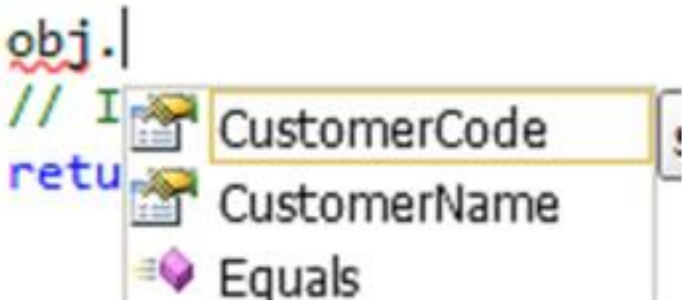
مزایا

Strongly typed به صورت در تمامی لایه ها قابل دسترسی و استفاده می باشد.

```

public class CustomerDal
{
    public List<CustomerEntity>
    {
        // fetch customer record
        return new List<CustomerEntity>()
    }
    public bool Add(CustomerEntity obj)
    {
        // Insert
        return true
    }
}

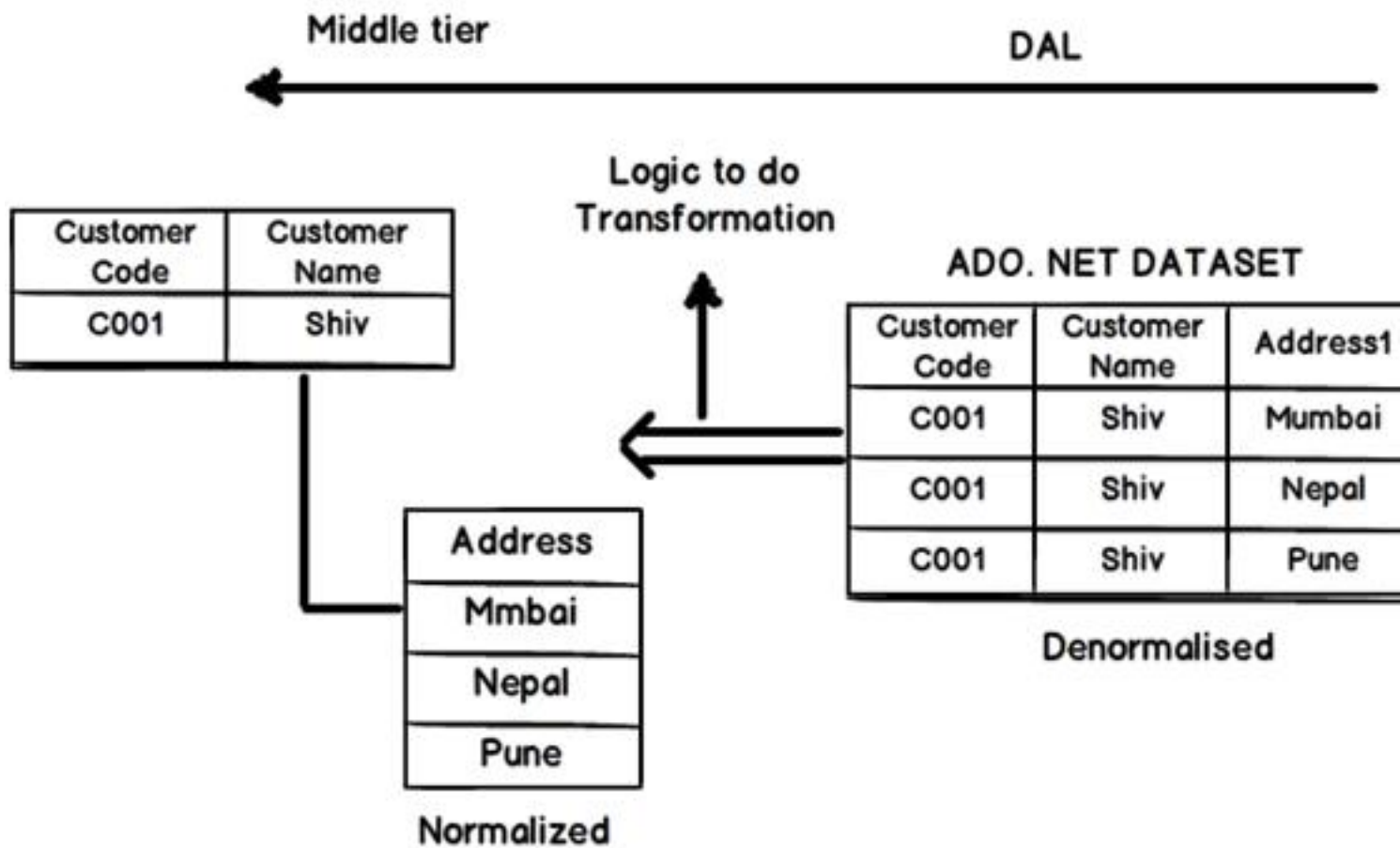
```



• به دلیل اینکه از ساختار عمومی Entity استفاده می‌کند، بنابراین فقط یکبار نیاز به تبدیل داده‌ها وجود دارد. به این معنی که کافی است یک بار دیتای واکنشی شده از دیتابیس را به یک ساختار Entity تبدیل کنید و در ادامه بدون هیچ تبدیل دیگری از این Entity استفاده کنید.

### معایب

• تنها مشکلی که این روش دارد، مشکلی است به نام Double Loop. هنگامیکه شما در مورد کلاس‌های entity بحث می‌کنید، ساختارهای دنیای واقعی را مدل می‌کنید. حال فرض کنید شما به دلیل یکسری مسایل فنی دیتابیس خود را Optimize کرده اید. بنابراین ساختار دنیای واقعی با ساختاری که شما در نرم افزار مدل کرده‌اید متفاوت می‌باشد. بگذارید یک مثال بزنیم؛ فرض کنید که یک customer دارید، به همراه یکسری Address. همان طور که ذکر کردیم، به دلیل برخی مسایل فنی (denormalized) به صورت یک جدول در دیتابیس ذخیره شده است. بنابراین سرعت واکنشی اطلاعات بیشتر است. اما خوب اگر ما بخواهیم این ساختار را در دنیای واقعی بررسی کنیم، ممکن است با یک ساختار یک به چند مانند شکل ذیل برخورد کنیم.



بنابراین مجبوریم یکسری کد جهت این تبدیل همانند کد ذیل بنویسیم.

```
foreach (DataRow o1 in oCustomers.Tables[0].Rows)
{
    obj.Add(new CustomerEntityAddress()); // Fills customer
    foreach (DataRow o in oAddress.Tables[0].Rows)
    {
        obj[0].Add(new AddressEntity()); // Fills address
    }
}
```

عنوان: تبادل داده‌ها بین لایه‌ها؛ قسمت آخر

نویسنده: سید ریوف مدرسی

تاریخ: ۲۰:۴۵ ۱۳۹۴/۰۶/۰۳

آدرس: [www.dotnettips.info](http://www.dotnettips.info)

گروه‌ها: ADO.NET, Design patterns, Architecture, OOP, N-Layer Architecture, Architectural Patterns

## روش سوم: DTO (Data transfer objects)

در قسمت‌های قبلی دو روش از روش‌های موجود جهت تبادل داده‌ها بین لایه‌ها، ذکر گردید و علاوه بر این، مزایا و معایب هر کدام از آنها نیز ذکر شد. در این قسمت دو روش دیگر، به همراه مزایا و معایب آنها برشمرده می‌شود. لازم به ذکر است هر کدام از این روش‌ها می‌تواند با توجه به شرایط موجود و نظر طراح نرم افزار، دارای تغییراتی جهت رسیدن به یکسری اهداف و فاکتورها در نرم افزار باشد.

در این روش ما سعی می‌کنیم طراحی کلاس‌ها را به اصطلاح مسطح (flatten) کنیم تا بر مشکل double loop که در قسمت قبل بحث کردیم غلبه کنیم. در کد ذیل مشاهده می‌کنید که چگونه کلاس CustomerDTO از CustomerEntity مشتق می‌شود و کلاس Address را با CustomerEntity ادغام می‌کند؛ تا برای افزایش سرعت لود و نمایش داده‌ها، یک کلاس de-normalized شده ایجاد نماید.

```
public class CustomerDTO : CustomerEntity
{
    public AddressEntity _Address = new AddressEntity();
}
```

در کد ذیل می‌توانید مشاهده کنید که چگونه با استفاده از فقط یک loop یک کلاس de-normalized شده را پر می‌کنیم.

```
foreach (DataRow o1 in oCustomers.Tables[0].Rows)
{
    CustomerDTO o = new CustomerDTO();
    o.CustomerCode = o1[0].ToString();
    o.CustomerName = o1[1].ToString();
    o._Address.Address1 = o1[2].ToString();
    o._Address.Address2 = o1[3].ToString();
    obj.Add(o);
}
```

UI هم به راحتی می‌تواند DTO را فراخوانی کرده و دیتا را دریافت کند.

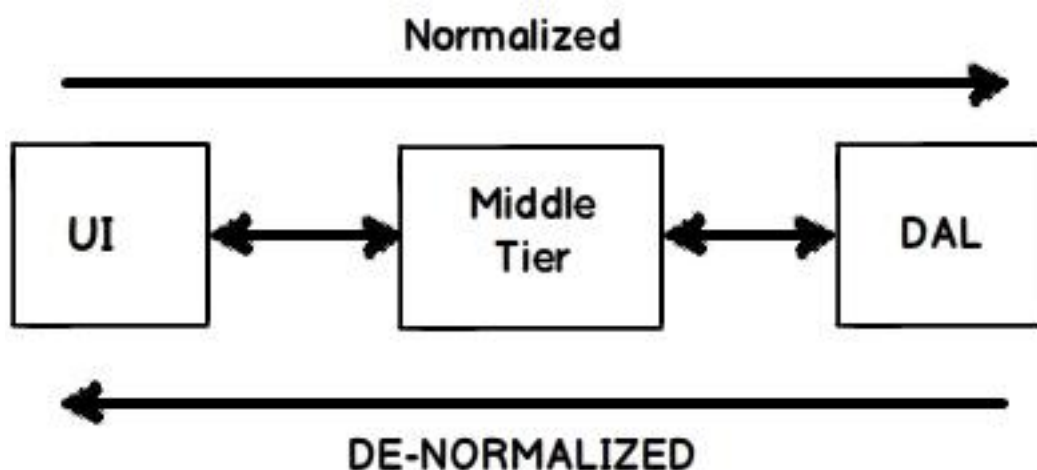
## مزایا و معایب روش DTO

یکی از بزرگترین مزایای این روش سرعت زیاد در بارگذاری اطلاعات، به دلیل استفاده کردن از ساختار de-normalized می‌باشد. اما همین مسئله خود یک عیب محسوب می‌شود؛ به این دلیل که اصول شی گرای را نقض می‌کند.

## روش چهارم: Hybrid approach (Entity + DTO)

از یک طرف کلاس‌های Entity که دنیای واقعی را مدل خواهند کرد و همچنین اصول شی گرای را رعایت می‌کنند و از یک طرف دیگر DTO نیز یک ساختار flatten را برای رسیدن به اهداف کارآیی دنبال خواهند کرد. خوب، به نظر می‌رسد که بهترین کار استفاده از هر دو روش و مزایای آن روش‌ها باشد.

زمانیکه سیستم، اهدافی مانند انجام اعمال CRUD را دنبال می‌کند و شما می‌خواهید مطمئن شوید که اطلاعات، دارای integrity می‌باشند و یا اینکه می‌خواهید این ساختار را مستقیماً به کاربر نهایی ارائه دهید، استفاده کردن از روش (Entity) به عنوان یک روش normalized می‌تواند بهترین روش باشد. اما اگر می‌خواهید حجم بزرگی از دیتا را نمایش دهید، مانند گزارشات طولانی، بنابراین استفاده از روش DTO با توجه به اینکه یک روش de-normalized به شمار می‌رود بهترین روش می‌باشد.



کدام روش بهتر است؟

**Non-uniform** : این روش برای حالتی است که متدهای مربوط به data access تغییرات زیادی را تجربه نخواهند کرد. به عبارت دیگر، اگر پروژه‌ی شما در آینده دیتابیس‌های مختلفی را مبتنی بر تکنولوژی‌های متفاوت، لازم نیست پشتیبانی کند، این روش می‌تواند بهترین روش باشد.

**Uniform : Entity, DTO, or hybrid** : اگر امکان دارد که پروژه‌ی شما با انواع مختلف دیتابیس‌ها مانند Oracle و Postgres ارتباط برقرار کند، استفاده کردن از این روش پیشنهاد می‌شود.