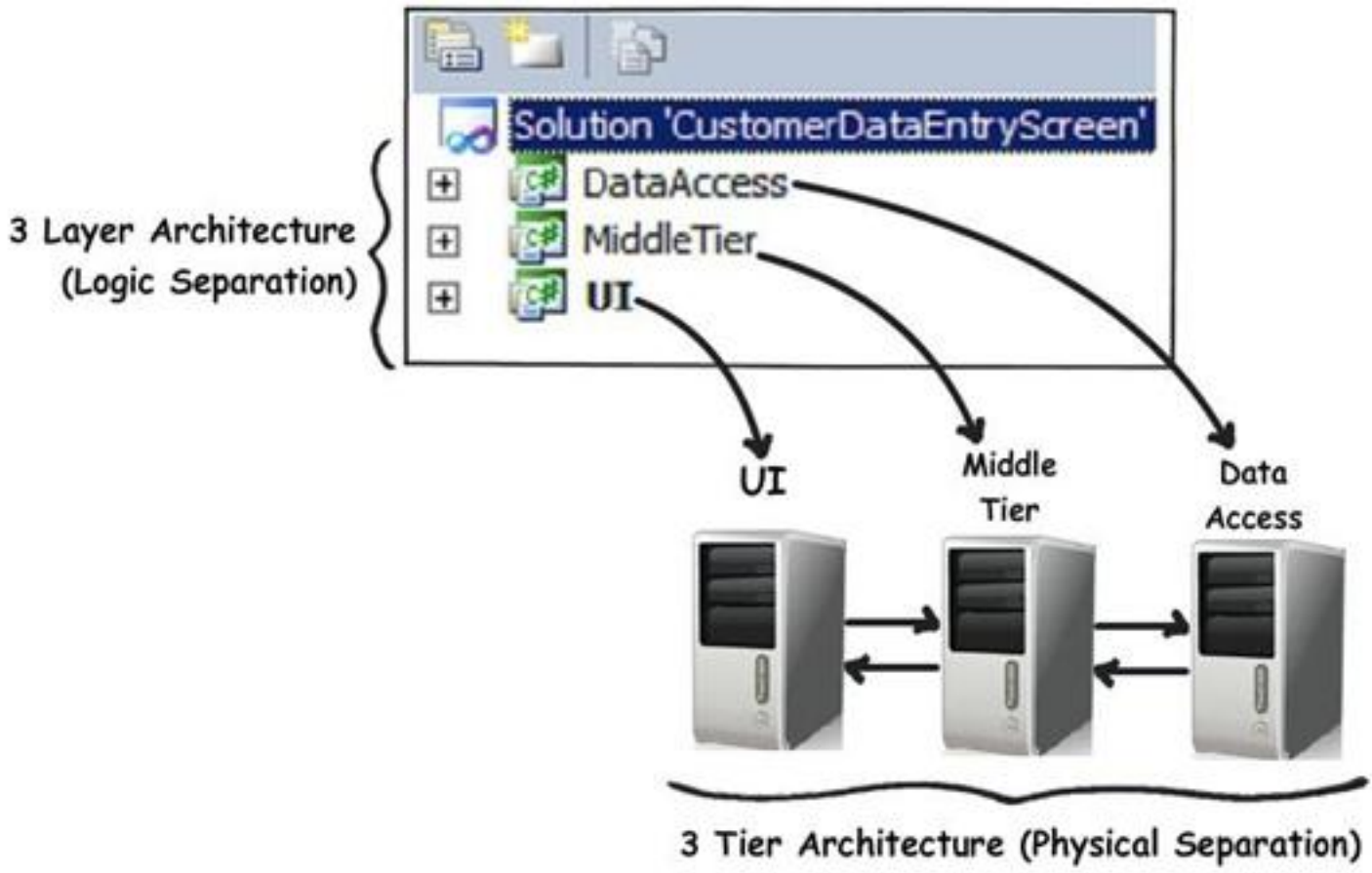
عنوان: **تبادل داده ها بین لایه ها- قسمت اول**
نویسنده: ریوف مدرسی
تاریخ: ۱۳۹۴/۰۳/۳۱ ۲۱:۳۰
آدرس: www.dotnettips.info
گروه‌ها: ADO.NET, Design patterns, Architecture, OOP, N-Layer Architecture

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

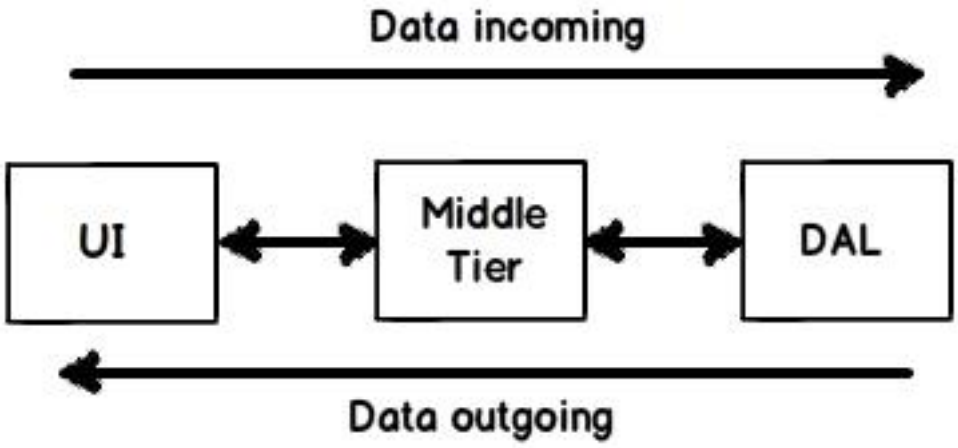

**Layers و Tiers با هم متفاوت هستند**

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




**داده‌های وارد شونده( incoming ) و خارج شونده( outgoing )**

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




**روش اول: Non-uniform**

این روش اولین روش و احتمالا 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";
```

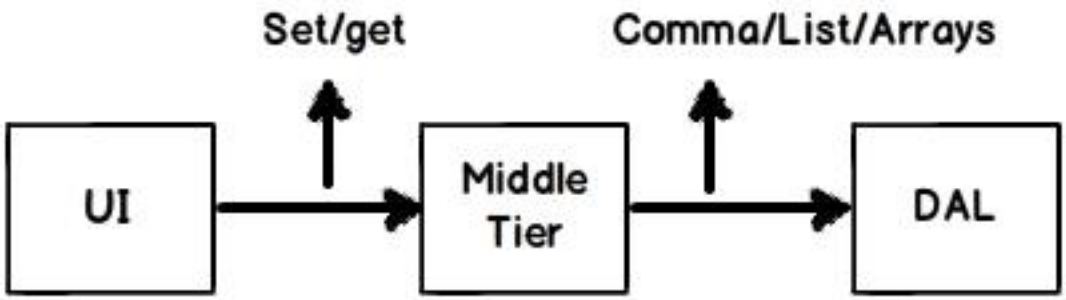
بعد از آن، از لایه Middle به لایه Data Access داده‌ها با استفاده از مجزاسازی به وسیله comma و آرایه‌ها و سایر روش‌های non-uniform پاس داده می‌شوند. در کد ذیل به متد Add دقت کنید که چگونه فراخوانی به لایه Data Access را با استفاده از پارامترهای ورودی انجام می‌دهد.

```
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 داده‌ها را در روش non-uniform بیان کنیم، شکل ذیل به زیبایی این مسئله را نشان می‌دهد.




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

- از لایه Middle به لایه data access با استفاده از array ، input ، comma

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

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

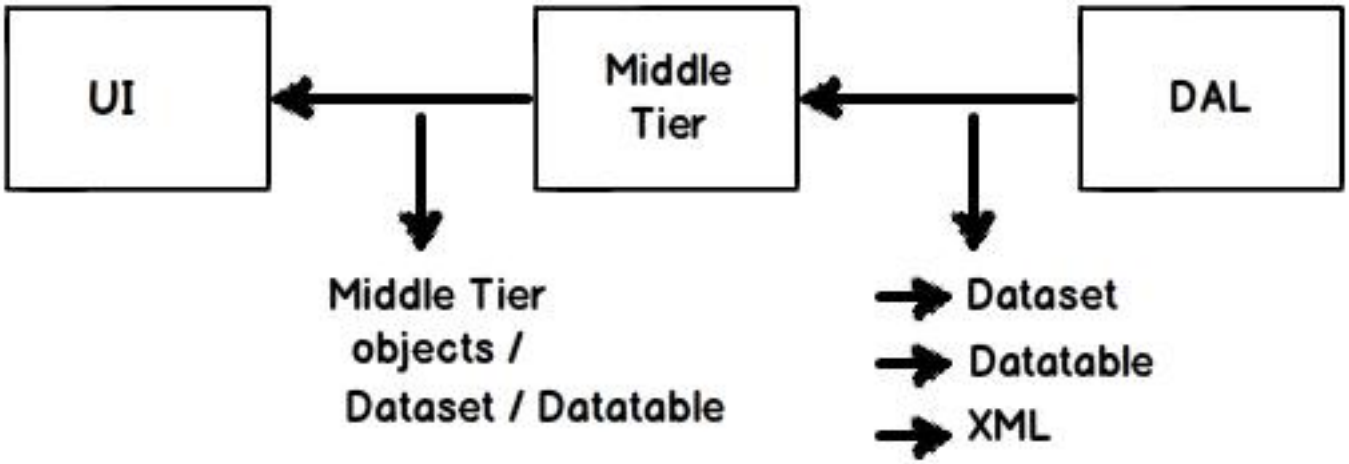
از لایه Middle هم داده‌ها با استفاده از datatable ، dataset و xml پاس خواهند شد. مهمترین مسئله برای لایه middle ، loop بر روی dataset و تبدیل آن به strong type object ها می‌باشد. برای مثال می‌توانید کد تابع getCustomers که بر روی ، dataset loop می‌زند و یک لیست از Customer ها را آماده می‌کند در ذیل مشاهده کنید. این تبدیل باید انجام شود، به این دلیل که UI به کلاس‌های 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 لایه‌ی 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: ۱۳۹۴/۰۴/۰۴ ۱۳:۲۰

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

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

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

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

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 Sepration Of Concern منافات دارد.