《现代软件开发——使用.NET与C#》系列第八讲

多层应用程序的开发

俞晖
MSDN讲师
dolphin@vip.163.com

《现代软件开发——使用.NET与C#》系列

- 回顾上节课的内容 *ADO.NET*
- 本节课讲述在C#中的多层应用程序开发的问题
- **LEVEL 300**
- 基础:
  - 初级编程
  - **DB&SQL**

Date	Topic
2005年8月16日	*ADO.NET*
2005年9月13日	Multi-tier Application Design
2005年9月26日	**Component-based Design and Programming**
.	.

Today's Objectives

“对于大部分商业应用程序，核心的设计在于数据层和商务逻辑层。没有一个‘正确’的解决方案，只能按需交替使用……” [Hummel]

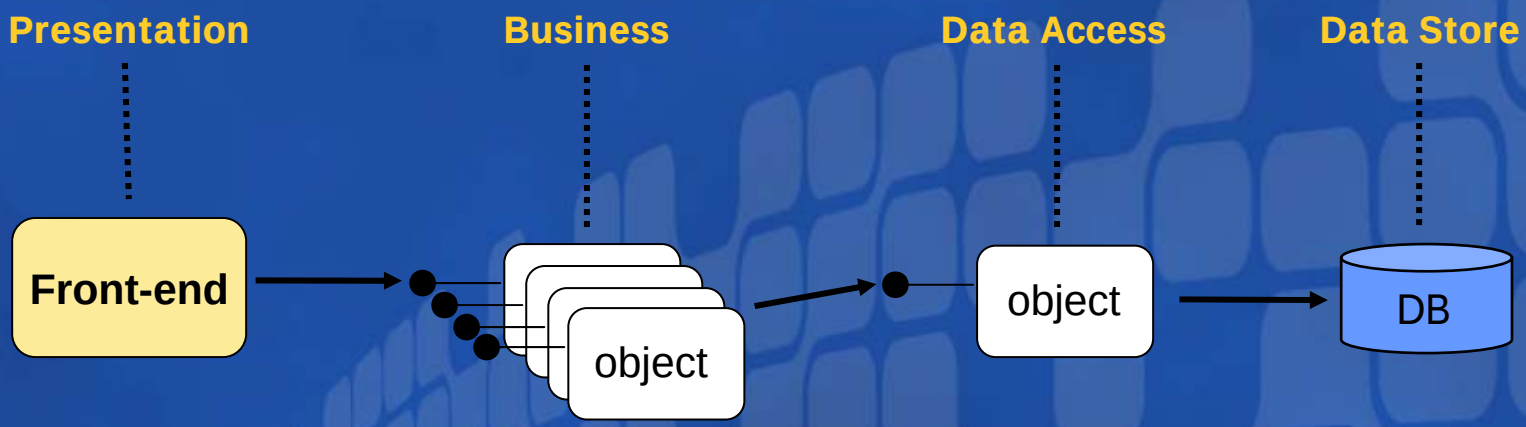
- 议题：
  - 多层设计的概述
  - 商务逻辑层
  - 数据访问层

日程

- *多层设计概述*
- 商务逻辑层
- 数据访问层
- 强类型**DataSet（Typed DataSets）**

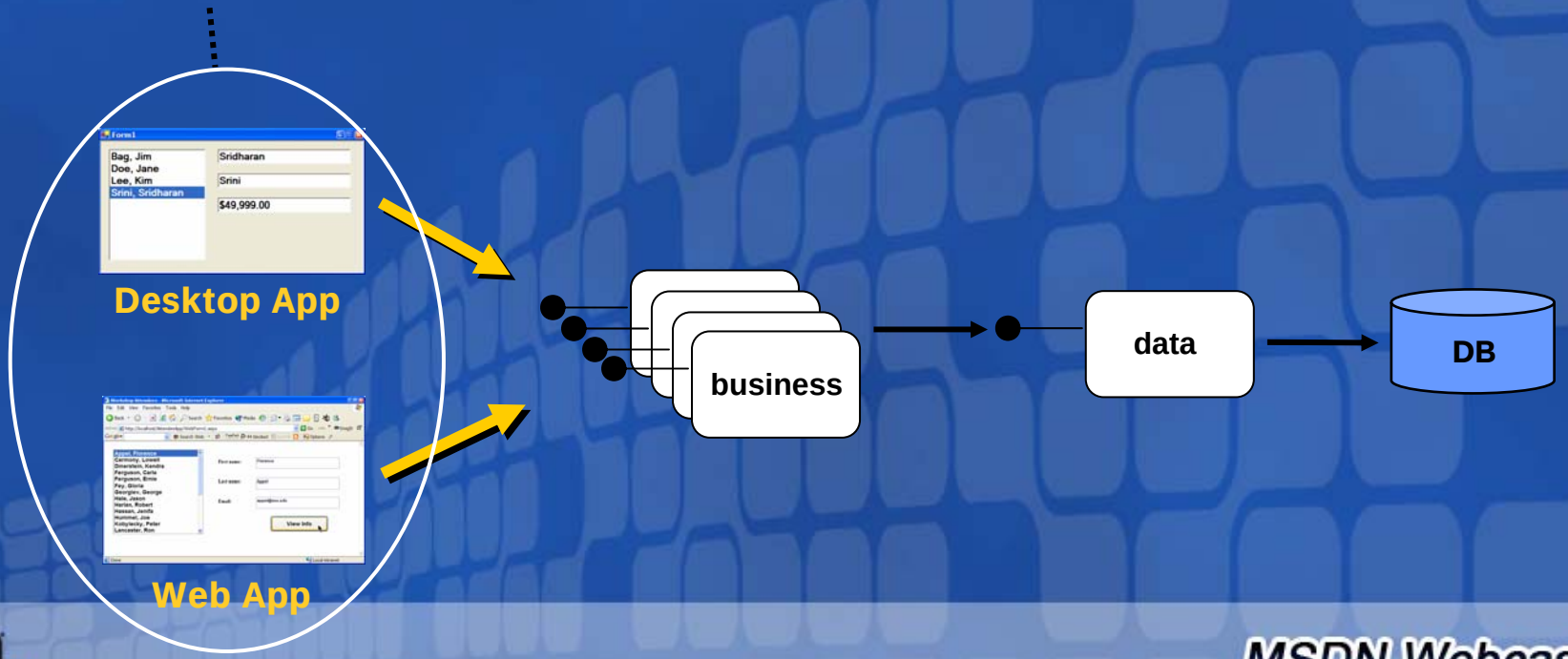
多层设计

- 商业应用程序一般会有多层
  - *表现层，商务逻辑层，数据访问层和数据存储*



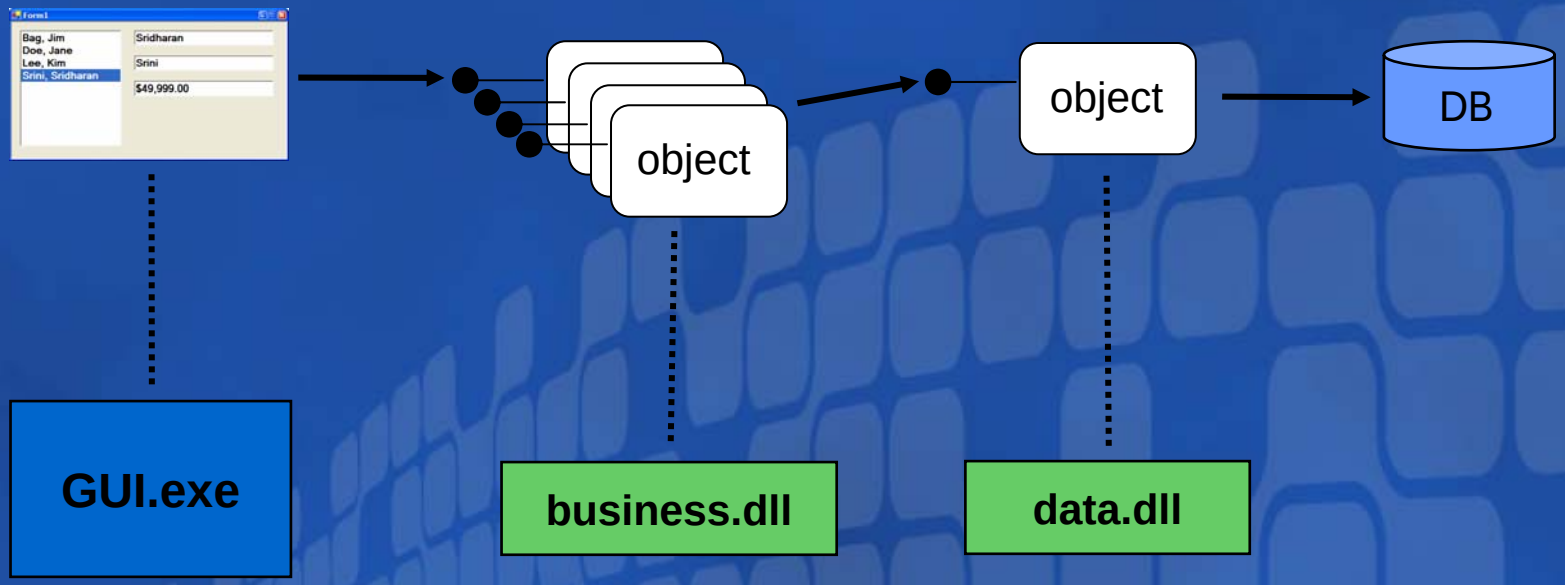
为什么？

- 为什么？
  - 分离逻辑思维的需要
  - 团队或多语言开发的需要
  - 重用商务逻辑层与数据层的需要

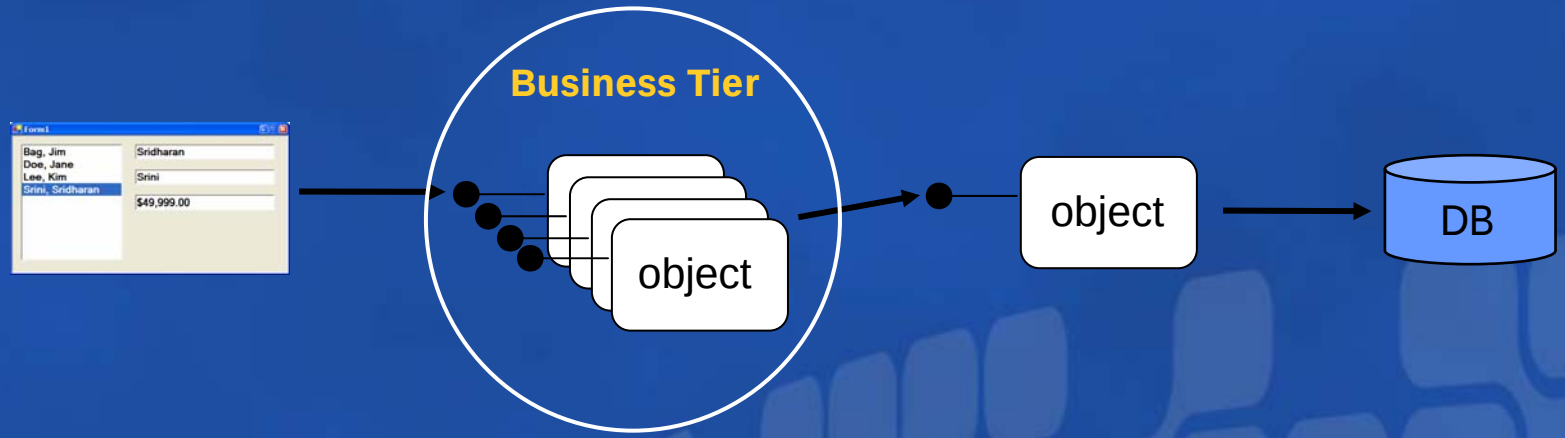


不要弄混逻辑与物理

- 使用多层是 *逻辑* 设计
- 打包和放置是 *物理* 设计



观点



- 目标?

“支持商务过程中的所有涉及安全，商务规则（逻辑）和数据的处理（校验，数据操作）。”

范例

- 对于Banking App来说，BT加强了银行的规则的处理：
  - 账户类型，最小余额，借款数额，等等
- 对于Sales App来说，BT加强了公司的规则处理：
  - 付款方式，运输费用，等等

问题

- 整体设计
- 安全性
- 商务规则
- 数据校验处理

(1) To OOP or Not to OOP

- 三种解决方案:
  - 完全 **OOP:**
    - *任何事物都是对象，大概会有1000多个对象存储在 RAM*
    - *例如： Customers, Products, Orders, Addresses, …*
    - *OOP的巨大能量能处理任何的问题，但也带来一定的压力。*
  - 大颗粒（**Large-grain**） **OOP:**
    - *任何事物都是对象，但是对象是大颗粒的。*
    - *少量的全面的对象，虽然不优雅，但是更实际。*
  - 不使用 **OOP:**
    - *没有对象— 返回过程样式的编程*
    - *没有类— 基于结构和静态方法*

例子：大颗粒设计

- 我们不要把所有的customers都存储在RAM中。
  - 通过ID获得唯一的customer。
  - 可以查找customer IDs的子集。

一个对象代表了所有的Customers

```
namespace BusinessTier
{
   public class Customers
   {
        public Customer Get(int customerID)
        { ... }

        public int[] Search(...)
        { ... }
```

返回IDs的数组，不是对象

搜索条件

```
namespace BusinessTier
{
   public class Customer
   {
        .
        .
        .
   }
}
```

(2) 安全性

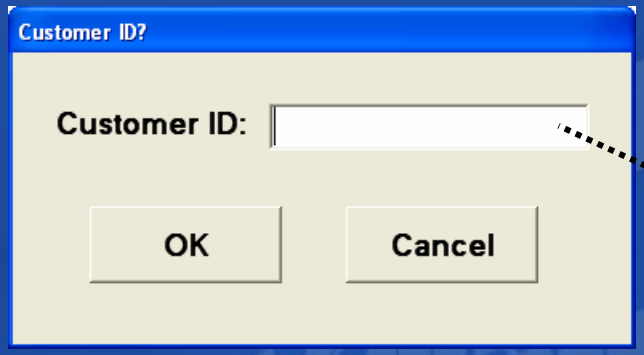
- 不幸的是，安全性非常的复杂，不是一次Webcast就能说清楚的。
- 安全性对于任何一层的设计都有潜在的影响
  - 谁将使用本系统？（验证）
  - 是否允许该用户进行此操作？（权限）
  - 在何处进行安全检查？
    - 商务逻辑层
    - 表现层是否需要反映出用户的权限？
    - 数据层是否需要进行检查？
- 最基本的，通过商务规则来进行检验.....

(3) 商务规则

- 典型地通过明确地编码表现
- 不同的商务情况有不同的规则
  - Does customer have a credit limit?
  - Does customer meet minimum age requirements?
  - Does user have permission to make this change?
  - ...
- 但，有一条通用的规则 – *（校验）validation*

检验（Validation）

- 表现层是第一级检验
- 例如：
  - 整形输入




```
try
{
   s = << user's input >>;
   i = int.Parse(s);
}
catch(FormatException)
{ ... }
catch(OverflowException)
{ ... }
catch(Exception ex)
{ ... }
```

Business Tier Validation

- 典型的情况，BT再做一次额外的检验
  - valid ID number?
  - valid credit card number?
  - valid email address?
- 如何?
  - 编写有力的代码控制
  - *正则表达式（Regular expressions) ...*

正则表达式

- REs 是强大的检验技术
- 过程:
  - *有效的输入被表现为一种模式*
  - *对于用户的输入，运行模式进行检验*
- 例如:
  - “\d” 指定单一的数字 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - “\w+@\w+” 是一个简单的Email地址模式。（1个或多个字符加上@再加上一个或多个字符）

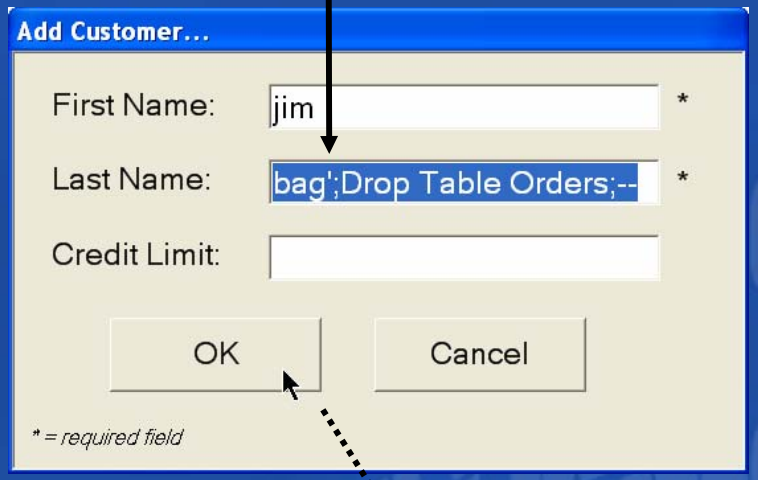
例子

- 检验Zipcode是否符合US的标准
  - Zip or Zip+4
  - 模式：^ 表示开始, ? 可选, $ 结束

```
using RE = System.Text.RegularExpressions.Regex;

public class Validation
{

   public static bool isUSZipcode(string s)
   {
      string  pattern;
      pattern = @"^(\d{5})((\-\d{4})?)$";

      return RE.Match(s, pattern).Success;
   }
```

防止SQL输入攻击

- 在数据被传入到数据层之前，对于文本框的内容必须检验
  - 把单引号转换为双引号




```
public static bool
isProperlyEscaped(string s)
{
   // allow either no ‘, or an even number of ‘, otherwise
fail…
  string  pattern = "^([^']|(''))*$";

  return RE.Match(s, pattern).Success;
}
```

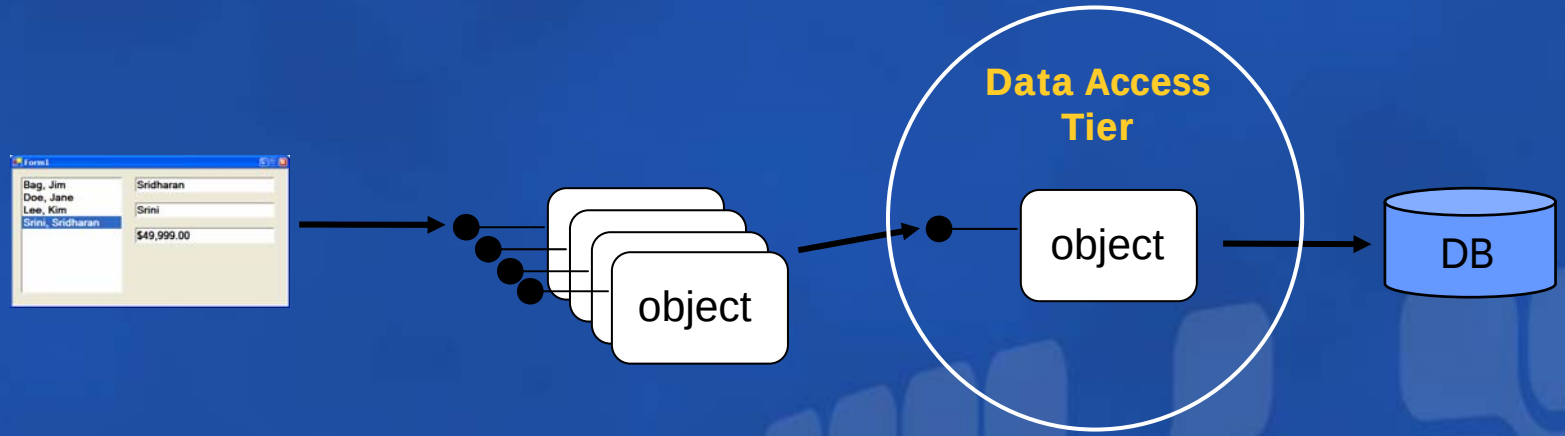
```
if (Validation.isProperlyEscaped(s))
  ;  // okay
else
  s = s.Replace(“’”, “’’”);
```

Demonstration One

demo

正则表达式...

观点



- 目标?

"数据访问层并不管理和存储数据，它只是提供商务逻辑层和数据库之间的接口。

[Lhotka]

如何设计接口？

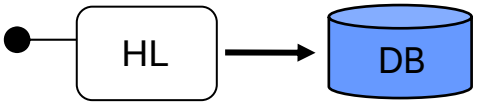
- 考虑的主要问题：
  - **High-level or low-level 接口?**
  - 存储过程或者动态**SQL**语句？

(1) HL 或者 LL 接口?

- high-level 接口隐藏BT传来的细节信息

Business Tier delegates all responsibilities to Data Access Tier

```
namespace BusinessTier
{
   public class Customers
   {
        private DataAccessTier.DataAccess  dataStore;
        .
        .
        .

        public void Update(Customer c)
        {
           dataStore.Update(c);
        }
```

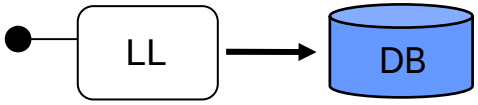


LL 接口?

- low-level 接口

Business Tier accepts some responsibility for data storage

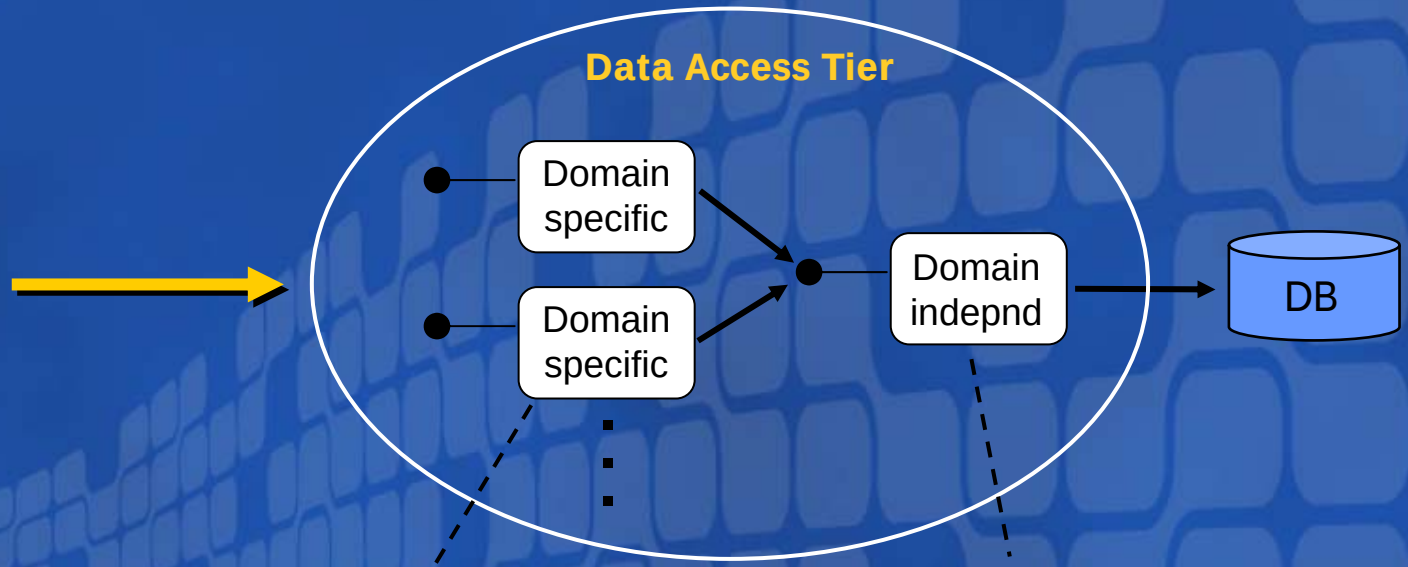
```
namespace BusinessTier
{
   public class Customers
   {
        private DataAccessTier.DataAccess  dataStore;
        .
        .
        .

        public void Update(Customer c)
        {
           string  sql;
           sql = String.Format("Update Customers Set ...", ...);
           dataStore.Execute(sql);
        }
```



- *劣势？数据存储细节将延伸到 BT*

明显的妥协方式

- 分为两级:
  - Domain specific数据访问代码
  - Domain indepnd “CRUD” (*Create/Read/Update/Delete*)



(2) 是否使用存储过程？

- 存储过程是存储到数据库的SQL语句
  - DBAs 编写
  - 数据库模式与应用程序分离
  - 提供数据访问控制的另一层
  - 封装复杂的SQL
  - 保证事务执行
  - 可能对提高性能产生很大的帮助

```
CREATE PROCEDURE sprocTopTen
As
Set RowCount 10
Select *
  From Customers
  Order By AccountBalance
Desc
```

执行存储过程

- 使用标准的ADO.NET机制来执行
- 例如:
  - 获取前10名最高账户余额的客户
  - 通过ID删除客户信息

```
sql = "Execute sprocTopTen;";

dbCmd = new SQL.SqlCommand(sql, dbConn);
dbReader = dbCmd.ExecuteReader();

while (dbReader.Read())    // retrieve records from server 1-by-1...
 ...
```

```
sql = string.Format("Execute sprocDeleteByCID {0};", id);

dbCmd = new SQL.SqlCommand(sql, dbConn);
rows  = dbCmd.ExecuteNonQuery(sql);
```

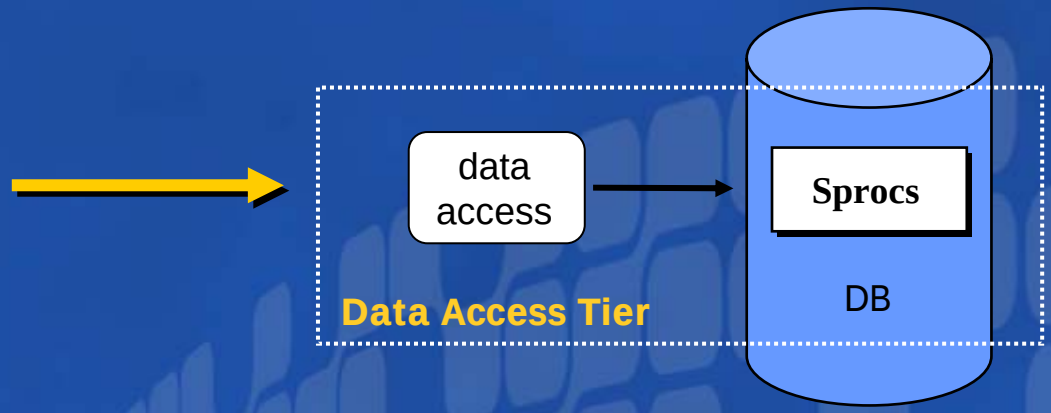
Demonstration Two


```
CREATE PROCEDURE sprocDeleteByCID
( @CID bigint )
As
Delete From Customers Where CID=@CID
Return(@@ROWCOUNT)
```

存储过程...（上节课已经做过一些）

如何合并?

- 一些或者所有的数据访问层，如今经常存在于DB中。

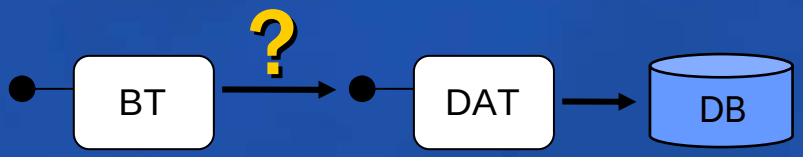


- *超级信徒把整个DAT实现在数据库...*
- ***http://www.adoguy.com/***

存储过程的劣势?

- Vendor-specific
- 灵活性降低
  - 对于数据访问层的改变将影响数据库

层间如何传递数据？



- 基本类型的数组？
- 对象的集合？
- 原始的 **DataSet?**
- *强类型* **DataSet?**

- **Typical trade-offs:**
  - **convenience, elegance, efficiency, quality of OOP-ness**

强类型 DataSets

- 强类型 dataset 是特殊化的 DataSet
  - 强类型DataSet是从DataSet继承的定制对象，它可以通过其显露的属性（properties）来对封装数据进行强类型的访问。
  - 强类型DataSet避免了字段的晚绑定。强类型DataSet提供了强类型的访问器，因为避免了到一个集合中查找列名或表名，访问时更快。
  - 除了能够提高运行时的性能，强类型DataSet还提供了类型检查，并且在设计时可以通过自定义字段名对字段智能感知。

例子

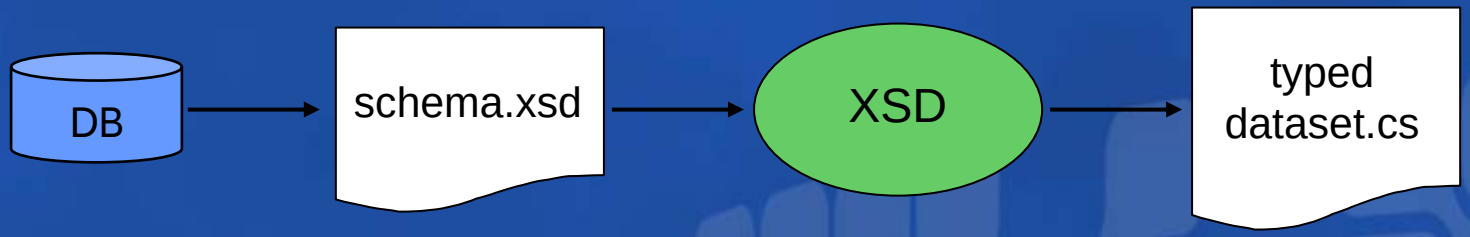
- 用户数据的例子

```
System.Data.DataSet                DS;
DataAccessTier.CustomersDataSet    customersDS;
  .
  .
  .

// Fill and access a standard DataSet:
dataAdapter.Fill(DS);
FirstName = DS.Tables[“Customers”].Rows[i][“FirstName”].ToString();

// Filling and accessing a typed DataSet:
dataAdapter.Fill(customersDS);
FirstName = customersDS.Customers[i].FirstName;
```

```
public class CustomersDataSet : System.Data.DataSet
{
   .
   .
   .
}
```

如何建立强类型

- 方便的，面向对象的，且易建立的
  - 可以使用**Visual Studio** 来建立



强类型DataSet == BT?

- 一些说法认为强类型dataset 可以当作BT!
  1. 建立强类型DataSet
  2. 添加一些实现商务逻辑的方法

- 可以工作，但一定要注意
  - 如果DB的模式发生了改变，你必须重新建立强类型，这时你将失去商务逻辑

其它参考资源

- **Webcast:**
  - http://www.microsoft.com/china/msdn/webcasts
- **Resources:**
  - MSDN Library
    http://www.microsoft.com/china/MSDN/library/default.mspx
- **MSDN flash:**
  - http://www.microsoft.com/msdn/china/newsletter/
- **Duwamish （Demo 中有安装文件）**

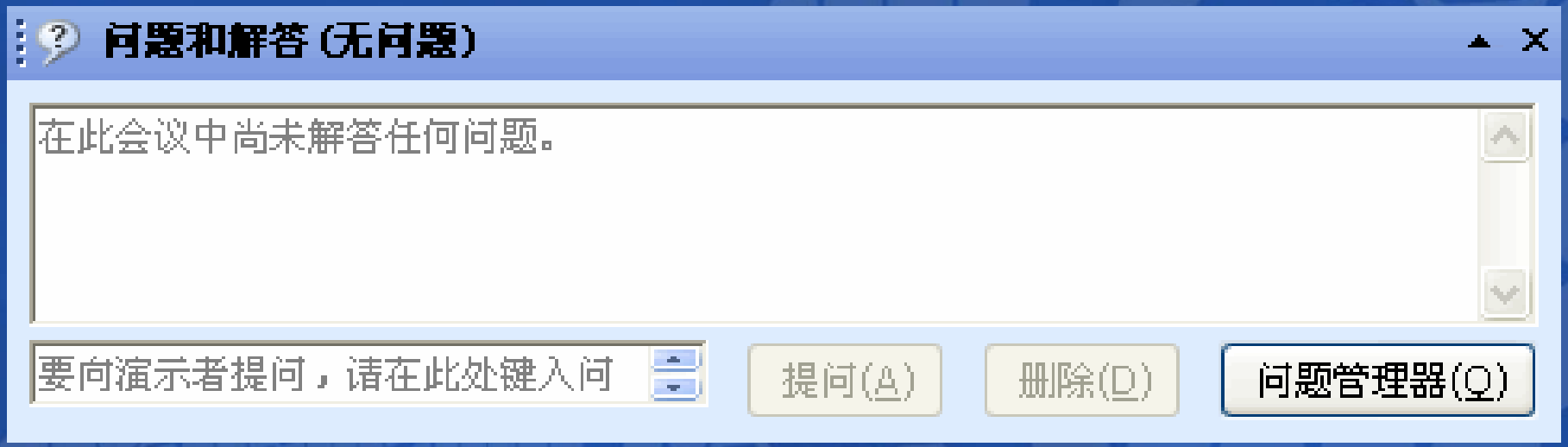
感谢大家的参与!

- 非常感谢大家的参与
- 您的反馈对我们来说是非常重要的

Date	Topic
2005年8月16日	ADO.NET
2005年9月	多层应用程序开发
2005年9月	基于组件的应用程序开发

Question & Answer

如需提出问题，请单击“提问”按钮并在随后显示的浮动面板中输入问题内容。一旦完成问题输入后，请单击“提问”按钮。




您的潜力，我们的动力
Microsoft
微软（中国）有限公司
Microsoft
msdn
MSDN Webcasts