- 基础
  - 文件结构
  - 生命周期
- Page 类
  - IsPostBack 属性
  - IsValid 属性
  - 内置对象
    - Response
    - Request
    - Server
  - web.config
  - 异常处理机制
- 服务器控件
  - 普通控件
  - 验证控件
- ADO.NET
  - 数据访问模式
  - 核心类
  - SqlConnection
    - 常用属性
    - 常用方法
    - 示例
  - 连接模式
    - 流程
    - SqlCommand
    - SqlDataReader
    - SqlParameter
  - 断开模式
    - 流程
    - DataSet
      - DataTable
      - DataColumn
      - DataRow
      - DataView
    - SqlDataAdapter

基础

文件结构

- .cs： 后台代码文件

- .aspx：标准 Web 页面文件
- .ascx：ASP.NET 控件
- .asmx：ASP.NET Web 服务，提供一系列方法来供其他应用程序远程调用
- web.config：基于 XML 的配置文件
- Global.asax：全局应用程序，定义在整个应用程序范围内可用的全局变量

文件夹:

- Bin：已编译好的 .NET 组件程序集，任何页面可用
- App_Code：源代码，与 Bin 的区别是动态编译
- App_Data：数据文件，也可存储在其他任何地方
- App_Broswers：标识个别浏览器并确定其功能的浏览器定义文件
- App_GlobalResources：所有页面都可用的资源
- App_LocalResources：该目录下资源的访问权限仅限单个页面
- App_WebReferences：Web 服务文件

生命周期

1. 用户从 Web 服务器请求应用程序资源
2. ASP.NET 接收应用程序的第一个请求
3. 为每个请求创建 ASP.NET 核心对象
4. 将 HttpApplication 对象分配给请求
5. 由 HttpApplication 管线处理请求

Page 类

ASP.NET 网页最终都继承自 Page 类

IsPostBack 属性

表示该页是否为响应客户端回发而加载

- true：页面回发
- false：首次加载访问

```
protected void Page_Load(object sender, EventArgs e) {
    if (!IsPostBack) { }
}
```

IsValid 属性

表示页面验证是否成功

内置对象

Response

HttpResponse 将响应发送到客户端

- Write(str)：将数据发送到浏览器， str 可为 HTML 标记
- WriteFile(path)：将 path 指定的文件写入输出流
- Redirect()：重定向

Request

- ApplicationPath：虚拟根路径
- PhysicalApplicationPath：服务器绝对路径
- MapPath(vPath)：将参数 vPath 指定的虚拟路径转化为实际路径
- SaveAs(filename, header)：将 HTTP 请求保存到磁盘

Server

- Transfer()：终止当前页的执行，执行新的页

web.config

```
<?xml version="1.0" ?>
<configuration>
    <configSections></configSections>
    <appSettings></appSettings>
    <connectionStrings></connectionStrings>
    <system.web></system.web>
    <system.codedom></system.codedom>
    <system.webServer></system.webServer>
    <runtime></runtime>
</configuration>
```

- appSettings：自定义信息块，如标题、开发者
- connectionStrings：连接数据库的信息
- system.web：ASP.NET 的设置，可包含多个

异常处理机制

1. try-catch 代码块
2. 页面级： Page_Error 事件

```
protected void Page_Error(object sender, EventArgs e) {
    Exception objErr = Server.GetLastError(); // 获取未处理的异常
    Server.ClearError(); // 清除异常
}
```

3. 页面级：ErrorPage 属性
    1. web.config 中 <customErrors mode="On">
    2. 指定 ErrorPage 属性

```
protected void Page_Load() {
    this.ErrorPage = "~/Error.html";
}
```

4. 应用程序级：Application_Error 事件
    1. 添加 Global.asax 文件
    2. 添加 Application_Error 事件

```
void Application_Error() {
    Exception ex = Server.GetLastError();
}
```

5. <customErrors> 配置项

```
<customErrors mode="On" defaultRedirect="ErrorPage.htm">
    <error statuscode="403" redirect="NoAccess.htm">
    <error statuscode="404" redirect="FileNotFound.htm">
</customErrors>
```

    - RemoteOnly：对远程用户显示通用的错误页，对本地用户显示详细页
    - Off
    - On

服务器控件

普通控件

- TextBox
    - AutoPostBack
    - MaxLength
    - TextChange()
- HyperLink：用于创建文本或图像超链接
    - ImageUrl
    - NavigateUrl
- Button、LinkButton、ImageButton
    - 只是样式不同
- DropDownList、ListItem：用于创建下拉列表
    - AutoPostBack
    - Items：选项集合
    - SelectedIndex：当前选项的下标
    - SelectedItem：当前选项对象

    - SelectedValue：当前选中项的值
    - SelectedIndexChanged()
    - ListItem 是列表子项，有以下属性
        - Text
        - Value
        - Selected
- ListBox：显示多个选项的列表框
    - Rows：行数
    - SelectionMode： Single / Multiple
- CheckBox 和 CheckBoxList：前者创建复选框，后者创建控件组
    - AutoPostBack
    - Checked
    - Text
    - CheckedChanged()

```
<asp:CheckBoxList>
    <asp:CheckBox ID="cb1" runat="server" Text="Opt">
</asp:CheckBoxList>
```

- RadioButton 和 RadioButtonList
    - GroupName：设置单选按钮所属的组名
    - Checked
- Panel：放置其他控件的容器
    - Visible
- MultiView：选项卡效果

验证控件

共有属性：

- ControlToValidate：指定要验证的输入控件
- Text：验证失败时，验证控件显示的文本，如果没有该属性，则显示 ErrorMessage 中的信息
- ErrorMessage：验证失败时，ValidationSummary 控件中显示的信息
- IsValid：每个控件公开该属性，用于判断

类型：

- RequiredFieldValidator
    - InitialValue：与初值相同则不通过
- CompareValidator
    - ControlToCompare：指定要与所验证的输入控件进行比较的控件
    - Operator
    - ValueToCompare
- RangeValidator

- RegularExpressionValidator
- CustomValidator

```
protected void Server_Validate(object source, ServerValidateEventArgs args) {
    args.IsValid = true;
}
```

- ValidationSummary，集中显示 ErrorMessage

ADO.NET

ADO.NET 是 .NET Framework 提供的数据访问类库，为数据源提供了一致性的访问

命名空间如下：

- System.Data.SqlClient：SQL Server
- System.Data.OleDb：Access、SQL Server 6.5 或更低版本
- System.Data.Obdc：OBDC
- System.Data.OracleClient：Oracle

数据访问模式

- 连接模式：连接到数据库，具有操作数据库数据的功能
- 断开模式：离线编辑和更新，处理完成交由连接类型进行更新

通过数据库连接对象建立与数据库的连接；使用数据适配器对象（DataAdapter）从数据库读取数据填充到内存的 DataSet，供应用程序使用

DataReader 一次只能把数据表中的一条记录读入内存

DataSet 是一个内存数据库，是 DataTable 容器，在 System.Data 下，用于在内存中暂存数据

核心类

- *Connection*：创建连接
- *Command*：执行查询数据、修改数据等命令
- *DataReader*：读取数据库数据
- *DataAdapter*：使用 *Command* 对象在数据源中执行 SQL 命令向 DataSet 中加载数据

SqlConnection

常用属性

- ConnectionString：获取或设置数据库连接字符串，通常包含以下参数：
  - Server

    - Database
    - AttachDBFilename：数据库路径和文件名
    - Uid：数据库的账户名
    - Pwd：密码
    - Integrated Security：是否启用 Windows 身份验证
        - true
        - false
        - SSPI
    - Connection Timeout
- ConnectionTimeout：获取超时时间
- Database：当前数据库名称
- DataSource：数据源的完整路径和文件名，若是 SQL Server 数据库，则获取连接服务器名称
- State：获取连接状态，其值是 ConnectionState 的枚举值

常用方法

- Open()：打开连接
- Close()：关闭连接
- Begintransaction()：执行事务
- ChangeDatabase()：更改数据库
- CreateCommand()：创建并返回 Command 对象
- Dispose()：调用 Close() 关闭数据库连接，并释放所占用资源

示例

使用命令创建连接：

```
SqlConnection dbo = new SqlConnection("Data Source=.\SQLEXPRESS; AttachDBFilename=|DataDirectory|\

// 或

SqlConnection dbo = new SqlConnection();
dbo.ConnectionString = "Data Source=.\SQLEXPRESS; AttachDBFilename=|DataDirectory|\db1.mdf";
```

使用配置文件：

```
<configuration>
    <connectionStrings>
        <add name="LinkName"
            connectionString="Data Source=10.1.122.74;
                Initial Catalog=student;
                User ID=sa;
                Password=pass;"
            providerName="System.Data.SqlClient"/>
    <connectionStrings>
<configuration>
```

使用 System.Configuration.ConfigurationManager 类读取连接字符串：

```
using System.Configuration;
string  strCnn = ConfigurationManager.connectionStrings["LinkName"].ConnectionString;
```

连接模式

流程

1. 创建 SqlConnection 对象，建立连接
2. 创建 SqlCommand 对象，对数据库 CRUD
3. 创建 SqlDataReader 对象，读取结果集

SqlCommand

常用属性：

- CommandText：获取或设置 SQL 命令、存储过程、数据表名称
- CommandType：获取或设置命令类型
  - Text
  - StoredProcedure
  - TableDirect
- Connection：获取或设置连接属性
- Parameters：命令参数集合
- Transaction：设置 Command 所属的事务

```
SqlCommand cmd = new SqlCommand("SQL", linkObj, tranObj);
```

常用方法：

- Cancel()
- ExecuteNonQuery()：执行命令，返回受影响的行数
- ExecuteReader()：执行命令，返回 DataReader 对象
- ExecuteScalar()：执行命令，以 Object 类型返回，用于查询单值

SqlDataReader

该对象不能直接实例化，需调用 `ExecuteReader()`

常用属性:

- `FieldCount`：字段数
- `IsClosed`
- `HasRows`：是否包含数据

常用方法:

- `Close()`
- `Read()`：记录指针指向下一条，返回 `true` 或 `false`
- `NextResult()`
- `GetValue(i)`：返回指定列的值
- `GetName(i)`：获得指定列的名称

SqlParameter

- `ParameterName`：参数名称
- `SqlDbType`：数据库类型
- `Size`：数值长度
- `Direction`： `Input` / `Output` / `InputOutput`
- `Value`：参数

断开模式

流程

1. 创建 SqlCommand 对象，建立连接
2. 创建 SqlDataApter 对象
3. 执行操作
    - 如果要查询，使用 SQLDataAdapter 的 `Fill()` 方法填充 DataSet 对象
    - 如果要操作，先更新 DataSet，再使用 SqlDataApter 的 `Update()` 方法更新数据

DataSet

- 表集合
    - 行集合
    - 列集合
    - 约束集合
- 关系集合

```
DataSet ds = new DataSet("set_name");
```

- DataSetName
- Table
- Clear()
- Copy()

DataTable

```
DataTable ds = new DataTable("table_name");
```

- Rows：获取所有行
- Columns：获取所有字段
- DataSet：获取 DataTable 对象所属的 DataSet 对象
- DefaultView
- PrimaryKey
- TableName
- Clear()
- NewRow()

创建好的数据表对象，可添加到数据集对象中

```
ds.Tables.Add(dt);
```

DataColumn

```
DataColumn col = new DataColumn("colName", System.Type.GetType("System.String"));
```

DataRow

不能通过 *new* 创建，而是通过 DataTable 的 NewRow 创建

DataView

```
DataView obj = new DataView(tb);
```

SqlDataAdapter

常用属性:

- SelectedCommand
- InsertCommand
- UpdateCommand
- DeleteCommand

常用方法:

- Fill()：自动执行 SelectedCommand，获取数据集并填充 DataTable

- Update()：自动执行更改操作