教学支持系统

编码文档

文件状态： [ ] 草稿 [√] 正式发布 [ ] 正在修改	**文件标识：**	教学支持系统
	当前版本：	1.0
	作　　者：	余阳舟
	完成日期：	2010-07-12

目录

1.引言.....3
1.1 编写目的.....3
1.2 项目背景.....3
1.3 定义.....3
1.4 参考资料.....3
2. 代码规范.....4
2.1 文件组织规范.....4
2.1.1 源文件.....4
2.1.2 目录.....4
2.2 命名规范.....4
2.2.1 命名空间.....4
2.2.2 变量.....5
2.2.3 类名.....5
2.2.4 控件名.....5
2.2.5 接口.....5
2.2.6 方法.....6
2.2.7 属性.....6
2.3 缩进.....6
2.4 空白.....7
2.4.1 空行.....7
2.4.2 内部空格.....7
2.5 注释.....8
2.5.1 块注释.....8
2.5.2 单行注释.....8
2.5.3 方法注释.....8
2.5.4 文件注释.....9
3. 源程序清单.....9
3.1 实体层.....9
3.2 数据访问层.....13
3.3 业务逻辑层.....27
3.4 显示层.....30

1.引言

1.1 编写目的

为了保证团队编写的程序都符合相同的规范，保证一致性、统一性，以及提高代码的可度性、可维护性而建立编码文档。

文档预期的读者是整个团队的编码人员。希望团队能根据此编码规范文档，编写出可读性高、质量好的程序代码。

1.2 项目背景

随着信息技术的日益发展，部分教学支持手段的信息化已成为必然趋势。教学支持的核心部分是对教学对象的教学问题进行快速反馈。为了更详细的了解教学支持过程中各项管理业务，调研人员和最终用户进行了多次讨论，并提出了双方认可的解决方案。

将要开发的教学支持系统，主要为教学支持部门解决日常办公和项目管理的需求，协助工作人员进行日常教学支持和资料下载，提高管理效率，降低运作成本，增强企业长期竞争。通过该系统，合作院校的教学负责人员能实现对本校教师的动态管理；支持人员能随时了解讲师授课情况；院校授课讲师能随时下载教学资料等。

1.3 定义

Bussiness Logic Layer	业务逻辑层
Data Access Layer	数据访问层

1.4 参考资料

a.《第四届 ATA-微软“校园之星”大赛决赛选拔任务书（软件开发方向）》

b.《.Net 代码设计简单规范》

c.《C#程序编码规范》

2. 代码规范

2.1 文件组织规范

2.1.1 源文件

文件名要尽量做到“顾名思义”，使用有意义的名词组合来作为文件名。

使用页面文件(.aspx)和隐藏代码页文件（.aspx.cs），这有助于更好的管理和维护源文件。

每个类放在一个单独的文件中，使用类名来命名文件名。

2.1.2 目录

每个命名空间独立出一个目录，便于管理和维护。比如 WEB 文件夹 、 BLL 文件夹 、 DAL 文件夹 、 Model 文件夹，分别存放不同命名空间的内容。

根据项目的实际情况，每个“用户角色”的页面存放在各自独立的文件夹中，便于管理和维护，以及便于实现访问权限的维护。比如 Lecturer 文件夹存放“授课讲师”的页面和部分数据，ResponsiblePerson 文件夹存放“院校负责人”的页面和部分数据，等等。

将一类相关作用的文件存放在同一文件夹，比如页面的 403、404 等错误代码，可以统一地存放在 SpecialPages 文件夹内，层叠样式文件可以统一地存放 css 文件夹内。

2.2 命名规范

2.2.1 命名空间

使用简短扼要的英文缩写。比如业务逻辑层(Bussiness Logic Layer)，则将其所在的命名空间命名为 BLL ，数据访问层(Data Access Layer)，则命名为 DAL 。

2.2.2 变量

变量名主要以易于表达概念的英语进行命名，同样对命名要求达到可以顾名思义。

单词第一个字符必须小写；如果是组合单词，后续的单词应该大写，如 firstName、schoolLeaderName。

对于类的成员变量，需要定义为“_”开头，比如：

```
public class Login
{
        private string _loginid;
        private string _loginpwd;
        private string _rolename;
}
```

2.2.3 类名

主要由名词性短语命名，类的第一个单词都是大写，如果是组合单词，后续的单词应该大写。比如 Course 、 LecturerInfo 、 Login、SupporterInfo 等。注意的是类命名不能加复数形式如 Course 表示人的集合，正确命名应该为 CourseCollection 。

2.2.4 控件名

控件的命名，分为两种情况：

1.控件跟代码有联系的，要求可以达到从命名里看出这个控件的类型、控件使用的意义。比如一个提交按钮，可以命名为 btnSumbit。

2.控件跟代码没有联系，比如一些仅用于显示的控件，如果 Label 、 Table 等控件，则不要求使用上述的命名规则，采用创建时默认的命名即可。

2.2.5 接口

接口命名可以参考类命名规则，但是其需要在命名前加”I”。比如 ICommandPool、

IDownloader、IPrintManager、IProductFinder 等

2.2.6 方法

主要以动名词短语构成，函数主要是实现某一功能或者执行某一任务，因此给函数取一个合适的名字非常重要。比如 SetConfig(设置配置) 、 GetConfig(读取配置) 等等。

对于某些操作，存在多种方式实现的，在方法名后统一使用“_bY_方式”，比如对于查询操作，可以通过 ID 号查询和姓名查询，则使用 Select_bY_ID 和 Select_bY_Name 来命名。

2.2.7 属性

属性通常为类的私有成员提供读写功能，因而使用与私有成员相同的名字但去除“_”，比如：

```
public string loginID
{
    set{ _loginid=value;}
    get{return _loginid;}
}
public string loginPWD
{
    set{ _loginpwd=value;}
    get{return _loginpwd;}
}
```


2.3 缩进

合理的缩进，可以增加代码的可读性，便于团队之间的交流，也便于对错误逻辑的发现。缩进时，使用若干个制表符来进行缩进，避免使用空格。

需要缩进的情况有：方法名与其中的代码开始、逻辑判断、循环体等等，同一逻辑或者

同一层的代码，保持缩进左对齐。比如：

```
protect int CalcSumofArray(int[] arr)
{
    int sum = 0;
    if ( arr.count > 0)
    {
        for (int i=0 ; i < arr.count ; i++ )
        {
            sum += arr[i];
        }
    }
    return sum;
}
```


2.4 空白

2.4.1 空行

空行使代码的逻辑更加清晰，可以提高可读性。它们分开了那些逻辑上自身相关联的代码块。两行空格行应该用于以下之间：

一个源文件的逻辑段。

一个方法中的局部变量和它的第一条语句。

同一方法内，逻辑区分比较明显的两个代码块之间。

2.4.2 内部空格

内部空格，同样可以使得代码的逻辑容易辨认，提高可读性。特别是在一些逻辑判断上组合比较紧密的语句上，比如：

For 循环语句：　　for (int i=0 ; i < arr.count ; i++)

三目选择语句： sum = (result == true) ? 1 : 0

2.5 注释

2.5.1 块注释

块注释通常应该是被避免的。推荐使用///注释作为 C#的标准声明。如果希望用块注释时你应该用以下风格：

```
/* Line 1
* Line 2
* Line 3
*/
```

因为这样可以将注释块与代码块区分开。虽然并不提倡使用 C 风格的单行注释，但仍然可以使用。一旦用这种方式，那么在注释行后应有断行，因为很难看清在同一行中前面有注释的代码，比如：

```
/* blah blah blah */
```


块注释在极少情况下是有用的。通常块注释用于注释掉大的代码段。

2.5.2 单行注释

用//注释风格“注释掉”代码，它也可以被用于代码功能的注释部分：

注释掉的代码应该放在第一行被注释掉以使注释掉的代码更容易看清。

单行注释被用于代码说明时必须缩进到相应的编进层级，与紧跟在该条语句之后，或者紧跟在某个代码块之上的。

2.5.3 方法注释

方法的注释使用如下格式：

```
/// <summary>
```

```
/// 备份数据库
/// </summary>
/// <param name="fullname">备份的路径和文件名</param>
/// <returns>备份是否成功</returns>
```

这种方格的注释不但比较清晰，而且可以为改方法增加“智能提示”功能。当在代码中调用该方法时，将提供相应的函数功能、参数信息、返回值信息等非常有意义的提示给编程人员。提高了可读性和编码效率。

2.5.4 文件注释

文件注释只要用于标识该文件的功能、版本、编写者等信息，便于团队间的维护。特别是数据库可编程性内容，比如存储过程的.sql 文件等，建议加上文件注释，便于进行维护时，快速地找到编写者并与之进行讨论。

3. 源程序清单

下面按照三层结构来展示部分源程序：

3.1 实体层

/Model/Login.cs

```
using System;
namespace Model
{
    /// <summary>
    /// 实体类 Login 。
    /// </summary>
    [Serializable]
    public class Login
```

```
{
    public Login()
    {}
    #region Model
    private string _loginid;
    private string _loginpwd;
    private string _rolename;
    private int _status;
    /// <summary>
    /// 登陆 ID 号，主键
    /// </summary>
    public string loginID
    {
        set{ _loginid=value;}
        get{return _loginid;}
    }
    /// <summary>
    /// 登陆密码
    /// </summary>
    public string loginPWD
    {
        set{ _loginpwd=value;}
        get{return _loginpwd;}
    }
    /// <summary>
    /// 角色名
    /// </summary>
    public string roleName
    {
        set{ _rolename=value;}
```

```
            get{return _rolename;}
        }
        /// <summary>
        ///  状态
        /// </summary>
        public int status
        {
            set{ _status=value;}
            get{return _status;}
        }
        #endregion Model

    }
}
```


/Model/Login.cs

```
using System;
namespace Model
{
    /// <summary>
    ///  实体类 Course 。(属性说明自动提取数据库字段的描述信息)
    /// </summary>
    [Serializable]
    public class Course
    {
        public Course()
        {}
        #region Model
        private int _courseid;
        private string _coursename;
```

```
private int? _coursehours;
private string _courseintroduction;
private string _coursepoint;
private string _courserequire;
/// <summary>
///  课程 ID 号，主键
/// </summary>
public int courseID
{
    set{ _courseid=value;}
    get{return _courseid;}
}
/// <summary>
///  课程名
/// </summary>
public string courseName
{
    set{ _coursename=value;}
    get{return _coursename;}
}
/// <summary>
///  课程时间
/// </summary>
public int? courseHours
{
    set{ _coursehours=value;}
    get{return _coursehours;}
}
/// <summary>
///  课程简介
```

```
        /// </summary>
        public string courseIntroduction
        {
            set{ _courseintroduction=value;}
            get{return _courseintroduction;}
        }
        /// <summary>
        ///  课程知识点
        /// </summary>
        public string coursePoint
        {
            set{ _coursepoint=value;}
            get{return _coursepoint;}
        }
        /// <summary>
        ///  课程要求
        /// </summary>
        public string courseRequire
        {
            set{ _courserequire=value;}
            get{return _courserequire;}
        }
        #endregion Model
```


3.2 数据访问层

/DAL/SQLHelper.cs

```
using System;
using System.Collections.Generic;
```

```
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using Model;

/*
 * sql 助手类
 */
namespace DAL
{
    public class SQLHelper
    {
        private SqlConnection conn = null;
        private SqlCommand cmd = null;
        private SqlDataReader sdr = null;

        private readonly string connectionString =
ConfigurationManager.ConnectionStrings["connectionString"].ConnectionString;

        #region 构造函数
        /// <summary>
        /// 构造函数
        /// </summary>
        public SQLHelper()
        {
            conn = new SqlConnection(connectionString);
        }
        #endregion
```

```
#region 实例化 SqlConnection 对象
/// <summary>
/// 实例化 SqlConnection 对象
/// </summary>
/// <returns></returns>
private SqlConnection SqlConnection()
{
    if (conn.State == ConnectionState.Closed)
    {
        conn.Open();
    }
    return conn;
}
#endregion
```



```
#region 实例化 SqlCommand 对象
/// <summary>
/// 实例化 SqlCommand 对象
/// </summary>
/// <param name="sql">sql 语句或存储过程名</param>
/// <param name="type">CommandType 类型</param>
/// <param name="conn">SqlConnection 对象</param>
/// <returns></returns>
public SqlCommand SqlCommand(string sql, SqlParameter[] paras, CommandType type)
{
    if (cmd == null)
    {
        cmd = new SqlCommand();
    }
    cmd.Connection = SqlConnection();
```

```
        if (conn.State == ConnectionState.Closed)
        {
            conn.Open();
        }
        cmd.CommandType = type;
        cmd.CommandText = sql;
        return cmd;
    }
    #endregion

    #region 执行 ExecuteNonQuery ..添加，删除，修改(带参数)
    /// <summary>
    /// 执行添加，删除，修改
    /// </summary>
    /// <param name="sql">sql 语句或者存储过程名</param>
    /// <param name="paras">参数数组名</param>
    /// <param name="type">命令类型</param>
    /// <returns></returns>
    public bool ExecuteNonQuery(string cmdText, SqlParameter[] paras, CommandType ct)
    {
        cmd = new SqlCommand(cmdText, SqlConnection());
        cmd.CommandType = ct;
        //    cmd.Connection = SqlConnection();
        cmd.Parameters.AddRange(paras);
        try
        {
            cmd.ExecuteNonQuery();
        }
        catch (SqlException ex)
        {
```

```
            throw ex;
            return false;
        }
        finally
        {
            cmd.Connection.Close();
        }
        return true;
    }
    #endregion

    #region 执行 ExecuteNonQuery..添加，删除，修改
    /// <summary>
    /// 执行添加，删除，修改
    /// </summary>
    /// <param name="sql">sql 语句或者存储过程名</param>
    /// <param name="type">命令类型</param>
    /// <returns></returns>
    public bool ExecuteNonQuery(string cmdText, CommandType ct)
    {
        cmd = new SqlCommand(cmdText, SqlConnection());
        cmd.CommandType = ct;
        //    cmd.Connection = SqlConnection();
        try
        {
            cmd.ExecuteNonQuery();
        }
        catch (SqlException ex)
        {
            // throw ex;
```

```
            return false;
        }
        finally
        {
            cmd.Connection.Close();
        }
        return true;
    }
    #endregion

    #region  执行 ExecuteScalar
    /// <summary>
    ///  执行 ExecuteScalar
    /// </summary>
    /// <param name="cmdText">sql 语句或者存储过程名</param>
    /// <param name="ct">命令类型</param>
    /// <returns>返回 count</returns>
    public int ExecuteScalar(string cmdText, CommandType ct)
    {
        int count = -1;
        cmd = new SqlCommand(cmdText, SqlConnection());
        cmd.CommandType = ct;

        try
        {
            count = (int)cmd.ExecuteScalar();
        }
        catch (SqlException ex)
        {
            throw ex;
```

```
        }
        finally
        {
            cmd.Connection.Close();
        }
        return count;
    }
    #endregion

    #region  执行带参数的 ExecuteScalar
    /// <summary>
    ///  执行带参数的 ExecuteScalar
    /// </summary>
    /// <param name="cmdText">sql 语句或者存储过程名</param>
    /// <param name="paras">SqlParameter 参数数组名称</param>
    /// <param name="ct">命令类型</param>
    /// <returns>返回 count</returns>
    public int ExecuteScalar(string cmdText, SqlParameter[] paras, CommandType ct)
    {
        int count = -1;
        cmd = new SqlCommand(cmdText, SqlConnection());

        if (cmd == null)
        {
            Console.WriteLine("here");
            return -1;
        }

        cmd.CommandType = ct;
        cmd.Parameters.AddRange(paras);
```

```
            try
            {
                count = (int)cmd.ExecuteScalar();
            }
            catch (SqlException ex)
            {
                throw ex;
            }
            finally
            {
                cmd.Connection.Close();
            }
            return count;
        }
        #endregion

        #region 执行 ExecuteReader (带参数)
        /// <summary>
        ///  执行 ExecuteReader
        /// </summary>
        /// <param name="cmdText">查询 SQL 语句或存储过程</param>
        /// <param name="ct">命令类型</param>
        /// <returns></returns>
        public DataTable ExecuteReader(string cmdText, SqlParameter[] paras, CommandType
ct)
        {
            DataTable dt = new DataTable();
            cmd = new SqlCommand(cmdText, SqlConnection());
            cmd.CommandType = ct;
            cmd.Parameters.AddRange(paras);
```

```
        using (sdr = cmd.ExecuteReader(CommandBehavior.CloseConnection))
        {
            dt.Load(sdr);
        }
        return dt;
    }
    #endregion

    #region 执行 ExecuteReader
    /// <summary>
    ///  执行 ExecuteReader
    /// </summary>
    /// <param name="cmdText">查询 SQL 语句或存储过程</param>
    /// <param name="ct">命令类型</param>
    /// <returns></returns>
    public DataTable ExecuteReader(string cmdText, CommandType ct)
    {
        DataTable dt = new DataTable();
        cmd = new SqlCommand(cmdText, SqlConnection());
        cmd.CommandType = ct;
        using (sdr = cmd.ExecuteReader(CommandBehavior.CloseConnection))
        {
            dt.Load(sdr);
        }
        return dt;
    }
    #endregion

    #region 执行 SqlDataAdapter 对象的 Fill 方法 (不分页)
    /// <summary>
```

```
/// 执行 SqlDataAdapter 对象的 Fill 方法
/// </summary>
/// <param name="cmdText">sql 语句或存储过程名</param>
/// <param name="paras">SqlParameter 数组对象名</param>
/// <param name="ct">命令类型</param>
/// <returns></returns>
public DataSet Fill(string cmdText, SqlParameter[] paras, CommandType ct)
{
    cmd = new SqlCommand(cmdText, SqlConnection());
    cmd.CommandType = ct;
    cmd.Parameters.AddRange(paras);
    DataSet ds = new DataSet();
    SqlDataAdapter dp = new SqlDataAdapter();
    dp.SelectCommand = cmd;
    dp.Fill(ds);
    return ds;
}
#endregion

#region 执行 SqlDataAdapter 对象的 Fill 方法 用于分页
/// <summary>
/// 执行 SqlDataAdapter 对象的 Fill 方法 用于分页
/// </summary>
/// <param name="cmdText">sql 语句或存储过程名</param>
/// <param name="paras">SqlParameter 数组对象名</param>
/// <param name="ct">命令类型</param>
/// <param name="currentPageIndex">当前显示页的页数</param>
/// <param name="pageSize">每页要显示的最大记录数</param>
/// <returns></returns>
public DataSet Fill(string cmdText, SqlParameter[] paras, CommandType ct, int
```

```
currentPageIndex, int pageSize)
        {
            int startRecord = (currentPageIndex - 1) * pageSize;
            cmd = new SqlCommand(cmdText, SqlConnection());
            cmd.CommandType = ct;
            cmd.Parameters.AddRange(paras);
            DataSet ds = new DataSet();
            SqlDataAdapter dp = new SqlDataAdapter();
            dp.SelectCommand = cmd;
            dp.Fill(ds, startRecord, pageSize, "page");
            return ds;
        }
        #endregion

        #region 执行 SqlDataAdapter 对象的 Fill 方法 用于分页
        /// <summary>
        /// 执行 SqlDataAdapter 对象的 Fill 方法 用于分页
        /// </summary>
        /// <param name="cmdText">sql 语句或存储过程名</param>
        /// <param name="ct">命令类型</param>
        /// <param name="currentPageIndex">当前显示页的页数</param>
        /// <param name="pageSize">每页要显示的最大记录数</param>
        /// <returns></returns>
        public DataSet Fill(string cmdText, CommandType ct, int currentPageIndex, int pageSize)
        {
            int startRecord = (currentPageIndex - 1) * pageSize;
            cmd = new SqlCommand(cmdText, SqlConnection());
            cmd.CommandType = ct;
            DataSet ds = new DataSet();
            SqlDataAdapter dp = new SqlDataAdapter();
```

```
            dp.SelectCommand = cmd;
            dp.Fill(ds, startRecord, pageSize, "page");
            return ds;
        }
        #endregion

    }
}
```

/DAL/CourseDAO.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using Model;

namespace DAL
{
    public class CourseDAO
    {
        private SQLHelper sqlHelper;

        #region 构造函数
        public CourseDAO()
        {
            sqlHelper = new SQLHelper();
        }
        #endregion
```

```
        #region 根据课程 ID 删除课程
        /// <summary>
        /// 根据课程 ID 删除课程
        /// </summary>
        /// <param name="courseID">课程 ID</param>
        /// <returns></returns>
        public bool DeleteCourse(int courseID)
        {
            SqlParameter[] paras = new SqlParameter[]{
                new SqlParameter("@courseID",courseID)
            };
            return sqlHelper.ExecuteNonQuery("Course_delete", paras,
CommandType.StoredProcedure);
        }
        #endregion

        #region 查询所有课程信息
        /// <summary>
        /// 查询所有课程信息
        /// </summary>
        /// <returns></returns>
        public DataTable GetAllCourse()
        {
            return sqlHelper.ExecuteReader("Course_GetAllCourse",
CommandType.StoredProcedure);
        }
        #endregion

        #region 查询所有课程信息（分页显示）
        /// <summary>
```

```
        /// 查询所有课程信息（分页显示）
        /// </summary>
        /// <returns></returns>
        public DataSet GetAllCourse(int currentPageIndex, int pageSize)
        {
            return sqlHelper.Fill("Course_GetAllCourse", CommandType.StoredProcedure,
currentPageIndex, pageSize);
        }
        #endregion

        #region 添加课程
        /// <summary>
        /// 添加课程
        /// </summary>
        /// <param name="course"></param>
        /// <returns></returns>
        public bool AddCourse(Model.Course course)
        {
            SqlParameter[] paras = new SqlParameter[]{
                new SqlParameter("@courseName",course.courseName),
                new SqlParameter("@courseHours",course.courseHours),
                new SqlParameter("@courseIntroduction",course.courseIntroduction),
                new SqlParameter("@coursePoint",course.coursePoint),
                new SqlParameter("@courseRequire",course.courseRequire)
            };
            return sqlHelper.ExecuteNonQuery("Course_insert", paras,
CommandType.StoredProcedure);
        }
        #endregion
```

```
        #region  修改课程信息
        /// <summary>
        ///  修改课程信息
        /// </summary>
        /// <param name="course"></param>
        /// <returns></returns>
        public bool UpdateCourse(Model.Course course)
        {
            SqlParameter[] paras = new SqlParameter[]{
                new SqlParameter("@courseID",course.courseID),
                new SqlParameter("@courseName",course.courseName),
                new SqlParameter("@courseHours",course.courseHours),
                new SqlParameter("@courseIntroduction",course.courseIntroduction),
                new SqlParameter("@coursePoint",course.coursePoint),
                new SqlParameter("@courseRequire",course.courseRequire)
            };
            return sqlHelper.ExecuteNonQuery("Course_update", paras,
CommandType.StoredProcedure);
        }
        #endregion
    }
}
```


3.3 业务逻辑层

/BLL/CourseManage.cs

```
using System;
using System.Collections.Generic;
using System.Text;
```

```
using System.Data;
using Model;
using DAL;

namespace BLL
{
    public class CourseManage
    {
        #region 根据课程 ID 删除课程
        /// <summary>
        /// 根据课程 ID 删除课程
        /// </summary>
        /// <param name="courseID">课程 ID</param>
        /// <returns></returns>
        public static bool DeleteCourse(int courseID)
        {
            return new DAL.CourseDAO().DeleteCourse(courseID);
        }
        #endregion

        #region 查询所有课程信息
        /// <summary>
        /// 查询所有课程信息
        /// </summary>
        /// <returns></returns>
        public static DataTable GetAllCourse()
        {
            return new DAL.CourseDAO().GetAllCourse();
        }
        #endregion
```

```
#region 查询所有课程信息（分页显示）
/// <summary>
/// 查询所有课程信息（分页显示）
/// </summary>
/// <returns></returns>
public static DataSet GetAllCourse(int currentPageIndex, int pageSize)
{
    return new DAL.CourseDAO().GetAllCourse(currentPageIndex, pageSize);
}
#endregion

#region 添加课程
/// <summary>
/// 添加课程
/// </summary>
/// <param name="course"></param>
/// <returns></returns>
public static bool AddCourse(Model.Course course)
{
    return new DAL.CourseDAO().AddCourse(course);
}
#endregion

#region 修改课程信息
/// <summary>
/// 修改课程信息
/// </summary>
/// <param name="course"></param>
/// <returns></returns>
```

```
        public static bool UpdateCourse(Model.Course course)
        {
            return new DAL.CourseDAO().UpdateCourse(course);
        }
        #endregion
    }
}
```

3.4 显示层

/Web/Default.aspx.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
    #region 维护人员登陆信息
    private static string UpholdPersonName = "adminperson";
    private static string UpholdPersonPWD = "login";
    #endregion

    protected void IButtonLogin_Click(object sender, ImageClickEventArgs e)
    {
        //点击登陆
        #region 检查验证码

        #region 验证码是否过期
        if (Session["Code"] == null)
        {
            ourUI.Alert(@"抱歉。你在页面停留的时间过久，验证码已经过期。\n 请重新输入。", Page);
            return;
        }
```

```
#endregion

string rightCode = Session["Code"].ToString().ToUpper();
string code = txtCode.Text.Trim().ToUpper();
if (rightCode != code)
{

    txtPWD.Text = "";
    txtCode.Text = "";
    ourUI.Alert(@"验证码错误!\n 请重新输入。", Page);
    return;
}
#endregion

#region  检查 ID 对应的密码
string LoginID = txtID.Text.Trim();
string pwd = ourSecurity.String2MD5(txtPWD.Text.Trim());

#region  首先验证是否是维护人员
if (LoginID == UpholdPersonName && txtPWD.Text.Trim() == UpholdPersonPWD)
{
    ourSecurity.Create(UpholdPersonName, "UpholdPerson");
    Response.Redirect("~/UpholdPerson/Default.aspx");
    return;
}
#endregion


//判断 ID 号是否存在
if (!BLL.LoginManage.LoginID_IsExist(LoginID))
{
    ourUI.Alert(@"抱歉。你所填写的 ID 号不存在。\n 请检查后重新输入。",Page);
    return;
}

//获取 ID 号对应的密码
System.Data.DataTable dt =    BLL.LoginManage.GetloginInfoByID(LoginID);
try
{
    if (dt.Rows[0]["loginPWD"].ToString().Trim() == pwd)
    {
        string role = dt.Rows[0]["roleName"].ToString().Trim();

        ourSecurity.Create(LoginID, role);
```

```
                    Response.Redirect("~/" + role + "/Default.aspx");

                }
                else
                {
                    ourUI.Alert(@"抱歉。你所填写的密码不正确。\n 请重新输入。", Page);
                }
            }
            catch { }

            #endregion
        }
}
```