.Net 编码规范

2018 年 11 月 1 日发布

云学堂网络科技有限公司

版本历史

修订日期	主要修订专家	主要评审专家	版本号	修订情况
2018-10-31	李广志 101001	王润念、史振兴、向明杰、 李广志、张斐、姜亚正、王 仪兴、陆英	V1.0	新建

目 录 Table of Contents

1 引言.....4
1.1 编写目的.....4
1.2 适用对象.....4
1.3 引用术语及缩写解释.....4
1.4 书写约定.....4
2 编码规范.....5
2.1 命名规范.....5
2.1.1 命名原则.....5
2.1.2 命名通用要求.....6
2.1.3 常用词.....7
2.2 注释规范.....9
2.2.1 注释原则.....9
2.2.2 注释通用要求.....9
2.2.3 平台级代码注释要求.....10
2.2.4 注意事项.....10
2.3 排版规范.....12
2.4 安全规范.....13
2.4.1 通用要求.....13
2.4.2 防跨站脚本攻击`.....13
3 .NET 相关.....14

3.1 网站目录结构.....14
3.1.1 业务类或契约类（OceanSoft.Itsp.Contracts）.....14
3.1.2 数据访问层（OceanSoft.Itsp.Data）（旧框架使用）.....15
3.1.3 业务逻辑层（OceanSoft.Itsp.Business）.....16
3.1.3.1 旧框架说明.....16
3.1.3.2 新框架—REST Api 规范.....18
3.1.4 UI 界面层（OceanSoft.eLearning.WebUI）.....22
3.2 控件说明.....26
3.3 公共方法.....28
4 其他规范.....28
4.1 异常处理.....28
4.2 多语言.....29
4.2.1 多语言文件定义及命名.....29
4.2.2 资源 KEY 的命名规则.....30
4.2.3 C#中多语言.....30
4.2.4 JS 中多语言.....30
4.2.5 HTML 中多语言.....30
4.2.6 多语言日期.....31
5 API 使用规范.....31
5.1 ResfulApiService 的实现.....31
5.2 Service 调用 API.....31

1 引言

1.1 编写目的

本文档是项目的开发规范，便于降低项目的维护成本以及新员工入职的编码约束。

1.2 适用对象

本规范的读者及使用对象主要为 Web 相关的编码.NET 开发人员。

1.3 引用术语及缩写解释

术语/缩写	解释
Pascal 风格	大写每个单词的首字母，如：BackColor
camel 风格	除了第一个单词的首字母,其他单词都应大写首字母，如：backColor

1.4 书写约定

- ::=，表示：定义为
- < >，尖括号中的内容表示为必填项
- [ ]，方括号中的内容表示可选项
- |，竖划线表示多项中选择一项
- 名称域之间用下划线(_)连接
- 不同域之间用减号（-）连接

2 编码规范

2.1 命名规范

2.1.1 命名原则

1. **使用正确的大小写风格；**
   1) 命名空间、类、方法、属性、类公开字段、枚举使用 **Pascal** 风格（大写每个单词的首字母，如：BackColor）；
   2) 在局部变量参数名或类私有字段上使用 **camel** 风格（除了第一个单词的首字母，其他单词都应大写首字母，如：backColor）。
2. **尽量避免在名称中使用数字；**
3. **尽量使用常用词；**

   参见：常用词
4. **尽量使用计算机专业术语；**

   尽量使用约定成俗的惯用语、计算机科学术语、算法名称、设计模式名称、数学名词等软件编程相关名词，例如：运用工厂模式类的命名应该是“名词+Factory”；
5. **同义词中使用固定的单词；**

   例如：摘要，有 Excerpt、Summary 等同义词，应统一使用其中一个；
6. **名词尽量简洁；**

   仅在语境中才有意义的名词，并且去除语境说明不会引起歧义，此时建议不要再包含语境说明，例如： book.BookTitle 可简写为 book.Title；
7. **实体类属性尽量与数据库字段保持一致；**

8. 缩写：

1) 常用控件缩写参考 3.2；

2) 只有绝对需要时再使用广为人知的缩写。可以使用 UI 来表示 User Interface；

3) 当缩写被组合使用时，对于缩写请使用 Pascal 风格或 camel 风格，例如，使用 HtmlButton 或 htmlButton。

9. 选择的词语词性要准确。

类名、对象名、字段、属性、参数、枚举应使用名词，方法应使用动词，接口可使用形容词或名词。

2.1.2 命名通用要求

描述符	大小写风格	说明	示例
命名空间	Pascal	命名空间的作用是对类进行组织，一般来说需要使用公司名称、项目名称、模块名称、技术应用范畴来作为命名空间。 命名规则：**<公司名/项目名>.<技术名称/模块名称>[.<特性>]。** 1. 第二层名称必须是稳定被广泛认可的； 2. 不要使用组织的结构层次作为命名空间的层次依据； 3. 不要让命名空间和类都使用同样的名字； 4. 命名空间结构不需和项目目录结构并行； 5. 命名空间建议不要超过三级，以两级为宜； 6. 应用模块的命名空间二级即可	Test.Caching
类	Pascal	1. 使用名词或名词短语命名类；	TagRepository

		2. 不要使用类型前缀； 3. 不要使用下划线字符（_）； 4. 慎用缩写； 5. 建议**使用组合词来命名派生类**，考虑使用基类名称作为派生类名称的结尾，例如：TagRepository 是合适的名称，它继承了 Repository；	
方法	Pascal	1. 使用动词或动词短语来命名方法； 2. 形参顺序按照其重要程度、稳定性由高到低依次排序；	RemoveAll
参数、类私有字段	camel	1. 必须有意义； 2. 必须清楚地表现出其作用；	topMember
属性、类公开字段	Pascal	1. 使用动词或动词短语来命名； 2. 类型是枚举、类时，建议命名和类型一致；	Title
枚举类型	Pascal	**使用单数形式来命名枚举**	ErrorLevel

2.1.3 常用词

1. 类名

名称	说明
[EntityName]Biz	业务逻辑类名
[EntityName]Service	服务类名称
[EntityName]	实体类名
[EntityName]View	用于表单呈现EntityName的实体

2. 方法名

名称	说明

Create	创建实体
Update	更新实体
Update<PropertyName>	更新单个实体的属性值
BatchUpdate<PropertyName>	批量设置多个实体的属性值
Delete	删除实体
Delete<EntityName(复数)>	删除实体
Get	获取单个实体
GetTop[EntityName(复数)]	获取前几个实体，用于获取排行数据
Get[EntityName(复数)]	获取实体分页集合
Clear<EntityName(复数)>	清空

3. 字段、属性常用词：参见《数据库设计说明书》的“常用字段”即可；

4. 参数

名称	说明
topNumber	前几条
sortBy	排序依据
pageIndex	页码

5. 枚举名

名称	说明
[EntityName]Type	实体类型
[EntityName]Status	实体状态
SortBy[EntityName]	实体排序依据

2.2 注释规范

2.2.1 注释原则

1. 注释的详细程度跟源代码的使用范围及使用频繁程度相关，注释的优先级为：

   1) 平台级代码>应用级代码；

   2) 接口>实现类；

   3) 工具类>业务逻辑类>仓储类>实体类&查询条件类>表现层（Controller、HttpHandler）；

2. 注释语言必须准确、易懂、简洁。

2.2.2 注释通用要求

注释内容	说明	注释要求	备注
源文件注释	文件头注释的主要是指对 C#文件进行的文件级的注释，作用是用来记录文件的变更历史，通过该注释团队成员就会很清楚该文件是谁在创建和维护，如果在使用该文件遇到问题需要和谁去沟通。一般在文件注释中需要保留以下信息： ➢ 文件当前版本号 ➢ 最初创建人 ➢ 最初创建日期 ➢ 版本变更历史：谁在什么时间曾经做过什么改动	必须注释	由于经常需要把源文件发 给客户，但是又不希望客户 到每个文件的详细变更历史 因此在写文件注释时，需要 文件注释采 //<Copyright> //</Copyright>包起来， 便于对外发布时可以把文 注释区域替换成统一的信息
类注释	描述类的功能	必须注释	
类方法注释	描述方法的功能，对重点参数或不易理解的参数进行注释	公开方法必须注释	
类属性注释	描述属性的功能	公开属性必须注释	
事件注释		必须注释	
委托注释		必须注释	
枚举注释	枚举类型及枚举项都必须注释	必须注释	
View 注释	注释格式为：<!— 注释内容[Begin|End] --> 其中[Begin|End]表示注释区域标签时，在区域标签开始之前和结束之后以“Begin”或“End”为结尾，表示区域注释的开始或结束；	内容块、布局文件中的各个区域标签必须加注释	

2.2.3 平台级代码注释要求

1. 类型（含类、接口、枚举、事件、委托）描述
   1) <summary>（必填）；
   2) <remark>（可选）：有特殊情况需要说明时必须添加；
2. 方法（适用于 public、Protected，private 暂不做要求）
   1) <summary>(必填)；
   2) <remark>（可选）：有特殊情况需要说明时必须添加；
   3) <typeparam>（可选）：类型参数有限制（例如，有明显的 where 条件）或需要特殊说明是必须填写；
   4) <param>（必填）；
   5) <returns>（可选）：如返回是 bool、字典等需要进行说明的类型时必须填写；
3. 属性
   1) <summary>(必填)；
4. 平台常用实例如下：

```
/// <summary>
/// 查询我的学分
/// 创建人：xxx
/// 时间：2012年3月30日20:32:19
/// 修改人：xxx
/// 修改时间：2012年08月30日20：05：08
/// 修改原因：将列表改成相对应的标签显示
/// </summary>
```

2.2.4 注意事项

1. 源文件头部注释

```
//<Copyright>
//------------------------------------------------------------
//<version>[文件版本号]</verion>
//<createdate>[创建日期]</createdate>
//<author>[作者姓名]</author>
//<email>[作者Email]</email>
//<log date="[变更历史创建日期]" version="[文件版本号]" author="[变更人]">[变更原因]</log>
//------------------------------------------------------------
//</Copyright>
```

例如：

```
//<Copyright>
//------------------------------------------------------------
//<version>V0.6</verion>
//<createdate>2011-11-16</createdate>
//<author>test</author>
//<email>test@test.com</email>
//<log date="2011-11-16" version="0.5">创建</log>
//<log date="2011-11-18" version="0.6" author="libsh">增加对缓存的支持</log>
//------------------------------------------------------------
//</Copyright>
```

2. 在注释中使用引用提高根据注释生成的类库文档的可读性，例如：

```
        private UserRankTypes userRankType;
        /// <summary>
        /// 用户等级名称体系<seealso cref="Test.Components.UserRankType"/>
        /// </summary>
        public UserRankTypes RankType
        {
            get { return this.userRankType; }
            set { this.userRankType = value; }
        }
```

2.3 排版规范

1. Visual Studio 必须使用统一的格式设置文件，其需要在 vs 默认设置的基础上，做以下修改：

   1) 所有语言/常规中的显示部分，选中“行号”；

   2) 所有语言/制表符中，修改以下设置项：

      a) 缩进：块

      b) 制表符：

         i. 制表符大小：4

         ii. 缩进大小：4

         iii. 选中“插入空格”

   3) Html/格式设置中的标记换行部分，取消勾选“超出指定长度时对标记执行换行”

   目前已导出 vs2010.vssettings，直接导入即可。详细说明见：《vs2010 格式设置与导入指南》

2. 在类文件中，使用外侧代码（Region）对相似内容的代码组织在一起，方便阅读，例如：在 Service 类中，使用“Create、Update&Delete”组织写操作方法，使用“Get & Gets”组织读操作方法。

2.4 安全规范

2.4.1 通用要求

1. 对用户录入信息进行处理，例如：

   1) 注意清除不必要的前后空格；

2) 清理敏感词；

3) 清理 JavaScript，特殊功能（例如广告）需要进行特殊处理；

4) 允许输入 html 时应该清理不允许的 html 标签及属性；

5) 不允许输入 html 时的多行文本应该考虑保持换行及空格的格式；

2. 严格控制“记录级安全”，比如删除某条记录时，必须判断用户具有删除该条记录的权限；

3. 所有发起对数据库直接修改的 http 请求不允许使用 get 方式；

2.4.2 防跨站脚本攻击`

1. 对于任何需要在页面中显示并是来自于客户端的数据，一律进行特殊字符处理，比如防止 js 注入，界面本身不是服务器控件提交时。

   后端可能出现注入的字段加

   CommonWebUtil.ReplaceKeyWord(string inputString)

   读出时界面展示如果渲染不了的加

   CommonWebUtil.DecodeKeyWord(string inputString)

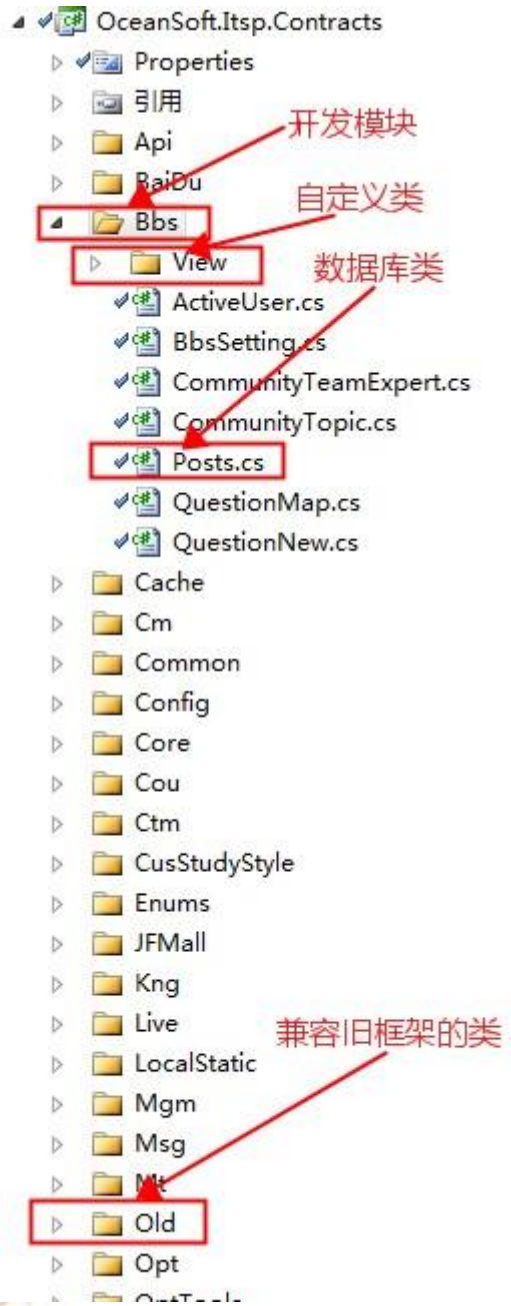
2. 对应的 js 方法 同名的有一套；

3 .NET 相关

3.1 网站目录结构

3.1.1 业务类或契约类（OceanSoft.Itsp.Contracts）

1. 用于存放整个项目中的所有数据类，原则上是与数据库中的表及视图一一对应的，当然也可以根据实际需要自定义一些数据类，这些类在整个项目中都可以被用到，原则上不可添加的自定义数据类，特殊情况需和项目经理商量。
2. 具体包含内容如下：



3.1.2 数据访问层(OceanSoft.Itsp.Data)(旧框架使用)

1. 每个模块原则上在数据访问层对应产生 2 个类 4 个文件
2. 类***ExcuteDao 用来处理新增/删除/更新相关操作，继承 ExecuteDaoBase
3. 类***QueryDao 用来处理查询相关操作，继承 QueryDaoBase
4. 数据访问层原则上不允许直接编写 SQL 语句，如现提供底层不能支持请确认

3.1.3 业务逻辑层（OceanSoft.Itsp.Business）

3.1.3.1 旧框架说明

1. 每个模块原则上在数据访问层对应产生 2 个类 4 个文件
2. 数据的处理
3. 当执行非查询操作时，必须使用 try …… catch……，且必须写日志
4. 业务层的一些公共方法可查看文档“公共方法使用说明.docx”。

```
/// <summary>
/// 新增知识目录权限设置表
/// </summary>
/// <param name="entity">实体对象</param>
/// <param name="context">用户上下文对象</param>
/// <returns>影响行数</returns>
public static int CreateCatalogPermission(Entity entity, MyContext context)
{
    int count = 0;
    try
    {
        entity.CreateDate = DateTime.Now;
        entity.CreateUserID = context.UserID;
        entity.CreateUserName = context.CnName;
        entity.UpdateDate = DateTime.Now;
        entity.UpdateUserID = context.UserID;
        entity.UpdateUserName = context.CnName;

        //创建DAO对象
        ExcuteDao excuteDao = DaoCreater.CreateDao<ExcuteDao>();

        count= excuteDao.Insert(entity);

        BizUtil.WriteOperateLog(context, "知识目录权限设置表", entity, LogCategory.Create);
    }
    catch (Exception ex)
    {
        BizUtil.HandleException(ex, context);
    }
    return count;
}
```

新增操作

```
/// <summary>
/// 更新知识目录权限设置表
/// </summary>
/// <param name="entity">实体对象</param>
/// <param name="context">用户上下文对象</param>
/// <returns>影响行数</returns>
public static int UpdateCatalogPermission(Entity entity, MyContext context)
{
    int count = 0;
    try
    {
        entity.UpdateDate = DateTime.Now;
        entity.UpdateUserID = context.UserID;
        entity.UpdateUserName = context.CnName;

        //创建DAO对象
        ExcuteDao excuteDao = DaoCreater.CreateDao<ExcuteDao>();

        count = excuteDao.Update(entity);

        BizUtil.WriteOperateLog(context, "知识目录权限设置表", entity, LogCategory.Update);
    }
    catch (Exception ex)
    {
        BizUtil.HandleException(ex, context);
    }
    return count;
}
```

更新操作

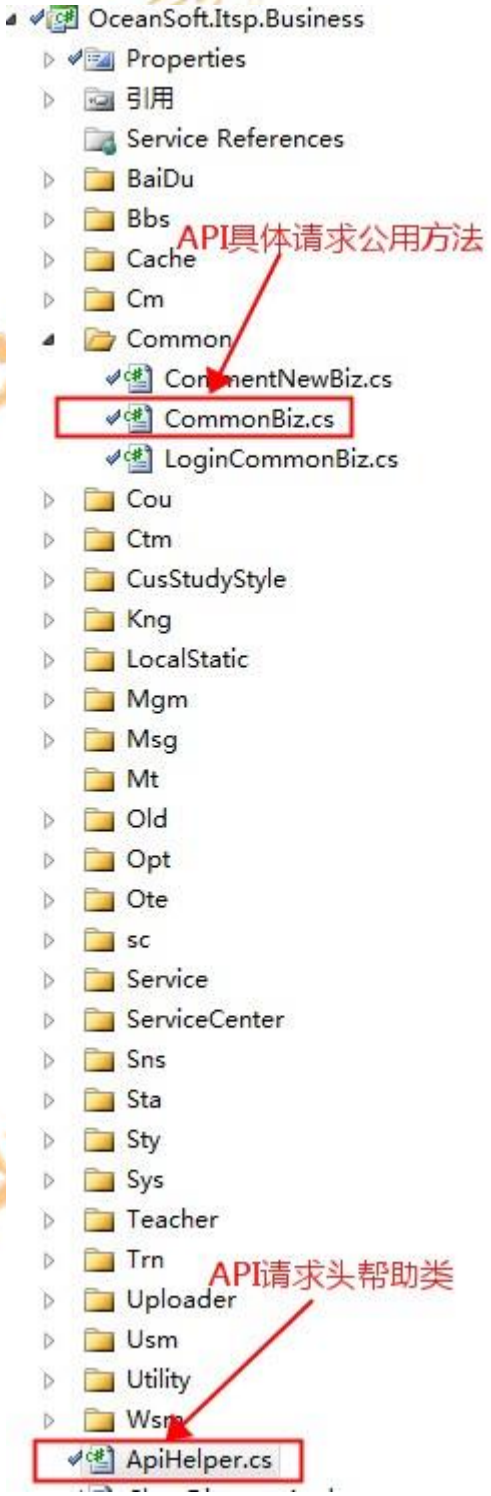
```
/// <summary>
/// 根据主键删除知识目录权限设置表
/// </summary>
/// <param name="id">主键</param>
/// <param name="context">用户上下文对象</param>
/// <returns>影响行数</returns>
public static int DeleteCatalogPermission(string id, MyContext context)
{
    int count = 0;
    try
    {
        //创建DAO对象
        ExcuteDao excuteDao = DaoCreater.CreateDao<ExcuteDao>();

        Entity entity = excuteDao.Get<Entity>(id);

        if (entity != null)
        {
            count = excuteDao.Delete<Entity>(id);

            BizUtil.WriteOperateLog(context, "知识目录权限设置表", entity, LogCategory.Delete);
        }
    }
    catch (Exception ex)
    {
        BizUtil.HandleException(ex, context);
    }
    return count;
}
```

删除操作

5. 对于业务层和静态缓存相关的功能需在执行操作时更新对象时缓存信息。

3.1.3.2 新框架—REST Api 规范

1. 新的框架采用的是 REST Api 规范，具体需要了解已封装的 ApiHelper.cs 与 CommonBiz.cs



2. ApiHelper 类主要声明 API 请求头的参数，如 token 即令牌，基于 token 身份验证；
_timeOut
即每个请求过期时间，单位秒等等，具体如图所示

```
public class ApiHelper
{
    public IApiServcie api = null;
    //Token 的身份验证
    private string token;
    //对接CRM 确认固定KEY
    private string key;
    private string sourceType;
    //自定义参数键值对
    private Dictionary<string, string> dicHeader;
    /// <summary>
    /// ApiHelper声明
    /// </summary>
    /// <param name="token">token</param>
    public ApiHelper(string token)
    {
        this.api = new RestfulApiService();
        this.token = token;
        this.sourceType = string.Empty;
        this.api.ExcutingRequest += new Action<string, string, System.Net.HttpWebRequest,
        this.api.Error += new Action<string, System.Net.HttpWebResponse, Exception>(api_Er
        this.api.ExcutedRequest += new Action<string, string, object, System.Net.HttpWebRe
    }
    /// <summary> ...
    public ApiHelper(string token, int _timeOut)...
    /// <summary> ...
    public ApiHelper(string token, string masterUrl)...
    /// <summary> ...
    public ApiHelper(string key, bool isKey, string masterUrl)...
    /// <summary> ...
    public ApiHelper(string key, bool isKey)...
    /// <summary> ...
    public ApiHelper(Dictionary<string, string> dicHeader)...
    /// <summary> ...
    public ApiHelper(Dictionary<string, string> dicHeader, string masterUrl)...

    private void api_ExcutedRequest(string method, string action, object resData, System.Ne

    private void api_Error(string action, System.Net.HttpWebResponse arg2, Exception arg3)

    private void api_ExcutingRequest(string method, string aciton, System.Net.HttpWebReques
    {
        if (req != null)
        {
            req.Headers.Add("source", "501");
            if (!string.IsNullOrEmpty(token))
```

3. CommonBiz.cs 包括所有 REST API 开发涵盖的公用请求方法，如 Get，Post，Put，Delete; 所有后端请求请使用如下图中方法，特例除外。

```
#region << 基础公用的方法 >>

/// <summary> ...
public static T GetInfo<T>(MyContext context, string url, string token)...

/// <summary> ...
public static T GetInfo<T>(MyContext context, string url, string token, out string msg)...

/// <summary> ...
public static bool PutUrlBase(MyContext context, string token, string url, out string msg)...

/// <summary> ...
public static bool UpdateBase<T>(MyContext context, T BaseInfo, string token, string url, out string msg)...

/// <summary> ...
public static bool AddBaseInfo<T>(MyContext context, T BaseInfo, string token, string url, out string msg)...

/// <summary> ...
public static bool AddBaseInfoReturnAll<T>(MyContext context, T BaseInfo, string token, string url, out string m

/// <summary> ...
public static bool DeleteBase(MyContext context, string token, string url, out string msg)...

/// <summary> ...
public static bool BatchDeleta<T>(MyContext context, T BaseInfo, string token, string url, out string msg)...

#endregion
```

3.1 Get 方式，一种不带返回错误信息，另一种带返回错误信息，最后还有一个分页列表（GetDataPage）

其中尤其注意错误返回（多语言处理方法—BizUtil.GetApiResultValue）

```
623         /// <summary> ...
633         public static T GetInfo<T>(MyContext context, string url, string token)...
651
652         /// <summary>
653         /// 根据URL获取信息
654         /// 创建人：向明杰
655         /// 修改人：李振华
656         /// 创建时间：2016-6-14 13:44:35
657         /// </summary>
658         /// <param name="context">上下文</param>
659         /// <param name="Id">id</param>
660         /// <param name="url">url</param>
661         /// <param name="token">身份</param>
662         /// <param name="msg">错误信息</param>
663         /// <returns></returns>
664         public static T GetInfo<T>(MyContext context, string url, string token, out string msg
665         {
666             var apiHelper = new ApiHelper(token);
667             msg = string.Empty;
668             try
669             {
670                 url = string.Format("{0}", url);
671                 ResponseHttpData<T> results = apiHelper.api.Get<T>(url);//请求保存配置
672                 if (results.IsSuccessStatusCode)
673                 {
674                     return results.content;
675                 }
676                 msg = BizUtil.GetApiResultValue(results.key, context);
677                 return default(T);
678             }
679             catch
680             {
681                 return default(T);
682             }
683         }
```

声明ApiHelper

如果请求成功，返回对象

如果请求失败，则返回对应的错误Key，通过Key获取特定语言对应的错误信息

3.2 Put 方式，一种不带请求体的，另一种带请求体的

```
685         /// <summary>
686         /// 根据url更新信息（无数据源）
687         /// 创建人：向明杰
688         /// 创建时间：2016-6-2 15:31:03
689         /// </summary>
690         /// <param name="context">上下文</param>
691         /// <param name="token"></param>
692         /// <param name="url">url</param>
693         /// <param name="msg">错误提示</param>
694         /// <returns>是否成功</returns>
695         public static bool PutUrlBase(MyContext context, string token, string url, out string msg)...
714
715         /// <summary>
716         /// 更新信息（有请求体）
717         /// 创建人：向明杰
718         /// 创建时间：2016-6-2 15:31:03
719         /// </summary>
720         /// <param name="context">上下文</param>
721         /// <param name="BaseInfo">数据源</param>
722         /// <param name="token"></param>
723         /// <param name="url">url</param>
724         /// <param name="msg">错误提示</param>
725         /// <returns>是否成功</returns>
726         public static bool UpdateBase<T>(MyContext context, T BaseInfo, string token, string url, out str
745
```

3.3 Post 方式，一种不带返回体，一种带返回体

```
746         /// <summary>
747         /// 创建数数据(不返回ID时使用)
748         /// 创建人：向明杰
749         /// 创建时间：2016-6-15 10:43:24
750         /// </summary>
751         /// <param name="context">上下文</param>
752         /// <param name="BaseInfo">数据源</param>
753         /// <param name="token"></param>
754         /// <param name="url">api地址</param>
755         /// <param name="msg">错误提示</param>
756         /// <returns></returns>
757         public static bool AddBaseInfo<T>(MyContext context, T BaseInfo, string token, string url, out string msg)
776
777         /// <summary>
778         /// 创建数数据(返回所有数据)
779         /// 创建人：向明杰
780         /// 创建时间：2016-6-15 10:43:24
781         /// </summary>
782         /// <param name="context">上下文</param>
783         /// <param name="BaseInfo">数据源</param>
784         /// <param name="token"></param>
785         /// <param name="url">api地址</param>
786         /// <param name="msg">错误提示</param>
787         /// <returns></returns>
788         public static bool AddBaseInfoReturnAll<T>(MyContext context, T BaseInfo, string token, string url, out st
808
```

3.4 Delete 方式，一种不带请求体的，另一种带请求体的

```
809         /// <summary>
810         /// 删除数据
811         /// 创建人：李振华
812         /// 创建时间：2016-9-21 15:35:02
813         /// </summary>
814         /// <param name="context">上下文</param>
815         /// <param name="token"></param>
816         /// <param name="url">url</param>
817         /// <param name="msg">错误提示</param>
818         /// <returns>是否成功</returns>
819         public static bool DeleteBase(MyContext context, string token, string url, out string msg)...
838
839         /// <summary>
840         /// 批量删除（带请求体）
841         /// 创建人：张士阳
842         /// 创建时间：2017年6月5日15:11:04
843         /// </summary>
844         /// <param name="context">上下文</param>
845         /// <param name="BaseInfo">数据源</param>
846         /// <param name="token"></param>
847         /// <param name="url">url</param>
848         /// <param name="msg">错误提示</param>
849         /// <returns>是否成功</returns>
850         public static bool BatchDeleta<T>(MyContext context, T BaseInfo, string token, string url, out st
869
```

3.1.4 UI 界面层（OceanSoft.eLearning.WebUI）

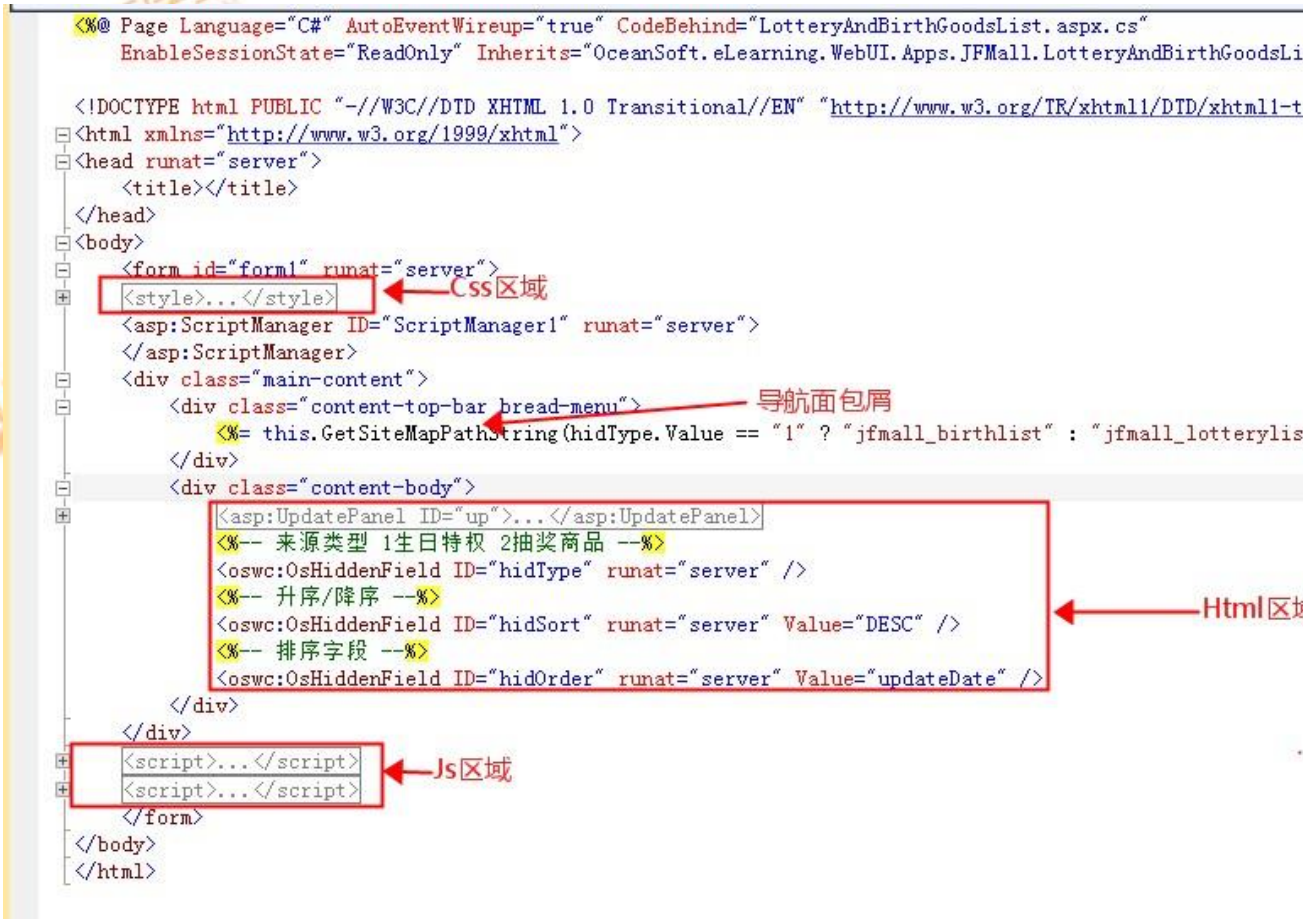
1. 用于存放整个项目中的所有数据类，原则上是与数据库中的表及视图一一对应的，当然也可以根据实际需要自定义一些数据类，这些类在整个项目中都可以被用到，原则上不可添加的自定义数据类，特殊情况需和项目经理商量
2. UI 界面层不能直接访问服务层和业务层，而是通过业务层访问；
3. 如果界面层呈现的数据量比较大并使用了局部刷新的话，需在界面非首次加载时对能影响数据呈现的控件注册客户端单击事件 ShowSearchingMsg；并在绑定数据完成后执行事件 CloseDealingMsg；

4. 所有的直接或间接继承 CommonPagebase 的界面都重写 SetPageCode 方法来设置界面编号（弹开界面可不设置）,界面编号是在后台添加导航菜单时设定，PageCode 的命名规范：功能名称小写；

   如果界面不用进行权限控制，则重写 IsCheckPermission 方法，并且固定返回 true

   如果界面可匿名访问，则重写方法 IsAllowAnonymous 方法，并且固定返回 false；

5. 如果操作的弹开界面为新增/编辑操作，界面包括四个按钮（保存&关闭；保存；新增；关闭），点击保存或新增不关闭当前弹开界面；如果操作的弹开界面知识一般的选择值，非左右“移除”“添加”操作的，点击保存直接关闭，否则不关闭当前弹开界面；
6. 弹开界面的执行关闭操作的按钮名称只可为“关闭”；跳转的返回操作的按钮名称只可为“返回”。
7. 界面的有效性验证（注意，描述结尾需要统一为中文句号，注意多语言提醒）：
   - 非空验证：请输入 XXX。
   - 选择验证：请选择 XXX。
   - 唯一性验证：XXX 已存在。
   - 文件类型：文件类型不是*.XXX 格式。
   - 文件大小：文件大小超过 XXX，请处理后再上传。
   - 格式不正确：XXX 格式不正确。
   - 长度验证：XXX 的长度不能超过 YYYY 个字。
   - 范围验证：您输入的 XXX 超过最大(小)分值，请重新输入。
   - 前台验证发现控件出现验证错误提示后应该让该控件获得焦点。如果不是同一选项卡则需要切换到相应选项卡。
8. 前端页面遵循 CSS 前置 JS 后置原则，如图所示



- 在 Dialog 页面默认引用

  <%= GlobalConfig.UIConfig.ReferenceDefaultJsAndCss2015%>；
- 导航面包屑使用到方法 GetSiteMapPathByParentPageCode；
- 页面根据 Pagecode 跳转方法

  <%=GetPageVirtualUrlByPageCode("jfmall_lotteryinfo")%>

9. 后端页面总体结构，包括页面属性、页面加载、控件事件、辅助方法，如下图

```
namespace OceanSott.eLearning.WebUI.Apps.JfMall
{
    /// <summary>
    /// 生日特权商品/积分抽奖商品 添加/编辑
    /// 创建人：李振华
    /// 创建时间：2017-11-7 13:25:01
    /// pagecode: 生日特权商品 jfmall_birthinfo  积分抽奖商品 jfmall_
    /// </summary>
    public partial class LotteryAndBirthGoodsInfo : CommonPagebase
    {
        << 页面属性 >>

        << 页面加载 >>

        << 控件事件 >>

        << 辅助方法 >>

    }
}
```

- 页面中所有的增加删除修改等主要业务都需要增加行为日志 BehaviorLog，如下图所示

```
154                 bool res = CommonAPIBiz.AddBaseInfoReturnAll(MyContext, entity, MyProfile.Toke
155                 if (res)
156                 {                                          具体业务处理成功之后增加行为日志
157                     #region << 行为日志写入 >>
158
159                     ///记录行为日志
160                     ///创建人：李振华
161                     ///创建时间：2017-11-22 10:30:42
162                     BehaviorLog log = new BehaviorLog(CommonWebUtil.GetSessionID());
163                     log.logtitle = hidType.Value == "1" ? "生日特权新增商品" : "积分抽奖新增商
164                     log.sex = CommonWebUtil.GetSessionID();
165                     log.method = LogMethod.增;
166                     log.target = GetPageFeaturesNum().Substring(0, 9) + "001";
167                     log.subtarget = "";
168                     log.attribute = entity.id;
169                     log.description = LogDescription.Single.ToString();
170                     log.logcontent = hidType.Value == "1" ? "生日特权新增商品" : "积分抽奖新增
171                     log.referstr1 = txtProductName.Value.Trim();//商品名称
172                     log.referstr2 = "";
173                     log.referstr3 = "";
174                     log.referstr4 = "";
175                     log.referstr5 = "";
176                     log.referstr6 = "";
177                     BizUtil.WriteBehaviorLog(log, this.MyContext);
178
179                     #endregion
180                     Script("closeForm();");
181                 }
182                 else
183                 {
184                     ShowWarningMessage(msg);
185                 }
```

- 列表页面一般情况下需重载 SetPageCode，设置页面唯一的 Pagecode；Dialog 页面可不重载
- 列表分页（最常使用）：

```
153         /// <summary> ...
158         protected void SearchData()
159         {
160             string url = "mall/admin/product";
161             Dictionary<string, object> dic = GetStatisticsPage();
162             PageContent<ProductInfo> pc = CommonAPIBiz.GetDataPage<PageContent<Product
163             if (pc != null)
164             {
165                 pageNav.TotalCount = pc.paging.count;
166                 pageNav.CurrentPageIndex = pc.paging.offset / pageNav.PageSize + 1;
167                 this.repProduct.DataSource = pc.datas.ToList();
168                 this.repProduct.DataBind();
169             }
170             Script("ChangeOrderInfo();closeDealingMsg();");
171         }
172
173         /// <summary> ...
178         public Dictionary<string, object> GetStatisticsPage()
179         {
180             Dictionary<string, object> dic = new Dictionary<string, object>();
181             //获取表单数据
182             int limit = pageNav.PageSize;
183             int offset = pageNav.PageSize * (pageNav.CurrentPageIndex - 1);
184             if (offset < 0)
185             {
186                 offset = 0;
187             }
188             string direction = this.hidSort.Value;
189             string orderby = this.hidOrder.Value;
190             dic.Add("limit", limit);
191             dic.Add("offset", offset);
192             dic.Add("direction", direction);
193             dic.Add("orderby", orderby);
194             string searchKey = this.txtSearch.Value.Trim();
195             if (!string.IsNullOrEmpty(searchKey))
196             {
197                 dic.Add("keywords", searchKey);
198             }
199             if (!string.IsNullOrEmpty(scStatus.SelectValue))
200             {
201                 dic.Add("status", scStatus.SelectValue);
202             }
```

3.2 控件说明

控件名称	控件命名规范	控件说明
OsTextBox	txt+英文意义名称	属性 InputType 对应程序的枚举，InputType="number" 控件只可输入数字，InputType="date"控件选择时间格式为 "yyyy-MM-dd"时分秒为 0，InputType="fulldate" 控件选择时间格式为 "yyyy-MM-dd 23：59：59"（一般用于搜索界面时间段截止时间）...

OsDropDownList	drp+英文意义名称	属性 CssClass="xxx" 可使用现有样式使控件以图片的形式显示； 绑定值方法：BindDropDownList； 赋值方法：SetDropDownListSelectValue。
OsButton	btn+英文意义名称	属性 FabidRepeatRequest="true"可防止重复请求。
OsCheckBox	chb+英文意义名称	
OsCheckBoxList	chbl+英文意义名称	现在程序中绑定的值为枚举信息或数据库键值对信息；绑定值方法：BindCheckBoxList。
OsRadioButton	rab+英文意义名称	
OsRadioButtonList	rabl+英文意义名称	现在程序中绑定的值为枚举信息或数据库键值对信息；绑定值方法：BindRadioButtonList；赋值方法：SetRadioButtonListSelectValue。
OsListBox	lib+英文意义名称	绑定值方法：BindListBox。
OsLabel	lbl+英文意义名称	
OsLinkButton	lbtn+英文意义名称	属性 CssClass="xxx" 可使用现有样式使控件以图片的形式显示；属性 Text="xxx" IsShowText="true"两个属性组合显示文本内容
OsHiddenField	hid+英文意义名称	
OsHyperLink	hl+英文意义名称	
OsRepeater	rep+英文意义名称	
OsNoResultMessageLable	nrml+英文意义名称	（只在 OsRepeater 中无数据使用）；属性 Colspan="0"，当 OsRepeater 中使用的 table 就需赋值为改控件跨列数，如果 OsRepeater 未使用的 table，可不赋值。
OsPageNavigation	nav+英文意义名称	和 OsRepeater 组合使用
OsPanel	pnl+英文意义名称	
SlUploadImage	Img+英文意义名称	属性 Enabled="false"表示只可查看，不可修改和删除图片；属性 ConfigKey="xxxx"根据具体附件设置来存储不同的图片信息，具体开发某块功能需具体配置枚举键。
SlUploadAttachments	osf+英文意义名称	属性 Enabled="false"表示只可查看，不可修改和删除附件；属性 ConfigKey="xxxx"根据具体附件设置来存储不同的附件信息，具体开发某块功能需具体配置枚举键。
OsControlManager	cvl+英文意义名称	如下标签：<oswc:OsControlManager ID="cvlValidator" runat="server"> <oswc:TemplateValidator ValidateGroup="btnSave"> <oswc:ReqValidateControl ControlToValidate="rblTransactionMode" MessageDailog="False"> <oswc:ValidatorItem Message="请选择交易类型" Value="true" Validator="Required" />

		</oswc:ReqValidateControl> </oswc:TemplateValidator> </oswc:OsControlManager> 说明：单击保存按钮时会判断交易模式是否选择； 属性 ValidateGroup="btnSave" 表示保存按钮触发该验证，ControlToValidate="rblTransactionMode"表示验证的控件名，Validator="Required" 验证控件值是否为空，Validator 对应的值未枚举可验证非空，是否是数字，比较大小等，Message="请选择交易类型"验证不通过时提示信息。
OsTooltip	tip+英文意义名称	
OsTagsAutoComplete	tac+英文意义名称	属性 ServiceMethod="xxxx"对应服务调用的方法名，具体功能需写具体方法。

3.3 公共方法

1. 业务单据的主键使用GetNewId()方法生成，特殊单据的主键生成规则可重载该方法
2. 所有界面存在MyContext上下文MyProfile用户上下文，微应用则是YxtContent
3. 对于界面上字符串过长使用方法LeftString 进行截取
4. 对于界面上时间类型使用方法 GetDateLang 自动格式化
5. 对于界面上由多语言 Key 获取对应语言信息 GetPropertiesText
6. 界面操作提示信息方法调用
   成功提示信息方法：ShowPageMessage；
   失败提示信息方法：ShowErrorMessage；
   警告提示信息方法：ShowWarningMessage
7. 弹出界面需调用方法 ShowIFrameDialog
8. 界面参数传递获取方法 GetQueryString；前端HTML的js则使用getQueryString
9. 页面权限控制方法
   CommonWebUtil.CheckPermission(pageCode, FunctionOptionType.QueryAllDept)
   其中 FunctionOptionType 中包含类型参考表 Core_AppfeaturesOption

4 其他规范

4.1 异常处理

1. 正常情况下，应该仅在表现层处理异常；

2. 如果包含以下情况，可以在表现层以外的层处理异常：

   1) 能够有针对性的处理该异常；

   2) 可以忽略该异常；

   3) 转换新异常继续抛出；

3. 如何处理异常：

   1) 可以忽略无关紧要的异常；

   2) 将异常转换为友好的错误信息呈现给用户；

   3) 对于可能涉及程序功能、性能问题的异常应该记入系统日志，以备技术人员排错；

   4) 如果是重大异常，可以考虑终止应用程序；

   5) 在可能抛出异常的地方，如果需要，请确保使用 finally 进行资源清理，而无论此处是否捕捉了异常。

4. 性能问题： 异常的捕获有一定的性能开销，因此可以通过合适的判断语句减少异常的抛出。

   1) 可以采用“Tester-Doer”模式，参见：http://msdn.microsoft.com/zh-cn/library/ms229009.aspx；

   2) 可以采用“Try-Parse”模式，应该优先使用.net 基础类库中的“Try-Parse”方法，例如：int.TryParse()；

4.2 多语言

4.2.1 多语言文件定义及命名

1. 所有国际化文件统一定义在 Languages 目录中；
2. 界面渲染国际化文件使用 jquery.i18n.properties.js 读取 Languages 文件进行初始化
3. 应用级多语言通过哈希值读取 redis

4.2.2 资源 KEY 的命名规则

1. 国际化 KEY 的命名规则是：模块名+功能名称+翻译自定义名称，并用下划线连接，如：发表群组话题时，“话题标题”这个标签名称的 KEY 为，“Group_Topic_Title”；
2. API 返回消息 key 来自 java 返回的 key 名称，这个是后端自己维护处理

4.2.3 C#中多语言

1. 代码事例：

```
GetPropertiesText('key')
CacheUtil.GetPropertiesText(strKey, MyContext)
ComBaseUtil.GetPropertiesText(strKey, MyContext)
在 UpdatePanel 中时绑定方法
protected void Page_Load(object sender, EventArgs e)
{
   BindUpdatePanelLanguage（UpdatePanelID）
}
```

4.2.4 JS 中多语言

1. 代码事例：

```
获值：jQuery.i18n.map['key']
占位符：$.i18n.prop('string_hello',data1,data2)
异步标签或动态加载 html 时 loadPropertiesForAsync(“外层 DIV id”);
```

4.2.5 HTML 中多语言

1. 代码事例：

```
正常值：data-localize
title：data-localizetitle
占位符：data-localizeph 例：<label data-localizeph="common_text"><label>222222</label>
<label>333333</label></label>
如果需要加样式:<label data-localizeph="common_text"><label><label class="color">222222</label></label>
<label>333333</label></label>;
```

4.2.6 多语言日期

C#

//date 时间，type：1 短格式（年月日）；10 是否启用几秒前短格式（年月日）；2 长格式（年月日时分）；20 是否启用几秒前长格式（年月日时分）；3(月日)；30 是否启用几秒前(月日)，lang 语言

GetDateLang(date, type, lang)

JS

获取时间：<param name="type">1 短格式（年月日）；10 是否启用几秒前短格式（年月日）；2 长格式（年月日时分）；20 是否启用几秒前长格式（年月日时分）；3(月日)；30 是否启用几秒前(月日)</param>

GetDateLang(object date, int type)

5 API 使用规范

5.1 ResfulApiService 的实现

略

5.2 Service 调用 API

call api 时，ResponseHttpData 无返回值(NoContent)的重载函数，PUT/Delete 请求时，通常没有返回内容，需要使用这种方式

如: api.Put("orgs/111", {}, usercontext)，直接返回不带泛型的 ResponseHttpData

。。。