C#开发微信门户及应用教程

作者：伍华聪

C#开发微信门户及应用(1)--开始使用微信接口

微信应用如火如荼，很多公司都希望搭上信息快车，这个是一个商机，也是一个技术的方向，因此，有空研究下、学习下微信的相关开发，也就 成为日常计划的重要事情之一了。本系列文章希望从一个循序渐进的角度上，全面介绍微信的相关开发过程和相关经验总结，希望给大家了解一下相关的开发历程。 本随笔主要针对微信开发过程的前期准备和一些初始的工作的介绍。

在写下本文的之前一周时间里 ,我主要就是参考一些介绍文章以及微信公众平台的相关接口说明，并结合 C#的代码开发，整理了自己公司的门 户界面，实现了微信工作号的初步用户交互和信息展示工作，随着工作的进一步开展，越来越多的功能可能加入，并希望从应用角度上扩展微信的接口，从而实现我 对微信接口的技术探秘和了解过程。

1、微信账号

要开发使用微信的平台 API，就需要到微信的公众平台（https://mp.weixin.qq.com/）去注册，拥有一个**服务号**或者**订阅号**，服务号主要面对企业和组织，订阅号主要面向组织和个人，他们之间有一定的差异，根据不同的需要自己申请对应的账号即可。

为了使用一些高级的接口 ,你可能需要拥有服务号和高级的认证。账号注册过程，需要下载一个申请表格，打印并盖公章，另外还需要申请人拿着身份证拍照（有点怪异，呵呵），然后上传到服务器进行审核，一般很快就能获取批复。

我以公司名义申请了服务号，账号注册后，会在主界面上显示你的相关信息，另外给你申请一个二维码的东西 ,扫描二维码即可进入公司的微信关注确认对话框，非常方便。如下就是我申请后的公司账号二维码，可以直接使用扫描。

2、微信菜单定义

微信有两种方式的菜单定义，一种是编辑模式，一种是开发模式，两者互斥，也就是说，一旦我们采用了开发模式，就不能使用编辑模式了，反过来也一样。编辑下的菜单，其实也是可以管理的，但是微信不支持，觉得很不爽。

一般情况下，如果我们刚刚申请了微信号码，可以使用编辑菜单测试一下，根据说明编辑一些菜单试试。虽然微信说 24 小时内更新，不过一般很快，最快可能一两分钟就更新了，感觉还是不错的。

使用开发者模式，你需要根据微信的要求，在服务器上放置一个页面链接，使用C#开发的 ,可以采用***.ashx 的命名方式 ,使用 Asp.NET 的一般处理程序即可，不需要使用普通的页面。

使用开发模式的菜单，也就是可以调用微信 API 进行菜单创建的工作，对于调用微信的 API（微信有很多 API 可以调用），我们需要知道，有几个参数的重要性，所以在开发模式打开的时候，会给你列出这些参数，如下所示。

3、接入微信的链接处理

上面说了，你申请开发模式对菜单或者对其他 API 的调用，你需要顺利通过接入微信的测试，也就是确认你填写的链接存在并能顺利经过微信的回调测试。微信提供了一个 PHP 的页面处理例子，如果我们是 C#开发的呢，可以搜一下就会得到答案，我的处理方式如下所示。

创建一个一般处理程序，然后在其处理页面里面增加一个处理逻辑，如果是非 POST 方式的内容，就是表示微信进行的 Get 测试，你需要增加一些处理逻辑，把它给你的内容传回去即可，如果是 POST 方式的，就是微信服务器对接口消息的请求操作了，后面介绍。

```
    /// <summary>
    /// 微信接口。统一接收并处理信息的入口。
    /// </summary>
    public class wxapi : IHttpHandler
    {
        public void ProcessRequest(HttpContext context)
        {
            string postString = string.Empty;
            if (HttpContext.Current.Request.HttpMethod.ToUpper() ==
"POST")
            {
                using (Stream stream =
HttpContext.Current.Request.InputStream)
                {
                    Byte[] postBytes = new Byte[stream.Length];
                    stream.Read(postBytes, 0, (Int32)stream.Length);
                    postString = Encoding.UTF8.GetString(postBytes);
```

```
                }


                if (!string.IsNullOrEmpty(postString))
                {
                    Execute(postString);
                }
            }
            else
            {
                Auth(); //微信接入的测试
            }
        }
```

一般来说，Auth 函数里面，就是要对相关的参数进行获取，然后进行处理返回给微信服务器。

```
            string token = "****";//你申请的时候填写的 Token


            string echoString = HttpContext.Current.Request.QueryString["echoStr"];
            string signature = HttpContext.Current.Request.QueryString["signature"];
```

```
            string timestamp =
HttpContext.Current.Request.QueryString["timestamp"];
            string nonce =
HttpContext.Current.Request.QueryString["nonce"];
```

完整的 Author 函数代码如下所示，其中我把业务逻辑进行进一步抽取到了一个新的类里面，方便业务逻辑的管理。

```
        /// <summary>
        /// 成为开发者的第一步，验证并相应服务器的数据
        /// </summary>
        private void Auth()
        {
            string token =
ConfigurationManager.AppSettings["WeixinToken"];//从配置文件获取
Token
            if (string.IsNullOrEmpty(token))
            {
                LogTextHelper.Error(string.Format("WeixinToken 配置
项没有配置！"));
            }
```

```
            string echoString =
HttpContext.Current.Request.QueryString["echoStr"];
            string signature =
HttpContext.Current.Request.QueryString["signature"];
            string timestamp =
HttpContext.Current.Request.QueryString["timestamp"];
            string nonce =
HttpContext.Current.Request.QueryString["nonce"];

            if (new BasicApi().CheckSignature(token, signature,
timestamp, nonce))
            {
                if (!string.IsNullOrEmpty(echoString))
                {
                    HttpContext.Current.Response.Write(echoString);
                    HttpContext.Current.Response.End();
                }
            }
        }
```

而对微信参数的签名并返回的操作 CheckSignature，代码如下所示。

```
        /// <summary>
        /// 验证微信签名
        /// </summary>
        public bool CheckSignature(string token, string signature, string timestamp, string nonce)
        {
            string[] ArrTmp = { token, timestamp, nonce };

            Array.Sort(ArrTmp);
            string tmpStr = string.Join("", ArrTmp);

            tmpStr = FormsAuthentication.HashPasswordForStoringInConfigFile(tmpStr, "SHA1");
            tmpStr = tmpStr.ToLower();

            if (tmpStr == signature)
            {
                return true;
            }
            else
            {
```

```
                return false;
            }
        }
```

4、使用开发方式创建菜单

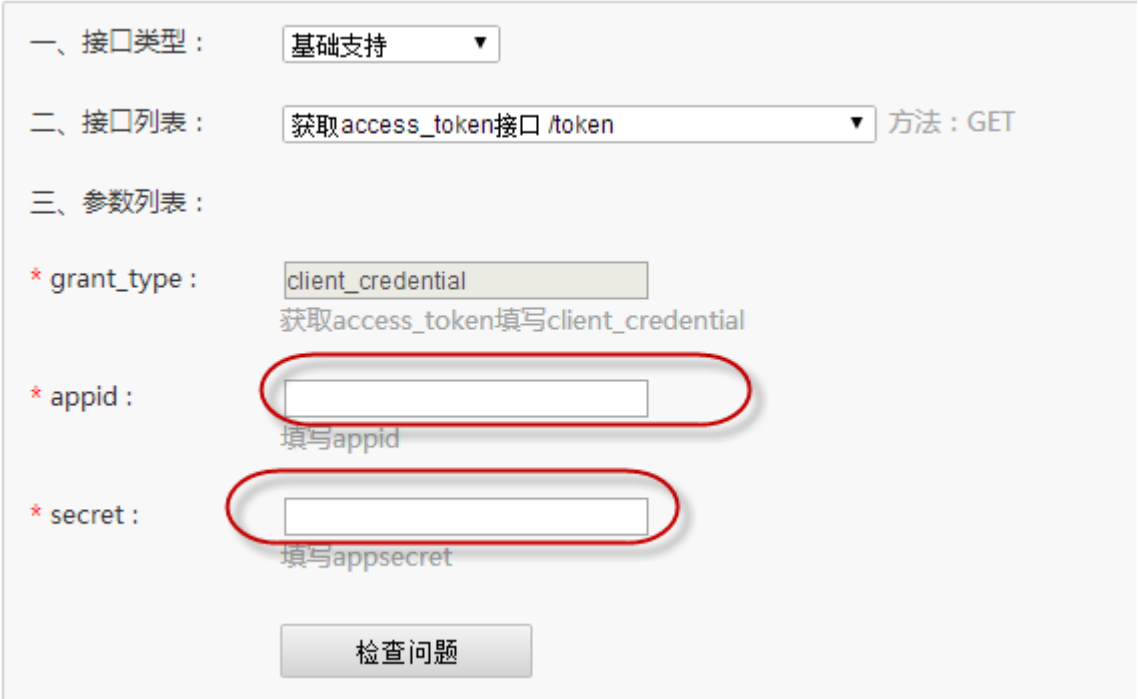
一旦你顺利通过微信的认证，那么它就让你以开发方式调用它的 API，并且可以随意创建你的菜单了。

创建菜单的方式，你可以通过下面的位置进入到他的 API 处理界面里面。

进入后，你会发现微信把很多消息的处理，分门别类放到不同的分类里面了。

其实我们现在初步要做的就是如何看看，使用代码方式调用创建菜单，进入菜单的 API 调试界面里面。

你会发现里面还需要输入一个 Access_Token 的东西 ,这个是一个会话身份认证，因此你还需要到接口里面去找这个如何创建的。下面图中的两个红色部分，就是我们开始的时候，微信提示我们“开发者凭据”的两个关键参数。



弄完这些，你就可以根据获得的 Access_Token 进行菜单的创建工作了，根据菜单的定义，它分为几类，可以分为 URL 方式（View），事件方式（Click）。

click：用户点击 click 类型按钮后，微信服务器会通过消息接口推送消息类型为 event 的结构给开发者（参考消息接口指南），并且带上按钮中开发者填写的 key 值，开发者可以通过自定义的 key 值与用户进行交互；

view：用户点击 view 类型按钮后，微信客户端将会打开开发者在按钮中填写的 url 值 （即网页链接），达到打开网页的目的，建议与网页授权获取用户基本信息接口结合，获得用户的登入个人信息。

5、我创建的菜单案例

在随笔的开始 ,我公布了一个二维码 ,一旦使用微信扫一扫 ,进行关注服务号后，那么就可以看到我自己创建的菜单了。主菜单一般最多三列，每个主菜单还可以有子菜单，他们的文字都有所限制的。

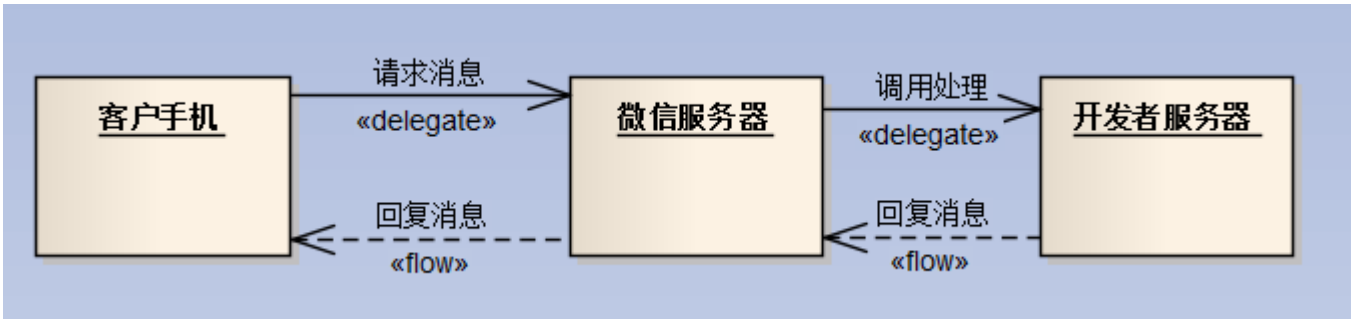
我们来看看我公司的微信门户菜单，看起来是不是很酷呢。

C#开发微信门户及应用(2)--微信消息的处理和应答

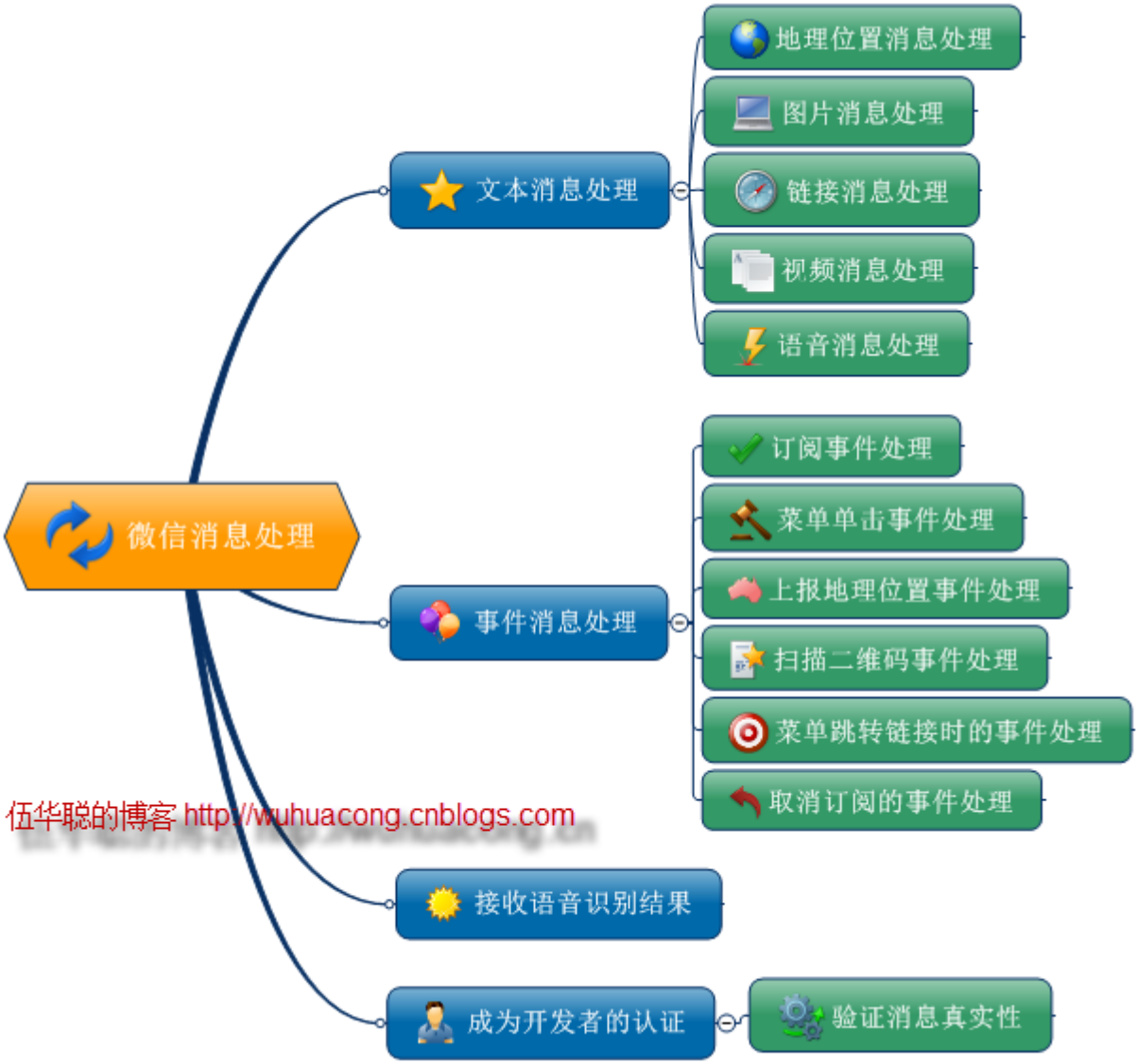
微信应用如火如荼，很多公司都希望搭上信息快车，这个是一个商机，也是一个技术的方向，因此，有空研究下、学习下微信的相关开发，也就 成为计划的安排事情之一了。本系列文章希望从一个循序渐进的角度上，全面介绍微信的相关开发过程和相关经验总结，希望给大家了解一下相关的开发历程。本篇 随笔主要基于上一篇《C#开发微信门户及应用(1)--开始使用微信接口》的基础上进行深入的介绍，介绍微信消息的处理和应答的过程。

1、微信的消息应答交互

我们知道，微信的服务器架起了客户手机和开发者服务器的一个桥梁，通过消息的传递和响应，实现了与用户的交互操作，下面是它的消息流程图。



微信向开发者服务器请求的消息包含了多种类型，不过基本来说，分为了文本消息处理、事件消息处理、语音消息的识别，以及成为开发者之前的那个消息认证操作基本分类，下面是我绘制的一个消息分类图，其中介绍了这几种关系，以及各自的消息细化分类。



对于这些消息的请求，我们在开发服务器端，需要编写相关的逻辑进行对应给的处理，然后给微信服务器平台回应消息即可。

在前一篇的随笔里面我贴过代码，介绍微信消息处理的入口操作，代码如下所示。

```
        public void ProcessRequest(HttpContext context)
        {
```

```
            //WHC.Framework.Commons.LogTextHelper.Info("测试记录
");

            string postString = string.Empty;
            if (HttpContext.Current.Request.HttpMethod.ToUpper() ==
"POST")
            {
                using (Stream stream =
HttpContext.Current.Request.InputStream)
                {
                    Byte[] postBytes = new Byte[stream.Length];
                    stream.Read(postBytes, 0, (Int32)stream.Length);
                    postString = Encoding.UTF8.GetString(postBytes);
                }

                if (!string.IsNullOrEmpty(postString))
                {
                    Execute(postString);
                }
            }
            else
            {
```

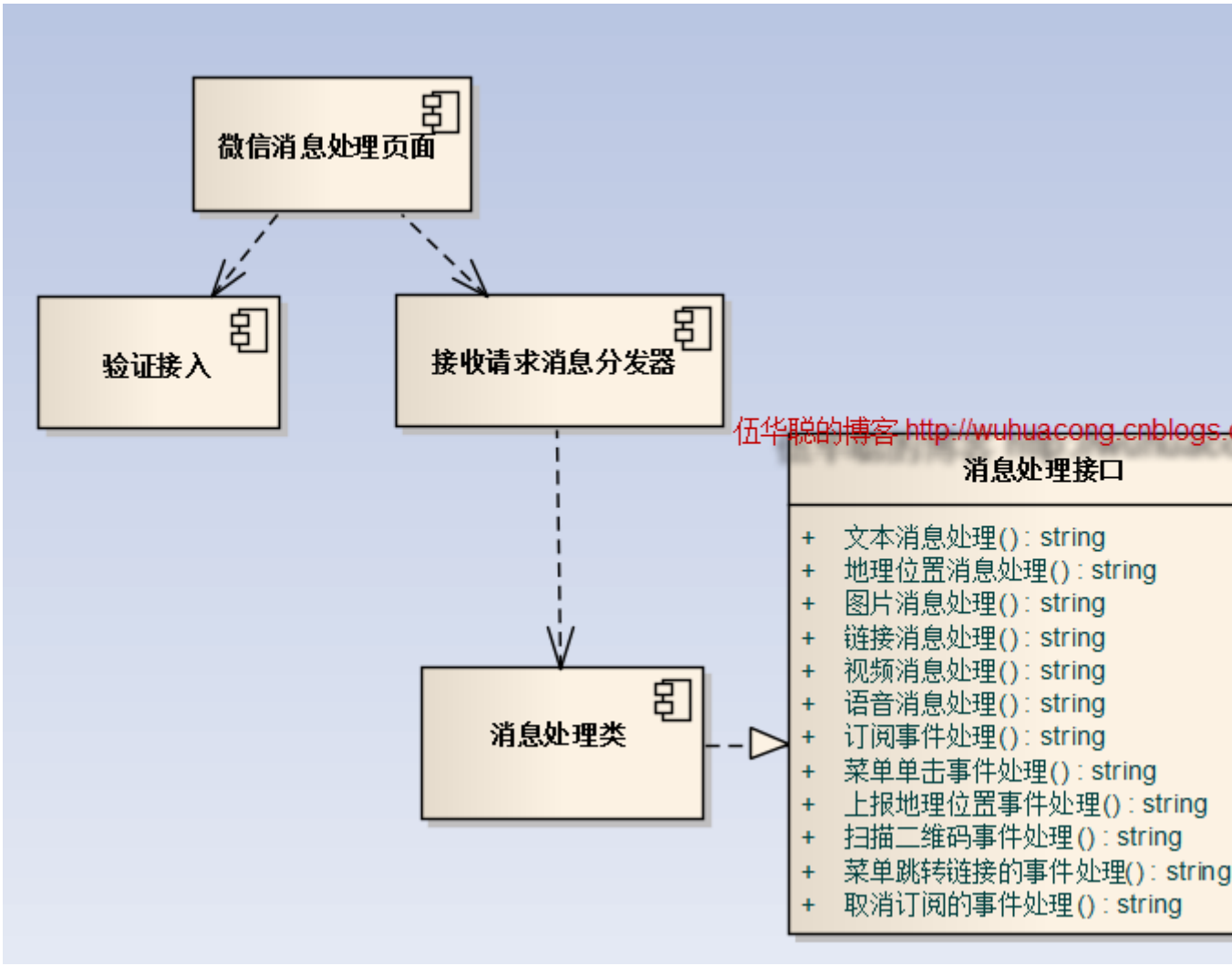
```
                Auth();
            }
        }
```

其中的 Execute(postString);就是对消息的处理函数，它实现了对不同消息的分发处理过程。 ‘

```
        /// <summary>
        /// 处理各种请求信息并应答（通过 POST 的请求）
        /// </summary>
        /// <param name="postStr">POST 方式提交的数据</param>
        private void Execute(string postStr)
        {
            WeixinApiDispatch dispatch = new WeixinApiDispatch();
            string responseContent = dispatch.Execute(postStr);

            HttpContext.Current.Response.ContentEncoding = Encoding.UTF8;
            HttpContext.Current.Response.Write(responseContent);
        }
```

里面的 WeixinApiDispatch 就是一个分发的管理类，它提取请求消息的内容，并构建不同类型的消息参数，传递给不同的响应函数进行处理，然后返回封装好的 XML 内容，作为响应。

具体的代码处理逻辑如下图所示。



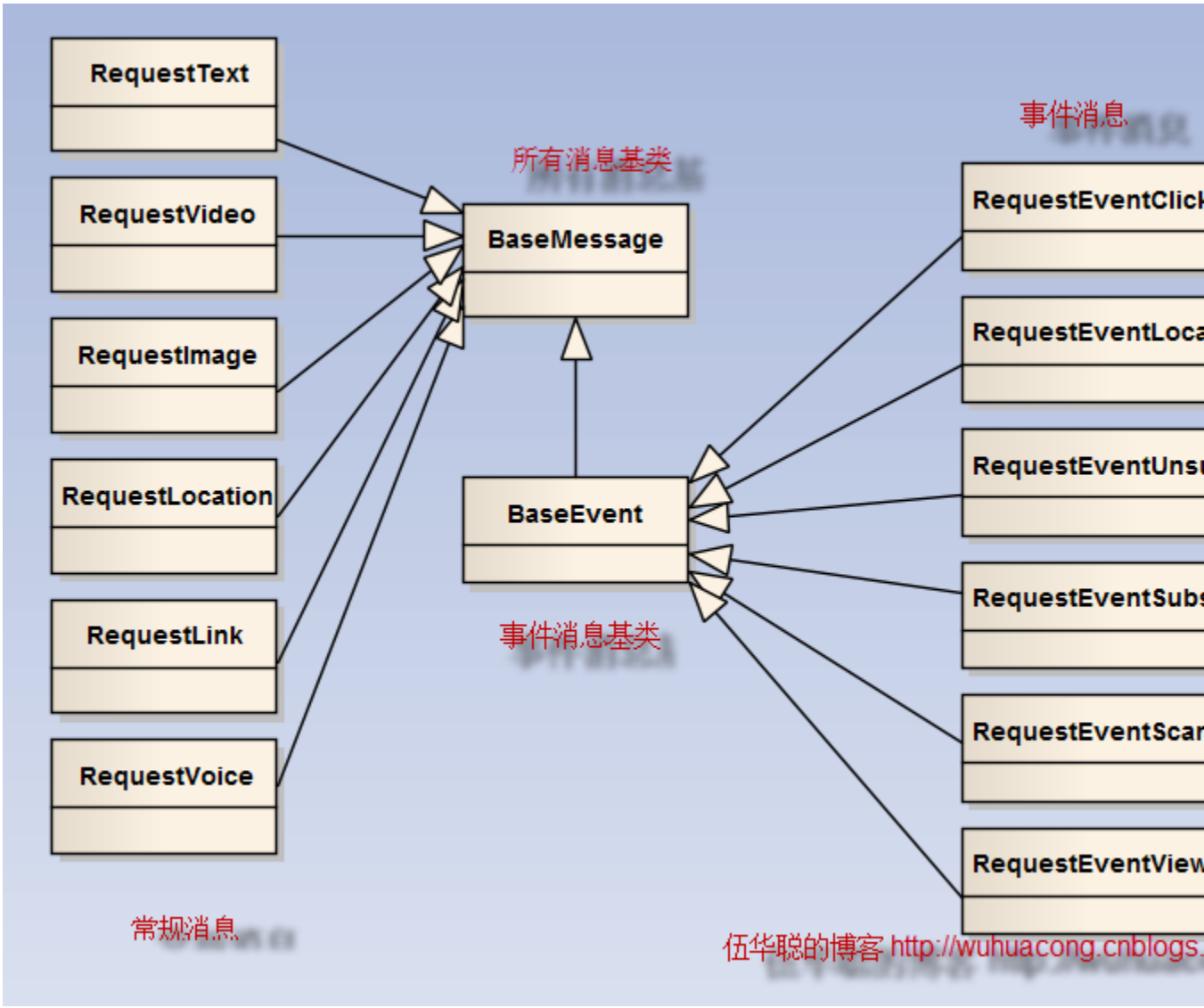
这个消息处理接口，其实就是定义好一系列的对请求消息的处理操作，参数是不同给的消息对象，具体的代码定义如下所示（由于篇幅原因，省略部分接口，具体可以参考上图）。

```
/// <summary>
/// 客户端请求的数据接口
/// </summary>
public interface IWeixinAction
{
    /// <summary>
    /// 对文本请求信息进行处理
    /// </summary>
    /// <param name="info">文本信息实体</param>
    /// <returns></returns>
    string HandleText(RequestText info);

    /// <summary>
    /// 对图片请求信息进行处理
    /// </summary>
    /// <param name="info">图片信息实体</param>
    /// <returns></returns>
    string HandleImage(RequestImage info);
```

.....

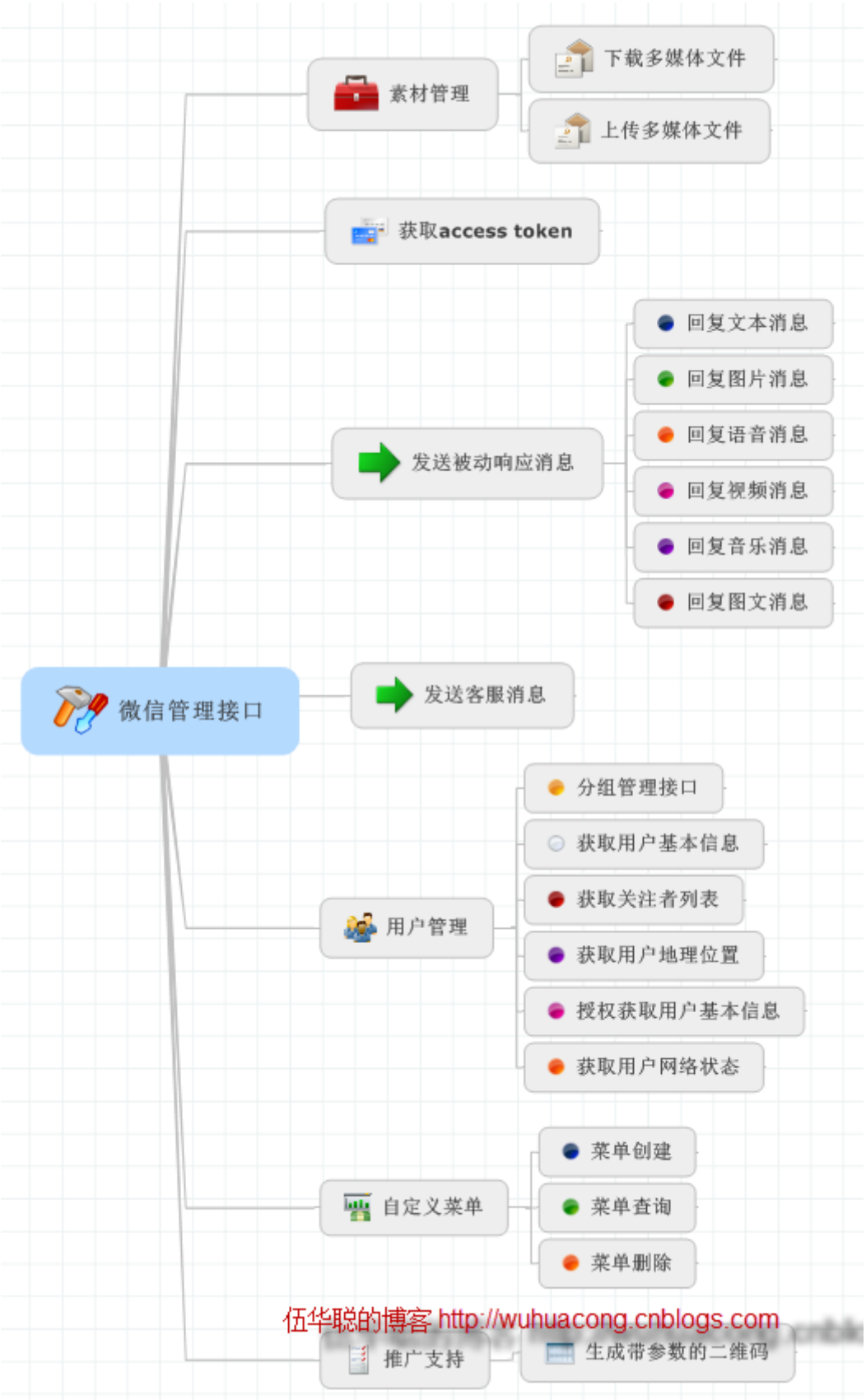
```
        /// <summary>
        /// 对订阅请求事件进行处理
        /// </summary>
        /// <param name="info">订阅请求事件信息实体</param>
        /// <returns></returns>
        string HandleEventSubscribe(RequestEventSubscribe info);

        /// <summary>
        /// 对菜单单击请求事件进行处理
        /// </summary>
        /// <param name="info">菜单单击请求事件信息实体</param>
        /// <returns></returns>
        string HandleEventClick(RequestEventClick info);

.....
    }
```


从上面的代码可以看出，不同的消息，到处理函数这里，就以不同的消息实体类的方式传递过来了（**注意：实体类是我根据程序开发需要自己定义的，非微信本身的实体类**），这样非常方便我们处理操作，否则每次需要解析不同的消息内容，很容易出现问题，这样强类型的数据类型，提高了我们开发微信应用的强壮型和高效性。这些实体类的对象有一定的继承关系的，他们的继承关系如下所示。



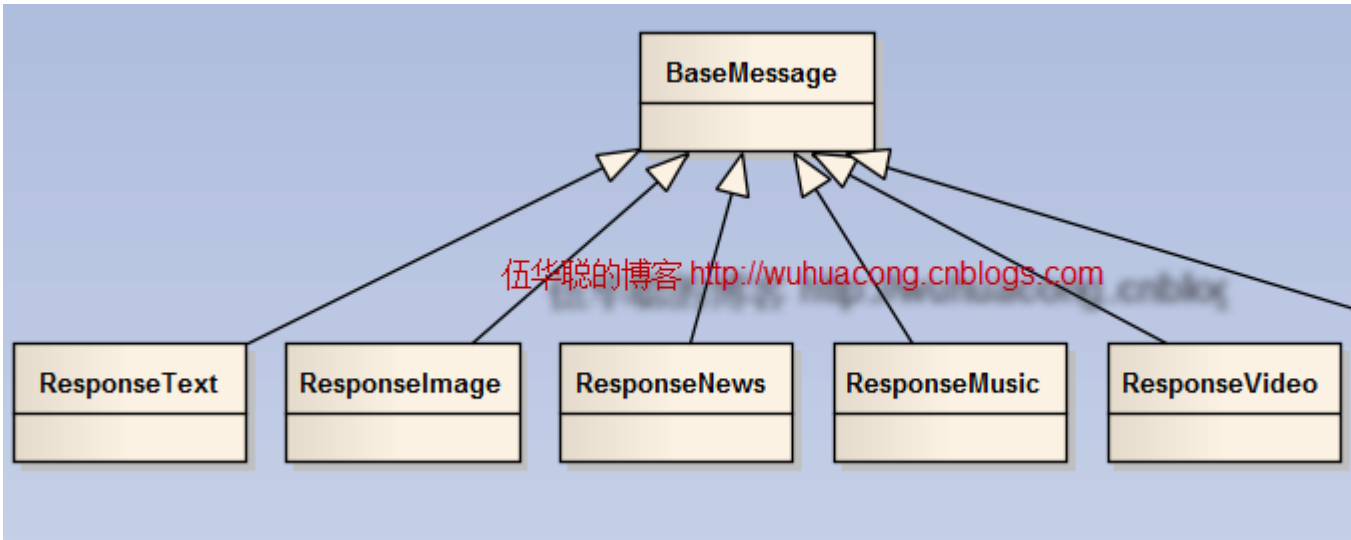
2、微信的管理接口

上面的消息分类是微信服务器向开发者服务器发送的消息请求操作，还有一种消息，是我们开发者服务器向微信服务器进行的消息请求或者响应，这种这里暂且称之为微信的管理接口，它表明了我们可以通过这些接口进行相关的消息回复或者数据管理操作。它的分类图如下所示。



伍华聪的博客 http://wuhuacong.cnblogs.com

微信的回复消息处理 ,它也和上面小节的信息一样 ,它也是继承自 BaseMessage 实体类的（**同样，下图的实体类及其继承关系也是自定义的，方便程序开发**），它的关系如下所示



回复的消息，一般用的最多的是文本消息和图文消息。

文本消息的效果如下所示。

图文消息 ,可以增加图片 ,还可以增加详细的链接页面 ,是非常好看的一种效果，对于一些内容比较多，希望展现更好效果的，一般采用这种，效果如下所示。

C#开发微信门户及应用(3)--文本消息和图文消息的应答

微信应用如火如荼，很多公司都希望搭上信息快车，这个是一个商机，也是一个技术的方向，因此，有空研究下、学习下微信的相关开发，也就 成为计划的安排事情之一了。本系列文章希望从一个循序渐进的角度上，全面介绍微信的相关开发过程和相关经验总结，希望给大家了解一下相关的开发历程。

在前面两篇两篇随笔《C#开发微信门户及应用(1)--开始使用微信接口》和《C#开发微信门户及应用(2)--微信消息的处理和应答》里面，大致介绍了我微信应

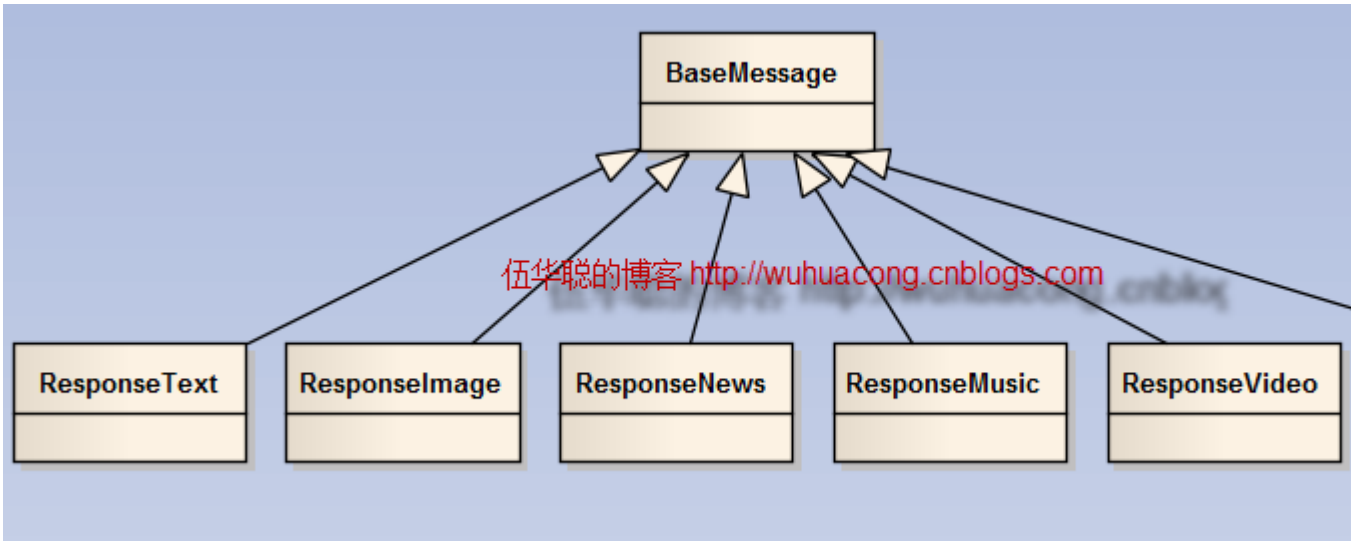
用的框架构建，本随笔继续介绍这一主题，介绍消息应答里面的文本应答和图文应答的过程。

我们知道 ,给手机用户发送响应消息 ,它可以分为好多种方式 ,如回复文本消息、回复图片消息、回复语音消息、回复视频消息、回复音乐消息、回复图文消息等，如下所示。

而其中图片、视频、语音这三种方式，是需要开通微信认证才可以向用户发送存在微信服务器上的媒体信息，一般没有认证的公众号或者服务号，是不能发送这几种内容的。

1、实体信息关系及定义

在上一篇微信开发的随笔中，我展示了对接收消息和回复消息的应用实体类，这些实体类是我根据需要，根据开发需要，在应用层面对它们进行了封装，如回复的消息关系如下所示。



消息基类 BaseMessage 的实体类定义如下所示，它对日期构造了一个整形数值，并具备了一些常规的属性，并且还有一个重要的 ToXML 方法，用来给方法传递这些 XML 数据的。

```
/// <summary>
/// 基础消息内容
/// </summary>
[XmlRoot(ElementName = "xml")]
public class BaseMessage
{
    /// <summary>
    /// 初始化一些内容，如创建时间为整形，
    /// </summary>
    public BaseMessage()
    {
```

```
        this.CreateTime = DateTime.Now.DateTimeToInt();
    }

    /// <summary>
    /// 开发者微信号
    /// </summary>
    public string ToUserName { get; set; }

    /// <summary>
    /// 发送方帐号（一个 OpenID）
    /// </summary>
    public string FromUserName { get; set; }

    /// <summary>
    /// 消息创建时间 （整型）
    /// </summary>
    public int CreateTime { get; set; }

    /// <summary>
    /// 消息类型
    /// </summary>
    public string MsgType { get; set; }
```

```
        public virtual string ToXml()
        {
            this.CreateTime = DateTime.Now.DateTimeToInt();//重新更新
            return MyXmlHelper.ObjectToXml(this);
        }

    }
```

回复的文本消息实体类代码如下所示，我们可以看到，它继承了很多通用的实体属性，并且还具备了一个 ToXml 的通用方法，我们需要把它转换为响应的 XML 的时候，就使用这个方法就可以了。

```
    /// <summary>
    /// 回复文本消息
    /// </summary>
    [System.Xml.Serialization.XmlRoot(ElementName = "xml")]
    public class ResponseText : BaseMessage
    {
        public ResponseText()
        {
```

```
            this.MsgType =
ResponseMsgType.Text.ToString().ToLower();
        }

        public ResponseText(BaseMessage info) : this()
        {
            this.FromUserName = info.ToUserName;
            this.ToUserName = info.FromUserName;
        }

        /// <summary>
        /// 内容
        /// </summary>
        public string Content   { get; set; }
    }
```

而图文消息对象类 ResponseNews，它包含更多的信息定义

```
    /// <summary>
    /// 回复图文消息
    /// </summary>
    [System.Xml.Serialization.XmlRoot(ElementName = "xml")]
```

```
    public class ResponseNews : BaseMessage
    {
        public ResponseNews()
        {
            this.MsgType =
ResponseMsgType.News.ToString().ToLower();

            this.Articles = new List<ArticleEntity>();
        }
        public ResponseNews(BaseMessage info) : this()
        {
            this.FromUserName = info.ToUserName;
            this.ToUserName = info.FromUserName;
        }

        /// <summary>
        /// 图文消息个数，限制为 10 条以内
        /// </summary>
        public int ArticleCount
        {
            get
            {
```

```
                return this.Articles.Count;
            }
            set
            {
                ;//增加这个步骤才出来 XML 内容
            }
        }

        /// <summary>
        /// 图文列表。
        /// 多条图文消息信息，默认第一个 item 为大图,注意，如果图文数超过 10，则将会无响应
        /// </summary>
        [System.Xml.Serialization.XmlArrayItem("item")]
        public List<ArticleEntity> Articles { get; set; }

    }
```

而其中的图文列表集合中的对象，它也是一个实体类型,包含了一些图文的链接，标题等信息，不在赘述。

2、消息的回复处理

如对于文本消息，我们可以用以下的方式进行处理。

```
ResponseText response = new ResponseText(info);
response.Content = "抱歉，此功能暂未开通。";
result = response.ToXml();
```

对于图文消息，我们可能需要录入更多的消息才能返回更好的效果。

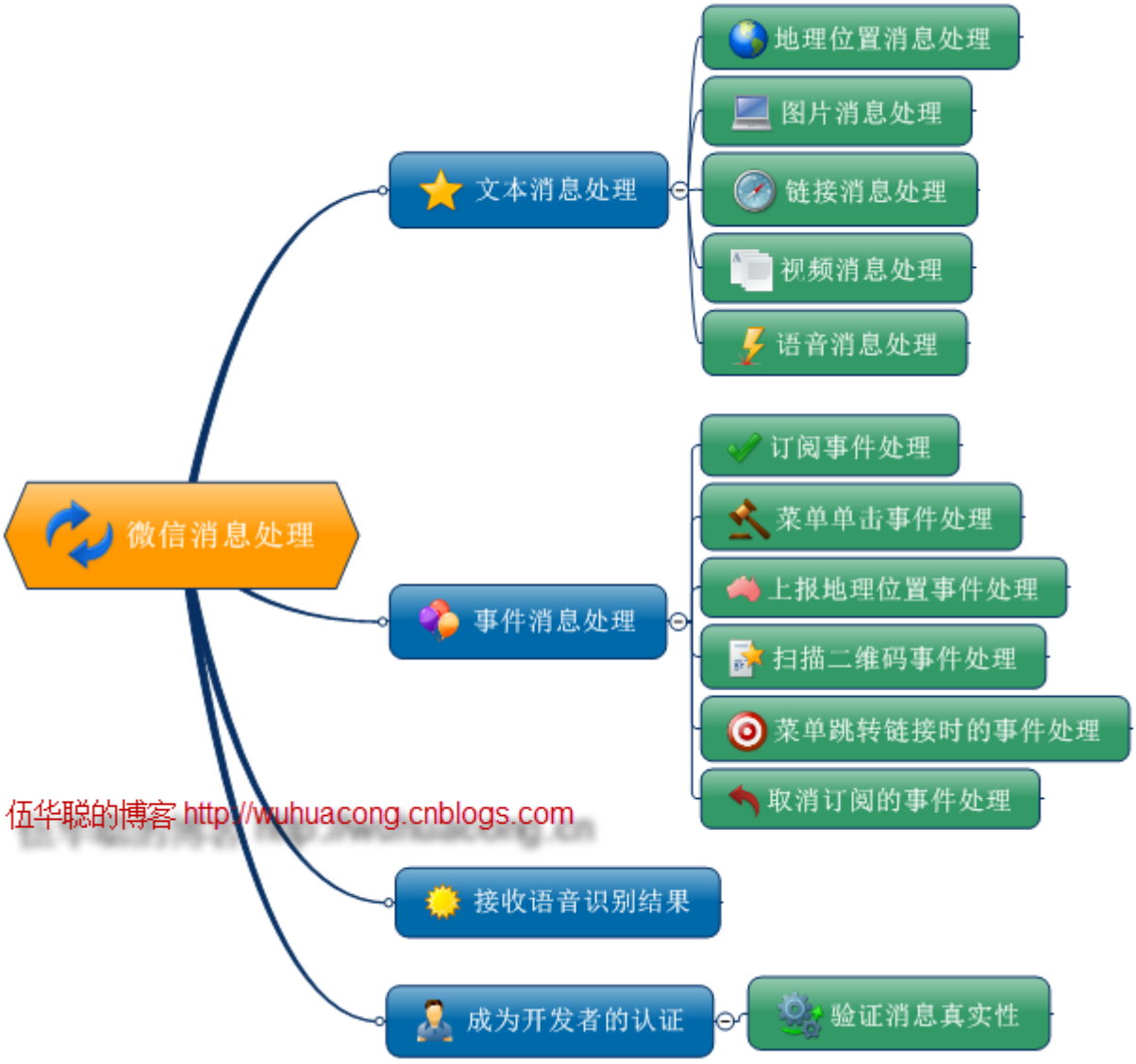
注意图文的消息 ,图片的尺寸最好按照官方的标准 ,否则在手机上看起来不好看，官方的标准好像是宽高是(360,200)像素

```
/// <summary>
/// 订阅或者显示公司信息
/// </summary>
/// <param name="info"></param>
/// <returns></returns>
private string ShowCompanyInfo(BaseMessage info)
{
    string result = "";
    //使用在微信平台上的图文信息(单图文信息)
    ResponseNews response = new ResponseNews(info);
    ArticleEntity entity = new ArticleEntity();
```

```
                entity.Title = "广州爱奇迪软件科技有限公司";
                entity.Description = "欢迎关注广州爱奇迪软件--专业的单位信息化软件和软件开发框架提供商，我们立志于为客户提供最好的软件及服务。\r\n";
                entity.Description += "我们是一家极富创新性的软件科技公司，从事研究、开发并销售最可靠的、安全易用的技术产品及优质专业的服务，帮助全球客户和合作伙伴取得成功。\r\n.....（此处省略 1000 字，哈哈）";
                entity.PicUrl = "http://www.iqidi.com/WeixinImage/company.png";
                entity.Url = "http://www.iqidi.com";

                response.Articles.Add(entity);
                result = response.ToXml();

                return result;
            }
```

我们来看看我公司的微信门户菜单，看起来是不是很酷呢。

对于这两种（文本消息、图文消息）用的地方是最多，很多微信门户，都主要是使用这两种方式进行响应。当然，我们还可以根据客户手机提交上来的各种消息进行不同的处理，请求消息的类型我在上一篇的随笔有介绍，如下所示。



需要关注了解整体效果，可以使用微信直接扫描二维码即可。

二维码

C#开发微信门户及应用(4)--关注用户列表及详细信息管理

在上个月的对 C#开发微信门户及应用做了介绍，写过了几篇的随笔进行分享，由于时间关系，间隔了一段时间没有继续写这个系列的博客了，并不是对这个方面停止了研究，而是继续深入探索这方面的技术，为了更好的应用起来，专心做好底层的技术开发。

微信的很重要的一个特点就是能够利用其平台庞大的用户群体，因此很容易整合在 CRM（客户关系管理）系统里面，服务号和订阅好都能够向关注者推送相 关的产品消息，还能和 48 小时内响应消息和事件的活跃用户进行交互对话，因此用户信息是微信 API 里面非常重要的一环，本随笔主要介绍获取关注用户、查看 用户信息、分组管理等方面的开发应用。

1、关注用户列表及用户分组信息

在微信的管理平台上，我们可以看到自己账号的关注者用户，以及用户分组信息，如下所示。

上面的管理界面，能看到关注者用户的基础信息，但是使用微信 API 获取到的是一个称之为 OpenID 的列表，我们先了解这个东西是什么？微信 API 的说明给出下面的解析：

关注者列表由一串 OpenID（加密后的微信号，每个用户对每个公众号的 OpenID 是唯一的。对于不同公众号，同一用户的 openid 不同）组成。公众号可通过本接口来根据 OpenID 获取用户基本信息，包括昵称、头像、性别、所在城市、语言和关注时间。

上面的解析意思很清楚了，就是一个用户关注我们的公众号，那么不管他是第几次关注，对我们公众号来说，都是一个确定的值；但是，一个用户对其他公众号，却有着其他不同的 OpenID。

微信提供了为数不多的几个关键字信息，用来记录用户的相关内容，根据用户的相关定义，我们定义一个实体类，用来放置获取回来的用户信息。

```
    /// <summary>
    /// 高级接口获取的用户信息。
    /// 在关注者与公众号产生消息交互后，公众号可获得关注者的 OpenID
    /// （加密后的微信号，每个用户对每个公众号的 OpenID 是唯一的。对于不同公众号，同一用户的 openid 不同）。
    /// 公众号可通过本接口来根据 OpenID 获取用户基本信息，包括昵称、头像、性别、所在城市、语言和关注时间。
    /// </summary>
    public class UserJson : BaseJsonResult
    {
        /// <summary>
        /// 用户是否订阅该公众号标识，值为 0 时，代表此用户没有关注该公众号，拉取不到其余信息。
        /// </summary>
        public int subscribe { get; set; }


        /// <summary>
        /// 用户的标识，对当前公众号唯一
        /// </summary>
```

```
public string openid { get; set; }

/// <summary>
/// 用户的昵称
/// </summary>
public string nickname { get; set; }

/// <summary>
/// 用户的性别，值为 1 时是男性，值为 2 时是女性，值为 0 时是未知
/// </summary>
public int sex { get; set; }

/// <summary>
/// 用户的语言，简体中文为 zh_CN
/// </summary>
public string language { get; set; }

/// <summary>
/// 用户所在城市
/// </summary>
public string city { get; set; }
```

```
        /// <summary>
        /// 用户所在省份
        /// </summary>
        public string province { get; set; }

        /// <summary>
        /// 用户所在国家
        /// </summary>
        public string country { get; set; }

        /// <summary>
        /// 用户头像，最后一个数值代表正方形头像大小（有 0、46、64、96、132 数值可选，0 代表 640*640 正方形头像），用户没有头像时该项为空
        /// </summary>
        public string headimgurl { get; set; }

        /// <summary>
        /// 用户关注时间，为时间戳。如果用户曾多次关注，则取最后关注时间
        /// </summary>
        public long subscribe_time { get; set; }
    }
```

根据分组信息定义，我们定义一个分组的实体类信息。

```
/// <summary>
/// 分组信息
/// </summary>
public class GroupJson : BaseJsonResult
{
    /// <summary>
    /// 分组 id，由微信分配
    /// </summary>
    public int id { get; set; }

    /// <summary>
    /// 分组名字，UTF8 编码
    /// </summary>
    public string name { get; set; }
}
```

2、获取 AIP 调用者的的 Token

在做微信 API 的开发，很多时候，我们都需要传入一个 AccessToken，这个就是区分调用者和记录会话信息的字符串，因此，在学习所有 API 开发之前，我们需要很好理解这个访问控制参数。

这个对象的定义，我们可以从微信的 API 说明中了解。

access_token 是公众号的全局唯一票据，公众号调用各接口时都需使用 access_token。正常情况下 **access_token 有效期为 7200 秒**，重复获取将导致上次获取的 access_token 失效。**由于获取 access_token 的 api 调用次数非常有限，建议开发者全局存储与更新 access_token，频繁刷新 access_token 会导致 api 调用受限，影响自身业务。**

根据上面的说明定义，我们可以看到，它是一个和身份，以及会话时间有关的一个参数，而且它的产生次数有限制，因此要求我们需要对它进行缓存并重复利用，在会话到期之前，我们应该尽可能重用这个参数，避免反复请求，增加服务器压力，以及调用的时间。

我定义了一个方法，用来构造生成相关的 Access Token，而且它具有缓存的功能，但具体如何缓存及使用，对我 API 的调用是透明的，我们只要用的时候，就对它调用就是了。

```
        /// 获取凭证接口
        /// </summary>
        /// <param name="appid">第三方用户唯一凭证</param>
        /// <param name="secret">第三方用户唯一凭证密钥，既appsecret</param>
        string GetAccessToken(string appid, string secret);
```

缓存主要是基于.NET4 增加的类库 MemoryCache，这个是一个非常不错的缓存类。

我的获取 AccessToken 的操作实现代码如下所示。

```
        /// <summary>
        /// 获取每次操作微信 API 的 Token 访问令牌
        /// </summary>
        /// <param name="appid">应用 ID</param>
        /// <param name="secret">开发者凭据</param>
        /// <returns></returns>
        public string GetAccessToken(string appid, string secret)
        {
```

```
            //正常情况下 access_token 有效期为 7200 秒,这里使用缓存设置
短于这个时间即可
            string access_token =
MemoryCacheHelper.GetCacheItem<string>("access_token", delegate()
                {
                    string grant_type = "client_credential";
                    var url =
string.Format("https://api.weixin.qq.com/cgi-bin/token?grant_type={0}
&appid={1}&secret={2}",
                                                    grant_type, appid, secret);

                    HttpHelper helper = new HttpHelper();
                    string result = helper.GetHtml(url);
                    string regex = "\"access_token\":\"(?<token>.*?)\"";
                    string token = CRegex.GetText(result, regex,
"token");
                    return token;
                },
                new TimeSpan(0, 0, 7000)//7000 秒过期
            );

            return access_token;
```

```
        }
```

由于我们知道，AccessToken 默认是 7200 秒过期，因此在这个时间段里面，我们尽可能使用缓存来记录它的值，如果超过了这个时间，我们调用这个方法的时候，它会自动重新获取一个新的值给我们了。

3、获取关注用户列表

获取关注用户列表，一次拉取 API 调用，最多拉取 10000 个关注者的 OpenID，可以通过多次拉取的方式来满足需求。微信的接口定义如下所示。

http 请求方式: GET（请使用 https 协议）

https://api.weixin.qq.com/cgi-bin/user/get?access_token=ACCESS_TOKEN&next_openid=NEXT_OPENID

这个接口返回的数据是

{"total":2,"count":2,"data":{"openid":["","OPENID1","OPENID2"]},"next_openid":"NEXT_OPENID"}

根据返回的 Json 数据定义，我们还需要定义两个实体类，用来存放返回的结果。

```
    /// <summary>
    /// 获取关注用户列表的 Json 结果
```

```
/// </summary>
public class UserListJsonResult : BaseJsonResult
{
    /// <summary>
    /// 关注该公众账号的总用户数
    /// </summary>
    public int total { get; set; }

    /// <summary>
    /// 拉取的 OPENID 个数，最大值为 10000
    /// </summary>
    public int count { get; set; }

    /// <summary>
    /// 列表数据，OPENID 的列表
    /// </summary>
    public OpenIdListData data { get; set; }

    /// <summary>
    /// 拉取列表的后一个用户的 OPENID
    /// </summary>
    public string next_openid { get; set; }
```

```
}


/// <summary>
/// 列表数据，OPENID 的列表
/// </summary>
public class OpenIdListData
{
    /// <summary>
    /// OPENID 的列表
    /// </summary>
    public List<string> openid { get; set; }
}
```

为了获取相关的用户信息，我定义了一个接口，用来获取用户的信息，接口定义如下所示。

```
/// <summary>
/// 微信用户管理的 API 接口
/// </summary>
public interface IUserApi
{
    /// <summary>
```

```
        /// 获取关注用户列表
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="nextOpenId">第一个拉取的 OPENID，不填默认从头开始拉取</param>
        /// <returns></returns>
        List<string> GetUserList(string accessToken, string nextOpenId = null);


        /// <summary>
        /// 获取用户基本信息
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="openId">普通用户的标识，对当前公众号唯一</param>
        /// <param name="lang">返回国家地区语言版本，zh_CN 简体，zh_TW 繁体，en 英语</param>
        UserJson GetUserDetail(string accessToken, string openId, Language lang = Language.zh_CN);
```

然后在实现类里面，我们分别对上面两个接口进行实现，获取用户列表信息如下所示。

```
        /// <summary>
        /// 获取关注用户列表
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="nextOpenId">第一个拉取的 OPENID，不填默认从头开始拉取</param>
        /// <returns></returns>
        public List<string> GetUserList(string accessToken, string nextOpenId = null)
        {
            List<string> list = new List<string>();

            string url = string.Format("https://api.weixin.qq.com/cgi-bin/user/get?access_token={0}", accessToken);
            if (!string.IsNullOrEmpty(nextOpenId))
            {
                url += "&next_openid=" + nextOpenId;
            }

            UserListJsonResult result = JsonHelper<UserListJsonResult>.ConvertJson(url);
```

```
            if (result != null && result.data != null)
            {
                list.AddRange(result.data.openid);
            }

            return list;
        }
```

我们看到，转换的逻辑已经放到了 JsonHelper 里面去了，这个辅助类里面分别对数值进行了获取内容，验证返回值，然后转换正确实体类几个部分的操作。

获取内容，通过辅助类 HttpHelper 进行，这个在我的公用类库里面，里面的逻辑主要就是通过 HttpRequest 进行数据的获取操作，不在赘述。

```
HttpHelper helper = new HttpHelper();
string content = helper.GetHtml(url);
```

由于返回的内容，我们需要判断它是否正确返回所需的结果，如果没有，抛出自定义的相关异常，方便处理，具体如下所示。

```
        /// <summary>
        /// 检查返回的记录，如果返回没有错误，或者结果提示成功，则不抛出异常
        /// </summary>
```

```
        /// <param name="content">返回的结果</param>
        /// <returns></returns>
        private static bool VerifyErrorCode(string content)
        {
            if (content.Contains("errcode"))
            {
                ErrorJsonResult errorResult = JsonConvert.DeserializeObject<ErrorJsonResult>(content);
                //非成功操作才记录异常，因为有些操作是返回正常的结果 ( {"errcode": 0, "errmsg": "ok"} )
                if (errorResult != null && errorResult.errcode != ReturnCode.请求成功)
                {
                    string error = string.Format("微信请求发生错误！错误代码：{0}，说明：{1}", (int)errorResult.errcode, errorResult.errmsg);
                    LogTextHelper.Error(errorResult);

                    throw new WeixinException(error);//抛出错误
                }
            }
            return true;
        }
```

然后转换为相应的格式，就是通过 Json.NET 的类库进行转换。

```
T result = JsonConvert.DeserializeObject<T>(content);
return result;
```

这样我们就可以在 ConvertJson 函数实体里面，完整的进行处理和转换了，转换完整的函数代码如下所示。

```
/// <summary>
/// Json 字符串操作辅助类
/// </summary>
public class JsonHelper<T> where T : class, new()
{
    /// <summary>
    /// 检查返回的记录，如果返回没有错误，或者结果提示成功，则不抛出异常
    /// </summary>
    /// <param name="content">返回的结果</param>
    /// <returns></returns>
    private static bool VerifyErrorCode(string content)
    {
        if (content.Contains("errcode"))
```

```
            {
                ErrorJsonResult errorResult =
JsonConvert.DeserializeObject<ErrorJsonResult>(content);
                //非成功操作才记录异常，因为有些操作是返回正常的结果
( {"errcode": 0, "errmsg": "ok"} )
                if (errorResult != null && errorResult.errcode !=
ReturnCode.请求成功)
                {
                    string error = string.Format("微信请求发生错误！错误
代码：{0}，说明：{1}", (int)errorResult.errcode, errorResult.errmsg);
                    LogTextHelper.Error(errorResult);

                    throw new WeixinException(error);//抛出错误
                }
            }
            return true;
        }

        /// <summary>
        /// 转换 Json 字符串到具体的对象
        /// </summary>
        /// <param name="url">返回 Json 数据的链接地址</param>
```

```
        /// <returns></returns>
        public static T ConvertJson(string url)
        {
            HttpHelper helper = new HttpHelper();
            string content = helper.GetHtml(url);
            VerifyErrorCode(content);

            T result = JsonConvert.DeserializeObject<T>(content);
            return result;
        }
}
```

调用这个 API 的界面层代码如下所示（测试代码）

```
            IUserApi userBLL = new UserApi();
            List<string> userList = userBLL.GetUserList(token)
```

4、获取用户详细信息

上面的获取列表操作，相对比较简单，而且不用 POST 任何数据，因此通过 Get 协议就能获取到所需的数据。

本小节继续介绍获取用户详细信息的操作，这个操作也是通过 GET 协议就可以完成的。

这个 API 的调用定义如下所示：

http 请求方式: GET

https://api.weixin.qq.com/cgi-bin/user/info?access_token=ACCESS_TOKEN&openid=OPENID&lang=zh_CN

通过传入一个 OpenId，我们就能很好获取到用户的相关信息了。

前面小节我们已经定义了它的接口，说明了传入及返回值，根据定义，它的实现函数如下所示。

```
        /// <summary>
        /// 获取用户基本信息
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="openId">普通用户的标识，对当前公众号唯一</param>
        /// <param name="lang">返回国家地区语言版本，zh_CN 简体，zh_TW 繁体，en 英语</param>
        public UserJson GetUserDetail(string accessToken, string openId, Language lang = Language.zh_CN)
```

```
        {
            string url =
string.Format("https://api.weixin.qq.com/cgi-bin/user/info?access_token
={0}&openid={1}&lang={2}",
                        accessToken, openId, lang.ToString());

            UserJson result = JsonHelper<UserJson>.ConvertJson(url);
            return result;
        }
```

最后，我们结合获取用户列表和获取用户详细信息的两个 API，我们看看调用的代码（测试代码）。

```
        private void btnGetUsers_Click(object sender, EventArgs e)
        {
            IUserApi userBLL = new UserApi();
            List<string> userList = userBLL.GetUserList(token);
            foreach (string openId in userList)
            {
                UserJson userInfo = userBLL.GetUserDetail(token,
openId);
                if (userInfo != null)
```

```
                {
                    string tips = string.Format("{0}:{1}",
userInfo.nickname, userInfo.openid);
                    Console.WriteLine(tips);
                }
            }
        }
```

C#开发微信门户及应用(5)--用户分组信息管理

在上个月的对 C#开发微信门户及应用做了介绍，写过了几篇的随笔进行分享，由于时间关系 ,间隔了一段时间没有继续写这个系列的博客了 ,并不是对这个 方面停止了研究，而是继续深入探索这方面的技术，为了更好的应用起来，专心做好底层的技术开发。本篇继续上一篇的介绍 ,主要介绍分组管理方面的开发应用，这篇的内容和上一篇，作为一个完整的用户信息和分组信息管理的组合。

1、用户分组管理内容

用户分组的引入，主要是方便管理关注者列表，以及方便向不同的组别发送消息的操作的，一个公众账号，最多支持创建 500 个分组。

用户分组管理，包含下面几个方面的内容：

1 创建分组

2 查询所有分组

3 查询用户所在分组

4 修改分组名

5 移动用户分组

微信对于创建分组的定义如下所示。

http 请求方式: POST（请使用 https 协议）

https://api.weixin.qq.com/cgi-bin/groups/create?access_token=ACCESS_TOKEN

POST 数据格式：json

POST 数据例子：{"group":{"name":"test"}}

正常返回的结果如下所示。


```
{
    "group": {
        "id": 107,
        "name": "test"
    }
}
```

其他接口，也是类似的方式，通过 POST 一些参数进去 URL 里面，获取返回的 Json 数据。

前面随笔定义了 GroupJson 的实体类信息如下所示。

```
/// <summary>
/// 分组信息
/// </summary>
public class GroupJson : BaseJsonResult
{
    /// <summary>
    /// 分组 id，由微信分配
    /// </summary>
    public int id { get; set; }

    /// <summary>
    /// 分组名字，UTF8 编码
    /// </summary>
    public string name { get; set; }
}
```

根据以上几个接口的定义，我定义了几个接口，并把它们归纳到用户管理的 API 接口里面。

```
/// <summary>
/// 查询所有分组
/// </summary>
/// <param name="accessToken">调用接口凭证</param>
/// <returns></returns>
List<GroupJson> GetGroupList(string accessToken);

/// <summary>
/// 创建分组
/// </summary>
/// <param name="accessToken">调用接口凭证</param>
/// <param name="name">分组名称</param>
/// <returns></returns>
GroupJson CreateGroup(string accessToken, string name);

/// <summary>
/// 查询用户所在分组
/// </summary>
/// <param name="accessToken">调用接口凭证</param>
/// <param name="openid">用户的 OpenID</param>
/// <returns></returns>
int GetUserGroupId(string accessToken, string openid);
```

```
        /// <summary>
        /// 修改分组名
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="id">分组 id，由微信分配</param>
        /// <param name="name">分组名字（30 个字符以内）</param>
        /// <returns></returns>
        CommonResult UpdateGroupName(string accessToken, int id,
string name);

        /// <summary>
        /// 移动用户分组
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="openid">用户的 OpenID</param>
        /// <param name="to_groupid">分组 id</param>
        /// <returns></returns>
        CommonResult MoveUserToGroup(string accessToken, string
openid, int to_groupid);
```

2、用户分组管理接口的实现

2.1 创建用户分组

为了解析如何实现创建用户分组的 POST 数据操作 ,我们来一步步了解创建用户的具体过程。

首先需要创建一个动态定义的实体类信息，它包含几个需要提及的属性，如下所示。

```
            string url =
string.Format("https://api.weixin.qq.com/cgi-bin/groups/create?access_t
oken={0}", accessToken);
            var data = new
            {
                group = new
                {
                    name = name
                }
            };
            string postData = data.ToJson();
```

其中我们把对象转换为合适的 Json 数据操作 ,放到了扩展方法 ToJson 里面了，这个主要就是方便把动态定义的实体类转换 Json 内容 ,主要就是调用 Json.NET 的序列号操作。

```
        /// <summary>
        /// 把对象为 json 字符串
        /// </summary>
        /// <param name="obj">待序列号对象</param>
        /// <returns></returns>
        public static string ToJson(this object obj)
        {
            return JsonConvert.SerializeObject(obj,
Formatting.Indented);
        }
```

准备好 Post 的数据后，我们就进一步看看获取数据并转换为合适格式的操作代码。

```
            GroupJson group = null;
            CreateGroupResult result =
JsonHelper<CreateGroupResult>.ConvertJson(url, postData);
            if (result != null)
```

```
        {
            group = result.group;
        }
```

其中 POST 数据并转换为合适格式实体类的操作，放在了 ConvertJson 方法里面，这个方法的定义如下所示，里面的 HttpHelper 是我公用类库的辅助类，主要就是调用底层的 httpWebRequest 对象方法，进行数据的提交，并获取返回结果。

```
        /// <summary>
        /// 转换 Json 字符串到具体的对象
        /// </summary>
        /// <param name="url">返回 Json 数据的链接地址</param>
        /// <param name="postData">POST 提交的数据</param>
        /// <returns></returns>
        public static T ConvertJson(string url, string postData)
        {
            HttpHelper helper = new HttpHelper();
            string content = helper.GetHtml(url, postData, true);
            VerifyErrorCode(content);

            T result = JsonConvert.DeserializeObject<T>(content);
```

```
            return result;
        }
```

这样，完整的创建用户分组的操作函数如下所示。

```
        /// <summary>
        /// 创建分组
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="name">分组名称</param>
        /// <returns></returns>
        public GroupJson CreateGroup(string accessToken, string name)
        {
            string url =
string.Format("https://api.weixin.qq.com/cgi-bin/groups/create?access_t
oken={0}", accessToken);
            var data = new
            {
                group = new
                {
                    name = name
                }
```

```
            };
            string postData = data.ToJson();

            GroupJson group = null;
            CreateGroupResult result =
JsonHelper<CreateGroupResult>.ConvertJson(url, postData);
            if (result != null)
            {
                group = result.group;
            }
            return group;
        }
```

2.2 查询所有分组

查询所有分组，可以把服务器上的分组全部获取下来，也就是每个分组的 ID 和名称。

```
        /// <summary>
        /// 查询所有分组
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <returns></returns>
```

```
        public List<GroupJson> GetGroupList(string accessToken)
        {
            string url =
string.Format("https://api.weixin.qq.com/cgi-bin/groups/get?access_tok
en={0}", accessToken);

            List<GroupJson> list = new List<GroupJson>();
            GroupListJsonResult result =
JsonHelper<GroupListJsonResult>.ConvertJson(url);
            if (result != null && result.groups != null)
            {
                list.AddRange(result.groups);
            }

            return list;
        }
```

2.3 查询用户所在分组

每个用户都属于一个分组，默认在 **未分组** 这个分组里面，我们可以通过 API 获取用户的分组信息，也就是获取所在用户分组的 ID。

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

```
        /// 查询用户所在分组
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="openid">用户的 OpenID</param>
        /// <returns></returns>
        public int GetUserGroupId(string accessToken, string openid)
        {
            string url =
string.Format("https://api.weixin.qq.com/cgi-bin/groups/getid?access_t
oken={0}", accessToken);
            var data = new
            {
                openid = openid
            };
            string postData = data.ToJson();

            int groupId = -1;
            GroupIdJsonResult result =
JsonHelper<GroupIdJsonResult>.ConvertJson(url, postData);
            if (result != null)
            {
                groupId = result.groupid;
```

```
            }
            return groupId;
        }
```

2.4 修改分组名称

也可以在实际中，调整用户所在的分组，操作代码如下。

```
        /// <summary>
        /// 修改分组名
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="id">分组 id，由微信分配</param>
        /// <param name="name">分组名字（30 个字符以内）</param>
        /// <returns></returns>
        public CommonResult UpdateGroupName(string accessToken,
int id, string name)
        {
            string url =
string.Format("https://api.weixin.qq.com/cgi-bin/groups/update?access
_token={0}", accessToken);
            var data = new
            {
```

```
            group = new
            {
                id = id,
                name = name
            }
        };
        string postData = data.ToJson();

        return Helper.GetExecuteResult(url, postData);
    }
```

这里的返回值 CommonResult 是，一个实体类，包含了 bool 的成功与否的标志，以及 String 类型的错误信息（如果有的话）。

对于这个 GetExecuteResult 函数体，里面主要就是提交数据，然后获取结果，并根据结果进行处理的函数。

```
    /// <summary>
    /// 通用的操作结果
    /// </summary>
    /// <param name="url">网页地址</param>
    /// <param name="postData">提交的数据内容</param>
    /// <returns></returns>
```

```
        public static CommonResult GetExecuteResult(string url, string
postData = null)
        {
            CommonResult success = new CommonResult();
            try
            {
                ErrorJsonResult result;
                if (postData != null)
                {
                    result =
JsonHelper<ErrorJsonResult>.ConvertJson(url, postData);
                }
                else
                {
                    result =
JsonHelper<ErrorJsonResult>.ConvertJson(url);
                }

                if (result != null)
                {
                    success.Success = (result.errcode == ReturnCode.
请求成功);
```

```
                success.ErrorMessage = result.errmsg;
            }
        }
        catch (WeixinException ex)
        {
            success.ErrorMessage = ex.Message;
        }

        return success;
    }
}
```

上面红色部分的意思，就是转换为实体类的时候，如果错误是微信里面定义的，那么记录错误信息，其他异常我不处理（也就是抛出去）。

2.5 移动用户到新的分组

移动用户到新的分组的操作和上面小节的差不多，具体看代码。

```
/// <summary>
/// 移动用户分组
/// </summary>
/// <param name="accessToken">调用接口凭证</param>
```

```
        /// <param name="openid">用户的 OpenID</param>
        /// <param name="to_groupid">分组 id</param>
        /// <returns></returns>
        public CommonResult MoveUserToGroup(string accessToken,
string openid, int to_groupid)
        {
            string url =
string.Format("https://api.weixin.qq.com/cgi-bin/groups/members/upd
ate?access_token={0}", accessToken);
            var data = new
            {
                openid = openid,
                to_groupid = to_groupid
            };
            string postData = data.ToJson();

            return Helper.GetExecuteResult(url, postData);
        }
```

3、用户分组接口的调用

上面小节，定义并实现了用户分组的各类接口，所有的用户相关的都已经毫无保留贴出代码，它的调用操作如下代码所示（测试代码）。

```
        private void btnGetGroupList_Click(object sender, EventArgs e)
        {
            IUserApi userBLL = new UserApi();
            List<GroupJson> list = userBLL.GetGroupList(token);
            foreach (GroupJson info in list)
            {
                string tips = string.Format("{0}:{1}", info.name, info.id);
                Console.WriteLine(tips);
            }
        }

        private void btnFindUserGroup_Click(object sender, EventArgs
e)
        {
            IUserApi userBLL = new UserApi();
            int groupId = userBLL.GetUserGroupId(token, openId);

            string tips = string.Format("GroupId:{0}", groupId);
```

```
            Console.WriteLine(tips);
        }

        private void btnCreateGroup_Click(object sender, EventArgs e)
        {
            IUserApi userBLL = new UserApi();
            GroupJson info = userBLL.CreateGroup(token, "创建测试分组
");
            if (info != null)
            {
                string tips = string.Format("GroupId:{0}
GroupName:{1}", info.id, info.name);
                Console.WriteLine(tips);

                string newName = "创建测试修改";
                CommonResult result =
userBLL.UpdateGroupName(token, info.id, newName);
                Console.WriteLine("修改分组名称：" + (result.Success ? "
成功" : "失败:" + result.ErrorMessage));
            }
        }
```

```
        private void btnUpdateGroup_Click(object sender, EventArgs e)
        {
            int groupId = 111;
            string newName = "创建测试修改";

            IUserApi userBLL = new UserApi();
            CommonResult result = userBLL.UpdateGroupName(token,
groupId, newName);
            Console.WriteLine("修改分组名称：" + (result.Success ? "成功
" : "失败:" + result.ErrorMessage));
        }

        private void btnMoveToGroup_Click(object sender, EventArgs e)
        {
            int togroup_id = 111;//输入分组 ID

            if (togroup_id > 0)
            {
                IUserApi userBLL = new UserApi();
                CommonResult result =
userBLL.MoveUserToGroup(token, openId, togroup_id);
```

```
                Console.WriteLine("移动用户分组名称：" +
(result.Success ? "成功" : "失败:" + result.ErrorMessage));
            }
        }
```

了解了上面的代码和调用规则 ,我们就能通过 API 进行用户分组信息的管理了。通过在应用程序中集成相关的接口代码 ,我们就能够很好的控制我们的关注用户列表和用户分组信息。从而为我们下一步用户的信息推送打好基础。

C#开发微信门户及应用(6)--微信门户菜单的管理操作

前面几篇继续了我自己对于 C#开发微信门户及应用的技术探索和相关的经验总结，继续探索微信 API 并分享相关的技术，一方面是为了和大家对这方面进行互动沟通，另一方面也是专心做好微信应用的底层技术开发，把基础模块夯实，在未来的应用中派上用途。本随笔继续介绍微信门户菜单的管理操作。

1、菜单的基础信息

微信门户的菜单，一般服务号和订阅号都可以拥有这个模块的开发，但是订阅号好像需要认证后才能拥有，而服务号则不需要认证就可以拥有 了。这个菜单可以有编辑模式和开发模式，编辑模式主要就是在微信门户的平台上，对菜单进行编辑；而开发模式，就是用户可以通过调用微信的 API 对菜单进行 定制开发，

通过 POST 数据到微信服务器，从而生成对应的菜单内容。本文主要介绍基于开发模式的菜单管理操作。

自定义菜单能够帮助公众号丰富界面，让用户更好更快地理解公众号的功能。目前自定义菜单最多包括 3 个一级菜单，每个一级菜单最多包含 5 个二级菜单。一级菜单最多 4 个汉字，二级菜单最多 7 个汉字，多出来的部分将会以“...”代替。目前自定义菜单接口可实现两种类型按钮，如下：

click：

用户点击 click 类型按钮后，微信服务器会通过消息接口推送消息类型为 event 的结构给开发者（参考消息接口指南），并且带上按钮中开发者填写的 key 值，开发者可以通过自定义的 key 值与用户进行交互；

view：

用户点击 view 类型按钮后，微信客户端将会打开开发者在按钮中填写的 url 值（即网页链接），达到打开网页的目的，建议与网页授权获取用户基本信息接口结合，获得用户的登入个人信息。

菜单提交的数据 ,本身是一个 Json 的数据字符串 ,它的官方例子数据如下所示。

```
{
    "button":[
    {
        "type":"click",
        "name":"今日歌曲",
```

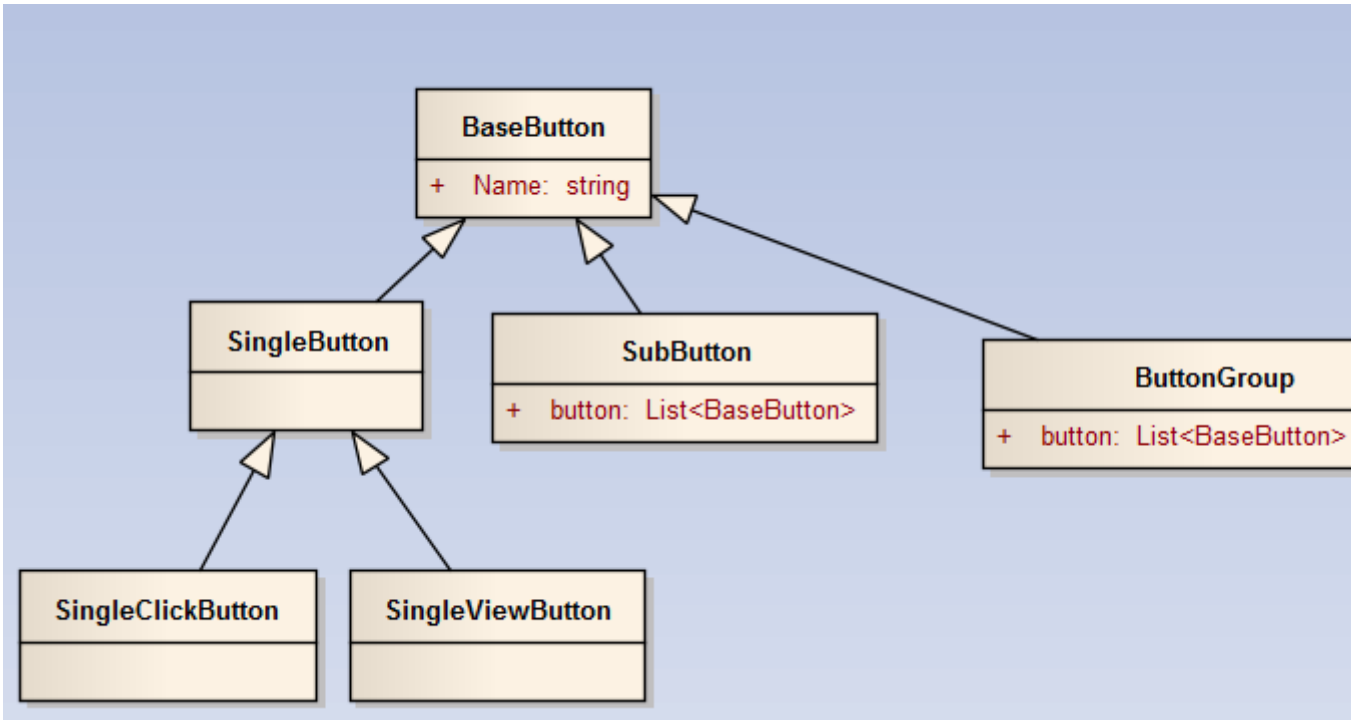
```
        "key":"V1001_TODAY_MUSIC"
    },
    {
        "type":"click",
        "name":"歌手简介",
        "key":"V1001_TODAY_SINGER"
    },
    {
        "name":"菜单",
        "sub_button":[
        {
            "type":"view",
            "name":"搜索",
            "url":"http://www.soso.com/"
         },
         {
            "type":"view",
            "name":"视频",
            "url":"http://v.qq.com/"
         },
         {
            "type":"click",
```

```
                "name":"赞一下我们",
                "key":"V1001_GOOD"
            }]
        }]
}
```

从上面我们可以看到，菜单不同的 type 类型，有不同的字段内容，如 type 为 view 的有 url 属性，而 type 为 click 的，则有 key 属性。而菜单可以有子菜单 sub_button 属性，总得来说，为了构造好对应的菜单实体类信息，不是一下就能分析的出来。

2、菜单的实体类定义

我看过一些微信接口的开发代码，把菜单的分为了好多个实体类，指定了继承关系，然后分别对他们进行属性的配置，大概的关系如下所示。



这种多层关系的继承方式能解决问题，不过我觉得并不是优雅的解决方案。其实结合 Json.NET 自身的 Attribute 属性配置，可以指定那些为空的内容在序列号为 Json 字符串的时候，不显示出来的。

[JsonProperty(NullValueHandling = NullValueHandling.Ignore)]

有了这个属性，我们就可以统一定义菜单的实体类信息更多的属性了，可以把 View 类型和 Click 类型的菜单属性的 url 和 key 合并在一起。

```
/// <summary>
/// 菜单基本信息
/// </summary>
public class MenuInfo
{
```

```
        /// <summary>
        /// 按钮描述，既按钮名字，不超过 16 个字节，子菜单不超过 40 个
字节
        /// </summary>
        public string name { get; set; }

        /// <summary>
        /// 按钮类型（click 或 view）
        /// </summary>
        [JsonProperty(NullValueHandling = NullValueHandling.Ignore)]
        public string type { get; set; }

        /// <summary>
        /// 按钮 KEY 值，用于消息接口(event 类型)推送，不超过 128 字节
        /// </summary>
        [JsonProperty(NullValueHandling = NullValueHandling.Ignore)]
        public string key { get; set; }

        /// <summary>
        /// 网页链接，用户点击按钮可打开链接，不超过 256 字节
        /// </summary>
        [JsonProperty(NullValueHandling = NullValueHandling.Ignore)]
```

```
        public string url { get; set; }


        /// <summary>
        /// 子按钮数组，按钮个数应为 2~5 个
        /// </summary>
        [JsonProperty(NullValueHandling = NullValueHandling.Ignore)]
        public List<MenuInfo> sub_button { get; set; }
```

.....

但是，这么多信息，不同的类型我需要指定不同的属性类型，那不是挺麻烦，万一我在 View 类型的菜单里面，把 key 属性设置了，那怎么办？

解决方法就是我们定义几个构造函数，分别用来构造不同的菜单信息，如下所示是对菜单不同的类型，赋值给不同的属性的构造函数。

```
        /// <summary>
        /// 参数化构造函数
        /// </summary>
        /// <param name="name">按钮名称</param>
        /// <param name="buttonType">菜单按钮类型</param>
        /// <param name="value">按钮的键值（Click），或者连接URL(View)</param>
```

```
        public MenuInfo(string name, ButtonType buttonType, string
value)
        {
            this.name = name;
            this.type = buttonType.ToString();

            if (buttonType == ButtonType.click)
            {
                this.key = value;
            }
            else if(buttonType == ButtonType.view)
            {
                this.url = value;
            }
        }
```

好了，还有另外一个问题，子菜单也就是属性 sub_button 是可有可无的东西，有的话，需要指定 Name 属性，并添加它的 sub_button 集合对象就可以了，那么我们在增加一个构造子菜单的对象信息的构造函数。

```
        /// <summary>
        /// 参数化构造函数,用于构造子菜单
```

```
        /// </summary>
        /// <param name="name">按钮名称</param>
        /// <param name="sub_button">子菜单集合</param>
        public MenuInfo(string name, IEnumerable<MenuInfo>
sub_button)
        {
            this.name = name;
            this.sub_button = new List<MenuInfo>();
            this.sub_button.AddRange(sub_button);
        }
```

由于只指定 Name 和 sub_button 的属性内容，其他内容为 null 的话，自然构造出来的 Json 就没有包含它们，非常完美！

为了获取菜单的信息，我们还需要定义两个实体对象，如下所示。

```
    /// <summary>
    /// 菜单的 Json 字符串对象
    /// </summary>
    public class MenuJson
    {
        public List<MenuInfo> button { get; set; }
```

```
        public MenuJson()
        {
            button = new List<MenuInfo>();
        }
    }

    /// <summary>
    /// 菜单列表的 Json 对象
    /// </summary>
    public class MenuListJson
    {
        public MenuJson menu { get; set; }
    }
```

3、菜单管理操作的接口实现

我们从微信的定义里面，可以看到，我们通过 API 可以获取菜单信息、创建菜单、删除菜单，那么我们来定义它们的接口如下。

```
    /// <summary>
    /// 菜单的相关操作
    /// </summary>
    public interface IMenuApi
```

```
{
        /// <summary>
        /// 获取菜单数据
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <returns></returns>
        MenuJson GetMenu(string accessToken);

        /// <summary>
        /// 创建菜单
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="menuJson">菜单对象</param>
        /// <returns></returns>
        CommonResult CreateMenu(string accessToken, MenuJson
menuJson);

        /// <summary>
        /// 删除菜单
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <returns></returns>
```

```
        CommonResult DeleteMenu(string accessToken);
    }
```

具体的获取菜单信息的实现如下。

```
        /// <summary>
        /// 获取菜单数据
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <returns></returns>
        public MenuJson GetMenu(string accessToken)
        {
            MenuJson menu = null;

            var url =
string.Format("https://api.weixin.qq.com/cgi-bin/menu/get?access_toke
n={0}", accessToken);
            MenuListJson list =
JsonHelper<MenuListJson>.ConvertJson(url);
            if (list != null)
            {
                menu = list.menu;
```

```
            }
            return menu;
        }
```

这里就是把返回的 Json 数据，统一转换为我们需要的实体信息了，一步到位。

调用代码如下所示。

```
        private void btnGetMenuJson_Click(object sender, EventArgs e)
        {
            IMenuApi menuBLL = new MenuApi();
            MenuJson menu = menuBLL.GetMenu(token);
            if (menu != null)
            {
                Console.WriteLine(menu.ToJson());
            }
        }
```

创建和删除菜单对象的操作实现如下所示。

```
        /// <summary>
        /// 创建菜单
        /// </summary>
```

```
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="menuJson">菜单对象</param>
        /// <returns></returns>
        public CommonResult CreateMenu(string accessToken,
MenuJson menuJson)
        {
            var url =
string.Format("https://api.weixin.qq.com/cgi-bin/menu/create?access_to
ken={0}", accessToken);
            string postData = menuJson.ToJson();

            return Helper.GetExecuteResult(url, postData);
        }

        /// <summary>
        /// 删除菜单
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <returns></returns>
        public CommonResult DeleteMenu(string accessToken)
        {
```

```
            var url =
string.Format("https://api.weixin.qq.com/cgi-bin/menu/delete?access_t
oken={0}", accessToken);

            return Helper.GetExecuteResult(url);
        }
```

看到这里，有些人可能会问，实体类你简化了，那么创建菜单是不是挺麻烦的，特别是构造对应的信息应该如何操作呢？前面不是介绍了不同的构造函数了吗，通过他们简单就搞定了，不用记下太多的实体类及它们的继承关系来处理菜单信息。

```
        private void btnCreateMenu_Click(object sender, EventArgs e)
        {
            MenuInfo productInfo = new MenuInfo("软件产品", new
MenuInfo[] {
                new MenuInfo("病人资料管理系统", ButtonType.click,
"patient"),
                new MenuInfo("客户关系管理系统", ButtonType.click,
"crm"),
                new MenuInfo("酒店管理系统", ButtonType.click,
"hotel"),
```

```
                new MenuInfo("送水管理系统", ButtonType.click,
"water")
            });

            MenuInfo frameworkInfo = new MenuInfo("框架产品", new
MenuInfo[] {
                new MenuInfo("Win 开发框架", ButtonType.click, "win"),
                new MenuInfo("WCF 开发框架", ButtonType.click, "wcf"),
                new MenuInfo("混合式框架", ButtonType.click, "mix"),
                new MenuInfo("Web 开发框架", ButtonType.click,
"web"),
                new MenuInfo("代码生成工具", ButtonType.click,
"database2sharp")
            });

            MenuInfo relatedInfo = new MenuInfo("相关链接", new
MenuInfo[] {
                new MenuInfo("公司介绍", ButtonType.click,
"Event_Company"),
                new MenuInfo("官方网站", ButtonType.view,
"http://www.iqidi.com"),
```

```
                new MenuInfo("提点建议", ButtonType.click,
"Event_Suggestion"),
                new MenuInfo("联系客服", ButtonType.click,
"Event_Contact"),
                new MenuInfo("发邮件", ButtonType.view,
"http://mail.qq.com/cgi-bin/qm_share?t=qm_mailme&email=S31yfX15f
n8LOjplKCQm")
            });

            MenuJson menuJson = new MenuJson();
            menuJson.button.AddRange(new MenuInfo[] { productInfo,
frameworkInfo, relatedInfo });

            //Console.WriteLine(menuJson.ToJson());

            if (MessageUtil.ShowYesNoAndWarning("您确认要创建菜单
吗") == System.Windows.Forms.DialogResult.Yes)
            {
                IMenuApi menuBLL = new MenuApi();
                CommonResult result = menuBLL.CreateMenu(token,
menuJson);
```

```
                Console.WriteLine("创建菜单：" + (result.Success ? "成功
" : "失败:" + result.ErrorMessage));
            }
        }
```

这个就是我微信门户里面的菜单操作了，具体效果可以关注我的微信门户：广州爱奇迪，也可以扫描下面二维码进行关注了解。

菜单的效果如下：

C#开发微信门户及应用(7)-微信多客服功能及开发集成

最近一直在弄微信的集成功能开发，发现微信给认证账户开通了一个多客服的功能，对于客户的咨询，可以切换至客服处理的方式，而且可以添加多个客服进 行处理，这个在客户咨询比较多的时候，是一个不错的营销功能。微信多客服的功

能，能够在很大程度上利用客服员工资源，及时迅速对客户咨询信息进行处理，为 企业带来更多的机会和市场。

默认这个多客服的功能，需要在微信公众平台中的服务中心进行主动开通，默认是不开通的，为了体验这个功能，我这里把多客服功能进行开通。

1、多客服准备工作

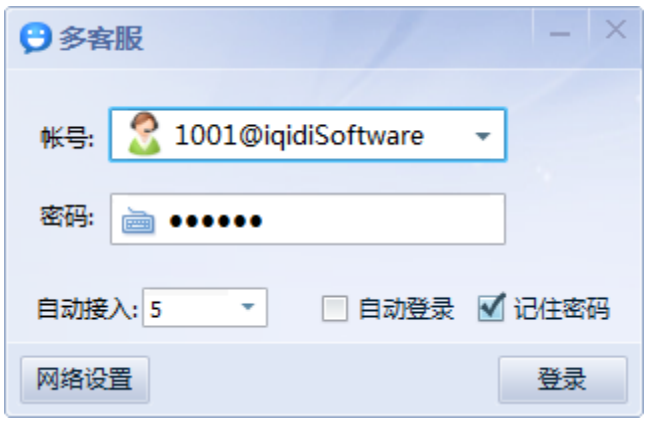
微信的多客服功能，对于客服的响应操作，既可以在电脑的客户端上进行操作，也可以在微信多客服助手进行信息处理，两者都能对客户的信息进行回应、结束会话等操作。

开通微信多客服功能后，就需要添加一些处理客户信息的客服工号了。

多客服账号采用“工号@微信号”的形式进行登录，请您在登录窗口依照下图形式输入帐号信息。

2、使用多客服客户端或助手操作

在电脑客户端上使用



在手机客户端上进行多客服的使用，就是关注一个账号，信息通过转发到这里进行处理。关注公众号”多客服助手“就搞定了。

通过上面两种途径，能够很好处理客户的相关信息，其实也就是类似电话坐席的方式，让不同的客服员工，对来访的客户进行处理。

3、微信多客服的开发使用

在微信的多客服开发介绍中，内容介绍的比较少，如下所示。

在新的微信协议中，开发模式也可以接入客服系统。 开发者如果需要使用客服系统，需要在接收到用户发送的消息时，返回一个 MsgType 为 transfer_customer_service 的消息，微信 服务器在收到这条消息时，会把用户这次发送的和以后一段时间内发送的消息转发客服系统。返回的消息举例如下。

```
<xml>
<ToUserName><![CDATA[touser]]></ToUserName>
<FromUserName><![CDATA[fromuser]]></FromUserName>
<CreateTime>1399197672</CreateTime>
<MsgType><![CDATA[transfer_customer_service]]></MsgType>
</xml>
```

而在开发的时候，我们一般把它封装为一个实体类信息，如下所示。主要就是指定消息类型，和翻转传入传出对象就可以了。

```
    /// <summary>
    /// 客服消息
```

```
    /// </summary>
    [System.Xml.Serialization.XmlRoot(ElementName = "xml")]
    public class ResponseCustomer : BaseMessage
    {
        public ResponseCustomer()
        {
            this.MsgType =
ResponseMsgType.transfer_customer_service.ToString().ToLower();
        }

        public ResponseCustomer(BaseMessage info) : this()
        {
            this.FromUserName = info.ToUserName;
            this.ToUserName = info.FromUserName;
        }
    }
```

然后调用处理的时候，代码如下所示。

```
 ResponseCustomer customInfo = new ResponseCustomer(info);
 xml = customInfo.ToXml();
```

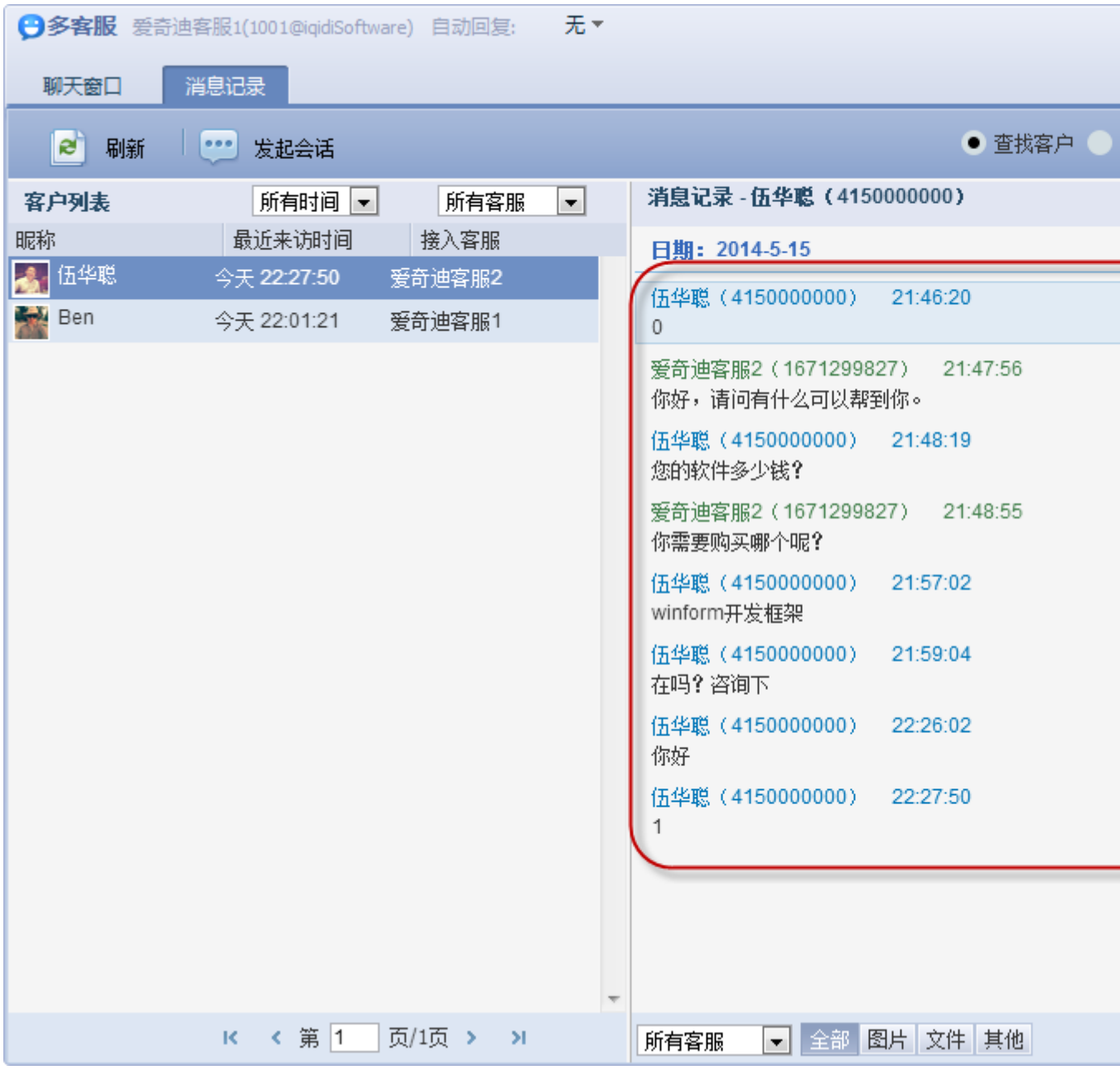
如我在客户应答处理里面，客户回应 0，我就切换进入客服模式，这样客户后续所有的输入内容，均不会触发微信门户里面的解析，而转发到客服模式，让客服的工号可以和客户进行交谈了。

```
                //处理 0 指令，人工客服
                if (string.IsNullOrEmpty(xml) && eventKey.Trim() ==
"0")
                {
                    xml = base.DealEvent(eventInfo,
"event_customservice");
                }
```

而在 DealEvent 里面，根据这个条件进行处理就可以了。

```
                //人工客服
                if (eventKey == "event_customservice")
                {
                    ResponseCustomer customInfo = new
ResponseCustomer(info);
                    xml = customInfo.ToXml();
                }
```

通过使用多客服的客户端，这样处理消息交互起来非常方便，能获得客户的对话信息了，在电脑客户端上，看到的界面如下所示。



手机上的谈话截图如下所示。

这样就能够通过多途径，及时响应客户的信息了。

如果感兴趣或者体验相关的客服应答功能，可以关注我的微信了解下。具体效果可以关注我的微信门户：广州爱奇迪，也可以扫描下面二维码进行关注了解。

C#开发微信门户及应用(8)-微信门户应用管理系统功能介绍

最近对微信接口进行深入的研究，通过把底层接口一步步进行封装后，逐步升级到自动化配置、自动化应答，以及后台处理界面的优化和完善上，力求搭建一个较为完善、适用的微信门户应用管理系统。

微信门户应用管理系统，采用基于 MVC+EasyUI 的路线，由于多数域名服务器上都只能支持.NET4.0，所以以 MVC3，C#4.0 作为开发基础，基本上能够部署在任何.NET 服务器上。

在微信门户系统里面，实现下面这些功能操作：

1）实现菜单的动态配置及更新到服务器上；

2）动态定义事件和响应消息，实现对不同行业，不同需求的菜单动作响应；

3）动态的应答指令配置处理，实现整套应答链的消息处理；

4）获取订阅用户和用户分组信息，并可以实现用户分组信息的维护等操作；

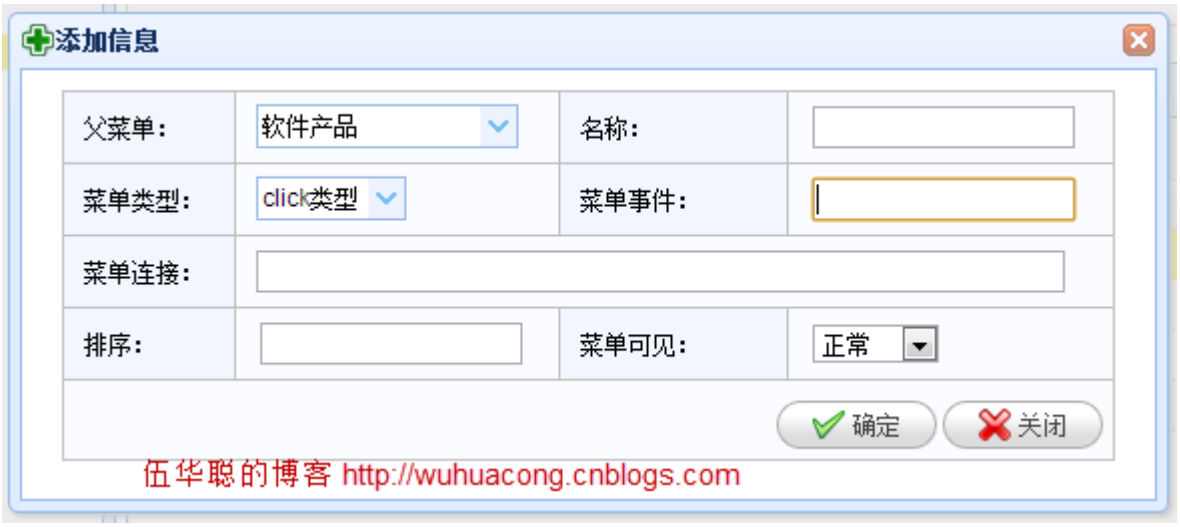
5）管理并更新多媒体文件、图文消息等内容，方便为客户推送消息做准备。

6）使用向选定订阅用户或者分组进行消息的群发功能。

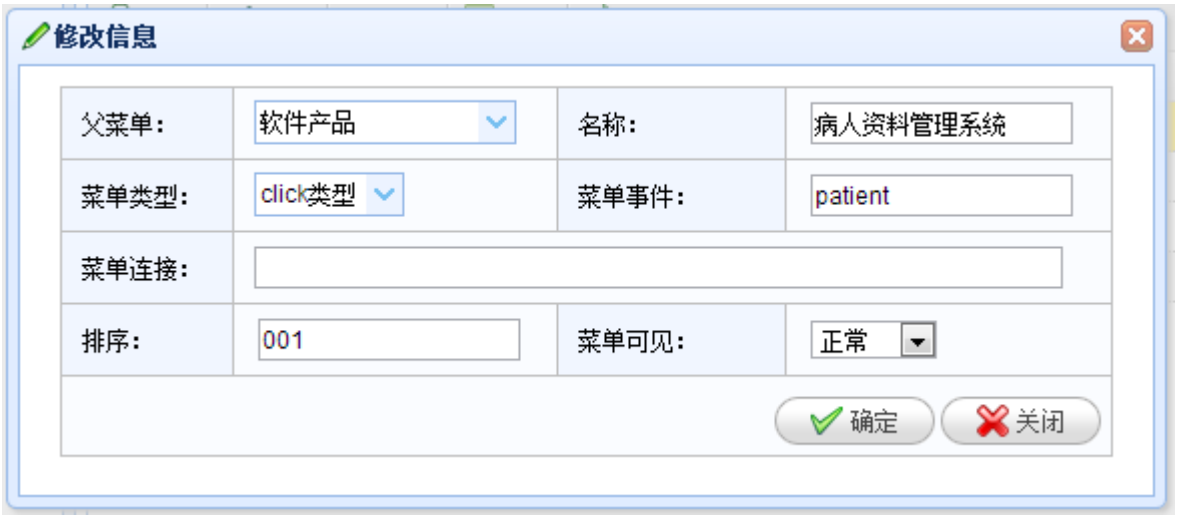
1、微信菜单管理

在系统中管理菜单 ,并通过把菜单提交到服务器上 ,实现菜单的动态配置和生成，能够为我们系统适应各种的需要，实现灵活的处理。

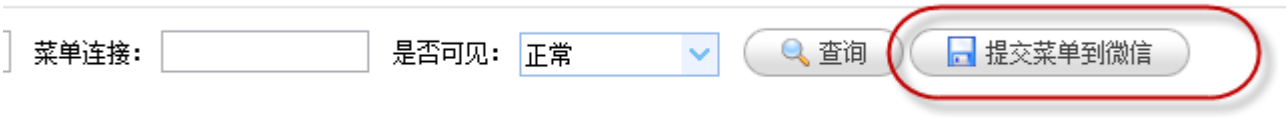
微信菜单的添加界面如下所示。



微信菜单的修改界面如下所示



微信菜单定义是存储在数据库里面，如果需要提交到微信服务器上并生效，则需要调用微信 API 接口进行处理，我在页面的 Controller 控制器里增加一个提交到服务器的处理方法。



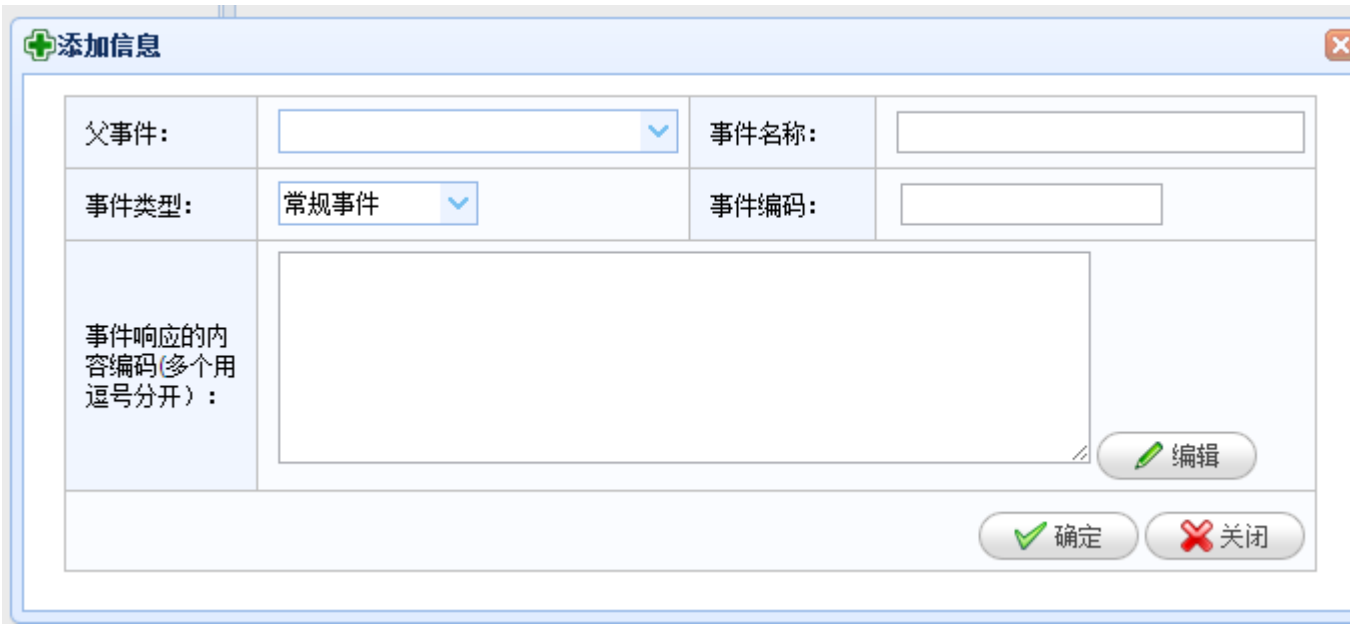
在微信服务账号的门户上，菜单的表现效果如下所示。

2、菜单事件的处理

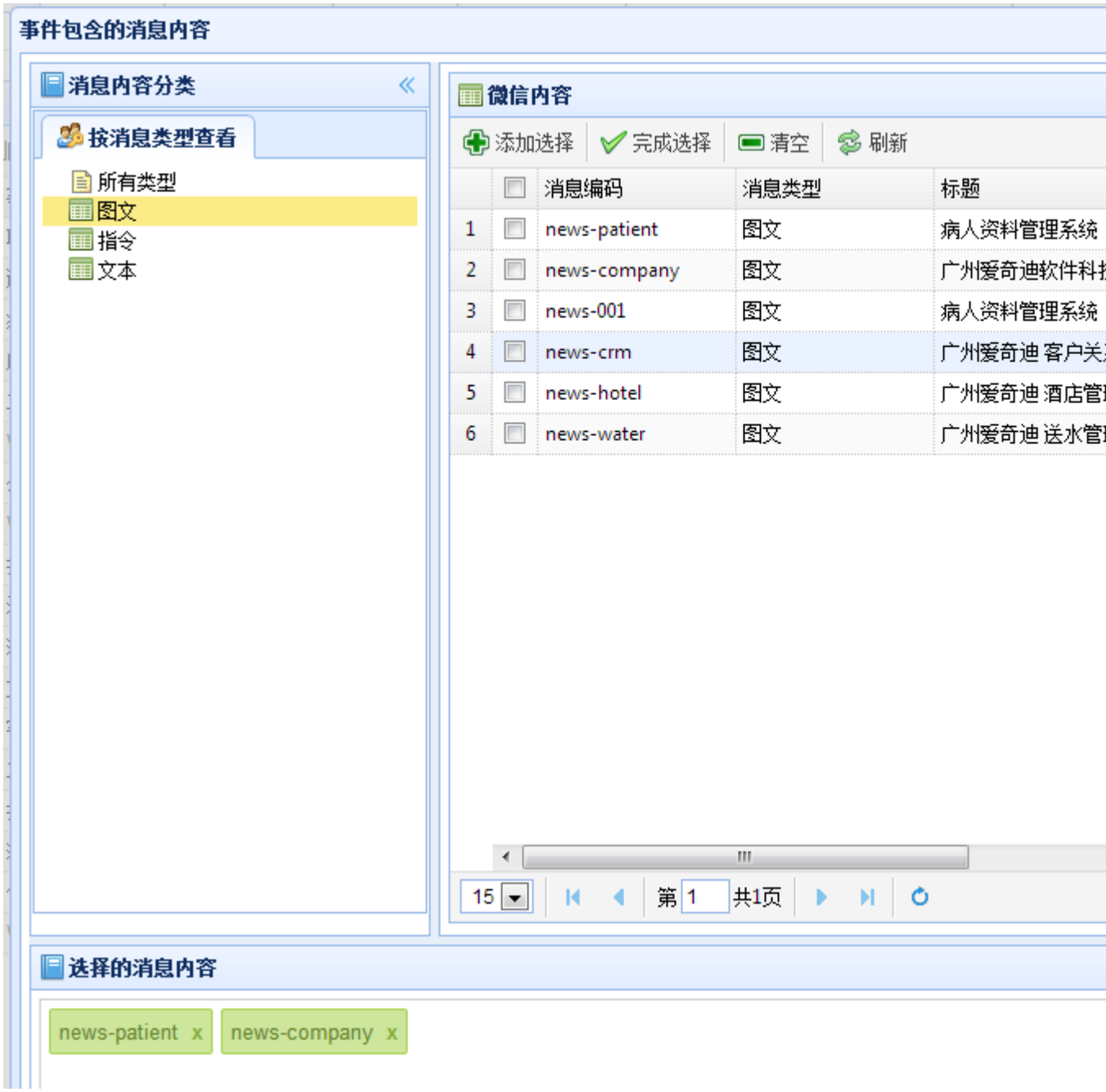
对于动态生成的菜单，大多数情况下是用作 Click 的方式，也就是需要定义每个菜单的事件响应操作，我们使用微信的话，可以了解到，微信的处理事件，一般可以响应用户文本消息、图片消息、图文消息等内容，常规下，一般使用文本消息或者图文消息居多。

为了进一步实现响应内容的重用，我们把菜单的事件定义和内容定义进行分开管理，事件定义可以使用多个文本消息，也可以使用多个图文消息进行组合，这样可以实现更加灵活的使用环境。

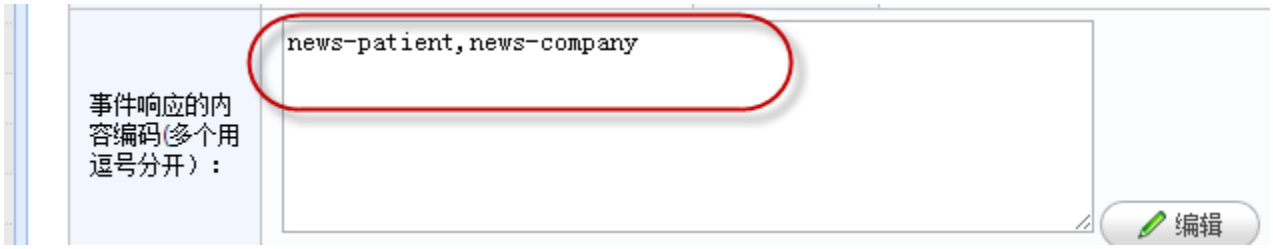
添加事件定义如下所示



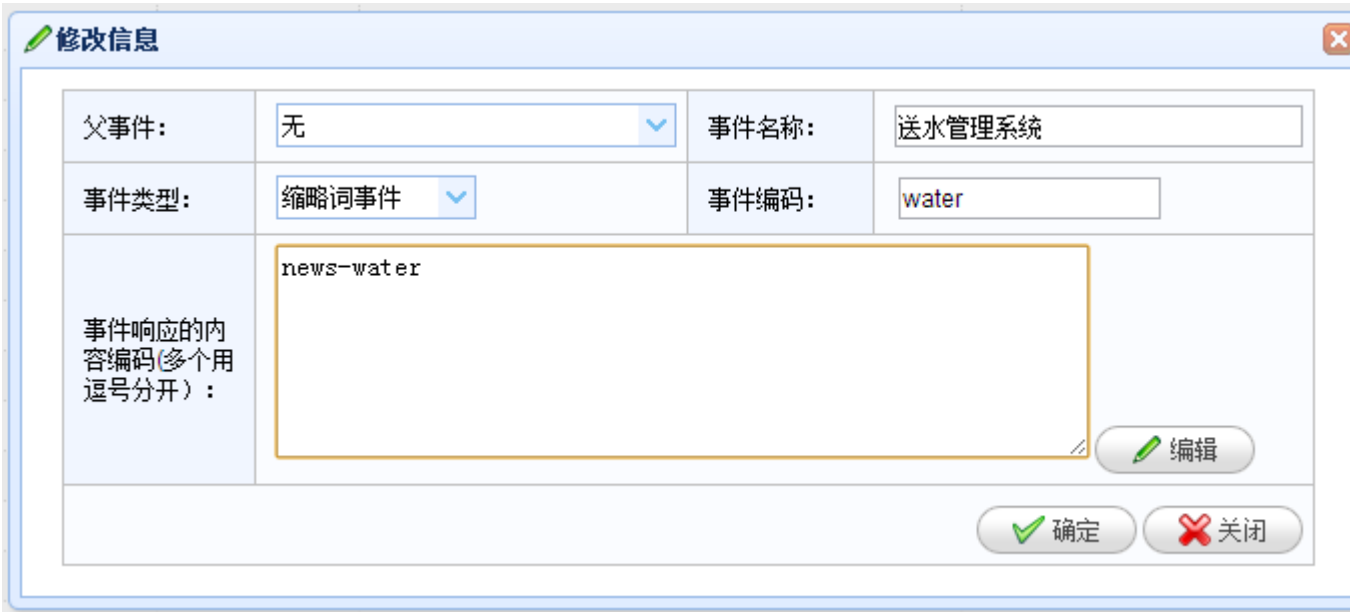
事件的响应内容编码，可以选择输入或者从“编辑”按钮中选择，当选择“编辑”按钮进行选择的时候，系统弹出一个对话框供用户对事件的响应内容编码选择。



完成选择后，回到原来的新增界面，将会看到返回的记录就是我们选择的记录。

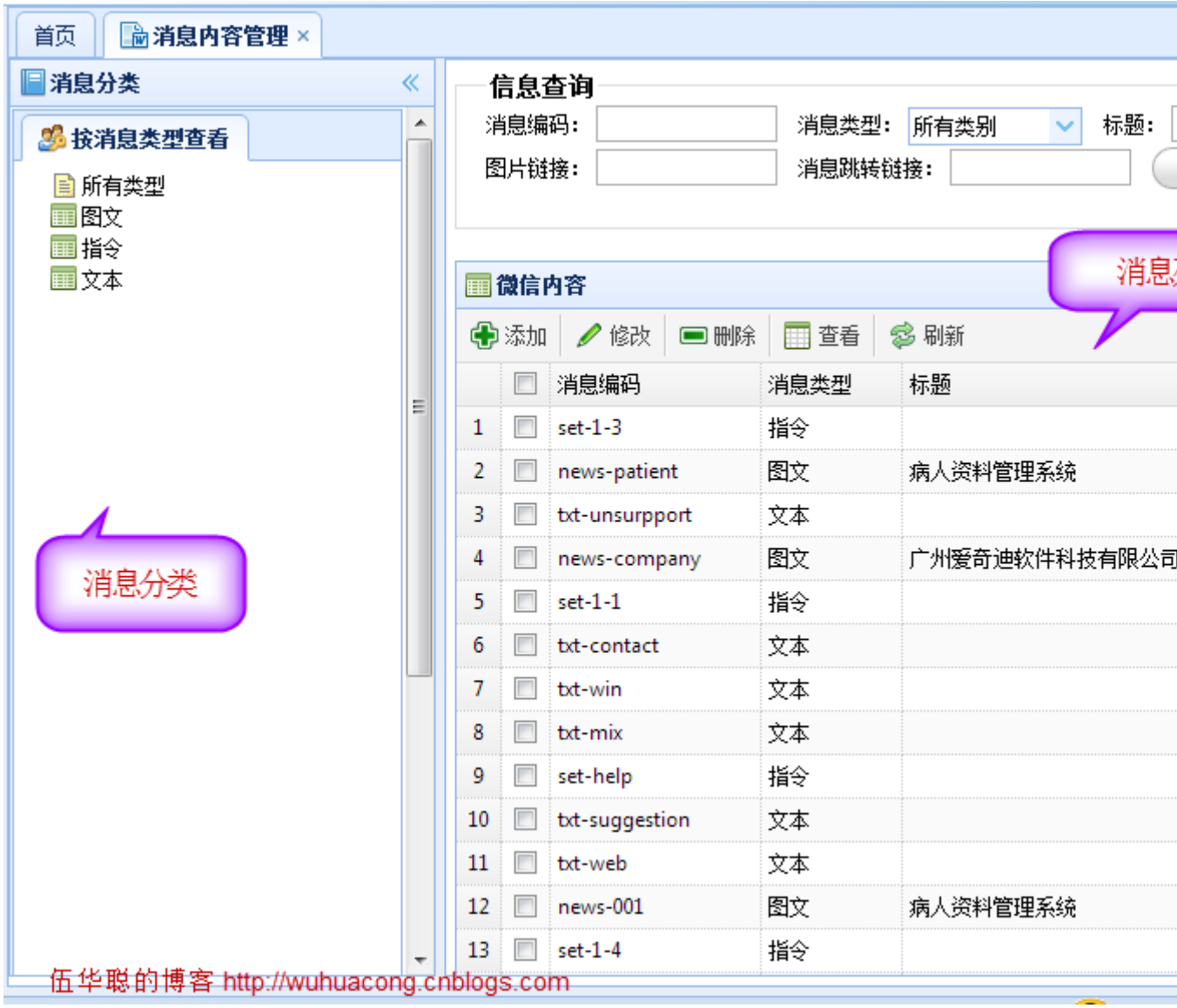


微信事件的编辑界面如下所示，类似新增界面的内容。



3、微信消息内容管理

上面说到，菜单的事件通过关联事件编码进行处理，而事件本身可以组合多个消息内容 ,因此消息内容是响应客户操作的最小单元 ,它们可以是一条文本消息、图文消息，也可以是多条消息的组合（同类型的话）。



为了方便管理，我把消息分为了图文、指令、文本类型，如果需要，还可以根据需要把它细化为其他类型的消息。

消息内容的添加界面如下所示。

文本消息的手机上界面效果如下所示。

这里不管是文本消息还是图文消息，我们统一以图文消息的定义来定义消息，如果是文本消息，我们只需要获取描述内容作为消息的主体即可。

图文消息的编辑界面如下所示，主要就是填写完整的内容和图片，以及页面详细的链接即可。

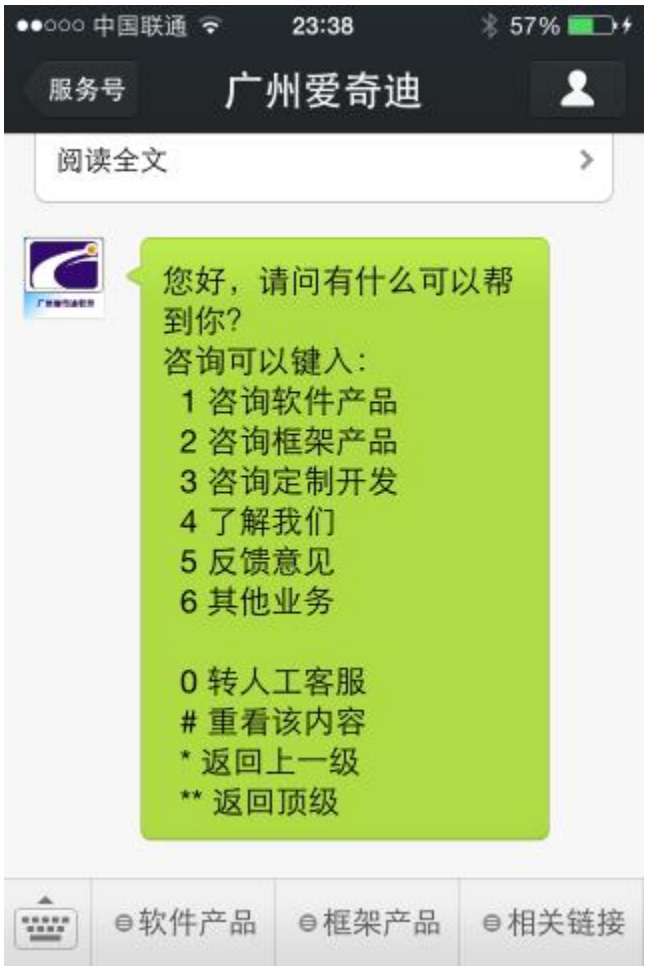
上面的这个客户关系管理系统的消息，在手机上显示的界面效果如下所示，单击链接，可以切换到消息跳转链接地址的。

4、应答指令的维护

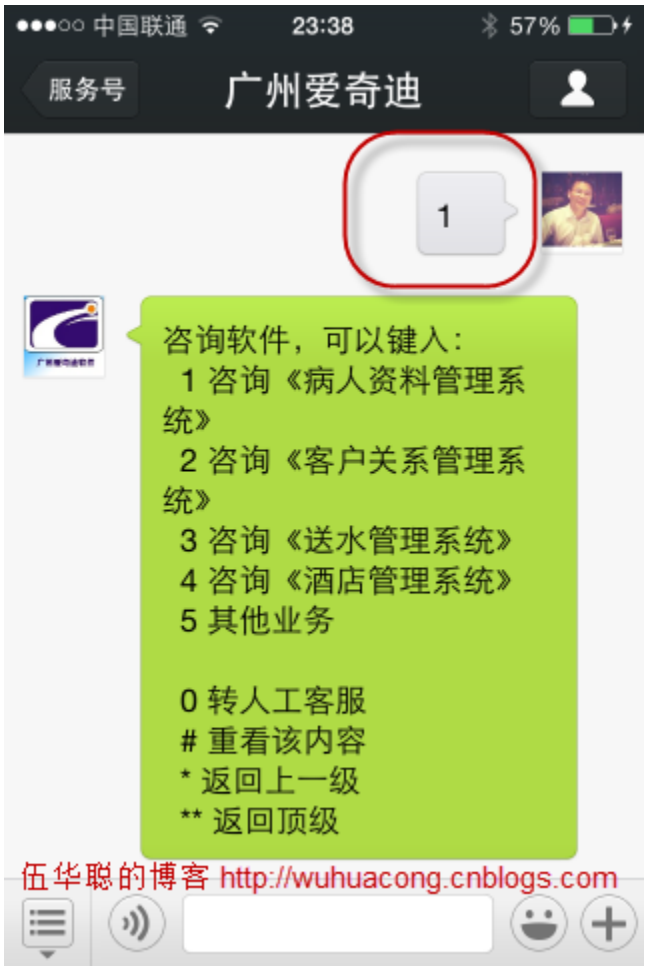
应答指令的维护，有点类似于事件的管理，主要就是定义一些用到的指令，方便构建应答系统的响应链，从而实现一步步的操作指令。

在后台设置好应答指令后，系统就能根据应答指令链进行处理了。首先我们需要提供一个进入应答链的提示界面，如下所示。

但我们在菜单选择应答系统后，系统返回一个文本提示界面，如下所示。



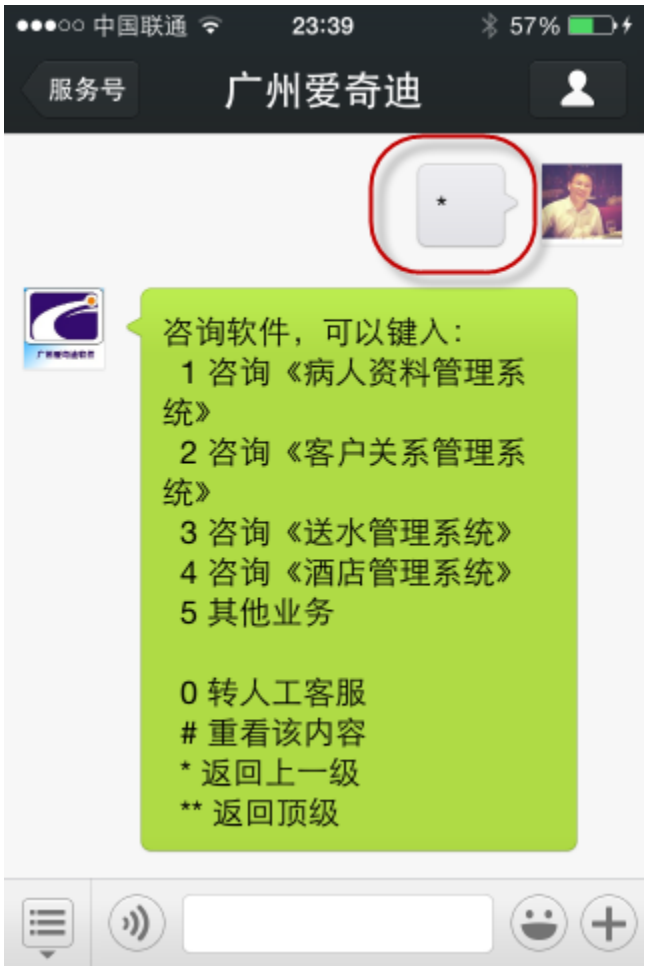
这个界面里面提示了一些按键，包括几个固定的按键和一些业务按键，输入简单的 1~6 可以对选择进行响应。



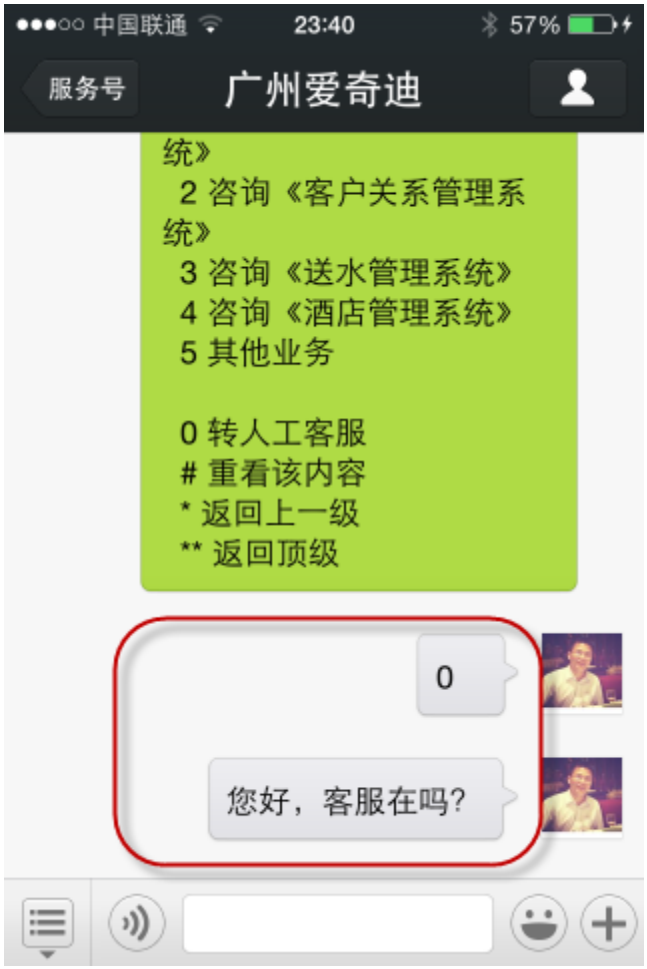
我们看到上面的界面，输入指令 1 后，系统进入下一层的应答指令，然后又列出几个可供输入的按键和内容提示。

当我们继续输入业务按键 1 后，响应的是一个图文消息，也是关于按键的详细说明。

这个时候，我们也还可以输入*号按键，返回上一级菜单的。



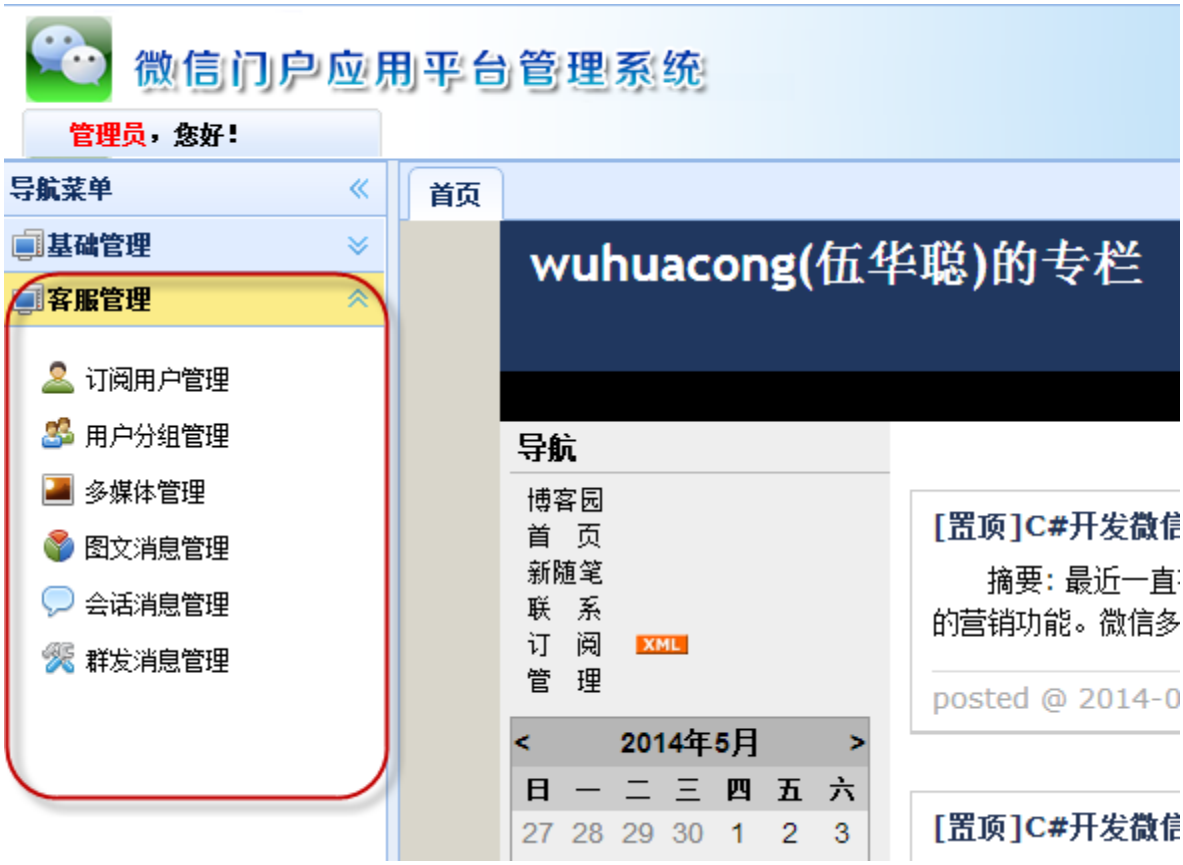
输入 0 则转入了客服对话模式，后续您发的任何消息，将会转发到多客服系统里面了。



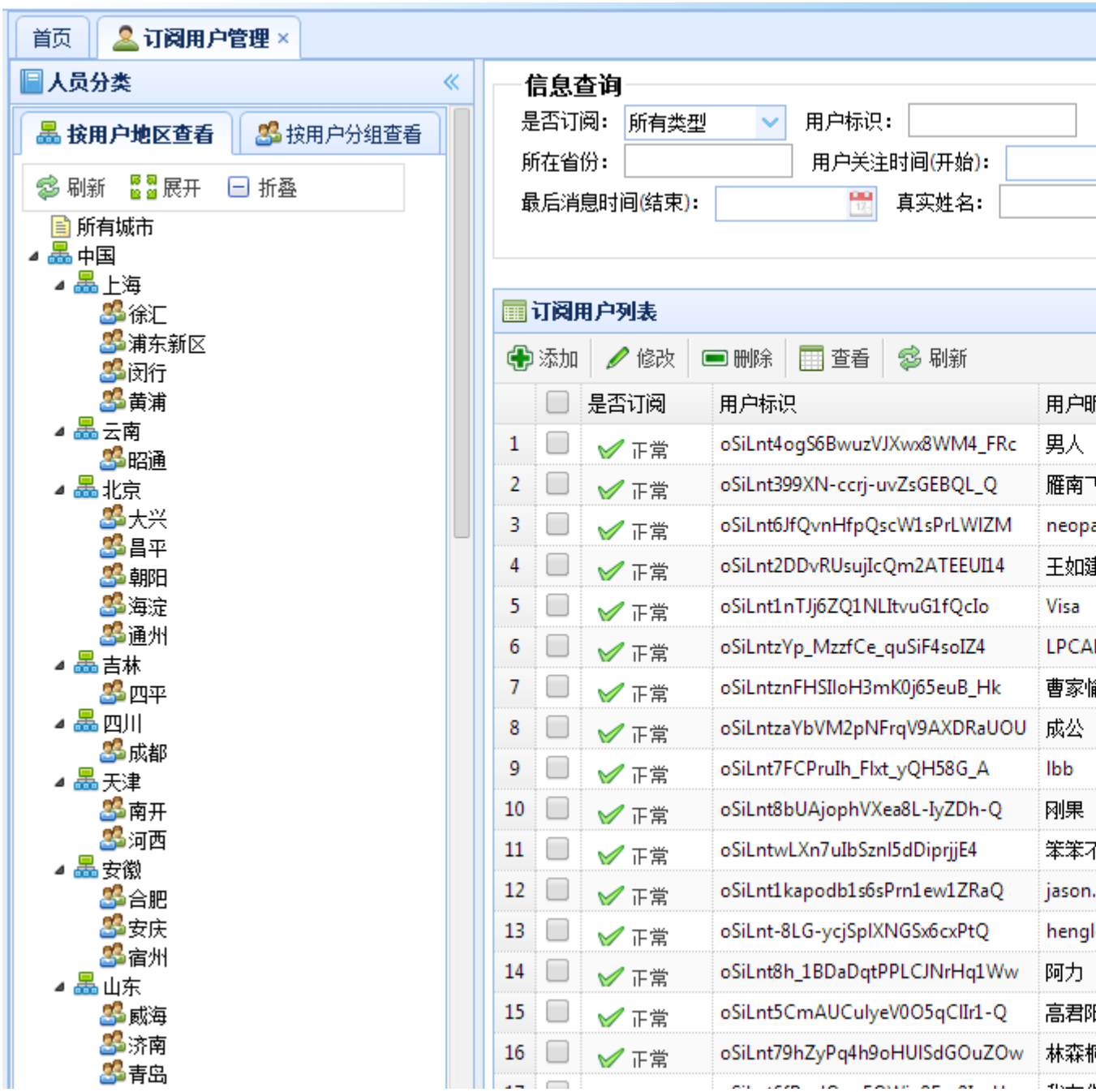
当用户发送消息后，客服助手就能及时收到消息并处理和客户的应答了。

5、订阅用户管理

为了更有效管理订阅用户以及分组信息，我们可以从微信服务器上获取相关的信息，供我们了解关注的用户信息，也可以为后续的群发消息做准备。



订阅用户的管理如下所示，默认可以通过用户的地区进行查看，地区根据：国家-省份-城市这样的级别进行展开。单击同步数据，可以把服务器上的用户数据下载到本地进行更新或者写入。



订阅用户，还可以根据分组进行查看

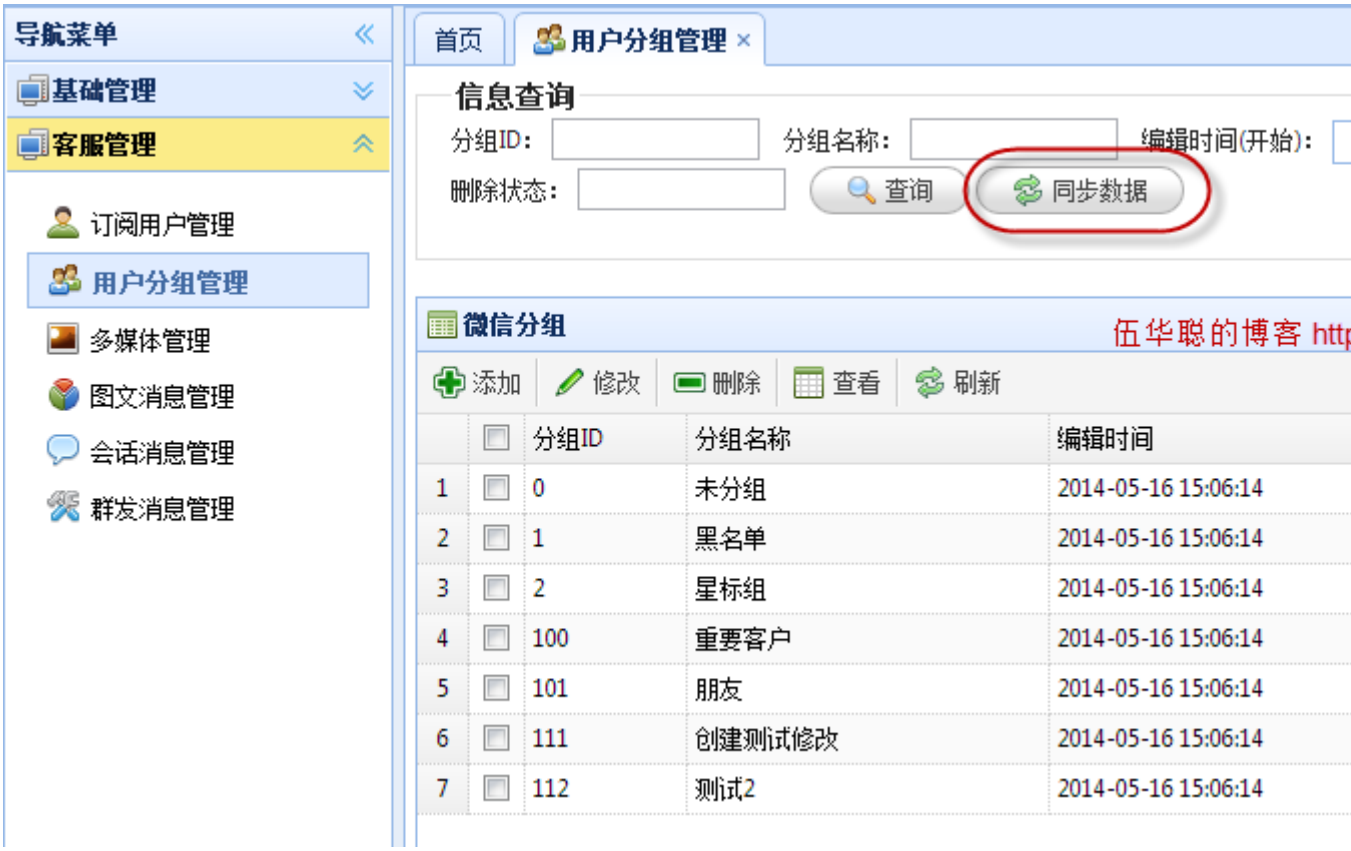
双击可以查看订阅用户信息，查看订阅用户的详细信息界面如下所示。

查看详细信息

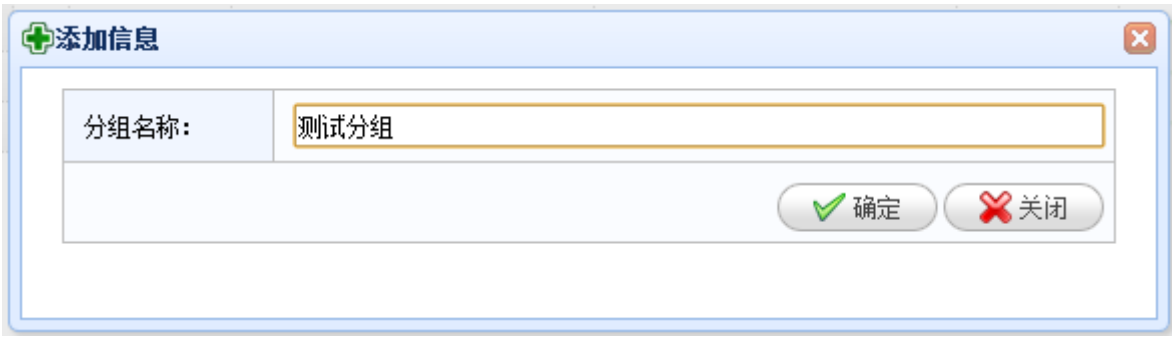
ID：	ffd82a58-c2ed-4ab1-870a-795d9969867a	是否订阅该公众号：	订阅
用户标识：	oSiLnt0jbmqhgAeom-IbyDgBegSg	用户昵称：	梦幻的不死鸟
性别：	男	用户语言：	zh_CN
所在城市：	武汉	所在省份：	湖北
所在国家：	中国	用户头像：	
用户关注时间：	2014-04-23 12:35:06	退订时间：	1900-01-01 00:00:00
用户分组：	未分组	最后消息时间：	2014-05-16 13:12:16
真实姓名：		移动电话：	
电子邮箱：		详细地址：	
邮政编码：		外部对接编码：	
备注信息：			

关

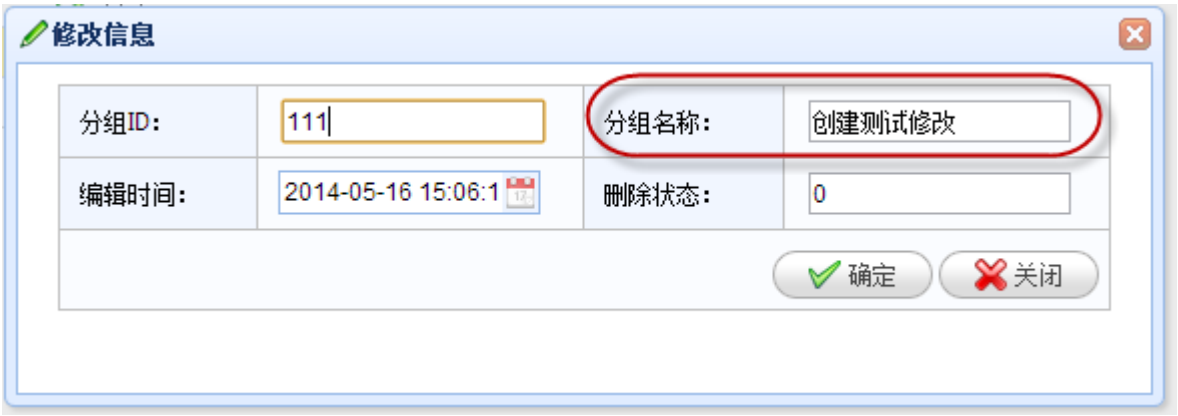
6、用户分组管理



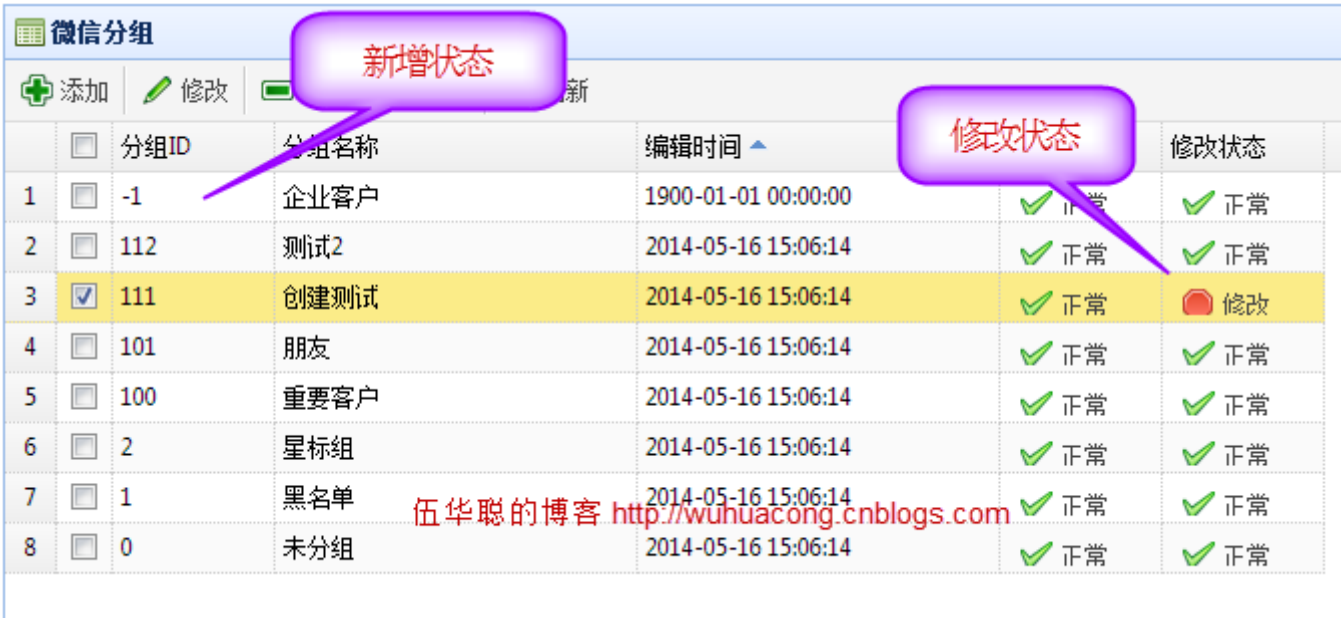
创建分组的界面如下所示。



编辑分组信息界面如下所示。



当对分组进行编辑保存后，系统会记住那些修改过的，同步的时候，把本地新增的内容，在服务器上创建分组；把修改的的分组名称，在服务器上进行修改，然后进行同步列表处理。

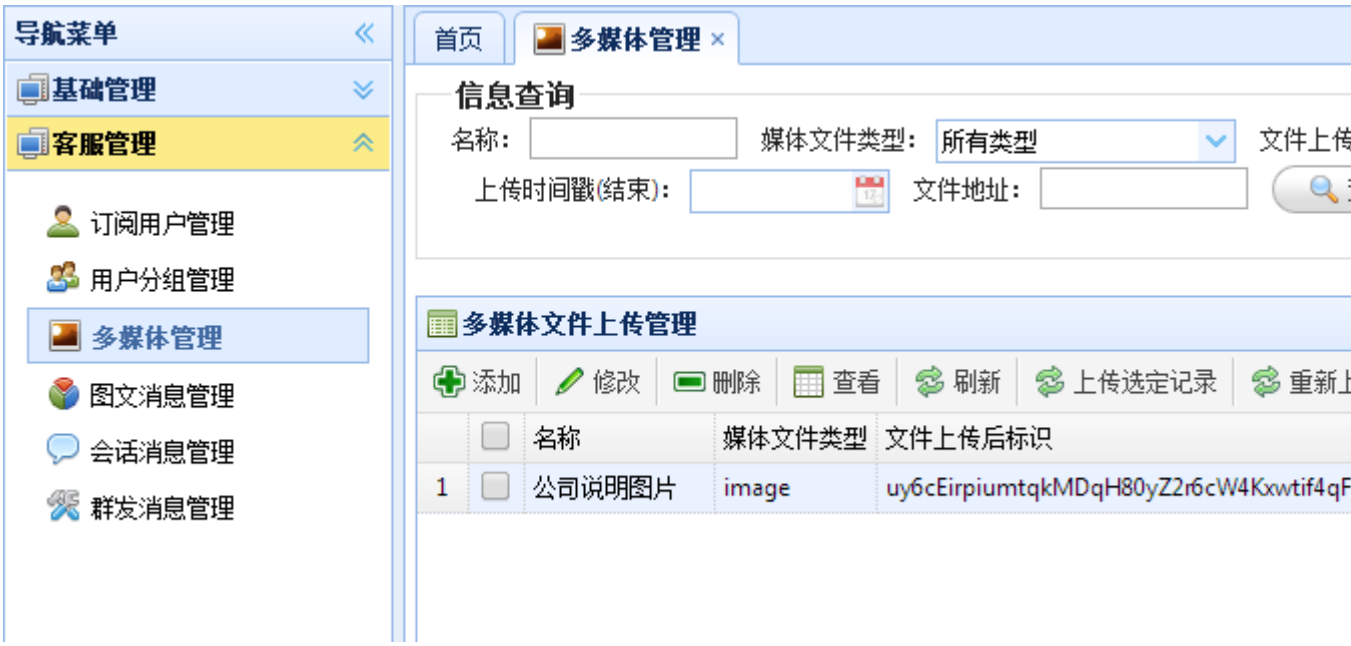


7、多媒体管理

多媒体管理是指把本地文件上传到微信服务器上进行保存，方便信息的发送等操作。微信要求，某些信息，必须是先上传到服务器上，然后才能使用它的媒体 ID 进行发送的。

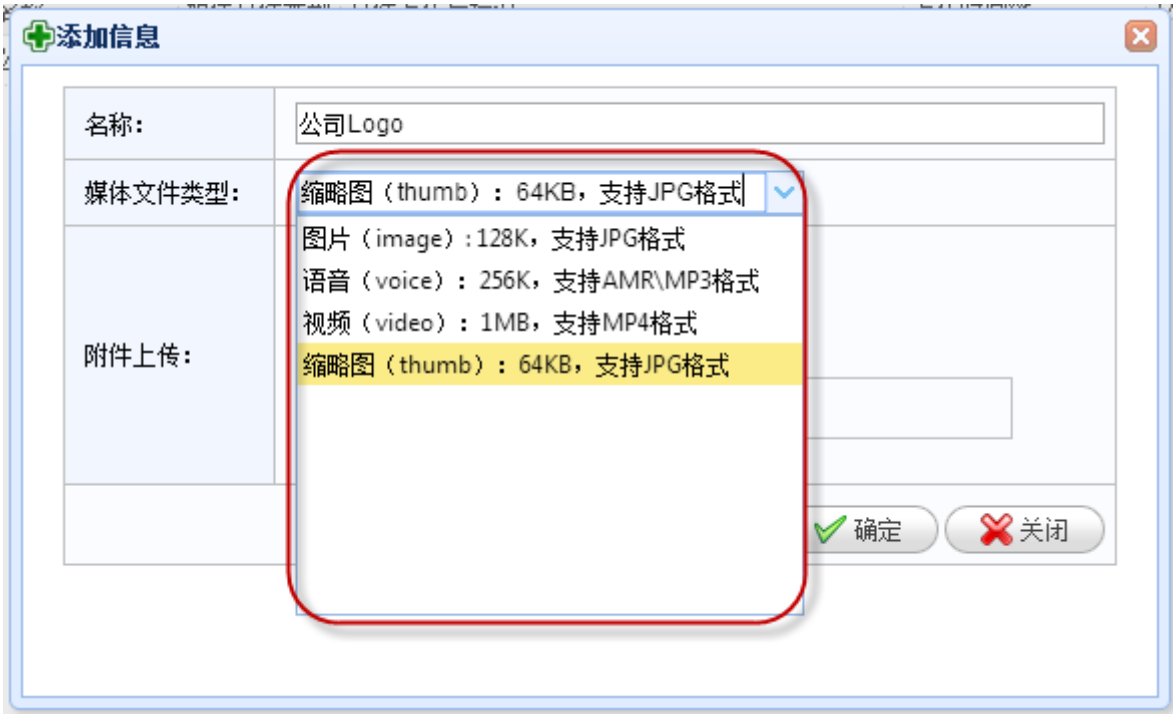
文件成功上传到服务器后，在列表里面的“文件上传标识，就是一串 BASE64 的编码数据，同时有一个上传的时间戳（因为微信服务器只保留了 3 天的媒体数据，超过期限的数据会被自动删除。

同时，在列表的上面，有两个重要的功能：上传选定的记录，重新上传过期的记录。方便我们对自己多媒体文件的重新更新操作。



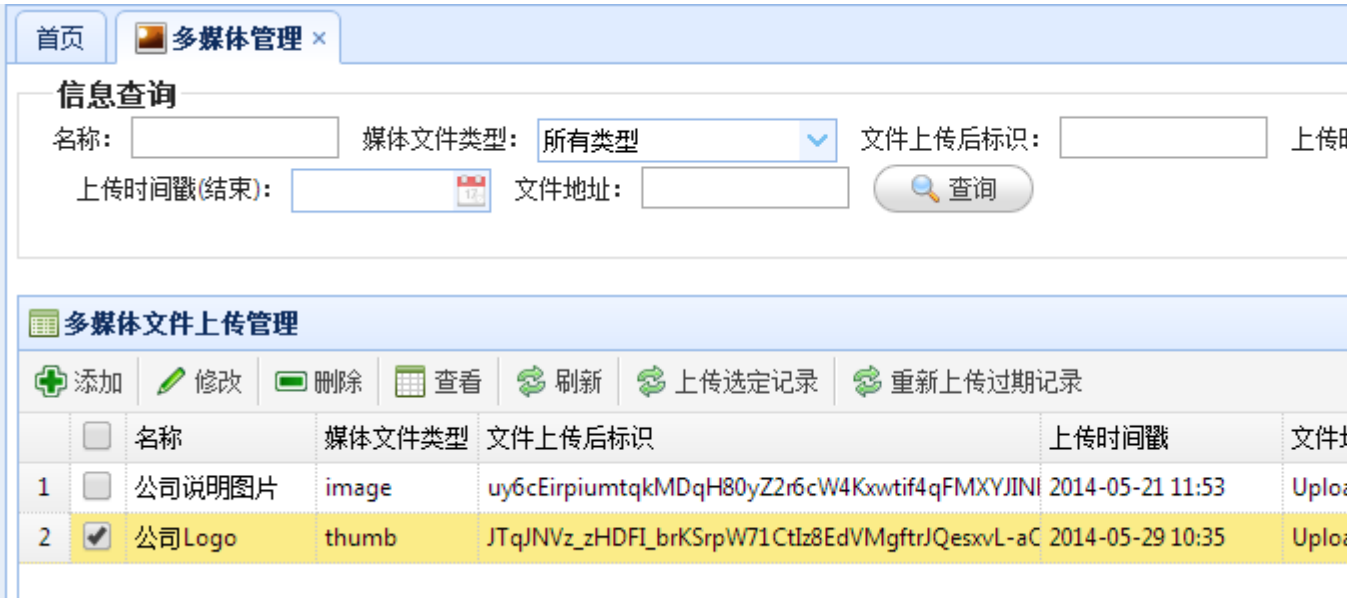
添加界面操作如下所示，其中引入了附件上传的控件进行文件的操作，非常方便。同时上传成功的文件，会在列表中列出。

多媒体文件可以是下面几种方式：图片、语音、视频、缩略图。



保存后的数据记录，文件上传标识和时间戳都是空的，我们如果要使用，必须把他们上传到微信的服务器上，然后根据它的 MediaId 进行信息的发送，上传选定的记录操作界面如下所示。

多媒体文件顺利上传后，记录的信息如下所示。



8、图文消息处理

图文消息分为单图文消息和多图文消息两种，单图文消息如下所示。

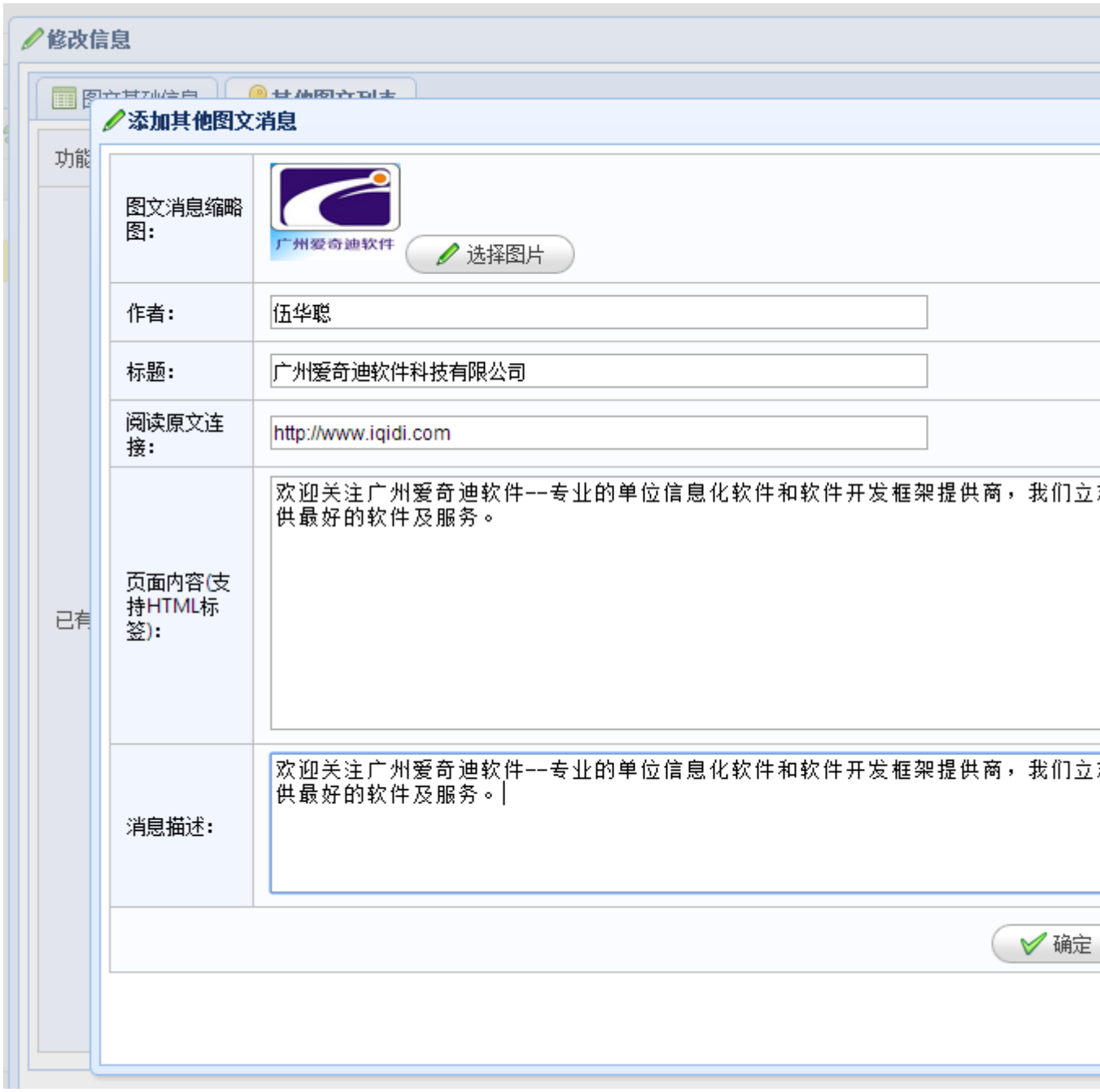
多图文消息如下所示：

和多媒体数据管理一样，图文消息也是通过同样的方式进行管理，先上传到服务器，然后在进行消息的发送操作，多媒体消息一样有时间方面的限制要求，具体在我们的微信门户平台里面管理界面如下所示。

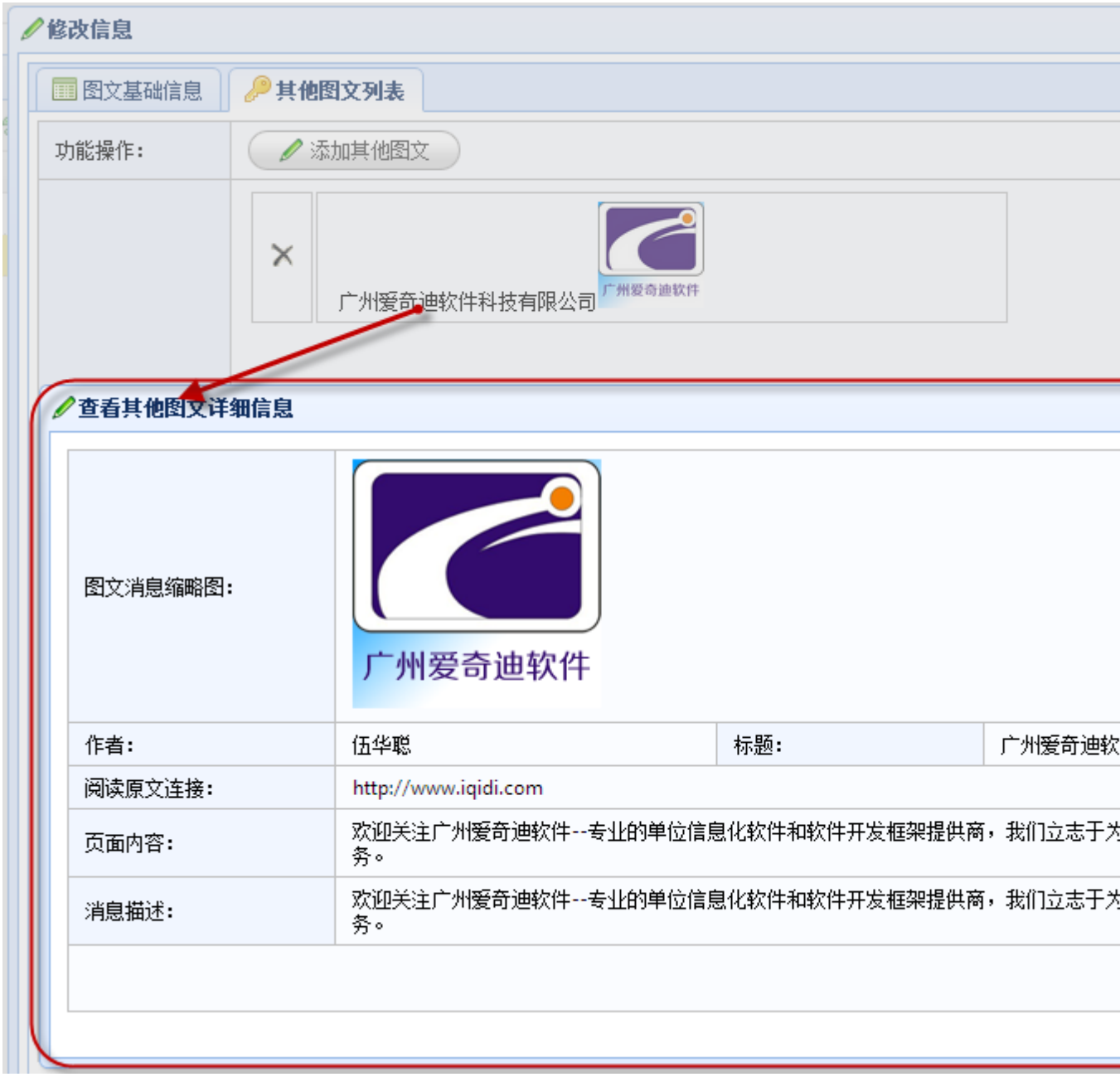
添加图文消息界面如下所示，保存后，可以在编辑界面中的“其他图文列表”里面，继续添加多图文的消息内容。

在添加界面中，选择图文消息的缩略图，都是通过选定指定的，已经上传到服务

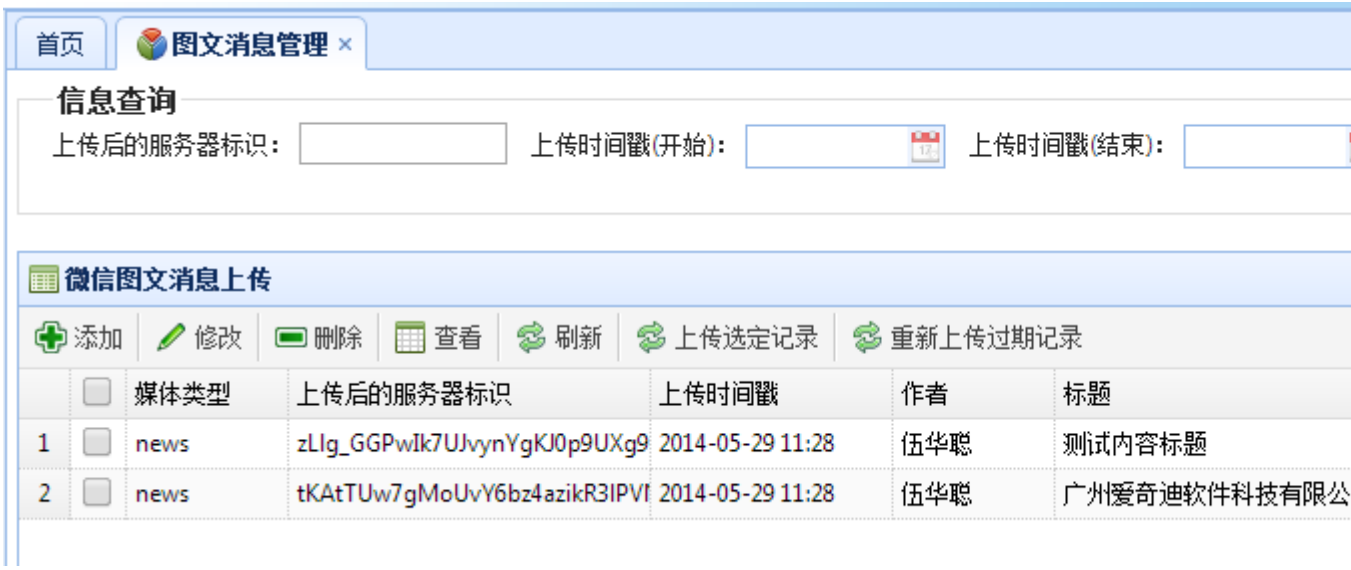
器上图片或者缩略图资源才可以的。



添加后的多图文列表，可以进行查看管理。

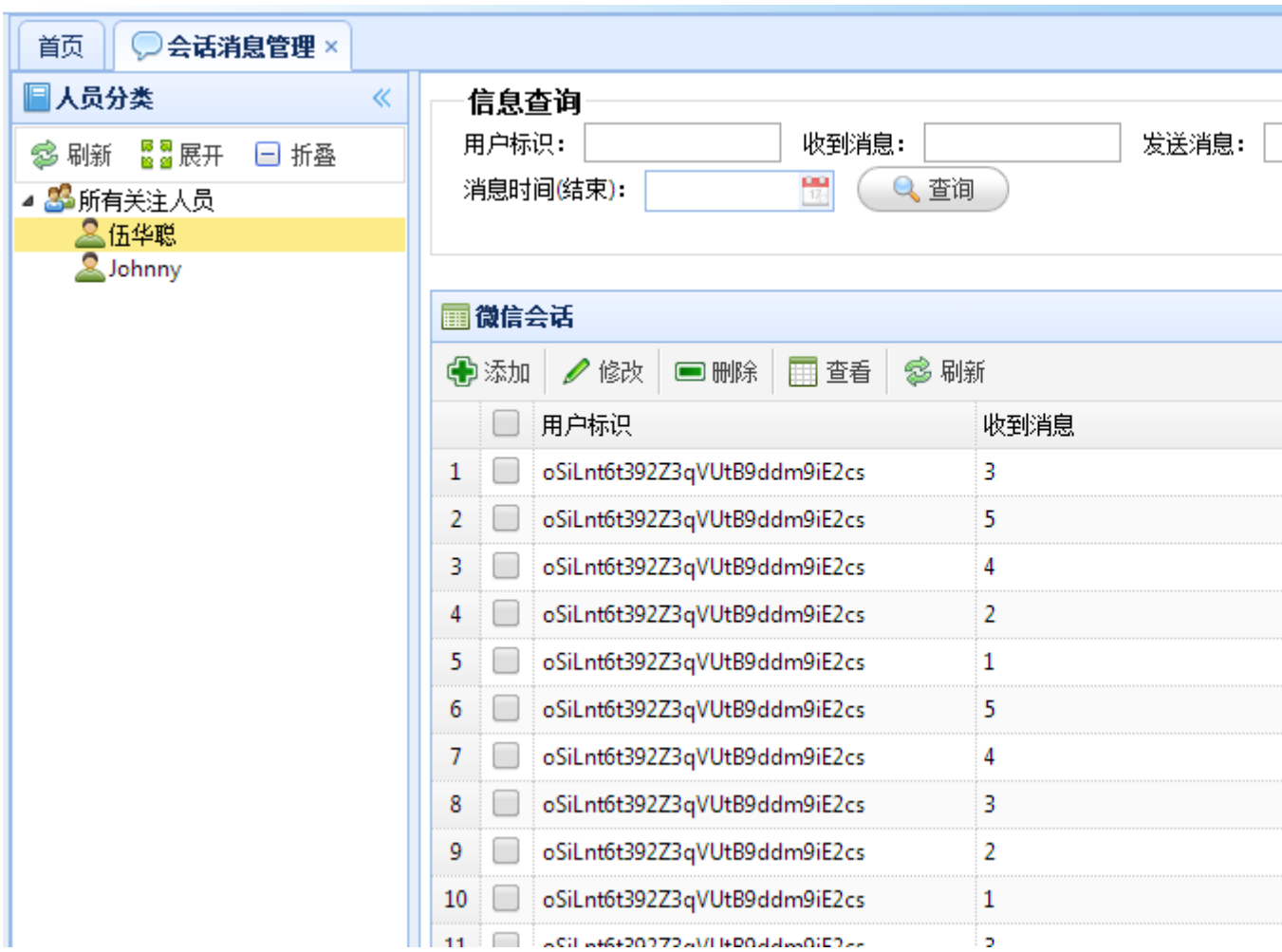


保存记录后，然后继续上传，上传后的记录界面如下所示，成功后返回一个上传后的服务器标识和时间戳，否则提示错误。



9、会话消息管理

为了方便记录客户的输入和发送信息 ,我们在微信门户管理平台里面记录用户的输入数据，具体会话消息管理界面如下所示。

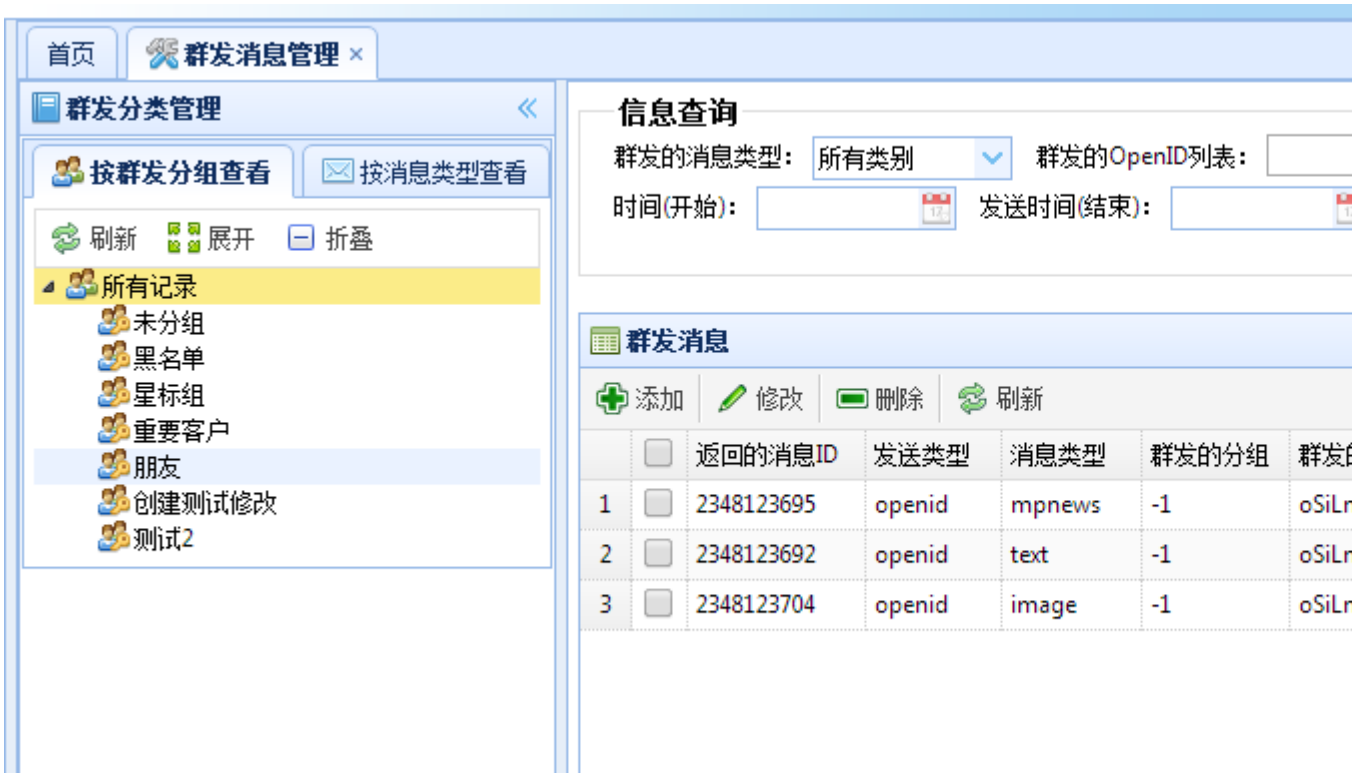


我们可以双击最近 48 小时内的任何一条记录，可以给关注的客户进行消息的发送操作，如果消息发送成功，用户在手机的微信账号里面就能收到相关的发送消息了。

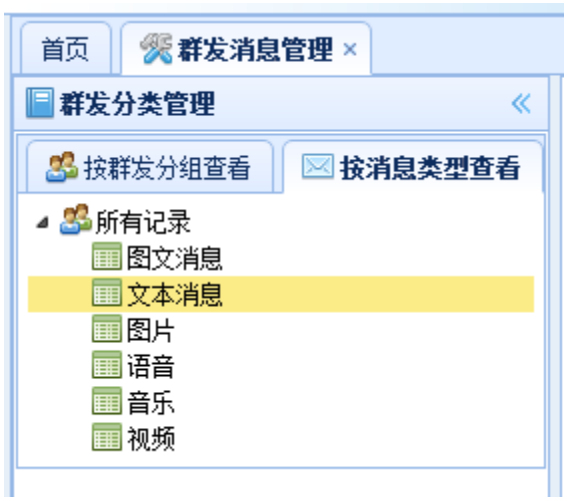
10、群发消息管理

为了对客户进行相应的营销操作，有时候我们需要对指定的群主或者人员进行消息的群发，让客户经常性的了解我们产品的信息和活动。

由于群发消息，除了文本消息，可以直接编辑发送外，其他数据，必须要求是上传到服务器的多媒体文件或者图文消息内容，因此前面的多媒体管理和图文消息管理，就是主要为了群发消息的目的引入的。有了上面的多媒体和多图文信息，我们从平台里面选择记录即可进行发送，从而省却麻烦的连带工作，实现高效的信 息群发操作。



群发的消息，可以按群发分组进行查看，也可以按照消息类型进行查看，使得我们管理起来根据方便。



添加图文消息，可以选择文本消息、图文消息、图片消息等内容，根据不同的内容，界面提供不同的选择操作。

消息的群发类型分为两种，一种是根据分组，那么从平台里面选择对应的分组即可；一种是根据用户的 OpenID 进行发送，提供给用户输入。主要的操作界面如下所示。

C#开发微信门户及应用(9)-微信门户菜单管理及提交到微信服务器

微信公众号（包括服务号和订阅号）都可以对菜单进行自定义设置，我们为了方便管理，一般先把菜单数据在本地管理维护，需要更新的时候，把它们更新到 微

信服务器上就可以了。本文基于这个方式，介绍我的微信门户平台管理系统中菜单提交到微信服务器上的操作。微信门户应用管理系统，采用基于MVC+EasyUI 的路线，由于多数域名服务器上都只能支持.NET4.0，所以以MVC3，C#4.0 作为开发基础，基本上能够部署在任何.NET 服 务器上。

1、微信菜单的要求及相关界面设计

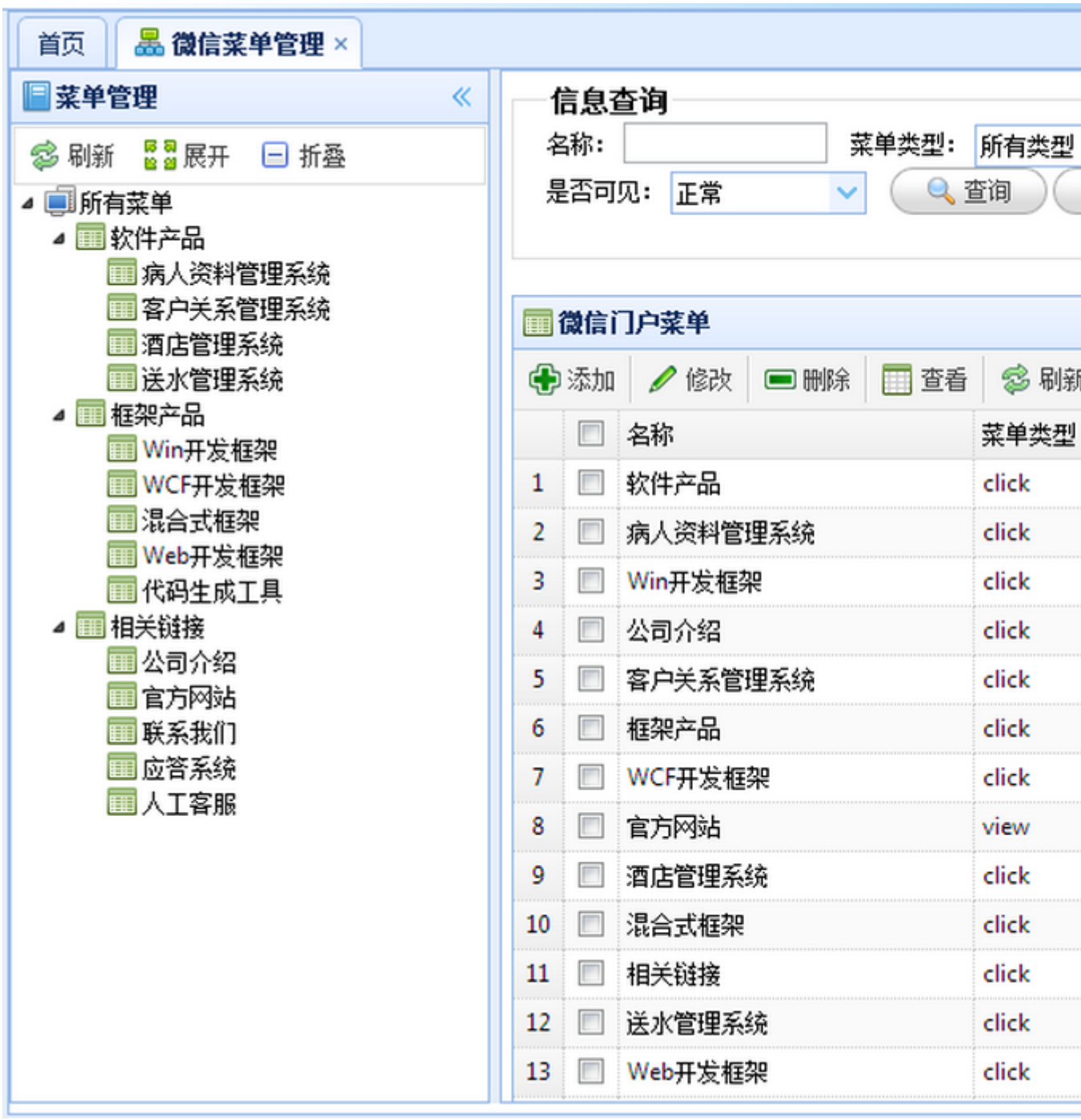
微信公众号的菜单我们可以通过网站进行本地的管理，维护好它们之间的层级关系，由于微信对自定义的菜单要求比较严格，以下是微信对自定义菜单的要求：

目前自定义菜单最多包括 3 个一级菜单，每个一级菜单最多包含 5 个二级菜单。一级菜单最多 4 个汉字，二级菜单最多 7 个汉字，多出来的部分将会以“...”代替。

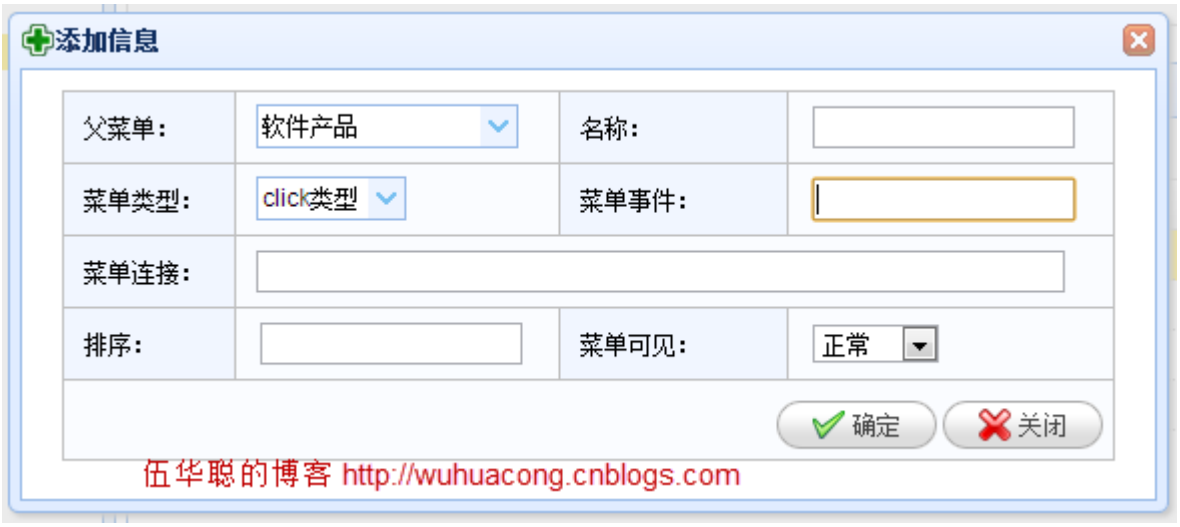
因此我们自己根据约定，不要越界即可，否则提交菜单到服务器，可能会返回一些错误，这些细节，我们在创建本地菜单管理的时候，注意一下就可以了。我在早期的一篇文章也介绍了自定义菜单的一些内容，需要可以进行回顾一下《C#开发微信门户及应用(6)--微信门户菜单的管理操作》，本篇主要是介绍在我的平台管理系统里面，调用前面介绍的菜单接口 API，实现菜单提交到服务器的操作。

根据微信的自定义菜单要求，我在管理系统里面，对微信的菜单几个基础性的界面设计如下。

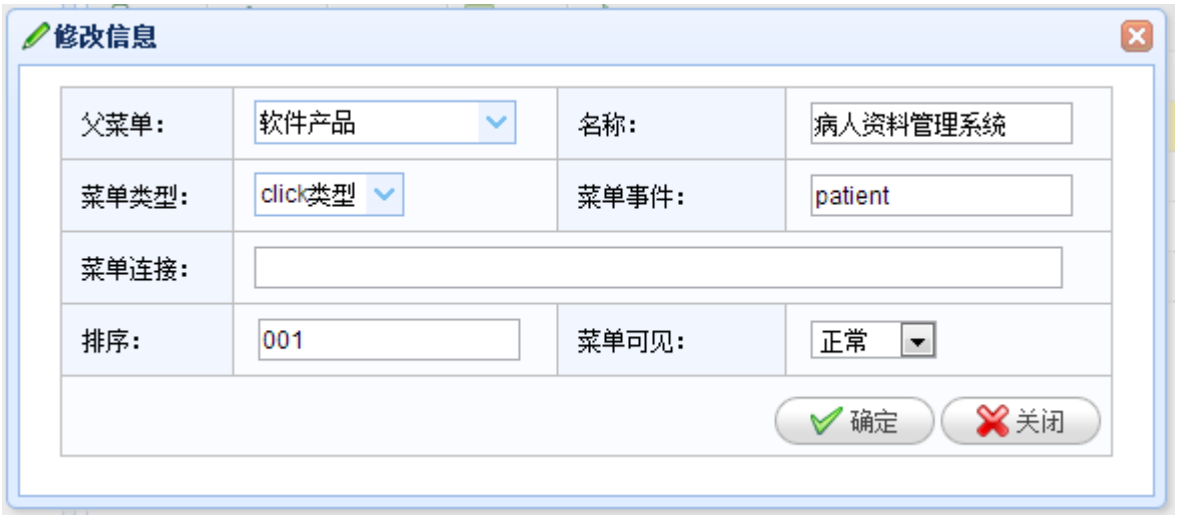
主菜单管理界面如下所示。



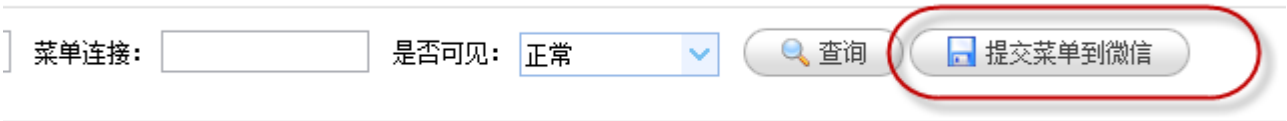
添加菜单的界面设计如下所示



微信菜单的修改界面如下所示



微信菜单定义是存储在数据库里面，如果需要提交到微信服务器上并生效，则需要调用微信 API 接口进行处理，我在页面的 Controller 控制器里增加一个提交到服务器的处理方法。



2、提交菜单到微信服务器的操作

上面几个界面，主要就是根据微信菜单的属性，对菜单进行维护管理，我们最终的目的是把它们放到服务器上去，供我们处理客户的相关事件操作的。

提交菜单的操作，我们在 MVC 的 View 页面里面，使用 JQuery 的 Ajax 提交即可（前提是我们在控制器里面添加相应的处理，后面介绍），界面脚本代码如下所示。

```
        //绑定提交按钮的的点击事件
        function BindSubmitEvent() {
            $("#btnSubmit").click(function () {
                $.messager.confirm("提交菜单确认", "您确认需要提交菜单到微信服务器吗？", function (action) {
                    if (action) {
                        //提交数据
                        $.ajax({
                            url: '/Menu/UpdateWeixinMenu',
                            type: 'post',
                            dataType: 'json',
                            success: function (data) {
                                if (data.Success) {
```

```
                                    $.messager.alert("提示", "提交微信
菜单成功");
                                }
                                else {
                                    $.messager.alert("提示", "提交微信
菜单失败:" + data.ErrorMessage);
                                }
                            },
                            data: ''
                        });
                    }
                });
            });
        }
```

上面红色的代码，就是我们在 MVC 的控制器里面定义的方法，我们只需要通过 POST 方法，对控制器方法调用，就能实现菜单提交到微信服务器上，至于具体里面的细节，我们可以把它挪到控制器或者更底层进行处理就是了，页面不需要涉及太多的逻辑就是了。

上面那个 Menu 控制器的 UpdateWeixinMenu 的方法代码如下所示（主要就是根据我前面介绍过的开发模型进行处理就是了）。

```
        /// <summary>
        ///更新微信菜单
        /// </summary>
        /// <returns></returns>
        public ActionResult UpdateWeixinMenu()
        {
            string token = base.GetAccessToken();
            MenuListJson menuJson = GetWeixinMenu();

            IMenuApi menuApi = new MenuApi();
            CommonResult result = menuApi.CreateMenu(token,
menuJson);
            return ToJsonContent(result);
        }
```

上面的几个方法这里逐一介绍一下。GetAccessToken 主要就是获得当前操作的访问令牌，这里的操作可以用缓存进行缓存，否则频繁的获取 AccessToken，达到每天指定的次数后，当天就不能再用了。

GetWeixinMenu 方法，主要就是为了方便，对获取构造微信的自定义菜单数据进行了一个函数封装，具体代码如下所示。

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

```
        /// 生成微信菜单的 Json 数据
        /// </summary>
        /// <returns></returns>
        private MenuListJson GetWeixinMenu()
        {
            MenuListJson menuJson = new MenuListJson();

            List<MenuNodeInfo> menuList =
BLLFactory<Menu>.Instance.GetTree();
            foreach (MenuNodeInfo info in menuList)
            {
                ButtonType type = (info.Type == "click") ?
ButtonType.click : ButtonType.view;
                string value = (type == ButtonType.click) ? info.Key :
info.Url;

                MenuJson weiInfo = new MenuJson(info.Name, type,
value);
                AddSubMenuButton(weiInfo, info.Children);

                menuJson.button.Add(weiInfo);
            }
```

```
            return menuJson;
        }
```

```
        private void AddSubMenuButton(MenuJson menu,
List<MenuNodeInfo> menuList)
        {
            if (menuList.Count > 0)
            {
                menu.sub_button = new List<MenuJson>();
            }
            foreach (MenuNodeInfo info in menuList)
            {
                ButtonType type = (info.Type == "click") ?
ButtonType.click : ButtonType.view;
                string value = (type == ButtonType.click) ? info.Key :
info.Url;

                MenuJson weiInfo = new MenuJson(info.Name, type,
value);
                menu.sub_button.Add(weiInfo);
```

```
                AddSubMenuButton(weiInfo, info.Children);
            }
        }
```

上面的代码，就是把本地存储的 MenuNodeInfo 数据，通过递归遍历的方 式，转换为微信的自定义菜单实体 MenuJson，这样我们调用 API 就非常方便了，这个函数主要负责构造对应的实体信息就是了。至于调用微信 API 提交 菜单的事情，还是让 API 自己亲自处理为好，他们的代码如下所示（也就是上面函数的部分代码）。

```
        IMenuApi menuApi = new MenuApi();
        CommonResult result = menuApi.CreateMenu(token,
menuJson);
        return ToJsonContent(result);
```

最终的结果是返回一个通用的结果 CommonResult，这个结果对象，非常方便脚本的处理，如果有错误，则提示错误，否则也方便判断布尔值，也就是上面的页面代码脚本。

```
success: function (data) {
        if (data.Success) {
```

```
                    $.messager.alert("提示", "提交微信菜单成功");
                }
                else {
                    $.messager.alert("提示", "提交微信菜单失败:" +
data.ErrorMessage);
                }
            },
```

通过以上几部分的代码，我们就可以实现前台 MVC 的视图界面，调用后台封装好的微信 API，实现菜单的提交处理了。

如果感兴趣或者体验相关的客服应答功能，可以关注我的微信了解下。具体效果可以关注我的微信门户：广州爱奇迪，也可以扫描下面二维码进行关注了解。

C#开发微信门户及应用(10)--在管理系统中同步微信用户分组信息

在前面几篇文章中，逐步从原有微信的 API 封装的基础上过渡到微信应用平台管理系统里面 ,逐步介绍管理系统中的微信数据的界面设计 ,以及相关的处理 操作过程的逻辑和代码，希望从更高一个层次，向大家介绍微信的应用开发过程。本篇主要介绍在管理系统中，如何实现微信用户分组信息的同步操作。

其实微信能够风风火火的原因，主要就是因为有用户信息，所以同步并管理好微信账号的关注用户数据是非常重要的。有了微信用户的数据，你 可以和你任何应用系统对接 ,实现系统-手机客户端的数据整合 ,还可以对用户进行营销管理，如发送用户感兴趣的产品消息、服务消息等，能够很好扩大企业的影 响力和市场行为。

在较早之前的一篇随笔《C#开发微信门户及应用(5)--用户分组信息管理》， 我曾经介绍了微信分组的各种底层的 API 封装操作 ,里面主要就是对微信提供 API 的.NET 高级分组，对所有的信息交换，通过实体性进行数据交换，使得我 们调用 API 来处理微信的各种事务更加方便 ,从而为微信应用平台的管理奠定基础。其中这篇文章介绍了所有微信分组管理的 API 封装过程，用户分组管理，包 含下面几个方面的内容：

1）创建分组

2） 查询所有分组

3） 查询用户所在分组

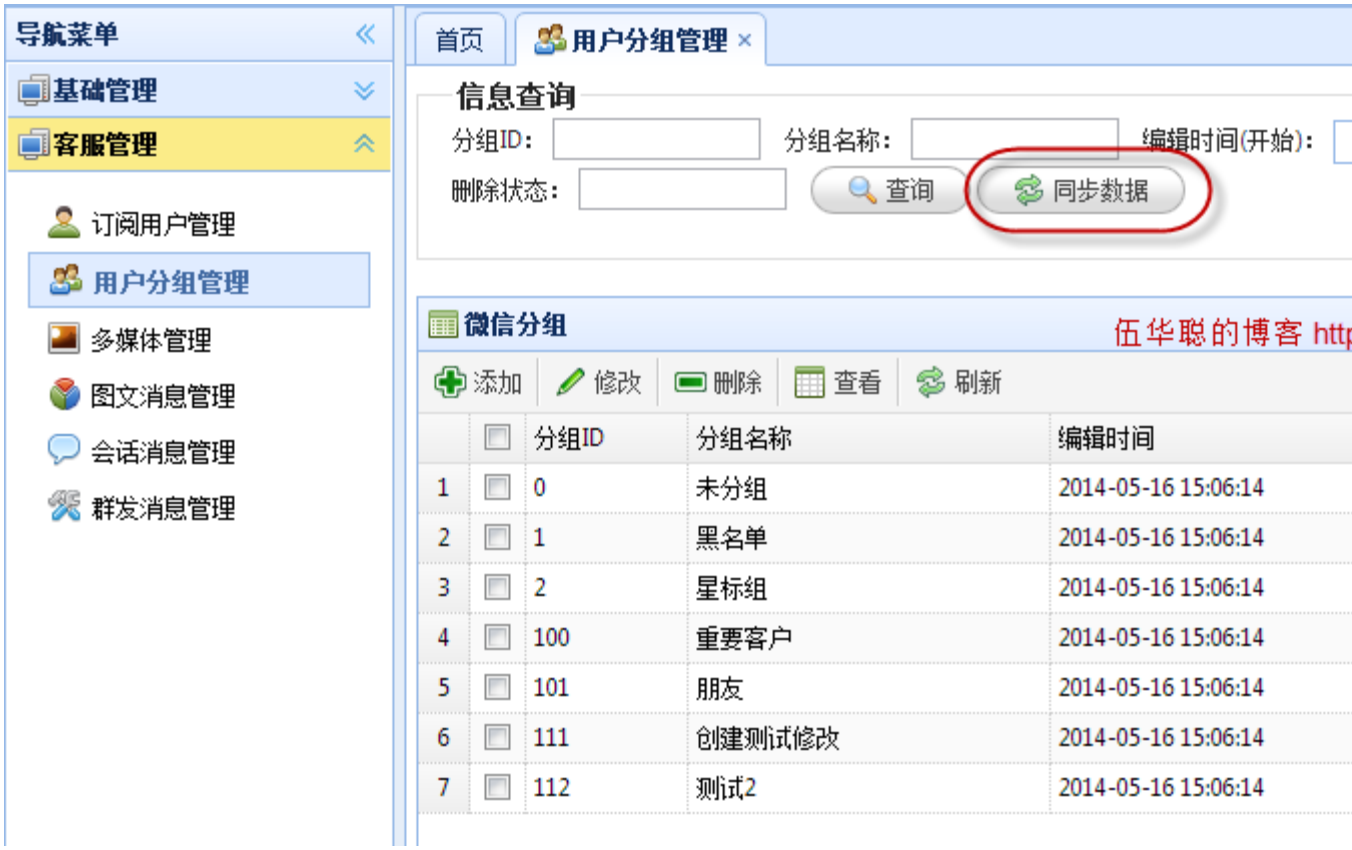
4） 修改分组名

5） 移动用户分组

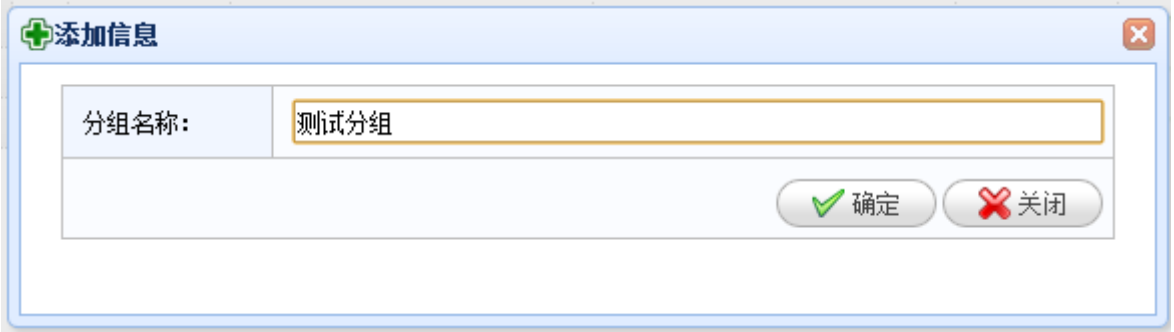
1、用户分组，在管理系统中的界面设计

针对以上微信分组的操作 ,我们可以在微信的应用管理系统里面 ,设计一个模块，用来管理微信的分组数据，在这个模块里面，可以创建分组，修改分组，查 看分组等基础操作，还可以实现同步微信分组的操作，同步操作，主要就是把新增的分组信息添加到微信里面，修改的分组也在微信中实现修改功能，删除目前微信 不支持，所以不用管了。最后，我们可以在此从微信服务器上，把修改后的数据同步下来，同步的时候为了避免对我们提交不成功的数据，我们需要对修改过的记录 做好标识，这个就是我对整个同步操作的逻辑处理了。

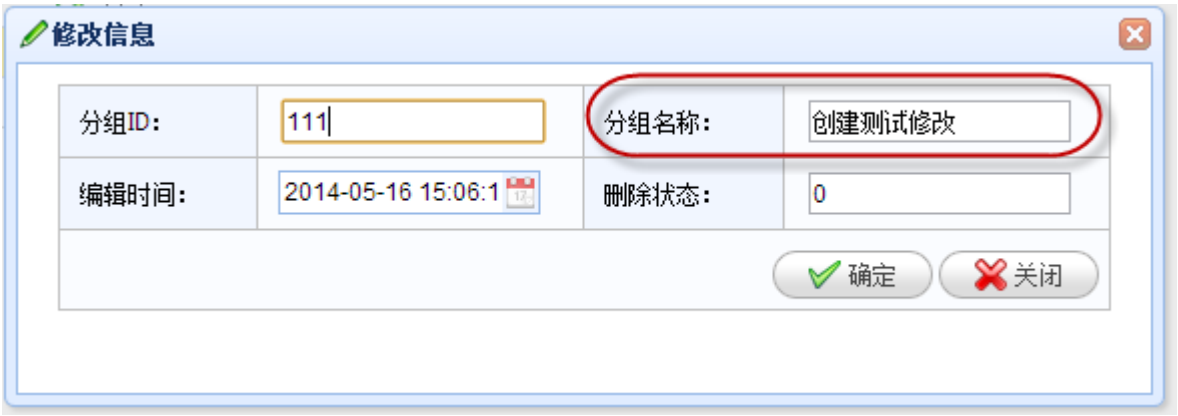
在管理系统里面，对微信分组的列表管理界面设计如下所示。



创建分组的时候，我们只需要添加一个分组名称就可以了，界面设计也简单，但是我们把创建的 ID 统一设计为-1，作为未同步的新增标识。



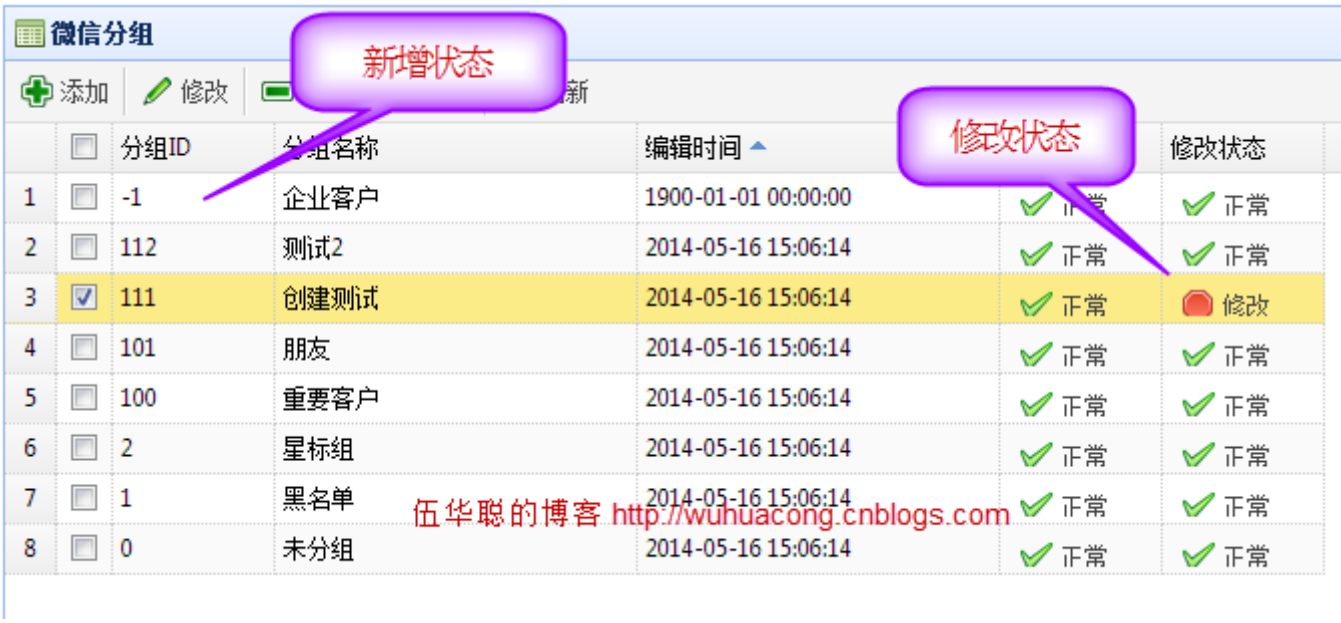
编辑分组信息界面如下所示。当对分组进行编辑保存后，系统会记住那些修改过的分组就是了。



2、分组同步操作代码展示

为了更好实现分组同步的管理，我把分组的操作代码，封装在一个 MVC 的控制器的方法里面，页面代码通过 Ajax 调用就可以实现同步操作了，同步成功，或者失败，都会提示用户，让我们对其结果进行了解。

同步的时候，把本地新增的内容，在服务器上创建分组；把修改的的分组名称，在服务器上进行修改，然后进行同步列表处理，同步操作前，列表界面可能如下所示，有新增记录 ID=-1 的，也有修改后，记录修改标志的。



用户分组的同步按钮操作，是调用一个脚本代码就可以了，具体代码如下所示。

```
        //绑定提交按钮的的点击事件
        function BindSyncDataEvent() {
            $("#btnSyncData").click(function () {
                $.messager.confirm("提交确认", "您确认需要和微信服务器同步分组信息吗？", function (action) {
                    if (action) {
                        //提交数据
                        $("#loading").show();
```

```
                $.ajax({
                    url: '/Group/SyncGroup',
                    type: 'post',
                    dataType: 'json',
                    success: function (data) {
                        if (data.Success) {
                            $("#grid").datagrid("reload");
                            $.messager.alert("提示", "同步成功
");
                        }
                        else {
                            $.messager.alert("提示", "同步失败:"
+ data.ErrorMessage);
                        }
                    },
                    data: ''
                });

                $("#loading").fadeOut(500);
            }
        });
    });
```

```
        }
```

其中上面红色部分就是通过 Jquery 调用的 MVC 的控制器方法，具体函数代码如下所示。

```
        /// <summary>
        /// 同步服务器的分组信息
        /// </summary>
        /// <returns></returns>
        public ActionResult SyncGroup()
        {
            string accessToken = GetAccessToken();
            CommonResult result =
BLLFactory<Group>.Instance.SyncGroup(accessToken);
            return ToJsonContent(result);
        }
```

从上面，我们没有看到太多的逻辑，为了方便我对他们进行了进一步的封装，把它放到了业务逻辑层进行处理了。具体我们看看它的代码逻辑吧，这里为了所有的数据库操作更加快捷和完整，使用了事务的操作，我把相关的代码贴出来，方便大家了解逻辑。

```
        /// <summary>
        /// 同步服务器的分组信息
        /// </summary>
        /// <returns></returns>
        public CommonResult SyncGroup(string accessToken)
        {
            CommonResult result = new CommonResult();

            try
            {
                IUserApi api = new UserApi();

                using (DbTransaction trans =
baseDal.CreateTransaction())
                {
                    //先把本地标志 groupId = -1 未上传的记录上传到服务
器,然后进行本地更新
                    string condition = string.Format("GroupID = '-1' ");
                    List<GroupInfo> unSubmitList =
base.Find(condition);
                    foreach (GroupInfo info in unSubmitList)
                    {
```

```
                    GroupJson groupJson =
api.CreateGroup(accessToken, info.Name);
                    if (groupJson != null)
                    {
                        info.GroupID = groupJson.id;
                        baseDal.Update(info, info.ID, trans);
                    }
                }

                //把标志为修改状态的记录，在服务器上修改
                condition = string.Format("GroupID >=0 and
Modified =1 ");
                List<GroupInfo> unModifyList =
base.Find(condition);
                foreach (GroupInfo info in unModifyList)
                {
                    CommonResult modifyed =
api.UpdateGroupName(accessToken, info.GroupID, info.Name);
                    if (modifyed != null && modifyed.Success)
                    {
                        info.Modified = 0;//重置标志
                        baseDal.Update(info, info.ID, trans);
```

```
            }
        }

        //删除具有删除标志的分组
        //condition = string.Format("GroupID >=100 and
Deleted=1 ");
        //List<GroupInfo> unDeletedList =
base.Find(condition);
        //foreach (GroupInfo info in unDeletedList)
        //{
        //    CommonResult deleted =
api.DeleteGroup(accessToken, info.GroupID, info.Name);
        //    if (deleted != null && deleted.Success)
        //    {
        //        baseDal.Delete(info.ID, trans);
        //    }
        //}

        List<GroupJson> list =
api.GetGroupList(accessToken);
        foreach (GroupJson info in list)
        {
```

```
                    UpdateGroup(info, trans);
                }

                try
                {
                    trans.Commit();
                    result.Success = true;
                }
                catch
                {
                    trans.Rollback();
                    throw;
                }
            }
        }
        catch (Exception ex)
        {
            result.ErrorMessage = ex.Message;
        }

        return result;
    }
```

在 Jquery 同步的时候，我们为了避免等待时间过久而无法判断程序是否正常在工作，最好增加一个忙碌的提示操作，因为我们使用了 Ajax 调用，所以我们可以统一设置 Ajax 的忙碌和完成状态，具体设置代码如下所示。

```
//用来统一请求忙碌显示的设置
$.ajaxSetup({
    beforeSend: function () {
        $("#loading").show();
    },
    complete: function () {
        $("#loading").hide();
    }
});
```

如果感兴趣或者体验相关的微信功能，可以关注我的微信了解下。具体效果可以关注我的微信门户：广州爱奇迪，也可以扫描下面二维码进行关注了解。

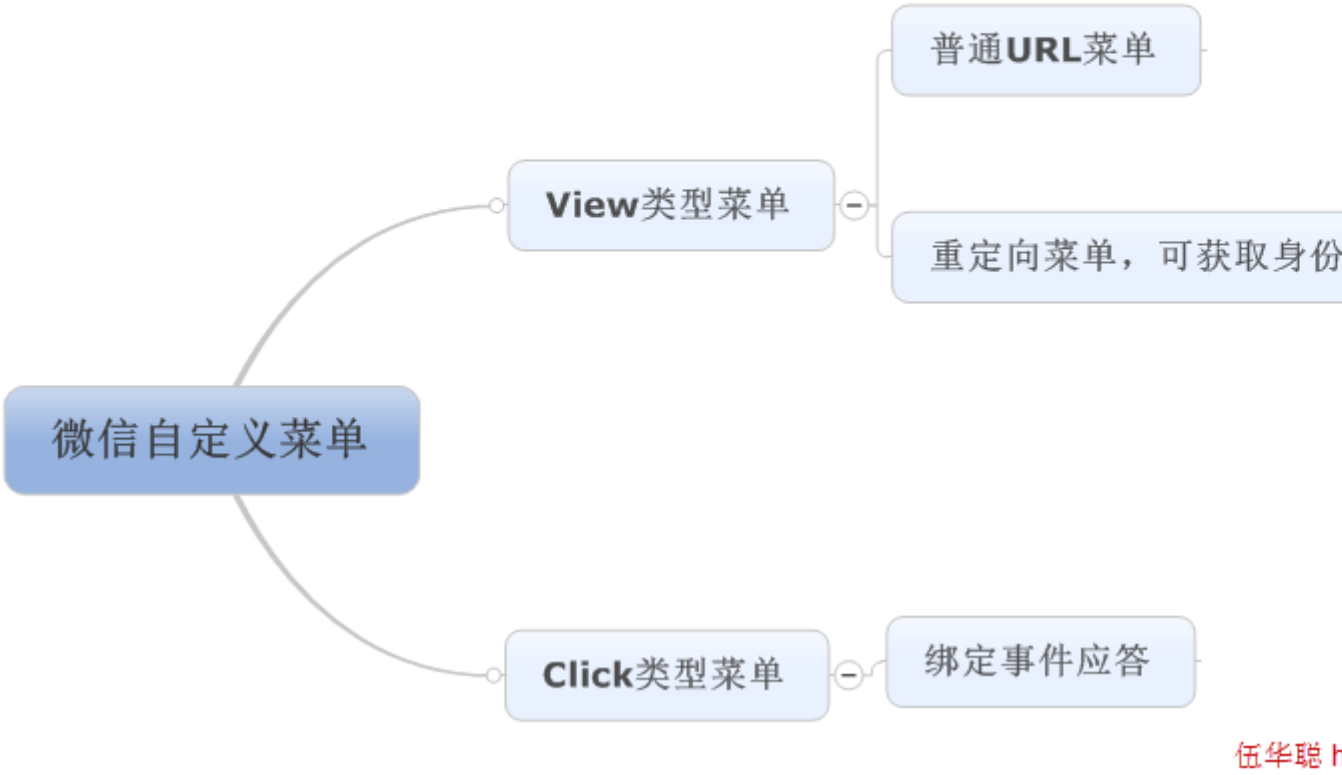
C#开发微信门户及应用(11)--微信菜单的多种表现方式介绍

在前面一系列文章中，我们可以看到微信自定义菜单的重要性，可以说微信公众号账号中 ,菜单是用户的第一印象 ,我们要规划好这些菜单的内容 ,布局等信 息。根据微信菜单的定义，我们可以看到，一般菜单主要分为两种，一种是普通的 Url 菜单（类型为 View 的菜单），一种是事件菜单（类型为 Click 的菜 单），一般情况下，微信的 Url 菜单，是无法获得用户的任何信息的，但微信用户信息非常重要，因此也提供了另外一种方式（类似重定向的方式）来给我们使 用，本篇主要介绍这种重新定向的方式菜单的使用 ,以使我们能够尽可能和用户进行交互。

1、微信自定义菜单的分类

微信对自定义菜单的要求：**目前自定义菜单最多包括 3 个一级菜单，每个一级菜单最多包含 5 个二级菜单。一级菜单最多 4 个汉字 ,二级菜单最多 7 个汉字，多出来的部分将会以“...”代替。**

根据菜单的分类，我们可以把它通过图形进行分类展示：



我对各种微信公众号进行了解，发现多数账号采用的都是普通的 View 类型的菜单链接方式，通过它们链接到自己的微网站上，但也有一些做的好的，如省 立中山图书馆，就能通过重定向的方式，提供一个绑定图书馆用户和微信 OpenID 的入口，绑定后，用户就可以查看借阅的书籍，然后可以通过一键续借功能实 现图书的快速续借功能。

对于这种重定向类型的 Url 菜单事件，微信的说明如下：

如果用户在微信中（Web 微信除外）访问公众号的第三方网页，公众号开发者可以通过此接口获取当前用户基本信息（包括昵称、性别、城市、国家）。利用用户信息，可以实现体验优化、用户来源统计、帐号绑定、用户身份鉴权等功能。**请 注意，“获取用户基本信息接口是在用户和公众号产生消息交互时，才能根据用户 OpenID 获取用户基本信息，而网页授权的方式获取用户基本信息，则无需消 息交互，只是用户进入到公众号的网页，就可弹出请求用户授权的界面，用户授权后，就可获得其基本信息(此过程甚至不需要用户已经关注公众号。)”**

2、重定向类型菜单的 URL

上面说了，重定向类型的菜单分为了两种，其实他们也仅仅是参数 Scope 类型的不同，其他部分也还是一样的。

为了展示，我们在假设用户单击菜单的时候，切换到 http://www.iqidi.com/testwx.ashx 这个页面，并带过来当前用户的 OpenID 等参数信息

对于 scope=snsapi_base 方式的链接如下：

https://open.weixin.qq.com/connect/oauth2/authorize?appid=wx3d81fc2886d86526&redirect_uri=http%3A%2F%2Fwww.iqidi.com%2Ftestwx.ashx&response_type=code&scope=snsapi_base&state=123#wechat_redirect

而对于 scope=snsapi_userinfo 方式的链接如下：

https://open.weixin.qq.com/connect/oauth2/authorize?appid=wx3d81fc2886d86526&redirect_uri=http%3A%2F%2Fwww.iqidi.com%2Ftestwx.ashx&response_type=code&scope=snsapi_userinfo&state=123#wechat_redirect

不过他们给手机客户端的体验是不同的，第一种可以平滑切换，但是第二种会弹出一个对话框供用户确认才能继续。

为了演示上面两种获取数据的不同，我把他们传过来的 code 的值，用户换取 OpenID 后进行用户信息的解析，他们两者的结果都是一样了。具体测试界面如下所示。

其中 TestWX.ashx 的页面后台代码如下所示：

```
    /// <summary>
    /// TestWX 的摘要说明
    /// </summary>
    public class TestWX : IHttpHandler
    {
        string appId = ""; //换成你的信息
        string appSecret = ""; //换成你的信息

        public void ProcessRequest(HttpContext context)
        {
            context.Response.ContentType = "text/plain";
            string content = "";

            if (context.Request != null && context.Request.Url != null)
            {
                NameValueCollection list =
HttpUtility.ParseQueryString(context.Request.Url.Query);
                foreach (string key in list.AllKeys)
                {
```

```
                content += string.Format("{0}:{1} \r\n", key,
list[key]);
            }
        }

        string code = context.Request.QueryString["code"] ?? "";
        if (!string.IsNullOrEmpty(code))
        {
            IBasicApi api = new BasicApi();
            try
            {
                AppConfig config = new AppConfig();
                appId = config.AppConfigGet("AppId");//从配置中
获取微信程序 ID
                appSecret = config.AppConfigGet("AppSecret");//
从配置中获取微信程序秘钥

                AccessTokenResult result =
api.GetAccessToken(appId, appSecret, code);
                if (result != null)
                {
```

```
                content += string.Format("openid:{0}\r\n",
result.openid);

                string token = api.GetAccessToken(appId,
appSecret);
                IUserApi userApi = new UserApi();
                UserJson userDetail =
userApi.GetUserDetail(token, result.openid);
                if (userDetail != null)
                {
                    content += string.Format("nickname:{0}
sex:{1}\r\n", userDetail.nickname, userDetail.sex);
                    content += string.Format("Location:{0} {1}
{2} {3}\r\n", userDetail.country, userDetail.province, userDetail.city,
userDetail.language);
                    content += string.Format("HeadUrl:{0}
\r\n", userDetail.headimgurl);
                    content +=
string.Format("subscribe:{0},{1}\r\n", (userDetail.subscribe == 1) ? "已订
阅" : "未订阅", userDetail.subscribe_time.GetDateTime());
                }
            }
```

```
                }
                catch { }
            }

            context.Response.Write(content);
        }
```

在上面的代码中，我主要分为几步，一个是打印当前用户重定向过来的链接的参数信息，代码如下。

```
                NameValueCollection list =
HttpUtility.ParseQueryString(context.Request.Url.Query);
                foreach (string key in list.AllKeys)
                {
                    content += string.Format("{0}:{1} \r\n", key,
list[key]);
                }
```

然后获取到 Code 参数后，通过 API 接口，获取 AccessTokenResult 的数据，这里面有用户的 OpenID

```
AccessTokenResult result = api.GetAccessToken(appId, appSecret, code);
```

当正常调用后，我们把用户标识的 OpenID 进一步进行解析，调用 API 获取用户的详细信息，具体代码如下所示。

UserJson userDetail = userApi.GetUserDetail(token, result.openid);

当我们把用户的相关信息获取到了，就可以做各种用户信息的展示了，如下代码所示。

```
                    if (userDetail != null)
                    {
                        content += string.Format("nickname:{0} sex:{1}\r\n", userDetail.nickname, userDetail.sex);
                        content += string.Format("Location:{0} {1} {2} {3}\r\n", userDetail.country, userDetail.province, userDetail.city, userDetail.language);
                        content += string.Format("HeadUrl:{0} \r\n", userDetail.headimgurl);
                        content += string.Format("subscribe:{0},{1}\r\n", (userDetail.subscribe == 1) ? "已订阅" : "未订阅", userDetail.subscribe_time.GetDateTime());
                    }
```

3、重定向链接菜单的用途

这种菜单就是需要指定域名，在微信后台中进行设置，重定向的链接必须属于这个域名之中，否则不会转到你希望的链接。

这个方式，让我们的微信应用程序后台可以获得用户的标识、用户详细信息等，我们就可以用来绑定和用户相关的业务信息了，如上面提到的图书馆借阅信 息，送水客户的信息，客户的积分信息，或者可以和后台账号进行关联实现更加复杂的应用等。用户的身份信息如此重要，如果结合到我们的 CRM 系统、业务管理系统，就可以发挥用户信息应用的作用了。

以上就是我对这个类型菜单链接的应用了解，具体还需要进一步深化其应用，希望和大家共同探讨这方面的应用场景。

C#开发微信门户及应用(12)-使用语音处理

我们知道，微信最开始就是做语音聊天而使得其更加流行的，因此语音的识别处理自然也就成为微信交流的一个重要途径，微信的开发接口，也提供了对语音 的消息请求处理。本文主要介绍如何利用语音的识别，对 C#开发的微信门户应用的整个事件链的处理操作，使得在我们的微信账号里面，更加方便和多元化对用户 的输入进行处理。

1、微信语音接口的定义 0

微信的 API 这么定义语音的识别的：**开通语音识别功能，用户每次发送语音给公众号时，微信会在推送的语音消息 XML 数据包中，增加一个 Recongnition 字段。**

语音的消息格式如下所示。

```
<xml>
<ToUserName><![CDATA[toUser]]></ToUserName>
<FromUserName><![CDATA[fromUser]]></FromUserName>
<CreateTime>1357290913</CreateTime>
<MsgType><![CDATA[voice]]></MsgType>
<MediaId><![CDATA[media_id]]></MediaId>
<Format><![CDATA[Format]]></Format>
<MsgId>1234567890123456</MsgId>
</xml>
```

参数	描述
	开发者微信号
ne	发送方帐号（一个 OpenID）
	消息创建时间 （整型）

	语音为 voice
	语音消息媒体 id，可以调用多媒体文件下载接口拉取数据。
	语音格式，如 amr，speex 等
	消息 id，64 位整型

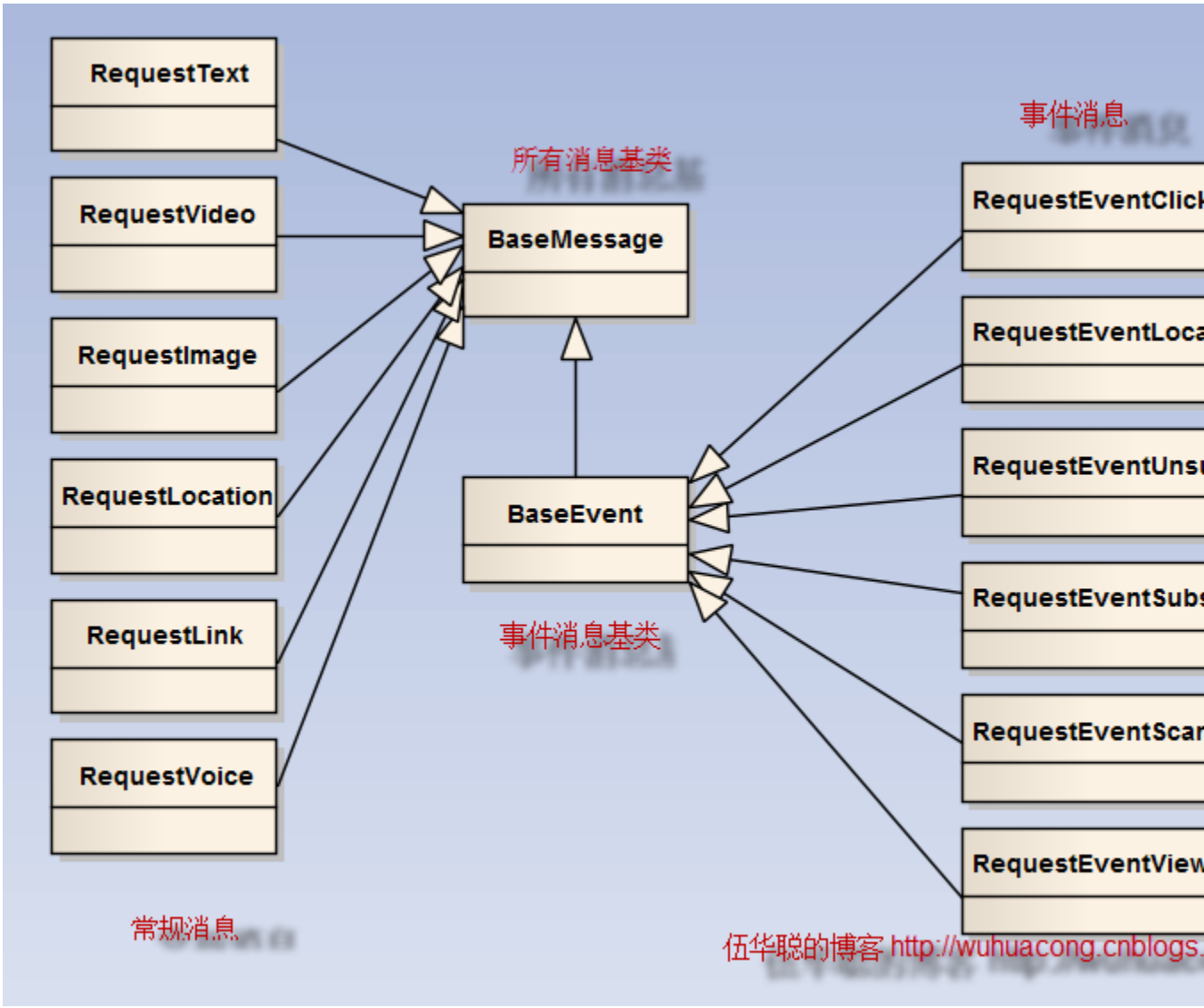
根据以上微信接口的定义，我们可以定义一个实体类来对消息的传递进行处理，如下所示。

```
    /// <summary>
    /// 接收的语音消息
    /// </summary>
    [System.Xml.Serialization.XmlRoot(ElementName = "xml")]
    public class RequestVoice : BaseMessage
    {
        public RequestVoice()
        {
            this.MsgType = RequestMsgType.Voice.ToString().ToLower();
        }

        /// <summary>
        /// 语音格式，如 amr，speex 等
```

```
        /// </summary>
        public string Format { get; set; }

        /// <summary>
        /// 语音消息媒体 id，可以调用多媒体文件下载接口拉取数据。
        /// </summary>
        public string MediaId { get; set; }

        /// <summary>
        /// 消息 ID
        /// </summary>
        public Int64 MsgId { get; set; }

        /// <summary>
        /// 语音识别结果，UTF8 编码
        /// </summary>
        public string Recognition   { get; set; }

    }
```

我们看到，这里我们最感兴趣的是语音的识别结果，也就是 Recognition 的字段，这个就是微信服务器自动根据用户的语音转换过来的内容，我测试过，识别率还是非常高的。

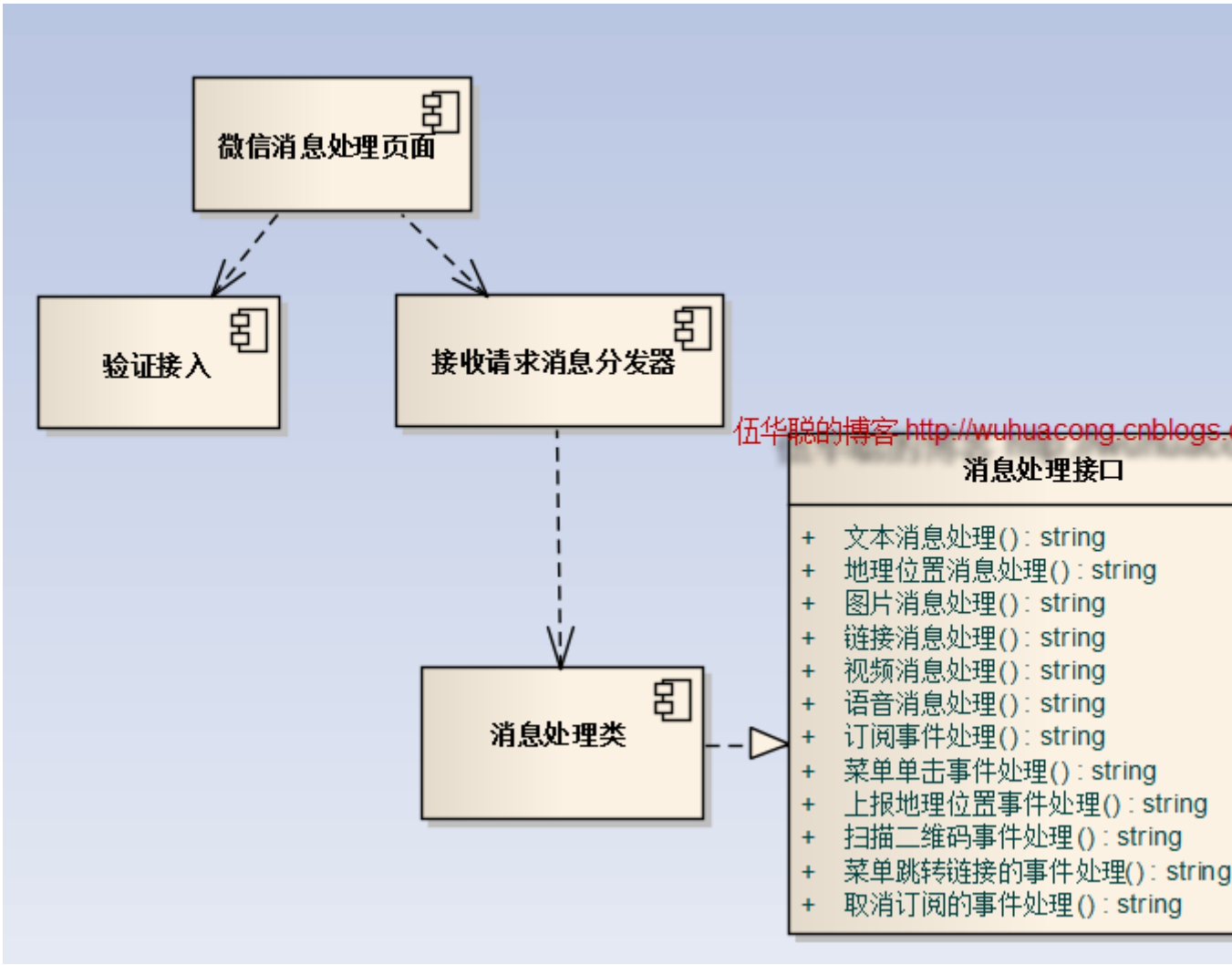
这个实体类，在整个微信应用的消息传递中的关系如下所示：



2、语音的处理操作

明确了上面的语音对象实体，我们就可以看看它们之间是如何处理的。

微信消息的处理逻辑如下图所示。



其中我们来看看语音的处理操作，我的代码处理逻辑如下所示。

```
/// <summary>
/// 对语音请求信息进行处理
/// </summary>
/// <param name="info">语音请求信息实体</param>
/// <returns></returns>
```

```
        public string HandleVoice(Entity.RequestVoice info)
        {
            string xml = "";
            // 开通语音识别功能，用户每次发送语音给公众号时，
            // 微信会在推送的语音消息 XML 数据包中，增加一个
Recongnition 字段。
            if (!string.IsNullOrEmpty(info.Recognition))
            {
                TextDispatch dispatch = new TextDispatch();
                xml = dispatch.HandleVoiceText(info, info.Recognition);
            }
            else
            {
                xml = "";
            }

            return xml;
        }
```

在这里，我先看看，是否获得了微信的语音识别结果，如果获得，那么这个时候，就是和处理用户文本输入的操作差不多了，因此把它转给 TextDispatch 的处理类进行处理。

其中这里面的处理逻辑如下所示。

首先我根据识别结果，寻找是否用户读出了微信门户的菜单名称，如果根据语音结果找到对应的菜单记录，那么我们执行菜单事件（如果是 URL 的 View 类型菜单，我们没办法重定向到指定的链接，因此给出一个链接文本提示，给用户单击进入；如果没有找到菜单记录，那么我们就把语音识别结果作为一般的事件进行处理，如果事件逻辑没有处理，那么我们最后给出一个默认的语音应答提示结果就可以了。

具体的处理代码如下所示。

```
        /// <summary>
        /// 如果用户用语音读出菜单的内容，那么我们应该先根据菜单对应的事件触发，最后再交给普通事件处理
```

```
        /// </summary>
        /// <param name="info"></param>
        /// <returns></returns>
        public string HandleVoiceText(BaseMessage info, string
voiceText)
        {
            string xml = "";
            MenuInfo menuInfo =
BLLFactory<Menu>.Instance.FindByName(voiceText);
            if (menuInfo != null)
            {
                #region 如果找到菜单对象的处理
                if (menuInfo.Type == "click")
                {
                    //模拟单击事件
                    RequestEventClick eventInfo = new
RequestEventClick();
                    eventInfo.CreateTime = info.CreateTime;
                    eventInfo.EventKey = menuInfo.Key;
                    eventInfo.FromUserName = info.FromUserName;
                    eventInfo.ToUserName = info.ToUserName;
```

```
                    xml = base.DealEvent(eventInfo,
eventInfo.EventKey);
                }
                else
                {
                    //由于无法自动切换到连接，
                    //转换为连接文本供用户进入
                    string content = string.Format("请单击链接进入<a
href=\"{0}\">{1}</a> ", menuInfo.Url, menuInfo.Name);

                    ResponseText textInfo = new ResponseText(info);
                    textInfo.Content = content;

                    xml = textInfo.ToXml();
                }
                #endregion
            }
            else
            {
                //交给事件机制处理
                if (string.IsNullOrEmpty(xml))
                {
```

```
                    xml = HandleText(info, voiceText);
                }
            }

            //最后如果没有处理到，那么提示用户的语音内容
            if (string.IsNullOrEmpty(xml))
            {
                ResponseText textInfo = new ResponseText(info);
                textInfo.Content = string.Format("非常抱歉，您输入的语音内容没有找到对应的处理方式。您的语音内容为：{0}", voiceText);
                xml = textInfo.ToXml();
            }

            return xml;
        }
```

微信门户测试界面效果如下所示。


●○○○○ 中国联通
21:12
36%
微信(5)
广州爱奇迪
21:12
2''
广州爱奇迪 客户关系管理系统
5月26日
客户关系管理系统
客户关系管理
系统管理
帮助
客户往来
客户管理
客户服务
商品及销售
通信管理
这是一款专业的客户关系管理软件(CRM管
按住 说话


●●○○○ 中国联通
21:12
36%
微信(5)
广州爱奇迪
21:12
1''
非常抱歉，您输入的语音内容没有找到对应的处理方式。您的语音内容为：你好
伍华聪的博客
http://wuhuacong.cnblogs.com
按住 说话

为了方便对客户会话的记录，我的微信门户后台，会记录用户的语音输入内容，如下所示。

当然，微信后台的管理界面，也能够查到相应的语音记录，界面如下所示。

以上就是我对微信语音的消息定义和事件处理的逻辑，其实语音是一个重要的输入，如果正确的识别内容，比手工输入的效果更好，给用户提供另外一种高效的输入和事件处理操作。

这样的处理模式，能够使得我们整个微信门户框架，不管是对于用户的语音输入，还是文本输入，还是菜单事件的处理，都可以融为一体，实现更加完美的衔接。

C#开发微信门户及应用(13)-使用地理位置扩展相关应用

本文继续上一篇《C#开发微信门户及应用(12)-使用语音处理》，继续介绍微信的相关应用。我们知道，地理位置信息可以用来做很多相关的应用，除了我们可以知道用户所在的位置，还可以关联出一些地理位置的应用，如天气， 热映影片，附近景点，附近影院，交通事件等等，反正所有和地理位置相关的信息，我们都可以根据需要做一些扩展应用。本文主要介绍利用地理位置信息，如何构 建使用这些应用的操作。

1、微信的地理位置信息

在使用前 ,我们先来看看微信的接口 ,为我们定义了那些关于与地理位置的信息。

其实地理位置的信息，微信分为了两个方面，一个是接收用户的地理位置请求，一个是用户允许上报地理位置操作，定时发送的地理位置信息。

本文主要介绍基于第一种，用户上报地理位置后，如何处理的相关应用。

地理位置的上报操作，就是在输入的地方，选择+号进行添加地理位置，然后选择当前或者指定的地理位置地图，具体操作如下所示。


中国联通
8:00
100%
微信(5)
广州爱奇迪
酒店管理系统、备件仓库管理系统、配电网络可视化管理系统等；开发框架类产品有：Winform开发框架、WCF开发框架、混合式开发框架、Web开发框架、Socket开发框架、Database
阅读全文
照片
拍摄
位置
名片
我的收藏
语音输入
大众点评
应用

地理位置消息

```
<xml>
<ToUserName><![CDATA[toUser]]></ToUserName>
<FromUserName><![CDATA[fromUser]]></FromUserName>
<CreateTime>1351776360</CreateTime>
<MsgType><![CDATA[location]]></MsgType>
<Location_X>23.134521</Location_X>
<Location_Y>113.358803</Location_Y>
<Scale>20</Scale>
```

```
<Label><![CDATA[位置信息]]></Label>
<MsgId>1234567890123456</MsgId>
</xml>
```

参数	描述
lame	**开发者**微信号
erName	发送方帐号（一个 OpenID）
me	消息创建时间 （整型）
e	location
_X	地理位置维度
_Y	地理位置经度
	地图缩放大小
	地理位置信息
	消息 id，64 位整型

有了上面的地理位置信息，我们在程序里面，需要在消息传递过来的时候，定义一个实体类信息，承载相关的地理位置信息，方便我们进一步的处理操作。

```
/// <summary>
/// 接收的地理位置消息
/// </summary>
```

```
    [System.Xml.Serialization.XmlRoot(ElementName = "xml")]
    public class RequestLocation : BaseMessage
    {
        public RequestLocation()
        {
            this.MsgType =
RequestMsgType.Location.ToString().ToLower();
        }

        /// <summary>
        /// 消息 ID
        /// </summary>
        public Int64 MsgId { get; set; }

        /// <summary>
        /// 地理位置维度
        /// </summary>
        public decimal Location_X { get; set; }

        /// <summary>
        /// 地理位置经度
        /// </summary>
```

```
        public decimal Location_Y { get; set; }

        /// <summary>
        /// 地图缩放大小
        /// </summary>
        public int Scale { get; set; }

        /// <summary>
        /// 地理位置信息
        /// </summary>
        public string Label { get; set; }

    }
```

有了这些信息，我们在信息传递的时候，就能很好得到用户的相关数据了。

如果仅仅为了返回给用户，告诉用户目前的地理位置信息，可以用下面的操作就可以了。

```
        /// <summary>
        /// 对地理位置请求信息进行处理
        /// </summary>
        /// <param name="info">地理位置请求信息实体</param>
```

```
        /// <returns></returns>
        public string HandleLocation(Entity.RequestLocation info)
        {
            string xml = "";

            ResponseText txtinfo = new ResponseText(info);
            txtinfo.Content = string.Format("您发送的地理位置是：{0}",
info.Label);
            xml = txtinfo.ToXml();

            return xml;
        }
```

2、地址位置的应用处理

不过上面的信息，显然不符合我们扩展应用的要求，因此我们进一步进行完善里面对地理位置信息处理的操作。我们进一步把关于地理位置的操作，放到事件处理模块里面进行处理，处理代码如下所示。

```
        /// <summary>
        /// 对地理位置请求信息进行处理
        /// </summary>
        /// <param name="info">地理位置请求信息实体</param>
```

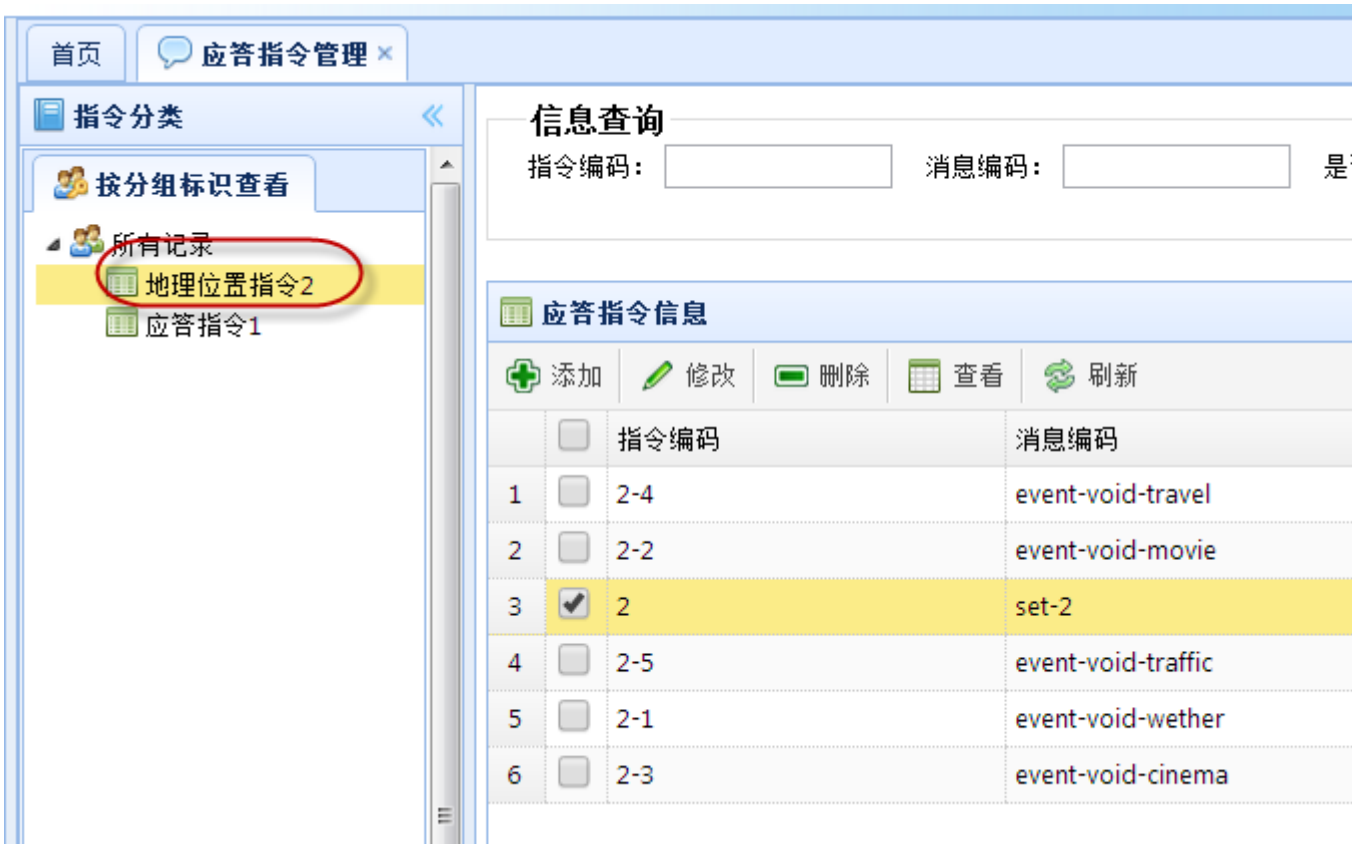
```
        /// <returns></returns>
        public string HandleLocation(Entity.RequestLocation info)
        {
            string xml = "";
            EventDispatch dispatch = new EventDispatch();
            xml = dispatch.DealLocation(info, info.Label, info.Location_Y,
info.Location_X);

            return xml;
        }
```

在处理的时候，我们需要先保存用户的地理位置信息，把它存储到用户的上下文记录里面。这样我们在处理指令的时候，把它获取到，然后传递给相关的方法就可以实现地理位置的扩展应用了。

```
            //保存经纬度
            string location = string.Format("{0},{1}", lat, lon);
            bool result =
BLLFactory<UserSet>.Instance.UpdateUserInput(info.FromUserName,
location);
```

首先对用户地理位置的请求，我根据数据库配置给出了一个用户选择的指令提示，如下所示。

为了对地理位置请求的处理，我定义了一个用于处理这个操作的指令操作



这样整个地理位置的指令操作，就在应答链里面进行很好的跳转管理了。那么为了实现天气、放映影片、附近影院、旅游线路、交通事件等方面的扩展应用，我们应该如何操作呢？

3、地址位置应用扩展

我们知道，百度或者腾讯都提供了一些开放平台，给我们进行各种方式的使用。那么我们这里以使用百度 LBS 平台应用来构建一些模块。

这上面都有很多相关的接口供使用，我们可以根据其提供的数据格式进行封装，然后进行调用处理就可以了。

刚才说了，我配置了一些指令，用来构建相关的应用，指令的最后是一些事件代码的定义，我们对这些末端的事件代码进行处理，就可以给用户返回相关的信息了，总体的操作代码如下所示。

```
        /// <summary>
        /// 其他插件操作，如天气，景点、电影影讯、交通等
        /// </summary>
        /// <param name="info">基础消息</param>
        /// <param name="eventKey">事件标识</param>
        /// <returns></returns>
        public string DealPlugin(BaseMessage info, string eventKey)
```

```
            {
                //LogTextHelper.Info(eventKey);
                string userInput =
BLLFactory<UserSet>.Instance.GetUserInput(info.FromUserName);

                string xml = "";
                switch (eventKey)
                {
                    case "event-void-wether":
                        xml = new WeatherPlugin().Response(info,
userInput);
                        break;
                    case "event-void-movie":
                        xml = new MoviePlugin().Response(info, userInput);
                        break;
                    case "event-void-cinema":
                        xml = new CinemaPlugin().Response(info,
userInput);
                        break;
                    case "event-void-travel":
                        xml = new TravelPlugin().Response(info, userInput);
                        break;
```

```
                case "event-void-traffic":
                    xml = new TrafficEventPlugin().Response(info,
userInput);
                    break;
                default:
                    break;
            }

            return xml;
        }
```

这里以天气为例，说明该如何调用百度的接口的，首先我们封装一下相关的接口调用。

```
        /// <summary>
        /// 根据参数调用百度接口，获取相关的结果数据
        /// </summary>
        /// <param name="location">地理位置</param>
        /// <param name="ak">API 调用键</param>
        /// <returns></returns>
        public BaiduWeatherResult Execute(string location, string ak)
        {
```

```
            location = HttpUtility.UrlEncode(location);
            var url =
string.Format("http://api.map.baidu.com/telematics/v3/weather?locatio
n={0}&output=json&ak={1}", location, ak);

            BaiduWeatherResult result =
BaiduJsonHelper<BaiduWeatherResult>.ConvertJson(url);
            return result;
        }
```

其中的 BaiduWeatherResult 是我根据调用返回的 Json 结果，构建的一个实体类，用来存储返回的内容。具体代码如下所示。

```
    /// <summary>
    /// 天气请求结果 Json 对象
    /// </summary>
    public class BaiduWeatherResult : BaiduResult
    {
        /// <summary>
        /// 天气预报信息
        /// </summary>
```

```
        public List<BaiduWeatherData> results = new
List<BaiduWeatherData>();
    }

    /// <summary>
    /// 城市的天气信息
    /// </summary>
    public class BaiduWeatherData
    {
        /// <summary>
        /// 当前城市
        /// </summary>
        public string currentCity { get; set; }

        /// <summary>
        /// 天气预报信息
        /// </summary>
        public List<BaiduWeatherJson> weather_data = new
List<BaiduWeatherJson>();
    }

    /// <summary>
```

```
/// 天气预报的单条记录 Json 信息
/// </summary>
public class BaiduWeatherJson
{
    /// <summary>
    /// 天气预报时间
    /// </summary>
    public string date { get; set; }

    /// <summary>
    /// 白天的天气预报图片 url
    /// </summary>
    public string dayPictureUrl { get; set; }

    /// <summary>
    /// 晚上的天气预报图片 url
    /// </summary>
    public string nightPictureUrl { get; set; }

    /// <summary>
    /// 天气状况
    /// </summary>
```

```
        public string weather { get; set; }

        /// <summary>
        /// 风力
        /// </summary>
        public string wind { get; set; }

        /// <summary>
        /// 温度
        /// </summary>
        public string temperature { get; set; }
    }
```

为了构建返回给客户的图文数据，我们需要构建一个 News 对象，然后生成 XML 数据返回给服务器进行处理即可。

```
        /// <summary>
        /// 响应用户请求，并返回相应的 XML 数据
        /// </summary>
        /// <param name="info">微信基础信息</param>
        /// <param name="location">地理位置：经纬度坐标或者地名
</param>
```

```
        /// <returns></returns>
        public string Response(BaseMessage info, string location)
        {
            string xml = "";

            //"广州" 或者 "116.305145,39.982368"
            if (!string.IsNullOrEmpty(location))
            {
                BaiduWeatherResult result = Execute(location,
baiduAK);
                if (result != null && result.results.Count > 0)
                {
                    BaiduWeatherData data = result.results[0];
                    if (data != null)
                    {
                        ArticleEntity first = new ArticleEntity();
                        first.Title = string.Format("{0} 天气预报",
data.currentCity);

                        ResponseNews news = new
ResponseNews(info);
                        news.Articles.Add(first);
```

```
                int i = 0;
                foreach (BaiduWeatherJson json in
data.weather_data)
                {
                    ArticleEntity article = new ArticleEntity();
                    article.Title = string.Format("{0}\n{1} {2} {3}",
json.date, json.weather, json.wind, json.temperature);
                    if (i++ == 0)
                    {
                        article.PicUrl = IsDayTime() ?
json.dayPictureUrl : json.nightPictureUrl;
                    }
                    else
                    {
                        article.PicUrl = json.dayPictureUrl;
                    }
                    news.Articles.Add(article);
                }

                xml = news.ToXml();
            }
```

```
                }
            }

            return xml;
        }
```

这样就很好实现了整体的功能了 ,具体界面功能可以访问我的微信(广州爱奇迪) 进行了解，下面是功能截图供参考。


中国联通
8:02
100%
微信(5)
广州爱奇迪
0 转人工客服 # 重看该内容 * 返回上一级 ** 返回顶级
1
广州市 天气预报
周三(今天, 实时：27°C)
多云 微风 34 ~ 25°C
周四
多云 微风 34 ~ 25°C


中国联通
10:49
98%
微信(5)
广州爱奇迪
X战警：逆转未来
归来
通过“陆焉识回家”呈现一段悲欢离合的爱情故事 陈道明/巩俐/闫妮 剧情
超凡蜘蛛侠2
一部精彩的青春爱情片，“蜘
软件产品
框架产品
相关链接


●○○○○ 中国联通
8:05
100%
微信(5)
广州爱奇迪
3
金逸国际电影城(广州太阳城店)
广州市广州大道北1811号太阳城广场3楼 (020)36732001 评分：4
华影梅花园影城
白云区广州大道北28号梅花园商业广场5楼(梅花园地铁站B出口附近) (020)37322088 评分：4
横店电影城
天河区长兴路13号高德汇购物中心3楼（近天河客运站150米处） 020-

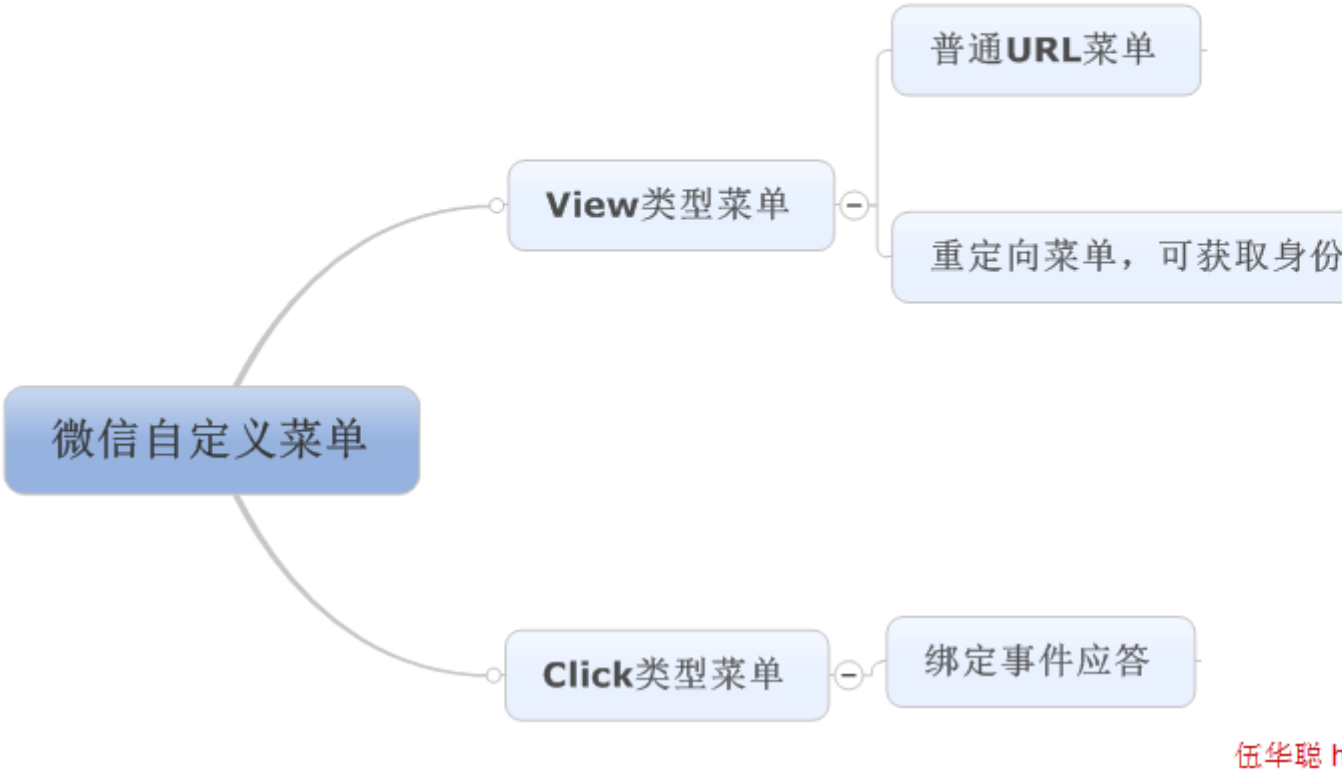

●●●○○ 中国联通
8:03
100%
微信(5)
广州爱奇迪
4
铺天盖地的粤语、大街小巷的鲜花、美味的粥品靓汤，让现代化的广州，仍保留着市井气息，漫步街头，各色靓仔靓女定会吸引你的眼球。
一日游
早上先去越秀公园。越秀公园是广州最大的综合性公园，里面有广州城的代表性雕塑：五羊石像。游览完之后可以在附近吃一下午饭，也可以把好胃口留到下午吃。
快中午的时候去陈家祠游览。陈家祠

C#开发微信门户及应用(14)-在微信菜单中采用重定向获取用户数据

我曾经在系列文章中的《C#开发微信门户及应用(11)--微信菜单的多种表现方式介绍》 中介绍了微信菜单里面的重定向操作，通过这个重定向操作，我们可以获取一个 code 值 ,然后获取用户的 openID ,进而就能获取到更多的用户信息，这个 在会员信息的场景里面用的很多，本篇介绍在网站中迅速配置这样的菜单链接 ,并介绍如何在后台获取相关的用户信息 ,实现页面数据个性化的展现操作。

我们知道 ,微信的自定义菜单分为两大类 ,分别对应 Click 类型和 View 类型的，

而重定向属于 View 类型的一种，如下所示。



伍华聪 h

1、微信重定向菜单的配置

微信重定向的菜单，就是通过传入一个地址参数，让微信服务器进行跳转，它的

主要规则如下所示。

对于 scope=snsapi_base 方式的链接如下：

https://open.weixin.qq.com/connect/oauth2/authorize?appid=wx3d81fc2886d86526&redirect_uri=http%3A%2F%2Fwww.iqidi.com%2Ftestwx.ashx&response_type=code&scope=snsapi_base&state=123#wechat_redirect

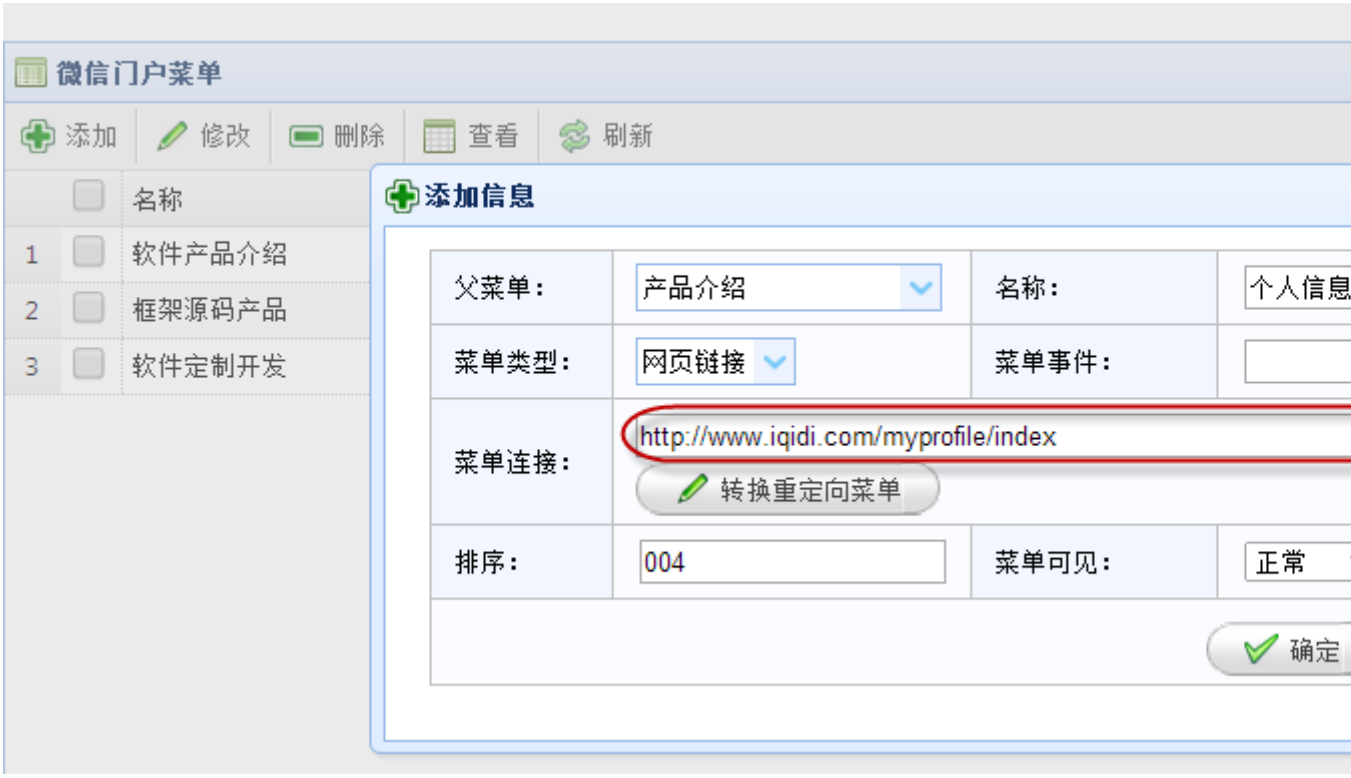
而对于 scope=snsapi_userinfo 方式的链接如下：

https://open.weixin.qq.com/connect/oauth2/authorize?appid=wx3d81fc2886d86526&redirect_uri=http%3A%2F%2Fwww.iqidi.com%2Ftestwx.ashx&response_type=code&scope=snsapi_userinfo&state=123#wechat_redirect

这两个菜单链接主要就是对我们给定的链接地址进行 UrlEncode 处理，然后把它赋值给参数 redirect_uri 实现的。

由于链接地址比较长，如果每次需要在配置菜单的时候，都复制过来修改，非常不方便，我们可以在自定义菜单的配置界面里面，增加一个按钮功能，对内容进行处理，以便实现我们需要的地址转换，我的门户应用平台对自定义菜单的操作就是基于这个思路实现。

默认我们只需要填写一个需要重定向的 url 地址就可以了，如下所示。



如果需要配置成重定向的菜单链接地址，那么调用【转换重定向菜单】按钮操作，

使用脚本函数进行转换就可以了，转换后的结果如下所示。

原来就是利用后台的javascript实现参数的URL转码，还需要获取后台的AppId，

这样才能构造成完整的地址连接。

2、脚本转换操作的实现代码

前面说了，第一是需要实现 URL 转码，第二是获取后台的 AppId，然后生成一个完整的 URL 就可以了。为了避免大家的重复研究，我把这部分代码贴出来一起学习下。

在使用前，我们还需要注意一个问题，就是重定向到指定页面后，这个页面会带有一个 code 的参数，这个参数非常重要，我们需要获取出来，当然也是通过 javascript 来获取对应的 code 参数了。

这个逻辑可以用一个脚本函数来实现，如下所示

```
function getUrlVars(){
    var vars = [], hash;
    var hashes =
window.location.href.slice(window.location.href.indexOf('?') +
1).split('&');
    for(var i = 0; i < hashes.length; i++)
    {
        hash = hashes[i].split('=');
        vars.push(hash[0]);
        vars[hash[0]] = hash[1];
    }
    return vars;
```

```
        }
```

定义了这个函数后，我们在重定向的页面里面，可以获取 code 参数的操作如下所示。

```
var code = getUrlVars()["code"];
```

先放下这些，我们先来讨论如何把链接地址转换为需要的链接地址操作。

我们为了实现链接地址的互相转换（为了方便），我们可以判断链接地址是否含有 qq 的域名就可以了。

```
if (url.indexOf("https://open.weixin.qq.com/connect/oauth2/authorize?") == 0) {
    var redirect_uri = getUrlVars(url)["redirect_uri"];
    if (redirect_uri != "") {
        var newUrl = decodeURIComponent(redirect_uri);
        $("#" + ctrlName).val(newUrl);
    }
}
```

而如果是我们输入的正常链接，那么就应该把它转换为重定向的链接地址，如下所示。

```
else {
                    var newUrl = encodeURIComponent(url);
                    var reNewUrl =
"https://open.weixin.qq.com/connect/oauth2/authorize?appid=@ViewB
ag.appid&redirect_uri=" + newUrl +
"&response_type=code&scope=snsapi_base&state=123#wechat_redire
ct";
                    $("#" + ctrlName).val(reNewUrl);
                }
```

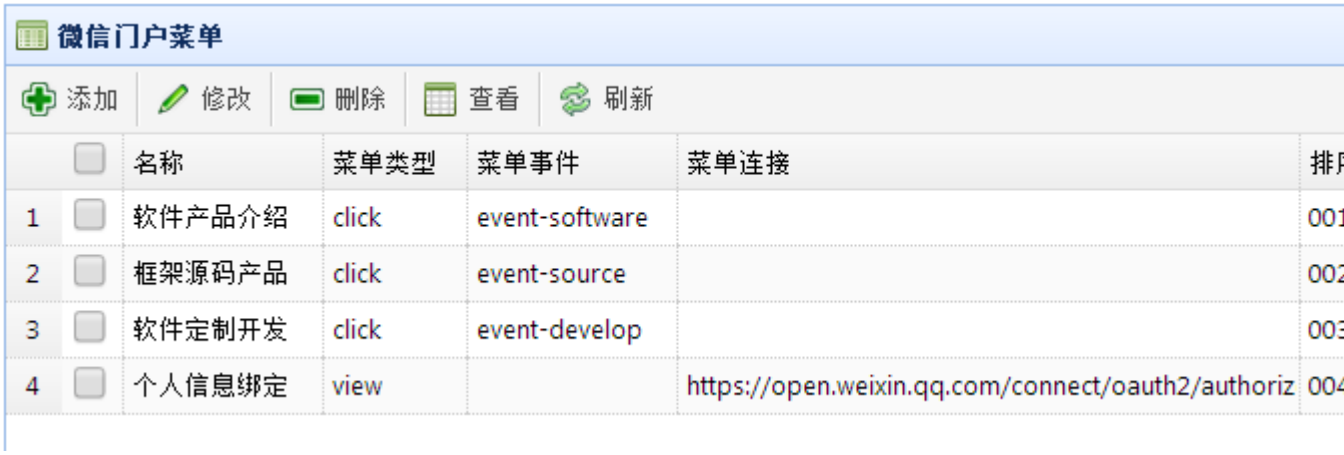
其中重定向链接需要带有一个当前微信开发用户的 appId,这个不是固定的，是不同的开发人员都不一样的东西，这里使用了 MVC 的动态对象进行绑定：@ViewBag.appid。

在对应的 MenuController 控制器里面，给它赋值就可以了。

```
        /// <summary>
        /// 默认的视图控制方法
        /// </summary>
        /// <returns></returns>
        public override ActionResult Index()
        {
            ViewBag.appid = GetAppId();
```

```
            return View();
        }
```

这样配置后的重定向菜单地址列表就如下所示了，我们打开对应的记录详细页面，可以通过页面里面的功能按钮，随时对重定向菜单的地址进行转换，方便了解详细的链接内容。



		名称	菜单类型	菜单事件	菜单连接	排
1		软件产品介绍	click	event-software		001
2		框架源码产品	click	event-source		002
3		软件定制开发	click	event-develop		003
4		个人信息绑定	view		https://open.weixin.qq.com/connect/oauth2/authoriz	004

3、重定向页面的设计及处理

配置了上面的链接地址后，我们需要在网站里面增加这样的一个页面进行处理用户的信息，一般情况下，我们可能是为了方便用户查看自己的微信基础信息， 也为了给用户绑定用户个人数据使用的用途的，如用户可以绑定手机、Email 邮箱等操作，还可以绑定和业务系统相关的用户名。这样用户就可以快速注册会员 或者和后台的系统进行关联了。

我设计的两个用户信息展示界面如下所示。

这两个界面主要使用了 Jquery Mobile 的相关内容，对界面进行了处理，整个模块结合了短信验证码的方式，对用户的手机进行验证处理，这样能够更高效的实现信息准确的绑定操作，当 然，还可以结合外部系统，绑定用户的账号密码，这样用户可以在微信进入微网站平台进行购物、数据维护、业务管理等操作了，其实一旦绑定外部系统的 ID，也 就是提供了一个快速进行外部系统的入口了。


正在搜索...
23:11
15%
返回
我的个人信息
重要信息绑定
基础信息
昵称: 伍华聪
性别: 男
地区: 中国-广东-广州
关注时间: 2014-06-09T09:21:33

具体的内容在下一篇继续介绍了。

C#开发微信门户及应用(15)-微信菜单增加扫一扫、发图片、发地理位置功能

前面介绍了很多篇关于使用 C#开发微信门户及应用的文章，基本上把当时微信能做的接口都封装差不多了 ,微信框架也积累了不少模块和用户 ,最近发现微 信公众平台增加了不少内容，特别是在自定义菜单里面增加了扫一扫、发图片、发

地理位置功能，这几个功能模块很重要，想想以前想在微信公众号里面增加一个扫 描二维码的功能，都做不了，现在可以了，还可以拍照上传等功能，本文主要介绍基于我前面的框架系列文章，进一步介绍如何集成和使用这些新增功能。

1、微信几个功能的官方介绍

1）. 扫码推送事件

用户点击按钮后，微信客户端将调起扫一扫工具，完成扫码操作后显示扫描结果（如果是 URL，将进入 URL），且会将扫码的结果传给开发者，开发者可以下发消息。

2）. 扫码推送事件，且弹出“消息接收中”提示框

用户点击按钮后，微信客户端将调起扫一扫工具，完成扫码操作后，将扫码的结果传给开发者，同时收起扫一扫工具，然后弹出“消息接收中”提示框，随后可能会收到开发者下发的消息。

3）. 弹出系统拍照发图

用户点击按钮后，微信客户端将调起系统相机，完成拍照操作后，将拍摄的相片发送给开发者，并推送事件给开发者，同时收起系统相机，随后可能会收到开发者下发的消息。

4）. 弹出拍照或者相册发图

用户点击按钮后，微信客户端将弹出选择器供用户选择“拍照”或者“从手机相册选择”。用户选择后即走其他两种流程。

5）. 弹出微信相册发图器

用户点击按钮后，微信客户端将调起微信相册，完成选择操作后，将选择的相片发送给开发者的服务器，并推送事件给开发者，同时收起相册，随后可能会收到开发者下发的消息。

6）. 弹出地理位置选择器

用户点击按钮后，微信客户端将调起地理位置选择工具，完成选择操作后，将选择的地理位置发送给开发者的服务器，同时收起位置选择工具，随后可能会收到开发者下发的消息。

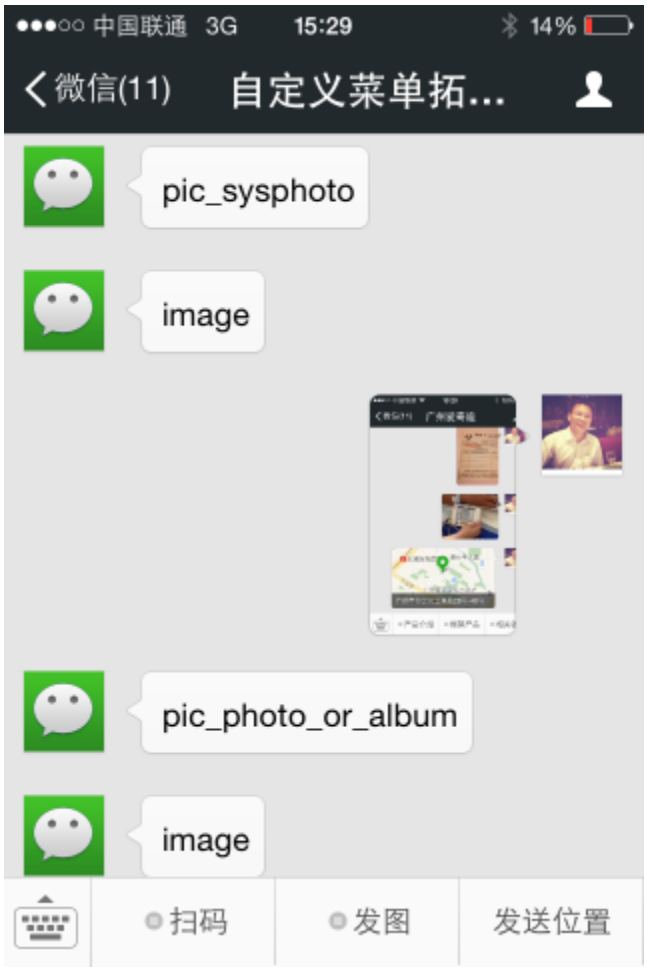
但请注意，以上新增能力，均仅支持微信 iPhone5.4.1 以上版本，和 Android5.4 以上版本的微信用户，旧版本微信用户点击后将没有回应，开发者也不能正常接收到事件推送。

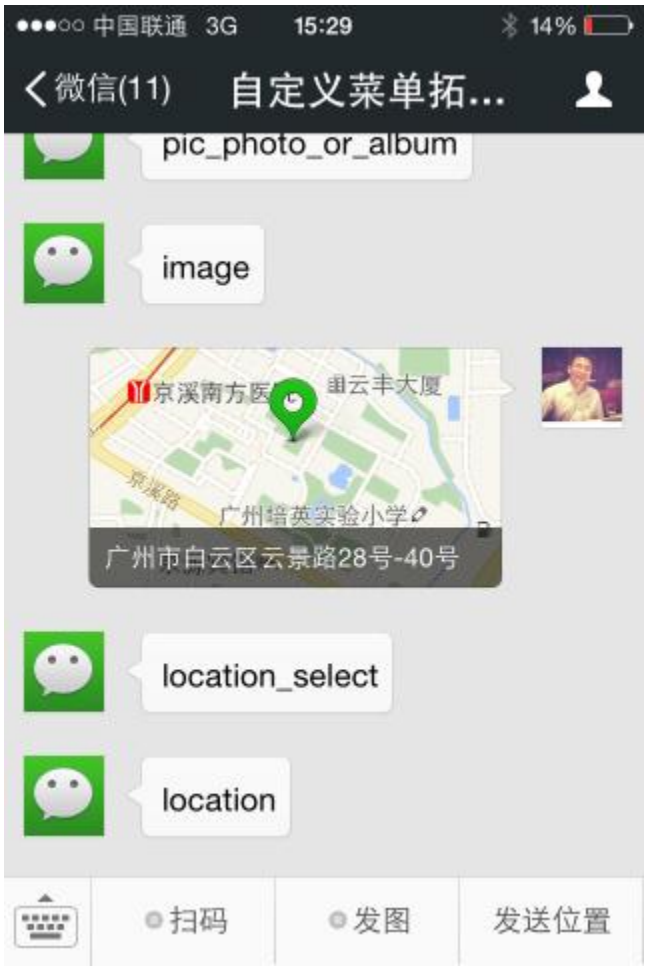
2、微信新菜单功能的测试公众号

微信不仅增加了这些功能模块的支持，还考虑到我们开发人员的方便，增加了一个叫做“menutest"的公众号，方便我们测试。我们在公众号搜索“menutest"，然后关注它即可进行测试几个新增功能了。

“menutest"的公众号名称是”自定义菜单拓展测试“，我关注它并进行了测试，二维码、图片、地理位置都很 OK，本身能够响应这些事件，并且图片、地理位置自身还能出现一个对应的事件，如下所示。

图片发送可以分为拍照、拍照和相册、微信相册三类，感觉后面两个有点类似，但有这些功能都很不错的。


●●●○○ 中国联通 3G 15:29 14%
微信(11)
自定义菜单拓...
pic_sysphoto
image
pic_photo_or_album
image
扫码
发图
发送位置



3、改进菜单对象和提交菜单

前面说了，微信提供这些功能，可以在菜单里面进行集成，也就是菜单的类型由原来 CLICK/VIEW 两种，变为现在 8 种类型，增加 2 个扫码操作、3 种图片操作、1 种地理位置操作。

因此把菜单的枚举类型扩展一下，如下所示。

```
/// <summary>
/// 菜单按钮类型
/// </summary>
```

```
public enum ButtonType
{
    /// <summary>
    /// 点击
    /// </summary>
    click,

    /// <summary>
    /// Url
    /// </summary>
    view,

    /// <summary>
    /// 扫码推事件的事件推送
    /// </summary>
    scancode_push,

    /// <summary>
    /// 扫码推事件且弹出“消息接收中”提示框的事件推送
    /// </summary>
    scancode_waitmsg,
```

```
        /// <summary>
        /// 弹出系统拍照发图的事件推送
        /// </summary>
        pic_sysphoto,

        /// <summary>
        /// 弹出拍照或者相册发图的事件推送
        /// </summary>
        pic_photo_or_album,

        /// <summary>
        /// 弹出微信相册发图器的事件推送
        /// </summary>
        pic_weixin,

        /// <summary>
        /// 弹出地理位置选择器的事件推送
        /// </summary>
        location_select
    }
```

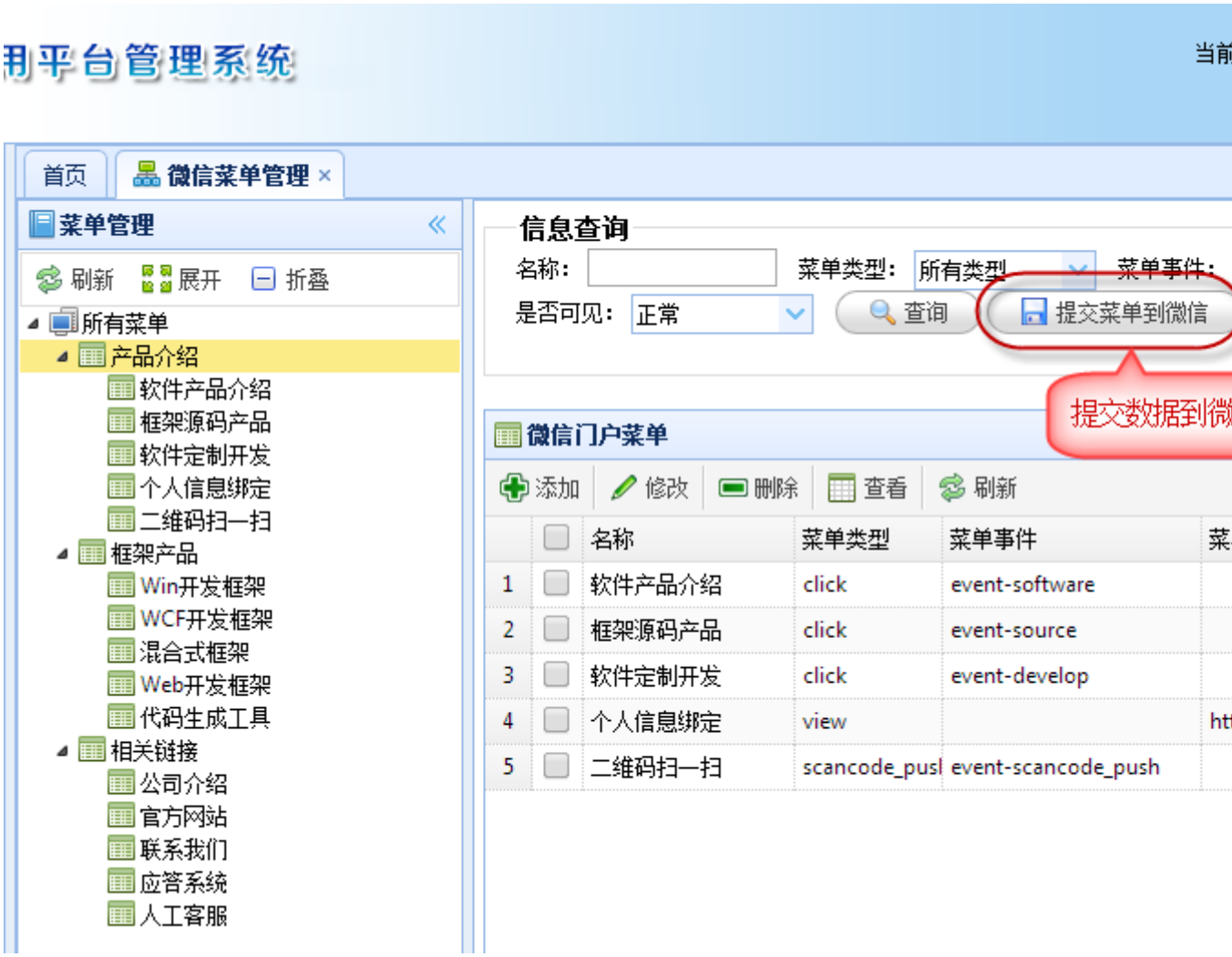
然后在 Winform 里面调用创建菜单操作代码如下所示：

```
        private void btnCreateMenu_Click(object sender, EventArgs e)
        {
            MenuJson productInfo = new MenuJson("新功能测试", new
MenuJson[] {
                new MenuJson("扫码推事件",
ButtonType.scancode_push, "scancode_push")
                ,new MenuJson("系统拍照发图",
ButtonType.pic_sysphoto, "pic_sysphoto")
                , new MenuJson("拍照相册发图",
ButtonType.pic_photo_or_album, "pic_photo_or_album")
                , new MenuJson("微信相册发图", ButtonType.pic_weixin,
"pic_weixin")
                , new MenuJson("地理位置选择",
ButtonType.location_select, "location_select")
            });

            MenuJson frameworkInfo = new MenuJson("框架产品", new
MenuJson[] {
                new MenuJson("Win 开发框架", ButtonType.click, "win"),
                new MenuJson("WCF 开发框架", ButtonType.click, "wcf"),
                new MenuJson("混合式框架", ButtonType.click, "mix"),
```

```
                    new MenuJson("Web 开发框架", ButtonType.click,
"web")
                    ,new MenuJson("代码生成工具", ButtonType.click,
"database2sharp")
                });

                MenuJson relatedInfo = new MenuJson("相关链接", new
MenuJson[] {
                    new MenuJson("公司介绍", ButtonType.click,
"event_company"),
                    new MenuJson("官方网站", ButtonType.view,
"http://www.iqidi.com"),
                    new MenuJson("联系我们", ButtonType.click,
"event_contact"),
                    new MenuJson("应答系统", ButtonType.click, "set-1"),
                    new MenuJson("人工客服", ButtonType.click,
"event_customservice")
                });

                MenuListJson menuJson = new MenuListJson();
                menuJson.button.AddRange(new MenuJson[] { productInfo,
frameworkInfo, relatedInfo });
```

```
            if (MessageUtil.ShowYesNoAndWarning("您确认要创建菜单吗") == System.Windows.Forms.DialogResult.Yes)
            {
                IMenuApi menuBLL = new MenuApi();
                CommonResult result = menuBLL.CreateMenu(token, menuJson);
                Console.WriteLine("创建菜单：" + (result.Success ? "成功" : "失败:" + result.ErrorMessage));
            }
        }
```

当然，一般情况下我们都是在 Web 后台系统进行的，维护菜单都是在自己微信平台上进行菜单管理，然后一次性提交到微信服务器即可。



而在 Web 后台，只需要把数据库的数据变化为 Json 数据提交即可，操作和上面的类似。

```
        /// <summary>
        ///更新微信菜单
        /// </summary>
        /// <returns></returns>
        public ActionResult UpdateWeixinMenu()
        {
```

```
            string token = base.GetAccessToken();
            MenuListJson menuJson = GetWeixinMenu();

            IMenuApi menuApi = new MenuApi();
            CommonResult result = menuApi.CreateMenu(token,
menuJson);
            return ToJsonContent(result);
        }
```

4、微信扫一扫功能集成

前面讲了，有了最新的功能，我们就可以实现扫一扫功能，从而可以扫描条形码，二维码的功能。有了条形码、二维码的快速和识别，我们就能开发一些如条码查询、商品处理等功能了。

这里我们介绍如何在我的微信开发框架里面整合这个扫一扫的功能处理操作。

前面已经增加了一些新功能的测试菜单，我们要做的就是响应这些事件处理，然后对他们进行应答处理就可以了。

下面是根据事件进行的一些 API 跳转处理，我们同时定义了几个相关的实体类用来处理他们的信息，如 RequestEventScancodePush、RequestEventScancodeWaitmsg、RequestEventPicSysphoto 等等。

RequestEventScancodeWaitmsg 实体类的代码如下所示，其他的类似处理。

```
    /// <summary>
    /// 扫码推事件且弹出“消息接收中”提示框的事件推送
    /// </summary>
    [System.Xml.Serialization.XmlRoot(ElementName = "xml")]
    public class RequestEventScancodeWaitmsg : BaseEvent
    {
        public RequestEventScancodeWaitmsg()
        {
            this.MsgType =
RequestMsgType.Event.ToString().ToLower();
            this.Event = RequestEvent.scancode_waitmsg.ToString();
            this.ScanCodeInfo = new ScanCodeInfo();
        }

        /// <summary>
        /// 事件 KEY 值，由开发者在创建菜单时设定
        /// </summary>
        public string EventKey { get; set; }

        /// <summary>
        /// 扫描信息
        /// </summary>
```

```
        public ScanCodeInfo ScanCodeInfo { get; set; }


    }
```

而根据实体类强类型的处理接口流转操作如下所示。

```
                                case RequestEvent.scancode_push:
                                {
                                    //扫码推事件的事件推送
                                    RequestEventScancodePush info =
XmlConvertor.XmlToObject(postStr, typeof(RequestEventScancodePush))
as RequestEventScancodePush;
                                    if (info != null)
                                    {
                                        responseContent =
actionBLL.HandleEventScancodePush(info);
                                    }
                                }
                                break;


                            case RequestEvent.scancode_waitmsg:
```

```
                {
                    //扫码推事件且弹出“消息接收中”
提示框的事件推送
                    RequestEventScancodeWaitmsg
info = XmlConvertor.XmlToObject(postStr,
typeof(RequestEventScancodeWaitmsg)) as
RequestEventScancodeWaitmsg;
                    if (info != null)
                    {
                        responseContent =
actionBLL.HandleEventScancodeWaitmsg(info);
                    }
                }
                break;

            case RequestEvent.pic_sysphoto:
                {
                    //弹出系统拍照发图的事件推送
                    RequestEventPicSysphoto info =
XmlConvertor.XmlToObject(postStr, typeof(RequestEventPicSysphoto))
as RequestEventPicSysphoto;
                    if (info != null)
```

```
                {
                    responseContent =
actionBLL.HandleEventPicSysphoto(info);
                }
            }
            break;
```

.....

处理扫描结果并返回的最终代码如下所示。

```
        /// <summary>
        /// 扫码推事件且弹出“消息接收中”提示框的事件推送的处理
        /// </summary>
        /// <param name="info">扫描信息</param>
        /// <returns></returns>
        public string
HandleEventScancodeWaitmsg(RequestEventScancodeWaitmsg info)
        {
            ResponseText response = new ResponseText(info);
            response.Content = string.Format("您的信息为：{0}，可以结
合后台进行数据查询。", info.ScanCodeInfo.ScanResult);
            return response.ToXml();
```

}

最后我们测试扫描一个条形码，可以看到返回的结果界面操作如下所示。

5、新菜单功能测试发现的问题

前面介绍了一些新菜单功能模块的集成，我个人对这种扫一扫菜单功能非常赞赏，这也是微信逐步整合更多硬件资源和接口处理的趋向，不过在集成使用的时候，发现公众号偶尔出现闪退的情况，还有就是这些新功能虽然后台能够实现数据的

处理和接收 ,但是有一些不能返回应答消息 ,很郁闷。也许随着版本研发的加 快，这些功能很快得到完善和解决。

另外微信开放平台也投入使用了，好些认证也是 300 元一年，不过暂时没有其应用的场景，我只是用到了它来获取微信账号的 unionid 的功能，其他功能慢慢了解吧。

还有就是微信的企业号也已经出来了，而且我也已经申请认证通过，它的开发用户的 API 也有不少，有空继续研究并整合到微信开发框架里面吧。

C#开发微信门户及应用(16)-微信企业号的配置和使用

在本系列随笔的前面，主要就是介绍微信公众号的门户应用开发，最近把整个微信框架进行了扩展补充，增加了最新的企业号的 API 封装和开发，后续主要介绍如何利用 C#进行微信企业号的开发工作，本篇作为微信企业号的开发的起步篇，介绍微信企业号的配置和使用。

1、微信企业号的注册和登陆

企业号是继公众号、订阅号的另外一种微信类型，它主要是面对企业的。企业号是微信为企业客户提供的移动应用入口。可以帮助企业建立员工、上下游供应 链与企业 IT 系统间的连接。利用 企业号 ，企业或第三方合作伙伴可以帮助企业

快速、低成本的实现高质量的移动轻应用，实现生产、管理、协作、运营的 移动化 。

个人觉得企业号最大的亮点是可以不限数量的消息发送，也就是可以在企业员工之间畅通交流。相对于公众号和订阅号，发送消息的谨慎程度，微信企业号可 谓给人眼前一亮的感觉。不过微信企业号是需要内部建立好通讯录，关注者需要匹配通讯录的微信号、邮箱、电话号码任一个通过才可以关注，也就是可以防止其他 外来人员的自由关注了，另外如果为了安全考虑，还可以设置二次验证，也就是一个审核过程。

企业号的认证和公众号一样，需要提供相关的企业资质文件，并且认证每年都要收取费用，否则可能有人员和功能的一些限制。觉得微信真是想着方法赚钱， 目前已有的收费模式有，订阅号、公众号、企业号、开放平台，好像都有认证收费的了，而且微信小店也还需要收 2 万的押金，一切都是钱呀。

好了，其他不多说，微信的注册地址是：https://qy.weixin.qq.com，一个邮箱不能同时注册微信公众号和微信企业号。

对于企业开通企业号并开始使用需要四步

1) 企业到微信官网（ http://qy.weixin.qq.com ）申请开通；

2) 开通后，企业在企业号管理后台导入成员，发布二维码；

3) 企业调用企业号 api 与企业自有系统对接开发；

4) 员工关注，收到微信信息，在微信中与企业交互

注册好企业号，就可以通过微信扫一扫，扫描企业二维码进行登录了，扫描的时候，需要微信进行确认，才可以继续输入密码进行登录，操作界面如下所示（左边是手机截图，右边是网页截图）。


●●●○○ 中国联通
9:02
100%
返回
应用登录
即将登录微信企业号，请确认是否本人操作
使用你的账号登录该应用
确认登录
取消

登录后我们就可以看到对应的电脑端的管理界面了。

2、设置开发回调模式

如果开发过微信公众号，那么我们就知道，如果需要在微信服务器和网站服务器之间建立连接关系，实现消息的转发和处理，那么就应该设置一个回调模式，需要配置好相关的参数。然后在自己 网站服务器里面建立一个处理微信服务器消息的入口。

进入配置后，我们需要修改相关的 URL、Token、EncodingAESKey 等参数，主要是 URL，这个就是和公众号的入口处理一样的，需要我们发布到网站服务器上的处理入口。

Token 和 AESKey 可以根据提示动态生成一个即可，AESKey 好像必须是 23 位的，所以这个一般是让它自己生成的，这个主要用来加密解密使用的。

URL、Token、EncodingAESKey 三个参数说明。

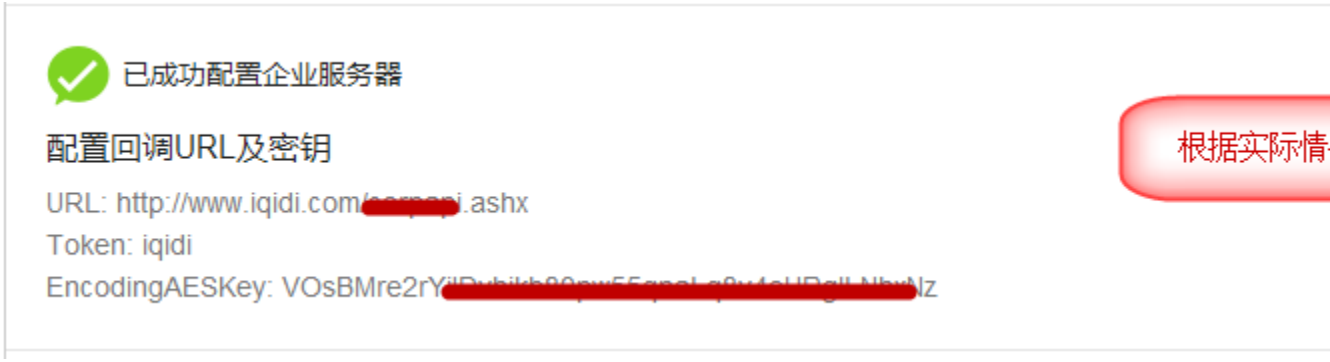
1）URL 是企业应用接收企业号推送请求的访问协议和地址，支持 http 或 https 协议。

2）Token 可由企业任意填写，用于生成签名。

3）EncodingAESKey 用于消息体的加密，是 AES 密钥的 Base64 编码。

验证 URL、Token 以及加密的详细处理请参考后续 “接收消息时的加解密处理”的部分。

我公司的企业号配置后的界面如下所示。



这个 URL 里面指向的页面功能，需要对数据进行解析并返回给微信服务器，因此我们需要在服务器上预先部署好这个处理功能入口。

除了上面的几个函数，还有一个 CorpID 的参数需要使用，我们可以在后台主界面-设置里面查看到。

然后我们为了方便网站后台使用，我们和公众号的配置一样，把它放到了 Web.Config 里面，如下所示。

```
<!--微信的Token-->
<add key="WeixinToken" value="iqidi"/>
<add key="AppId" value="wx3d81f
<add key="AppSecret" value="7f9256d
<!--企业号配置信息-->
<add key="CorpToken" value="iqidi"/>
<add key="CorpId" value="wx4ce70a
<add key="EncodingAESKey" value="VOsBMre2rYi
```

3、实现回调页面的功能开发

前面介绍了几个配置项，需要在回调页面里面使用的，本小节继续介绍如何实现企业号信息的回发，使之通过回调测试的操作。

由于回调测试的数据是通过 Get 方式发送的，因此我们的处理逻辑代码如下所示，和公众号的类似处理，只是实现部分不太一样而已。

```
/// <summary>
/// 企业号回调信息接口。统一接收并处理信息的入口。
/// </summary>
```

```
public class corpapi : IHttpHandler
{
    /// <summary>
    /// 处理企业号的信息
    /// </summary>
    /// <param name="context"></param>
    public void ProcessRequest(HttpContext context)
    {
        string postString = string.Empty;
        if (HttpContext.Current.Request.HttpMethod.ToUpper() ==
"POST")
        {
            using (Stream stream =
HttpContext.Current.Request.InputStream)
            {
                Byte[] postBytes = new Byte[stream.Length];
                stream.Read(postBytes, 0, (Int32)stream.Length);
                postString = Encoding.UTF8.GetString(postBytes);
            }

            if (!string.IsNullOrEmpty(postString))
            {
```

```
                    Execute(postString);
                }
            }
            else
            {
                Auth();
            }
        }

        /// <summary>
        /// 成为开发者的第一步，验证并相应服务器的数据
        /// </summary>
        private void Auth()
        {
            #region 获取关键参数
            string token =
ConfigurationManager.AppSettings["CorpToken"];//从配置文件获取
Token
            if (string.IsNullOrEmpty(token))
            {
                LogTextHelper.Error(string.Format("CorpToken 配置项
没有配置！"));
```

```
            }
            string encodingAESKey =
ConfigurationManager.AppSettings["EncodingAESKey"];//从配置文件获
取 EncodingAESKey
            if (string.IsNullOrEmpty(encodingAESKey))
            {
                LogTextHelper.Error(string.Format("EncodingAESKey 配
置项没有配置！"));
            }
            string corpId =
ConfigurationManager.AppSettings["CorpId"];//从配置文件获取 corpId
            if (string.IsNullOrEmpty(corpId))
            {
                LogTextHelper.Error(string.Format("CorpId 配置项没有
配置！"));
            }
            #endregion

            string echoString =
HttpContext.Current.Request.QueryString["echoStr"];
```

```
            string signature =
HttpContext.Current.Request.QueryString["msg_signature"];//企业号的
msg_signature
            string timestamp =
HttpContext.Current.Request.QueryString["timestamp"];
            string nonce =
HttpContext.Current.Request.QueryString["nonce"];

            string decryptEchoString = "";
            if (new CorpBasicApi().CheckSignature(token, signature,
timestamp, nonce, corpId, encodingAESKey, echoString, ref
decryptEchoString))
            {
                if (!string.IsNullOrEmpty(decryptEchoString))
                {

HttpContext.Current.Response.Write(decryptEchoString);
                    HttpContext.Current.Response.End();
                }
            }
        }
```


具体的处理代码如下所示，里面的一个加解密处理的类是微信企业号附录里面提供的，我使用了 C#版本的 SDK 而已。

```
/// <summary>
/// 企业号基础操作 API 实现
/// </summary>
public class CorpBasicApi : ICorpBasicApi
{
    /// <summary>
    /// 验证企业号签名
    /// </summary>
    /// <param name="token">企业号配置的 Token</param>
    /// <param name="signature">签名内容</param>
    /// <param name="timestamp">时间戳</param>
    /// <param name="nonce">nonce 参数</param>
    /// <param name="corpId">企业号 ID 标识</param>
    /// <param name="encodingAESKey">加密键</param>
    /// <param name="echostr">内容字符串</param>
    /// <param name="retEchostr">返回的字符串</param>
    /// <returns></returns>
```

```
        public bool CheckSignature(string token, string signature, string timestamp, string nonce, string corpId, string encodingAESKey, string echostr, ref string retEchostr)
        {
            WXBizMsgCrypt wxcpt = new WXBizMsgCrypt(token, encodingAESKey, corpId);
            int result = wxcpt.VerifyURL(signature, timestamp, nonce, echostr, ref retEchostr);
            if (result != 0)
            {
                LogTextHelper.Error("ERR: VerifyURL fail, ret: " + result);
                return false;
            }

            return true;

            //ret==0 表示验证成功，retEchostr 参数表示明文，用户需要将 retEchostr 作为 get 请求的返回参数，返回给企业号。
            // HttpUtils.SetResponse(retEchostr);
        }
```

C#开发微信门户及应用(17)-微信企业号的通讯录管理开发之部门管理

前面一篇随笔企业号的一些基础信息,以及介绍如何配置企业号的回调方式实现和企业号服务器进行沟通的桥梁。本篇主要还是继续介绍企业号的开发工作的开展，介绍微信企业号通讯录管理开发功能，介绍其中组织机构里面如何获取和管理部门的信息等内容。

1、企业组织的创建和配置

首先我们可以在企业号的管理后台里面创建一个组织机构,里面创建一些部门和人员列表，方便我们开发和使用。

例如创建一个广州爱奇迪的根结构，然后在其中在创建一些组织机构，如下图所示。

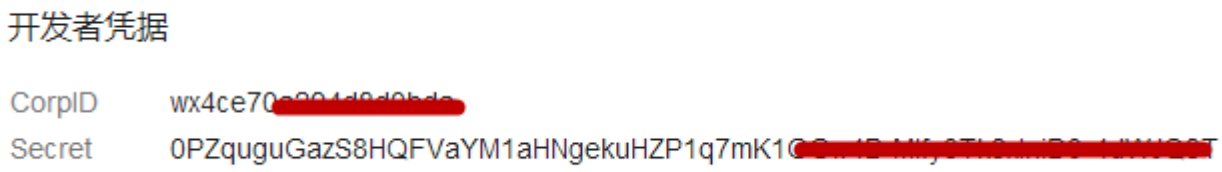
然后给组织结构根节点“广州爱奇迪”增加一个管理员权限 ,以后再开发接口里面也就可以使用这个管理员所属的权限 Secret 值进行调用了。

CorpID 是企业号的标识，每个企业号拥有一个唯一的 CorpID；Secret 是管理组凭证密钥。

系统管理员可通过管理端的权限管理功能创建管理组，分配管理组对应用、通讯录、接口的访问权限。完成后，管理组即可获得唯一的 secret。系统管理员可通过权限管理查看所有管理组的 secret ,其他管理员可通过设置中的开发者凭据查看。

我的企业号的创建者和“广州爱奇迪”组织结构的管理员是不同的 ,由于 Secret 是管理组凭证密钥，因此管理者负责不同的组织机构管理的话，自己的管理

Secret 值可能就不同了。如果我们需要调用接口，就需要用到这个属于自己权限级别的 Secret 值，如下图所示。



如果不是企业号的创建者，那么可能不能修改里面的一些权限分配，只能查看。

2、API 访问的全局唯一票据 AccessToken 的获取

和公众号一样，我们调用企业号 API 的第一步也是需要先获取访问的票据 AccessToken。这个票据是全局性的，有一定的时效和频率控制，因此需要适当的进行缓存，不能每次调用都去刷新获取。

企业号获取访问票据的主要的逻辑代码如下所示 ,其主要的就是需要使用管理者的 Secret 值去获取对应的口令 ,这样它就能够知道管理的是那个组织结构的了。

```
        /// <summary>
        /// 获取每次操作微信 API 的 Token 访问令牌
        /// </summary>
        /// <param name="corpid">企业 Id</param>
        /// <param name="corpsecret">管理组的凭证密钥</param>
        /// <returns></returns>
        public string GetAccessTokenNoCache(string corpid, string
corpsecret)
        {
            var url =
string.Format("https://qyapi.weixin.qq.com/cgi-bin/gettoken?corpid={0}
&corpsecret={1}",
                                            corpid, corpsecret);

            HttpHelper helper = new HttpHelper();
            string result = helper.GetHtml(url);
            string regex = "\"access_token\":\"(?<token>.*?)\"";

            string token = CRegex.GetText(result, regex, "token");
```

```
            return token;
        }
```

微信企业号的说明如下所示：

当企业应用调用企业号接口时，企业号后台为根据此次访问的 AccessToken,校验访问的合法性以及所对应的管理组的管理权限以返回相应的结果。

注：*你应该审慎配置管理组的权限，够用即好，权限过大会增加误操作可能性及信息安全隐患。*

AccessToken 是企业号的全局唯一票据，调用接口时需携带 AccessToken。AccessToken 需要用 CorpID 和 Secret 来换取，不同的 Secret 会返回不同的 AccessToken。**正 常情况下 AccessToken 有效期为 7200 秒，有效期内重复获取返回相同结果，并自动续期。由于获取 access_token 的 api 调用次数非常有限，建议企业全局存储与更新 access_token，频繁刷新 access_token 会导致 api 调用受限，影响自身业务。**

2、通讯录管理之部门信息的维护

有了第一节里面的访问票据，我们就可以利用 API 来做很多事情了，包括组织结构的获取、创建、删除等等功能。

创建部门的官方接口定义如下所示。

- 请求说明

Https 请求方式: POST

https://qyapi.weixin.qq.com/cgi-bin/department/create?access_token=ACCESS_TOKEN

请求包结构体为:

```
{
    "name": "邮箱产品组",
    "parentid": "1"
}
```

- 参数说明

参数	必须	说明
s_token	是	调用接口凭证
	是	部门名称。长度限制为 1~64 个字符
tid	是	父亲部门 id。根部门 id 为 1

- 返回结果

```
{
```

```
    "errcode": 0,
    "errmsg": "created",
    "id": 2
}
```

根据上面的一些类似的接口定义说明，我们先来定义下组织机构部门数据的维护接口，然后在逐步实现和调用。

```
        #region 部门管理
        /// <summary>
        /// 创建部门。
        /// 管理员须拥有“操作通讯录”的接口权限，以及父部门的管理权限。
        /// </summary>
        CorpDeptCreateJson CreateDept(string accessToken, string name, string parentId);

        /// <summary>
        /// 更新部门。
        /// 管理员须拥有“操作通讯录”的接口权限，以及该部门的管理权限。
        /// </summary>
        CommonResult DeleteDept(string accessToken, int id);

        /// <summary>
```

```
        /// 删除部门.
        /// 管理员须拥有“操作通讯录”的接口权限，以及该部门的管理权限。
        /// </summary>
        CorpDeptListJson ListDept(string accessToken);


        /// <summary>
        /// 获取部门列表.
        /// 管理员须拥有’获取部门列表’的接口权限，以及对部门的查看权限。
        /// </summary>
        CommonResult UpdateDept(string accessToken, int id, string name);
        #endregion
```

如创建部门的接口实现如下所示，主要就是构建 URL 和 POST 的数据包，然后统一调用并获取返回数据，转换为具体的 Json 对象实体即可。其他接口的实现方式类似，不在赘述。

```
        /// <summary>
        /// 创建部门。
        /// 管理员须拥有“操作通讯录”的接口权限，以及父部门的管理权限。
        /// </summary>
```

```
        public CorpDeptCreateJson CreateDept(string accessToken,
string name, string parentId)
        {
            string urlFormat =
"https://qyapi.weixin.qq.com/cgi-bin/department/create?access_token=
{0}";
            var data = new
            {
                name = name,
                parentId = parentId
            };
            var url = string.Format(urlFormat, accessToken);
            var postData = data.ToJson();

            CorpDeptCreateJson result =
CorpJsonHelper<CorpDeptCreateJson>.ConvertJson(url, postData);
            return result;
        }
```

CorpDeptCreateJson 对象实体类的定义如下所示，我们主要是根据返回结果进行定义的。

```
/// <summary>
/// 创建部门的返回结果
/// </summary>
public class CorpDeptCreateJson : BaseJsonResult
{
    /// <summary>
    /// 返回的错误消息
    /// </summary>
    public CorpReturnCode errcode { get; set; }

    /// <summary>
    /// 对返回码的文本描述内容
    /// </summary>
    public string errmsg { get; set; }

    /// <summary>
    /// 创建的部门 id。
    /// </summary>
    public int id { get; set; }
}
```

3、部门管理的 API 调用

上面小节介绍了如何封装部门管理的 API，那么我们封装好了对应的接口和接口实现，怎么样在实际环境里面进行调用处理的呢，为了方便我创建一个小的 Winform 程序来测试对应 API 的功能，如下所示。

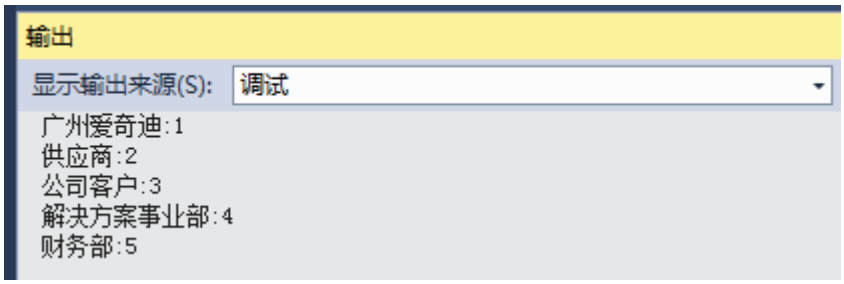
下面我们来介绍一下调用的代码和效果展示。

```
        private void btnCreateDeleteDept_Click(object sender,
EventArgs e)
        {
            ICorpAddressBookApi bll = new CorpAddressBookApi();
            string name = "测试部门";
```

```
                CorpDeptCreateJson json = bll.CreateDept(token, name,
"2");
                if (json != null)
                {
                    Console.WriteLine("创建了部门：{0}， ID:{1}", name,
json.id);

                    //更新部门信息
                    name = "测试部门修改名称";
                    CommonResult result = bll.UpdateDept(token, json.id,
name);
                    if(result != null)
                    {
                        Console.WriteLine("修改部门名称:{0} {1}",
(result.Success ? "成功" : "失败"), result.ErrorMessage);
                    }

                    //删除部门
                    result = bll.DeleteDept(token, json.id);
                    if (result != null)
                    {
```

```
                    Console.WriteLine("删除部门名称:{0} {1}",
(result.Success ? "成功" : "失败"), result.ErrorMessage);
                }
            }

        }
```


```
        /// <summary>
        /// 获取部门列表
        /// </summary>
        private void btnListDept_Click(object sender, EventArgs e)
        {
            ICorpAddressBookApi bll = new CorpAddressBookApi();
            CorpDeptListJson list = bll.ListDept(token);
            foreach (CorpDeptJson info in list.department)
            {
                string tips = string.Format("{0}:{1}", info.name, info.id);
                Console.WriteLine(tips);
            }
        }
```



C#开发微信门户及应用(18)-微信企业号的通讯录管理开发之成员管理

在上篇随笔《C#开发微信门户及应用(17)-微信企业号的通讯录管理开发之部门管理》介绍了通讯录的部门的相关操作管理，通讯录管理包括部门管理、成员管理、标签管理三个部分，本篇主要介绍成员的管理操作，包括创建、删除、更新、获取、获取部门成员几个操作要点。

1、成员的创建操作

为了方便，我们可以创建一个部门组织结构，这是开发的前提，因为我们通讯录管理，也是基于一个组织机构下的，如上篇介绍的组织结构层次一样。我这里创建一个广州爱奇迪的根结构，然后在其中在创建一些组织机构，如下图所示。

在后台可以通过功能操作添加人员，本篇主要介绍如何调用微信企业号 API 进行人员管理的操作。

创建人员的 API 定义如下所示。

- 请求说明

Https 请求方式: POST

https://qyapi.weixin.qq.com/cgi-bin/user/create?access_token=ACCESS_TOKEN

请求包结构体为:

{

```
    "userid": "zhangsan",
    "name": "张三",
    "department": [1, 2],
    "position": "产品经理",
    "mobile": "15913215421",
    "gender": 1,
    "tel": "62394",
    "email": "zhangsan@gzdev.com",
    "weixinid": "zhangsan4dev"
}
```


- 参数说明

参数	必须	说明
access_token	是	调用接口凭证
userid	是	员工 UserID。对应管理端的帐号，企业内必须唯一。长度为 1~64 个字符
name	是	成员名称。长度为 1~64 个字符
department	否	成员所属部门 id 列表。注意，每个部门的直属员工上限为 1000 个
position	否	职位信息。长度为 0~64 个字符
mobile	否	手机号码。企业内必须唯一，mobile/weixinid/email 三者不能同时为空
gender	否	性别。gender=0 表示男，=1 表示女。默认 gender=0

tel	否	办公电话。长度为 0~64 个字符
email	否	邮箱。长度为 0~64 个字符。企业内必须唯一
weixinid	否	微信号。企业内必须唯一

- 权限说明

管理员须拥有“操作通讯录”的接口权限，以及指定部门的管理权限。

- 返回结果

```
{
    "errcode": 0,
    "errmsg": "created"
}
```

我们在 C#里面，需要定义对应给的接口，然后根据需要构造对应的传递实体信息。

这里我把人员管理的接口全部定义好，接口定义如下所示。

```
#region 部门成员管理
/// <summary>
/// 创建成员
/// </summary>
```

```
        CommonResult CreateUser(string accessToken, CorpUserJson
user);

        /// <summary>
        /// 更新成员
        /// </summary>
        CommonResult UpdateUser(string accessToken,
CorpUserUpdateJson user);

        /// <summary>
        /// 删除成员
        /// </summary>
        CommonResult DeleteUser(string accessToken, string userid);

        /// <summary>
        /// 根据成员 id 获取成员信息
        /// </summary>
        CorpUserGetJson GetUser(string accessToken, string userid);

        /// <summary>
        /// 获取部门成员
        /// </summary>
```

```
        CorpUserListJson GetDeptUser(string accessToken, int
department_id, int fetch_child = 0, int status = 0);
        #endregion
```

然后根据信息定义，创建一个承载人员信息的 CorpUserJson 实体对象，创建人员的实现操作代码如下所示。

```
        /// <summary>
        /// 创建成员
        /// </summary>
        public CommonResult CreateUser(string accessToken,
CorpUserJson user)
        {
            string urlFormat =
"https://qyapi.weixin.qq.com/cgi-bin/user/create?access_token={0}";
            var data = new
            {
                userid = user.userid,
                name = user.name,
                department = user.department,
                position = user.position,
                mobile = user.mobile,
```

```
                gender = user.gender,
                tel = user.tel,
                email = user.email,
                weixinid = user.weixinid
            };
            var url = string.Format(urlFormat, accessToken);
            var postData = data.ToJson();

            return Helper.GetCorpExecuteResult(url, postData);
        }
```

2、成员的更新操作

成员的数据更新和创建操作类似，它的企业号定义如下所示。

- 请求说明

Https 请求方式: POST

https://qyapi.weixin.qq.com/cgi-bin/user/update?access_token=ACCESS_TOKEN

请求包示例如下（如果非必须的字段未指定，则不更新该字段之前的设置值）：

```
{
    "userid": "zhangsan",
    "name": "李四",
    "department": [1],
    "position": "后台工程师",
    "mobile": "15913215421",
    "gender": 1,
    "tel": "62394",
    "email": "zhangsan@gzdev.com",
    "weixinid": "lisifordev",
    "enable": 1
}
```

由于它的操作数据类似，因此它的实现代码也差不多，如下所示就是。

```
        /// <summary>
        /// 更新成员
        /// </summary>
        public CommonResult UpdateUser(string accessToken,
CorpUserUpdateJson user)
        {
            string urlFormat =
"https://qyapi.weixin.qq.com/cgi-bin/user/update?access_token={0}";
```

```
            //string postData = user.ToJson();
            var data = new
            {
                userid = user.userid,
                name = user.name,
                department = user.department,
                position = user.position,
                mobile = user.mobile,
                gender = user.gender,
                tel = user.tel,
                email = user.email,
                weixinid = user.weixinid,
                enable = user.enable
            };
            var url = string.Format(urlFormat, accessToken);
            var postData = data.ToJson();

            return Helper.GetCorpExecuteResult(url, postData);
        }
```

3、成员的删除、成员的获取、部门成员的获取操作

这些操作和上面的类似，不在赘述，主要就是根据需要定义他们对应的返回数据信息，然后解析 Json 数据即可转换为对应的实体。

1）删除人员的定义如下：

- 请求说明

Https 请求方式: GET

https://qyapi.weixin.qq.com/cgi-bin/user/delete?access_token=ACCESS_TOKEN&userid=lisi

- 参数说明

参数	必须	说明
access_token	是	调用接口凭证
userid	是	员工 UserID。对应管理端的帐号

- 返回结果

```
{
    "errcode": 0,
    "errmsg": "deleted"
}
```

2）成员的获取定义如下：

- 请求说明

Https 请求方式: GET

https://qyapi.weixin.qq.com/cgi-bin/user/get?access_token=ACCESS_TOKEN&userid=lisi

- 参数说明

参数	必须	说明
access_token	是	调用接口凭证
userid	是	员工 UserID

- 返回结果

{

"errcode": 0,

"errmsg": "ok",

"userid": "zhangsan",

"name": "李四",

"department": [1, 2],

"position": "后台工程师",

"mobile": "15913215421",

"gender": 1,

"tel": "62394",

"email": "zhangsan@gzdev.com",

"weixinid": "lisifordev",

"avatar":

"http://wx.qlogo.cn/mmopen/ajNVdqHZLLA3WJ6DSZUfiakYe37PKnQhBIeOQBO4czqrnZDS79FH5Wm5m4X69TBicnHFlhiafvDwklOpZeXYQQ2icg/0",

"status": 1

}

3）部门成员的获取定义如下：

- 请求说明

Https 请求方式: GET

https://qyapi.weixin.qq.com/cgi-bin/user/simplelist?access_token=ACCESS_TOKEN&department_id=1&fetch_child=0&status=0

- 参数说明

参数	必须	说明
access_token	是	调用接口凭证

department_id	是	获取的部门 id
fetch_child	否	1/0：是否递归获取子部门下面的成员
status	否	0 获取全部员工，1 获取已关注成员列表，2 获取禁用成员列表，4 获取未关注成员列表。status 可叠加

- 权限说明

管理员须拥有’获取部门成员’的接口权限，以及指定部门的查看权限。

- 返回结果

```
{
    "errcode": 0,
    "errmsg": "ok",
    "userlist": [
            {
                "userid": "zhangsan",
                "name": "李四"
            }
      ]
}
```

这个返回值我们定义一个实体对象用来存储数据即可。

```
/// <summary>
/// 获取部门成员返回的数据
/// </summary>
public class CorpUserListJson : BaseJsonResult
{
    public CorpUserListJson()
    {
        this.userlist = new List<CorpUserSimpleJson>();
    }

    /// <summary>
    /// 返回的错误消息
    /// </summary>
    public CorpReturnCode errcode { get; set; }

    /// <summary>
    /// 对返回码的文本描述内容
    /// </summary>
    public string errmsg { get; set; }

    /// <summary>
    /// 成员列表
```

```
        /// </summary>
        public List<CorpUserSimpleJson> userlist { get; set; }
    }
```

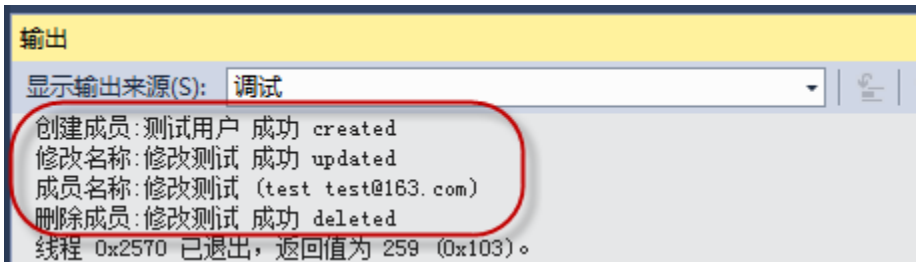
7、综合例子调用代码

上面介绍了一些企业号的接口定义和我对 API 的 C#封装接口和部分实现代码，实现了功能后，我们就可以在代码中对它进行测试，确信是否正常使用。

```
        /// <summary>
        /// 人员管理综合性操作（创建、修改、获取信息、删除）
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnCorpUser_Click(object sender, EventArgs e)
        {
            CorpUserJson user = new CorpUserJson();
            user.userid = "test";
            user.name ="测试用户";
            user.department = new List<int>(){2};
            user.email = "test@163.com";
```

```
                ICorpAddressBookApi bll = new CorpAddressBookApi();
                CommonResult result = bll.CreateUser(token, user);
                if (result != null)
                {
                    Console.WriteLine("创建成员:{0} {1} {2}", user.name,
(result.Success ? "成功" : "失败"), result.ErrorMessage);

                    string name = "修改测试";
                    user.name = name;
                    CorpUserUpdateJson userUpdate = new
CorpUserUpdateJson(user);
                    result = bll.UpdateUser(token, userUpdate);
                    if (result != null)
                    {
                        Console.WriteLine("修改名称:{0} {1} {2}", name,
(result.Success ? "成功" : "失败"), result.ErrorMessage);
                    }

                    CorpUserGetJson userGet = bll.GetUser(token,
user.userid);
                    if (userGet != null)
```

```
                {
                    Console.WriteLine("成员名称:{0} ({1} {2})",
userGet.name, user.userid, user.email);
                }

                result = bll.DeleteUser(token, user.userid);
                if (result != null)
                {
                    Console.WriteLine("删除成员:{0} {1} {2}", name,
(result.Success ? "成功" : "失败"), result.ErrorMessage);
                }
            }
        }
```



获取部门人员的操作代码如下所示。

```
        /// <summary>
        /// 获取部门人员
```

```
        /// </summary>
        private void btnCorpUserList_Click(object sender, EventArgs e)
        {
            int deptId = 1;
            ICorpAddressBookApi bll = new CorpAddressBookApi();
            CorpUserListJson result = bll.GetDeptUser(token, deptId);
            if (result != null)
            {
                foreach(CorpUserSimpleJson item in result.userlist)
                {
                    Console.WriteLine("成员名称:{0} {1}", item.name,
item.userid);
                }
            }
        }
```

人员的管理，相对来说比较简单，主要是在一定的部门下创建人员，然后也可以给标签增加相应的人员，基本上就是这些了，不过一定需要确保有相应的权限进行操作。

C#开发微信门户及应用(19)-微信企业号的消息发送（文本、图片、文件、语音、视频、图文消息等）

我们知道，企业号主要是面向企业需求而生的，因此内部消息的交流显得非常重要，而且发送、回复消息数量应该很可观，对于大企业尤其如此，因此可以 结合企业号实现内部消息的交流。企业号具有关注安全、消息无限制等特点，很适合企业内部的环境。本文主要介绍如何利用企业号实现文本、图片、文件、语音、视频、图文消息等消息的发送操作。

1、企业号特点

对于企业号，有以下一些特点：

1）关注更安全

–只有企业通讯录的成员才能关注企业号，分级管理员、保密消息等各种特性确保企业内部信息的安全。

企业可以设置自行验证关注者身份，进行二次安全验证，保证企业信息使用和传递安全。

若员工离职，企业管理员可在通讯录中删除该成员，该成员即自动取消关注企业号，同时微信中的企业号历史记录也会被清除。

2）应用可配置

–企业可自行在企业号中可配置多个服务号，可以连接不同的企业应用系统，只有授权的企业成员才能使用相应的服务号。

3）消息无限制

–发送消息无限制 ,并提供完善的的管理接口及微信原生能力 ,以适应企业复杂、个性化的应用场景。

企业可以主动发消息给员工，**消息量不受限制**。

4）使用更便捷

–企业号在微信中有统一的消息入口，用户可以更方便地管理企业号消息。微信通讯录也可以直接访问企业号中的应用。

2、企业号的管理接口内容

目前企业号的内容可以用下面的分层图来展示，分别包含素材管理、被动响应消息、通讯录管理、自定义菜单等内容，详细可以看下面图示。


用于上传图片、语音、视频等媒体资源文件以及普通文件
素材管理
下载多媒体文件
上传多媒体文件
发送被动响应消息
回复文本消息
回复图片消息
回复语音消息
回复视频消息
回复图文消息
企业号管理接口
通讯录管理
部门管理
创建部门
修改部门
删除部门
获取部门列表
成员管理
创建成员
修改成员
删除成员
获取部门成员
授权获取用户基本信息
伍华聪的博客 http://wuhuacong.cnblogs.com
标签管理
创建标签
修改标签名称
删除标签
获取标签成员
增加标签成员
删除标签成员
自定义菜单
菜单创建
菜单列表获取
菜单删除
OAuth2验证
通过OAuth2.0来获取员工的身份信息
获取access token

3、企业号消息和事件的处理

企业号和公众号一样，可以分为消息处理和事件处理，下面是他们两种类型的处理操作，也就发送的消息有文本消息、图片消息、文件消息、视频消息、语音消息、地理文字消息、图文和多媒体消息等。

事件处理主要就是关注、取消关注事件，以及菜单 click 类型和 view 类型两种操作，还有就是地理位置上报事件等。

两种类型的处理图如下所示。

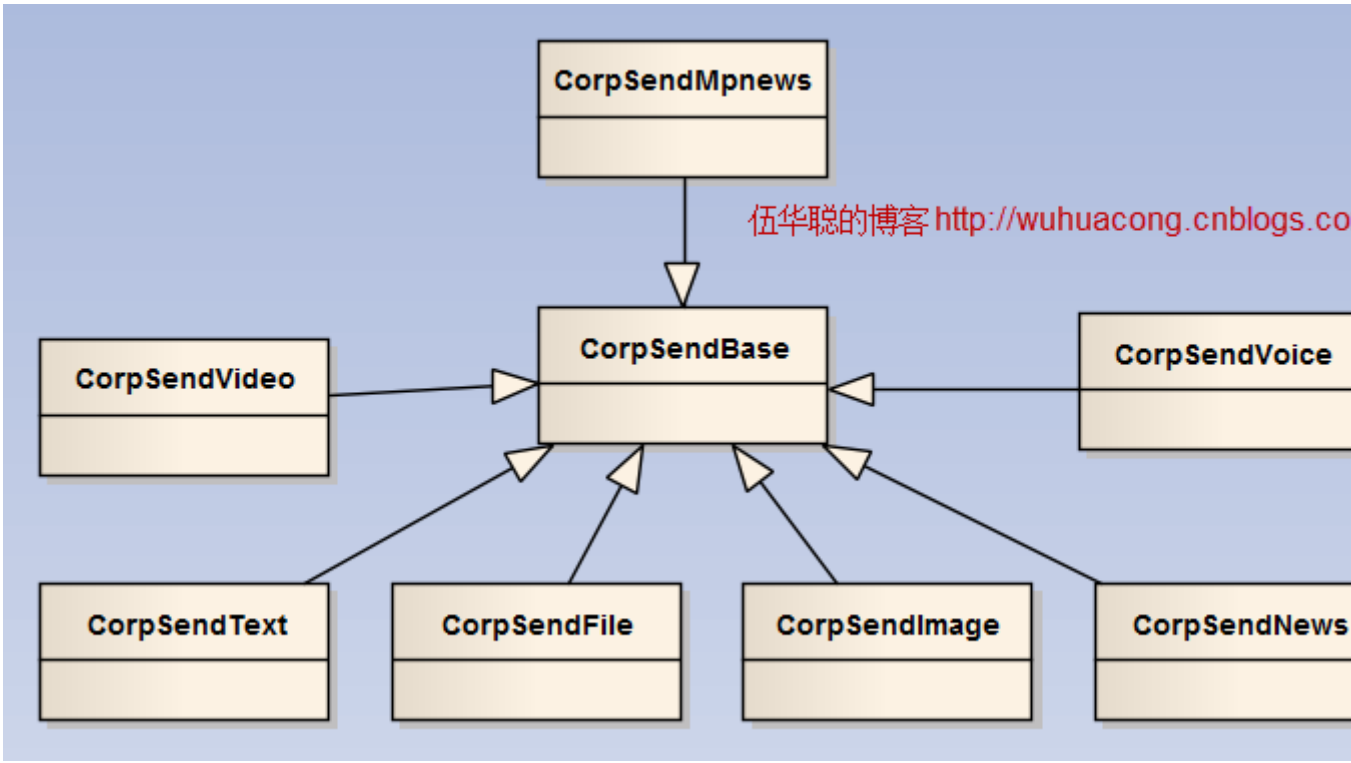
伍华聪的博客 http://wuhuacong.cnblogs.com

4、企业号消息管理

在企业的管理后台，和公众号一样，可以看到对应信息交流记录，包括文字、图片、地理位置等等，如下所示。

由于消息分为几种类型 ,包括文本(Text)、图片(Image)、文件(File)、语音(Voice)、视频(Video)、图文消息等(News)、MpNews 等。

因此我们需要分别对它们进行一定的定义和封装处理 ,如下是它们的信息对象设计图。



企业号发送消息的官方定义如下:

企业可以主动发消息给员工，**消息量不受限制。**

调用接口时，使用 Https 协议、JSON 数据包格式，数据包不需做加密处理。

目前支持文本、图片、语音、视频、文件、图文等消息类型。除了 news 类型，其它类型的消息可在发送时加上保密选项，保密消息会被打上水印，并且只有接收者才能阅读。

我们以发送的文本消息为例进行说明，它的定义如下所示。

- text 消息

```
{
    "touser": "UserID1|UserID2|UserID3",
    "toparty": " PartyID1 | PartyID2 ",
    "totag": " TagID1 | TagID2 ",
    "msgtype": "text",
    "agentid": "1",
    "text": {
        "content": "Holiday Request For Pony(http://xxxxx)"
    },
    "safe":"0"
}
```

参数	必须	说明
touser	否	UserID 列表（消息接收者，多个接收者用‘|’分隔）。特殊情况：指定为@all，则向关注该企业应用的全部成员发送
toparty	否	PartyID 列表，多个接受者用‘|’分隔。当 touser 为@all 时忽略本参数

totag	否	TagID 列表，多个接受者用‘|’分隔。当 touser 为@all 时忽略本参数
msgtype	是	消息类型，此时固定为：text
agentid	是	企业应用的 id，整型。可在应用的设置页面查看
content	是	消息内容
safe	否	表示是否是保密消息，0 表示否，1 表示是，默认 0

其中每种消息都会包含以下消息所示，也就是它们共同的属性：

```
 touser": "UserID1|UserID2|UserID3",
"toparty": " PartyID1 | PartyID2 ",
"totag": " TagID1 | TagID2 ",
"msgtype": "text",
"agentid": "1",
```

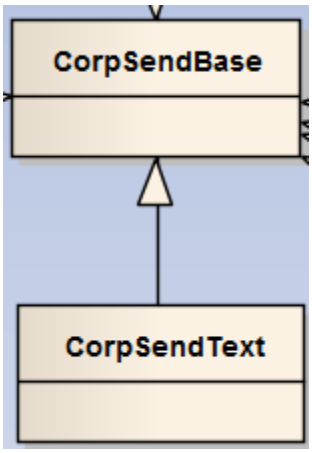
因此我们可以定义一个基类用来方便承载这些共同的信息。

```
/// <summary>
/// 企业号发送消息的基础消息内容
/// </summary>
```

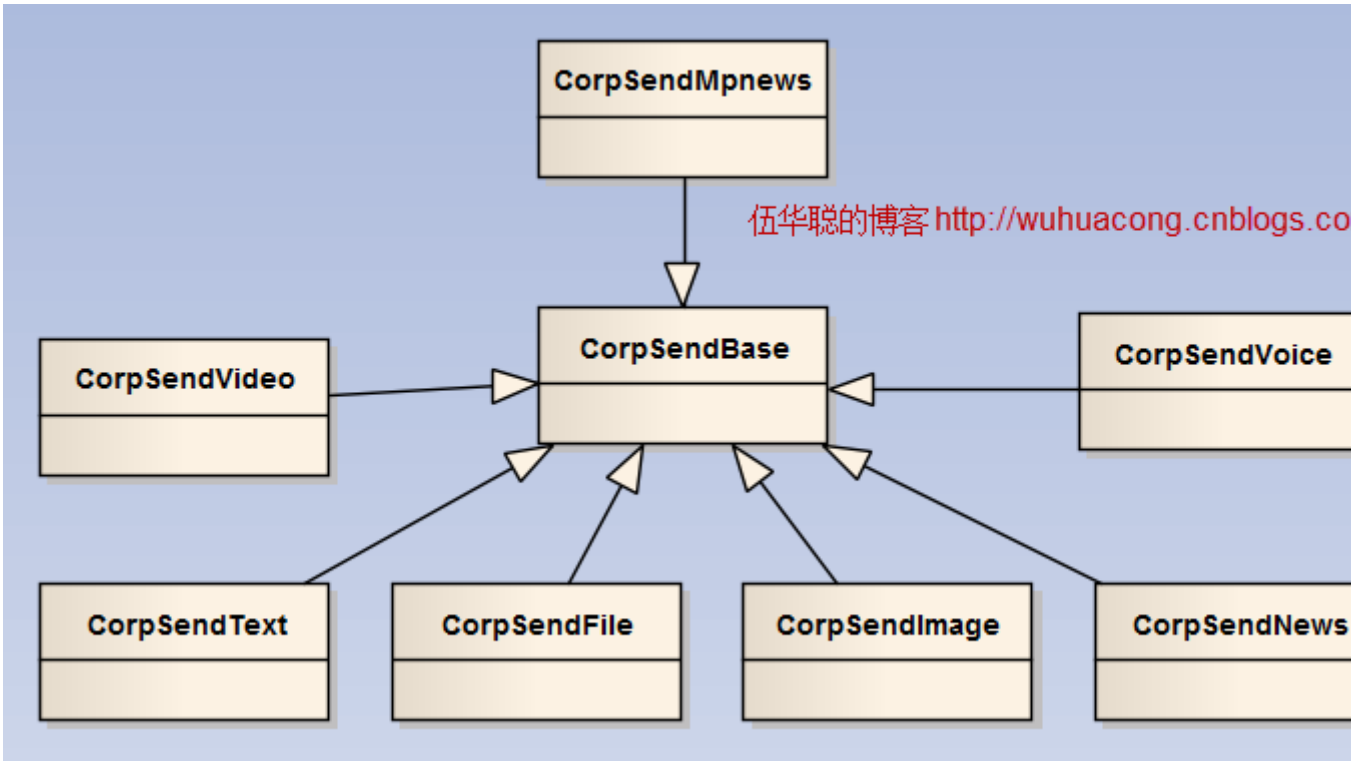
```
public class CorpSendBase
{
    /// <summary>
    /// UserID 列表(消息接收者，多个接收者用‘|’分隔)。特殊情况：指定为@all，则向关注该企业应用的全部成员发送
    /// </summary>
    public string touser { get; set; }

    /// <summary>
    /// PartyID 列表，多个接受者用‘|’分隔。当 touser 为@all 时忽略本参数
    /// </summary>
    public string toparty { get; set; }

    /// <summary>
    /// TagID 列表，多个接受者用‘|’分隔。当 touser 为@all 时忽略本参数
    /// </summary>
    public string totag { get; set; }

    /// <summary>
    /// 消息类型
```

```
        /// </summary>
        public string msgtype { get; set; }

        /// <summary>
        /// 企业应用的 id，整型。可在应用的设置页面查看
        /// </summary>
        public string agentid { get; set; }

        /// <summary>
        /// 表示是否是保密消息，0 表示否，1 表示是，默认 0
        /// </summary>
        [JsonProperty(NullValueHandling = NullValueHandling.Ignore)]
        public string safe { get; set; }

    }
```

然后其他消息逐一继承这个基类即可，如下所示。



最终会构成下面这个继承关系图。



5、消息接口的定义和实现

定义好相关的发送对象后，我们就可以定义它的统一发送接口了，如下所示。

```
    /// <summary>
    /// 企业号消息管理接口定义
    /// </summary>
    public interface ICorpMessageApi
    {
        /// <summary>
        /// 发送消息。
        /// 需要管理员对应用有使用权限，对收件人 touser、toparty、totag 有查看权限，否则本次调用失败。
        /// </summary>
        /// <param name="accessToken"></param>
        /// <returns></returns>
        CommonResult SendMessage(string accessToken, CorpSendBase data);
    }
```

最终，文本等类型的消息会根据接口定义进行实现，实现代码如下所示。注意，发送过程**不需要**调用加密类进行加密。

```
    /// <summary>
    /// 企业号消息管理实现类
    /// </summary>
```

```
    public class CorpMessageApi : ICorpMessageApi
    {
        /// <summary>
        /// 发送消息。
        /// 需要管理员对应用有使用权限，对收件人 touser、toparty、totag 有查看权限，否则本次调用失败。
        /// </summary>
        /// <param name="accessToken"></param>
        /// <returns></returns>
        public CommonResult SendMessage(string accessToken, CorpSendBase data)
        {
            CommonResult result = new CommonResult();

            string urlFormat = "https://qyapi.weixin.qq.com/cgi-bin/message/send?access_token={0}";
            var url = string.Format(urlFormat, accessToken);
            var postData = data.ToJson();

            //数据不用加密发送
            CorpSendResult sendResult = CorpJsonHelper<CorpSendResult>.ConvertJson(url, postData);
```

```
                if (sendResult != null)
                {
                    result.Success = (sendResult.errcode ==
CorpReturnCode.请求成功);
                    result.ErrorMessage =
string.Format("invaliduser:{0},invalidparty:{1},invalidtag:{2}",
                        sendResult.invaliduser, sendResult.invalidparty,
sendResult.invalidtag);
                }

                return result;
            }
        }
```

6、消息的发送操作和实际效果

定义好相应的发送对象后，我们就可以进行统一的消息发送操作，包括文本、图片、文件、语音等等类型的消息，注意有些消息是需要上传到服务器上，然后在根据 mediaId 进行发送出去的。

发送文本和图片的操作代码如下所示。

```
        private void btnSendText_Click(object sender, EventArgs e)
```

```
        {
            //发送文本内容
            ICorpMessageApi bll = new CorpMessageApi();

            CorpSendText text = new CorpSendText("API 中文测试
(http://www.iqidi.com)");
            text.touser = "wuhuacong";
            text.toparty = "4";//部门 ID
            text.totag = "0";

            text.safe = "0";
            text.agentid = "0";

            CommonResult result = bll.SendMessage(token, text);
            if (result != null)
            {
                Console.WriteLine("发送消息:{0} {1} {2}",
text.text.content, (result.Success ? "成功" : "失败"), result.ErrorMessage);
            }
        }
        private void btnSendImage_Click(object sender, EventArgs e)
        {
```

```
                btnUpload_Click(sender, e);


                if (!string.IsNullOrEmpty(image_mediaId))
                {
                    //发送图片内容
                    ICorpMessageApi bll = new CorpMessageApi();


                    CorpSendImage image = new
CorpSendImage(image_mediaId);
                    CommonResult result = bll.SendMessage(token,
image);
                    if (result != null)
                    {
                        Console.WriteLine("发送图片消息:{0} {1} {2}",
image_mediaId, (result.Success ? "成功" : "失败"), result.ErrorMessage);
                    }
                }
            }
```

最后在微信企业号上截图效果如下所示 ,包括了文本测试、文件测试、图文测试、语音测试均正常。


●●●○○ 中国联通
21:51
17%
iqidi
广州爱奇迪-企业号
http://www.iqidi.com)
10:02
营业执照.jpg
44k
10:12
广州爱奇迪 客户关系管理系统
9月26日
客户关系管理系统
个人待办
公司信息

C#开发微信门户及应用(20)-微信企业号的菜单管理

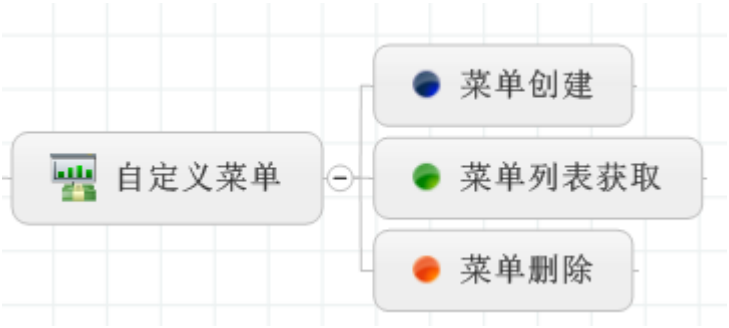
前面几篇陆续介绍了很多微信企业号的相关操作，企业号和公众号一样都可以自定义菜单，因此他们也可以通过 API 进行菜单的创建、获取列表、删除的操作，因此本篇继续探讨这个主体，介绍企业号的菜单管理操作。

菜单在很多情况下，能够给我们提供一个快速入口，也可以用来获取用户信息的主要入口，通过 OAuth2 验证接口，以及自定义的重定向菜单，我们就可以获取对应的用户 ID，然后进一步获取到用户的相关数据，可以显示给客户。

1、菜单的总体介绍

菜单的事件处理如下所示，包括了单击和跳转两个操作，未来企业号可能会增加一些和公众号一样的扫码操作，拍照操作等功能的，目前只有两个。

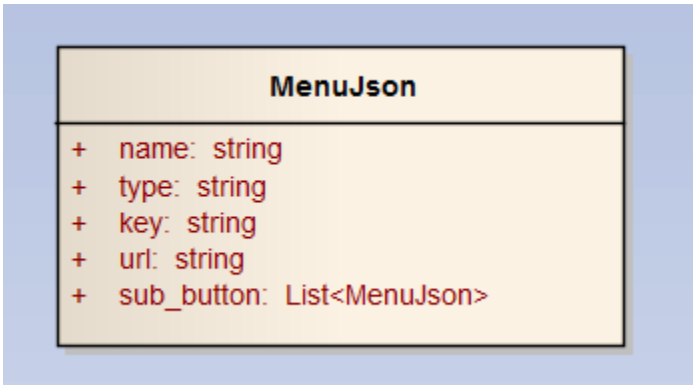
官方的菜单定义接口包含了下面三种操作，菜单创建、列表获取和菜单删除，这点和公众号操作几乎一样了。



2、菜单的实体类定义和接口定义处理

我们定义菜单，包括定义它的一些属性，包含有 name, type, key,url,以及一个指向自身引用的子菜单引用，因此菜单就可以循环构造多个层次，虽然严格意义上来讲，企业号的菜单和公众号菜单一样，一级三个，二级最多五个，而且没有三级菜单了。

实体类的 UML 图示如下所示。



菜单管理的创建操作，官方定义如下所示。

- 请求说明

Https 请求方式: POST

https://qyapi.weixin.qq.com/cgi-bin/menu/create?access_token=ACCESS_TOKEN&agentid=1

请求包如下：

{

```
"button":[
    {
        "type":"click",
        "name":"今日歌曲",
        "key":"V1001_TODAY_MUSIC"
    },
    {
        "name":"菜单",
        "sub_button":[
            {
                "type":"view",
                "name":"搜索",
                "url":"http://www.soso.com/"
            },
            {
                "type":"click",
                "name":"赞一下我们",
                "key":"V1001_GOOD"
            }
        ]
    }
]
```

}

- 参数说明

参数	必须	说明
access_token	是	调用接口凭证
agentid	是	企业应用的 id，整型。可在应用的设置页面查看
button	是	一级菜单数组，个数应为 1~3 个
sub_button	否	二级菜单数组，个数应为 1~5 个
type	是	菜单的响应动作类型，目前有 click、view 两种类型
name	是	菜单标题，不超过 16 个字节，子菜单不超过 40 个字节
key	click 类型必须	菜单 KEY 值，用于消息接口推送，不超过 128 字节
url	view 类型必须	网页链接，员工点击菜单可打开链接，不超过 256 字节

- 权限说明

管理员须拥有应用的管理权限，并且应用必须设置在回调模式。

返回结果

```
{
    "errcode":0,
    "errmsg":"ok"
```

}

根据上面官方的定义语义，我们菜单管理的 C#管理接口定义如下所示。

```
/// <summary>
/// 企业号菜单管理接口定义
/// </summary>
public interface ICorpMenuApi
{
    /// <summary>
    /// 获取菜单数据
    /// </summary>
    /// <param name="accessToken">调用接口凭证</param>
    /// <returns></returns>
    MenuListJson GetMenu(string accessToken, string agentid);


    /// <summary>
    /// 创建菜单
    /// </summary>
    /// <param name="accessToken">调用接口凭证</param>
    /// <param name="menuJson">菜单对象</param>
```

```
        /// <returns></returns>
        CommonResult CreateMenu(string accessToken, MenuListJson menuJson, string agentid);

        /// <summary>
        /// 删除菜单
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <returns></returns>
        CommonResult DeleteMenu(string accessToken, string agentid);
    }
```

我们以创建菜单的实现为例来介绍微信企业号菜单的操作，其他的操作类似处理，都是返回一个公共的消息类，方便处理和读取，代码如下所示。

```
        /// <summary>
        /// 创建菜单
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="menuJson">菜单对象</param>
        /// <returns></returns>
```

```
        public CommonResult CreateMenu(string accessToken,
MenuListJson menuJson, string agentid)
        {
            var url =
string.Format("https://qyapi.weixin.qq.com/cgi-bin/menu/create?access
_token={0}&agentid={1}", accessToken, agentid);
            string postData = menuJson.ToJson();

            return Helper.GetCorpExecuteResult(url, postData);
        }
```

3、企业号菜单管理接口的调用和处理效果

调用的代码和效果图如下所示。

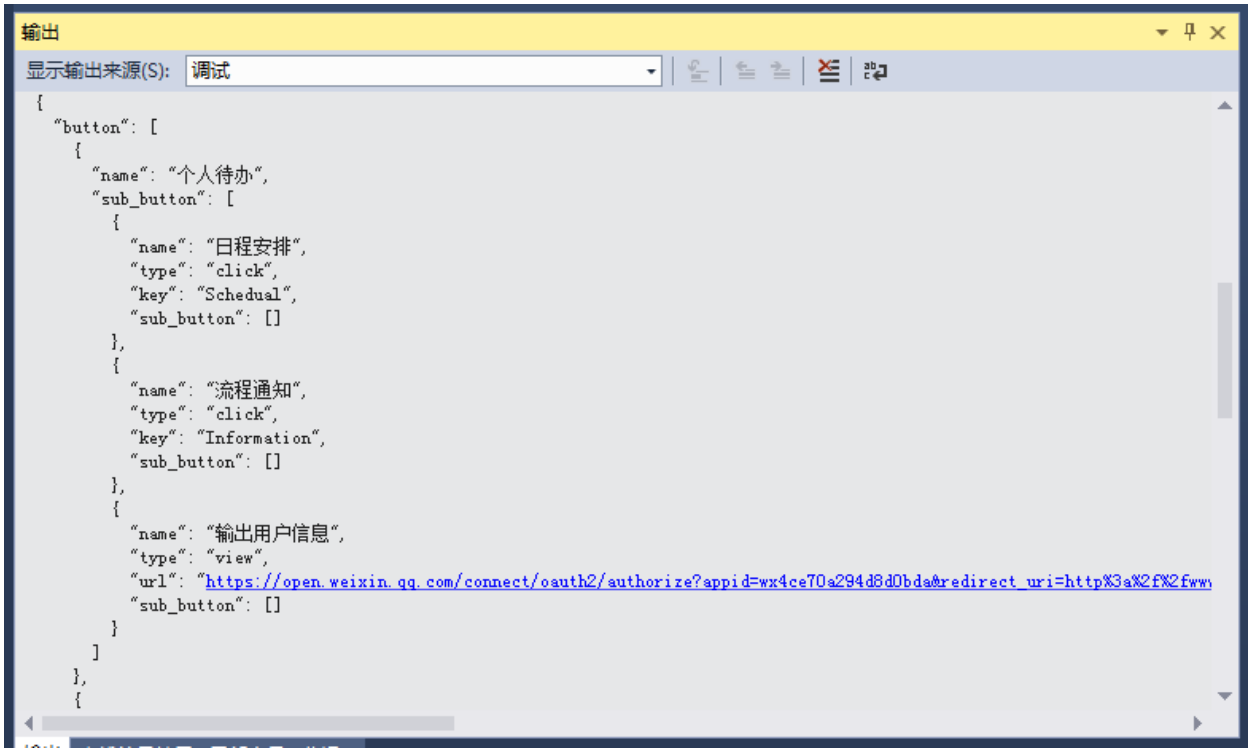
```
        private void btnMenuCreate_Click(object sender, EventArgs e)
        {
            MenuJson productInfo = new MenuJson("产品介绍", new
MenuJson[] {
                new MenuJson("软件产品介绍", ButtonType.click,
"event-software")
```

```
                , new MenuJson("框架源码产品", ButtonType.click,
"event-source")
                , new MenuJson("软件定制开发", ButtonType.click,
"event-develop")
            });

            MenuJson frameworkInfo = new MenuJson("框架产品", new
MenuJson[] {
                new MenuJson("Win 开发框架", ButtonType.click, "win"),
                new MenuJson("WCF 开发框架", ButtonType.click, "wcf"),
                new MenuJson("混合式框架", ButtonType.click, "mix"),
                new MenuJson("Web 开发框架", ButtonType.click,
"web")
                ,new MenuJson("代码生成工具", ButtonType.click,
"database2sharp")
            });

            MenuJson relatedInfo = new MenuJson("相关链接", new
MenuJson[] {
                new MenuJson("公司介绍", ButtonType.click,
"event_company"),
```

```
                    new MenuJson("官方网站", ButtonType.view,
"http://www.iqidi.com"),
                    new MenuJson("联系我们", ButtonType.click,
"event_contact"),
                    new MenuJson("应答系统", ButtonType.click, "set-1"),
                    new MenuJson("发邮件", ButtonType.view,
"http://mail.qq.com/cgi-bin/qm_share?t=qm_mailme&email=S31yfX15f
n8LOjplKCQm")
                });

                MenuListJson menuJson = new MenuListJson();
                menuJson.button.AddRange(new MenuJson[] { productInfo,
frameworkInfo, relatedInfo });

                //Console.WriteLine(menuJson.ToJson());

                if (MessageUtil.ShowYesNoAndWarning("您确认要创建菜单
吗") == System.Windows.Forms.DialogResult.Yes)
                {
                    ICorpMenuApi bll = new CorpMenuApi();
                    CommonResult result = bll.CreateMenu(token,
menuJson, agentid);
```

```
                Console.WriteLine("创建菜单：" + (result.Success ? "成功
" : "失败:" + result.ErrorMessage));
            }
        }

        private void btnMenuGet_Click(object sender, EventArgs e)
        {
            ICorpMenuApi bll = new CorpMenuApi();
            MenuListJson menu = bll.GetMenu(token, agentid);
            if (menu != null)
            {
                Console.WriteLine(menu.ToJson());
            }
        }
```

调用代码的测试输出如下所示。


输出
显示输出来源(S): 调试
{
"button": [
{
"name": "个人待办",
"sub_button": [
{
"name": "日程安排",
"type": "click",
"key": "Schedual",
"sub_button": []
},
{
"name": "流程通知",
"type": "click",
"key": "Information",
"sub_button": []
},
{
"name": "输出用户信息",
"type": "view",
"url": "https://open.weixin.qq.com/connect/oauth2/authorize?appid=wx4ce70a294d8d0bda&redirect_uri=http%3a%2f%2fww
"sub_button": []
}
]
},
{