**[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](http://www.cnblogs.com/wuhuacong/p/3735671.html)**

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。

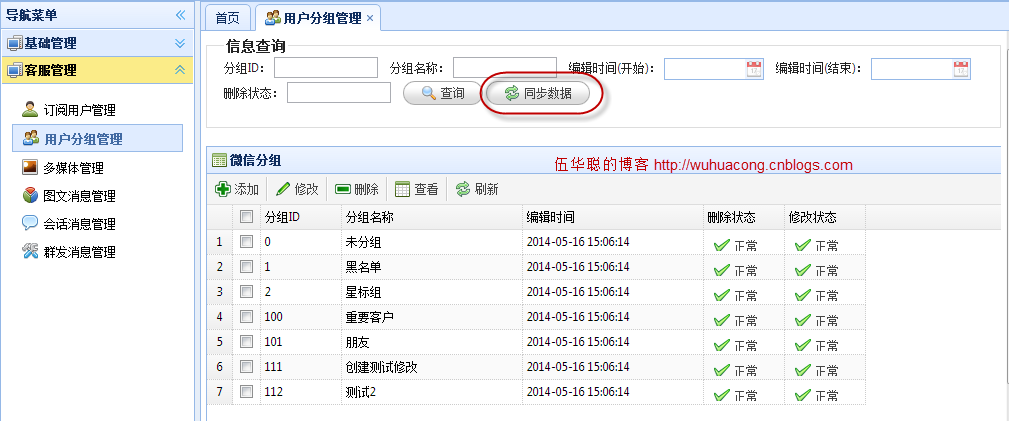
在较早之前的一篇随笔《[C#开发微信门户及应用(5)--用户分组信息管理](http://www.cnblogs.com/wuhuacong/p/3695351.html)》，我曾经介绍了微信分组的各种底层的API封装操作，里面主要就是对微信提供API的.NET高级分组，对所有的信息交换，通过实体性进行数据交换，使得我们调用API来处理微信的各种事务更加方便，从而为微信应用平台的管理奠定基础。其中这篇文章介绍了所有微信分组管理的API封装过程，用户分组管理，包含下面几个方面的内容：

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

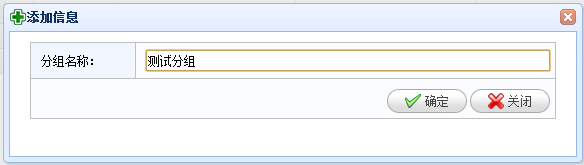
**1、用户分组，在管理系统中的界面设计**

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。

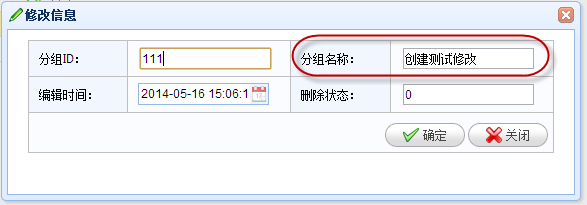
在管理系统里面，对微信分组的列表管理界面设计如下所示。



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。



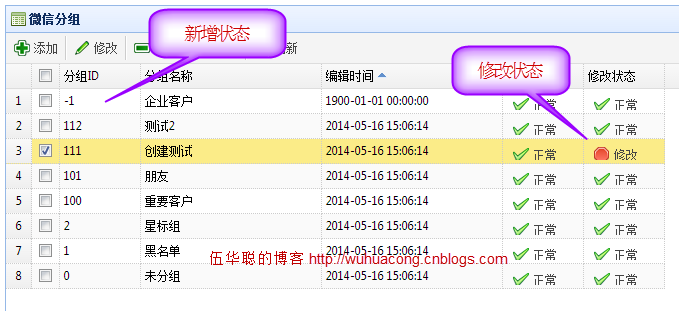
编辑分组信息界面如下所示。当对分组进行编辑保存后，系统会记住那些修改过的分组就是了。



**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。

[复制代码](javascript:void(0);)

//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);

}

});

});

}

[复制代码](javascript:void(0);)

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。

[复制代码](javascript:void(0);)

/// <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);

}

[复制代码](javascript:void(0);)

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。

[复制代码](javascript:void(0);)

/// <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;

}

[复制代码](javascript:void(0);)

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。

[复制代码](javascript:void(0);)

//用来统一请求忙碌显示的设置

$.ajaxSetup({

beforeSend: function () {

$("#loading").show();

},

complete: function () {

$("#loading").hide();

}

});