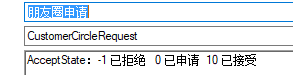
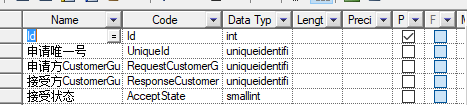
动工——朋友圈

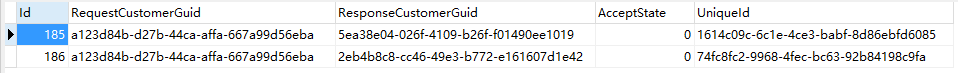
是

crm数据库，四个表：

（1）







A和B，比方花花和阿瓜妞，A，也就是花花请求与B（阿瓜妞）建立朋友关系，发出申请，这时候就多了一条数据：

请求方：花花id 响应方：阿瓜妞的id 接受状态（AcceptState）0

这时候，阿瓜妞那边肯定弹出弹窗，这个是轮询吗，阿瓜妞那边弹出是否接受的弹窗。

我自己觉得，中介者模式，后者说观察者模式，这边点确定了，send事件直接也就触发了另一个用户的Receive方法，也就是弹窗事件。

接下来，阿瓜妞来处理请求，如果接受，AcceptState改为1，如果不同意，AcceptState就是-1.相应地，就是改变了这条数据库的内容。

整理：通知对方，以及弹窗，前端去做。我后台做的就是：

1.请求

A请求B，我写入一条数据。AcceptState为0

2.响应

B是否接受，就是去修改该AcceptState的值，为1或者-1.

3.这个不确定，如果要拉黑了。

是否要将AcceptState由1变成-1.

4.这个感觉是必要的，一旦上面将AcceptState设置为0，那意味着两个建立了朋友关系。那就在我的朋友圈表，MyCustomerCircle中加入两个人的id吗。

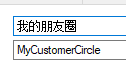
如果是这样，还要加一条IsDeleted，因为拉黑了，就置为true了。

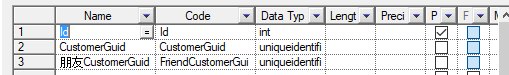
用户表用的是状态字段：

State：-4 已删除 -3 已禁用 -2 已拉黑 -1 已冻结 0 已注册（未激活） 10 已激活（待实名认证）15 实名认证未通过 16 实名认证审核中 17 运营审核通过 20 已实名认证



（2）







A和B建立了朋友关系，A比方是花花，那去查所有花花的朋友，就是CustomerGuid=花花的guid

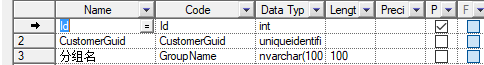
这里如果删除好友的话，要加字段。IsDeleted

用户表用的是状态字段：

State：-4 已删除 -3 已禁用 -2 已拉黑 -1 已冻结 0 已注册（未激活） 10 已激活（待实名认证）15 实名认证未通过 16 实名认证审核中 17 运营审核通过 20 已实名认证

（3）







这里，对应一个的医生Id，也就是CustomerGuid，他可以有n多的分组，比方老师，同学，朋友，家人。等等，他每次添加的时候都加到这个数据库。而当他要添加一个新的朋友时候，输出显示他所有的分组信息。供他选择，然后将新朋友放到该分组当中。

（4）







上面的朋友分组，如家人，朋友，老师，这种，已经建立。这里添加了新朋友，只要选择了对应的标签，就添加到该医生（花花）的分组A（好朋友）中了。

而这里使用了GroupId，通过GroupId与分组表CustomerCircleGroup建立连接join查询，或者说直接以该GroupId去GroupId表中获取了当前的医生id，CustomerGuid，以及分组名GroupName，这样就获得了医生和朋友间的一一对应，并且显示了分组的信息。

比方 花花——恶友——僵僵

要做：

0.

前置页，也就是通讯录页面的首页：

0.1——显示通讯录首页



这里获取所有朋友的信息，信息包括：

朋友列表：包含头像、姓名、科室、所在医院名称（限定最大长度，超出的显示...

0.2 如果有新的朋友，进行处理，同意或者拒绝，分别将数据库的AcceptState进行修改。

这里应该是一个ajax，当显示首页的同时，一个ajax请求，看表CustomerCircleRequest中对应的响应方ResponseCustomerGuid，查出所有响应方是当前登陆用户自己，以及AcceptState为0的所有数据，然后显示在新朋友中。

接下来处理的方式就是修改该条数据中AcceptState是1还是-1.而对应的我的朋友圈中，应该也是要增加这条数据，并且state设置为1，正常。0还是-1是删除状态。

新朋友界面：



（1）

交互1：进入新的朋友入口：通过通讯录

a：

进入新朋友界面后，就查ResponseCustomerGuid是当前用户，并且AcceptState是0的情况。然后列表。

当点同意之后，就AcceptState置为1.并且在我的朋友圈表MyCustomerCircle增加一条数据。

b：

还要查ResponseCustomerGuid是当前用户，并且AcceptState是1的已添加的记录。

这里不用了，这里是上面的操作成功后，比方返回了msg，是成功的ok，就显示已添加。

c：

自己的添加好友申请也显示

查RequestCustomerGuid是当前用户的所有信息，AcceptState为0的显示“等待验证”，AcceptState为1的显示“已添加”。

总结就是这里要进行两次查询，然后将结果整合传给UI端，也就是手机端。

交互2：返回到通讯录

就是跳回通讯录的首页，继续调用通讯录首页的接口

交互3：添加：点击”添加“图标，同意对方添加请求，状态显示为已添加

0.3 对现有的朋友列表进行搜索

前端还是后端去做呢，感觉所有列表都到前端了，前端自己去玩吧。



交互1：在搜索输入框内输入姓名或手机号，进行检索（此处注意IOS自带搜索功能，安卓的需要在页面加一个搜索按钮

do：

根据搜索的结果显示该用户，这里要确定要前端还是后端做。

交互2：选中某位朋友，点击进入朋友详情页

do：根据该朋友的CustomerId去查找详细信息

这里的搜索历史要求：

搜索历史：最多显示9个，按搜索时间最新的显示

应该是在前端的缓存当中。

0.4 添加朋友

跳到下面，详解

0.5 删除好友，

对应的数据库还没有加字段 state

就是当前CustomerId，要删除的好友的CustomerId，然后查到这条数据，将state置为删除

0.6 选中某条朋友记录，点击进入详情页（看到这里，接下来没仔细看了）

获取该朋友CustomerId对应的许多数据吧

1.

分组，加好友，删好友。

2.

添加朋友



加号“+”点了，搜索朋友，这里姓名或者手机，找到该customer。

2.1——搜索用户

交互：在搜索输入框内输入姓名或手机号，进行检索（此处注意IOS自带搜索功能，安卓的需要在页面加一个搜索按钮

根据手机或者姓名去用户表中获取该用户的信息：



搜索的结果这样的标准去显示：

搜索结果列表：包含头像、姓名、科室、所在医院（限定最大长度，超出的显示...）

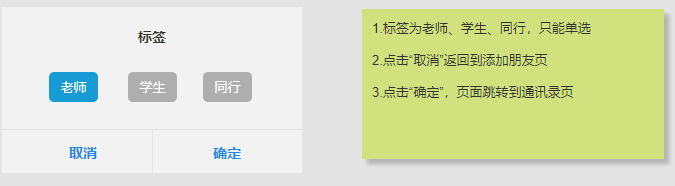


2.2——当点击了添加

交互：点击“添加”按钮，弹出朋友标签分类弹窗

（1）点击添加按钮，post请求返回当前用户的所有分类表

朋友圈分组表中获取该用户的所有分组信息，比方，家人，学生，老师，朋友



（2）单项选择了一个分组后，发送请求加好友的申请

请求参数，当前CustomerId，对方CustomerId，然后在表CustomerCircleRequest中增加这条数据。

FriendRequest

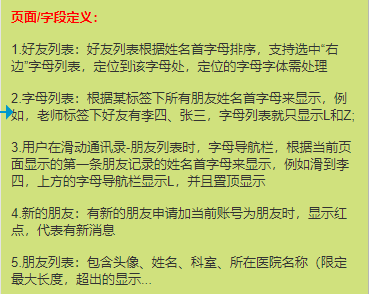
添加好友，dal层用void

bll层用apiModel

3.

逻辑分析:

1.



这里一堆排序，按首字母这些操作，感觉我后台这里只要返回了所有好友信息的list集合，里面带名字了吗？好像没唉，所以要加name，在dal那里，然后给返回到前端，这样拿到了名字前端自己去玩。

还有一种，就是我后台处理好，但这样，前端每点击一次姓名，字母，就要后后台交互一次，感觉费操作，流量了。还是放前端里面好。

2.

朋友列表，以为着每个好友的guid拿到，要去表里查，这个接口应该已经有了。

问题，和李阳确认：

1.

这个分组，老师，学生，同行，是一开始就确认了吗，如果是，就每增加一个医生，就要默认地给他加这个三个分组内容。

2.

3.

开工：

1.先写dal层。操作数据库。

2.再写bll层。写在crm，manager里面。

实现都写在这里面。

但仅限于crm，其他的模块未必这样。

3.wcf实现层，就一句话，因为都写在crm manager里面。

李阳表示，本来dal层的代码是可以写在这里，但因为之前都这么写了，所以现在也这么写。

4.写上契约

5.去web api中添加控制器方法

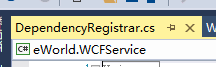
在UserController中写

或者加一个分组的Controller也行。

写接口，HttpPost请求。

6.bll层注入

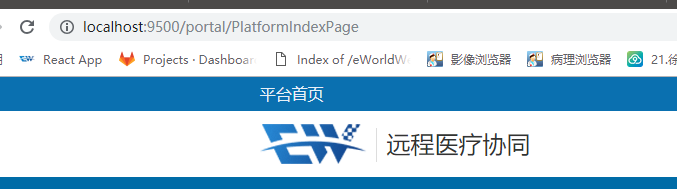
在这里注入，加入这么句话，就出现了。



//朋友圈获取分组

builder.RegisterType<CustomerCircleService>().As<ICustomerCircleService>().InstancePerHttpRequest();

首页终于正常了！开森！



添加，到我这里，customerId已经传过来了，我拿到，访问数据库查询分组信息，默认地给建立老师，学生，朋友，这三个标签。

Tag

List<ServiceCenterManager> GetTagListByCustomerGuid(Guid customerGuid);

List<CustomerCircleGroupModel> GetCustomerCircleGroupListByCustomerGuid(Guid customerGuid);

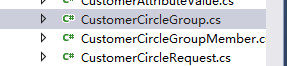
CustomerCircleGroup

遇到：

1.

在eWorld.Entity中，发现数据中的内容，已经建立好了，dbFirst。

我差点去entity中自己建立model类了，好傻。



2.

因为返回的是全部，所以就不需要建dto类了。



3.

ui那边反应过来，先把三个都作为分组，老师，学生，同行。

目前，同行就是显示所有的医生好友，也就是当前的所有好友。

以后可能会加患者，那这样的话，患者+同行才是所有好友了。

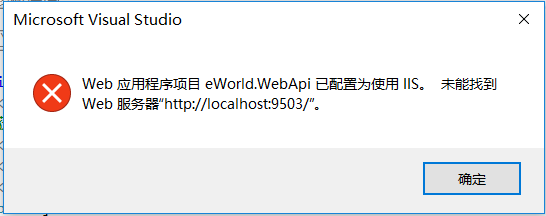
4.

bll层，dal层都加一个注入。

发现：

1.

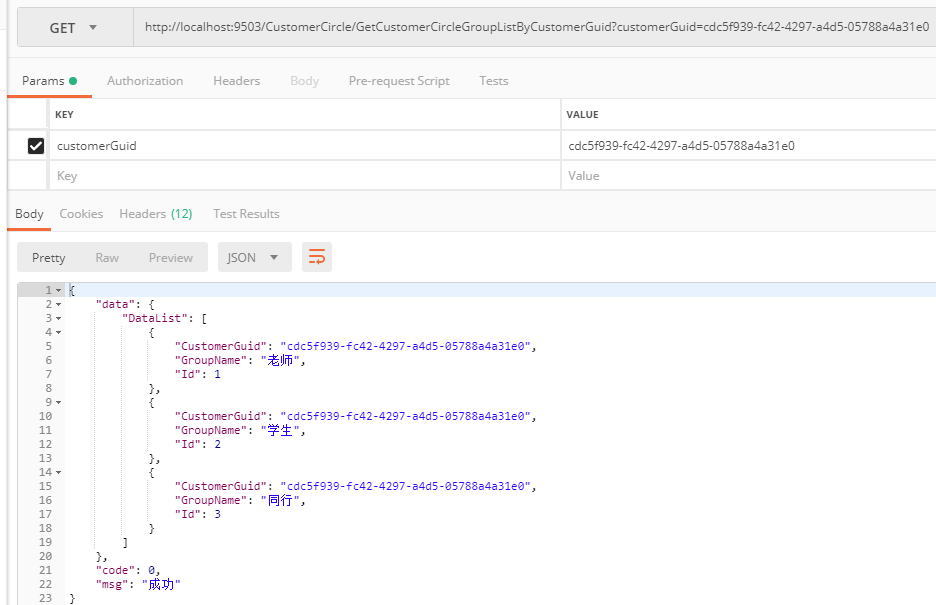
回来，架起来web api



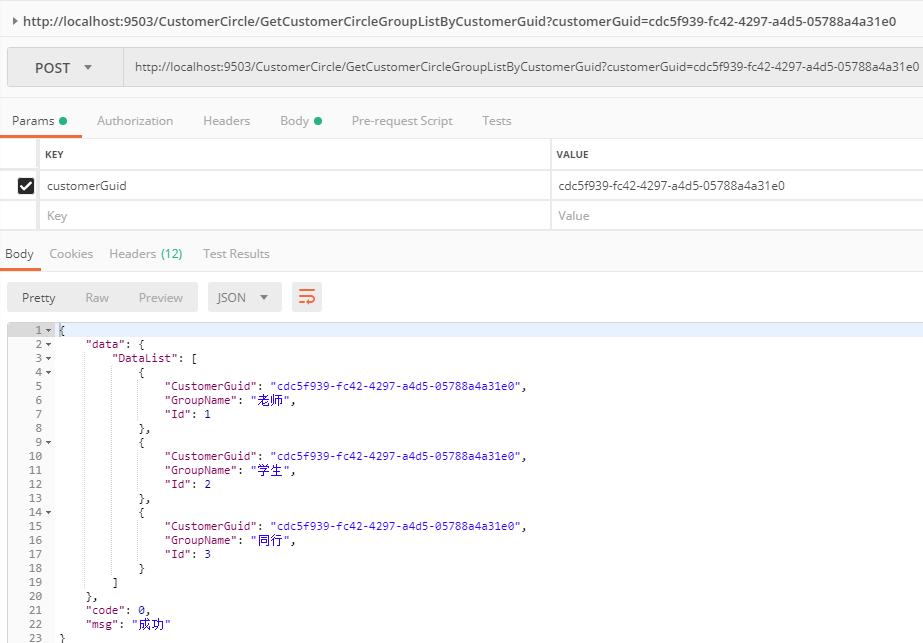
发现不是9500端口尼玛，明天回公司重新用9503端口去获取。

2.

用get测试，搞定了！！！！



哈哈哈。



让我guid不用返回，用outtomap转一下。

又开始学习api：

1.



返回体，现在也是有code，msg，还有泛型T data属性。

还有充分利用http状态码





2.FromUri和FromBody的查找

**FromUri和FromBody用途**

首先，我们来看一段代码：

1 [Route("all01\_2/{id:int?}")] 2 [AcceptVerbs("POST", "GET")] 3 public HttpResponseMessage GetAllStudents01\_2(int id = 0) 4 { 5 var students = db.GetAll(); 6 7 if (id > 0) 8 { 9 students = students.Where(b => b.Id == id).ToList();10 }11 12 return Request.CreateResponse(HttpStatusCode.OK, students);13 }

这里的action传递进来了一个名称id的参数，然后下面做了 id > 0 的判断来获取不同的数据，这种场景我们经常会遇到，并且查询条件的参数不止一个，这样来看我们会不断的在action里面增加传递的参数，格式可能会如此： public HttpResponseMessage GetAllStudents01\_2(int id = 0,参数2,参数3,参数...) ，这样如果需要用到的参数有10个以上，通常看起来不是很方便，这个时候**FromUri和FromBody**的用途就来了；先来看**FromUri**，首先需要我们把上面的那个Action方法的入参格式改成这样：

1 [Route(@"all01\_3/{id:int=0}/{name?}")] 2 [AcceptVerbs("POST", "GET")] 3 public HttpResponseMessage GetAllStudents01\_3([FromUri]MoSearch moSearch) 4 { 5 var students = db.GetAll().AsEnumerable(); 6 7 if (moSearch.Id > 0) 8 { 9 students = students.Where(b => b.Id == moSearch.Id);10 }11 12 if (!string.IsNullOrWhiteSpace(moSearch.Name))13 {14 students = students.Where(b => b.Name.Contains(moSearch.Name));15 }16 17 return Request.CreateResponse(HttpStatusCode.OK, students);18 }

通过实体类 MoSearch 来获取调用端传递的参数，实体类MoSearch定义的属性字段如：

public class MoSearch

{

public int Id { get; set; }

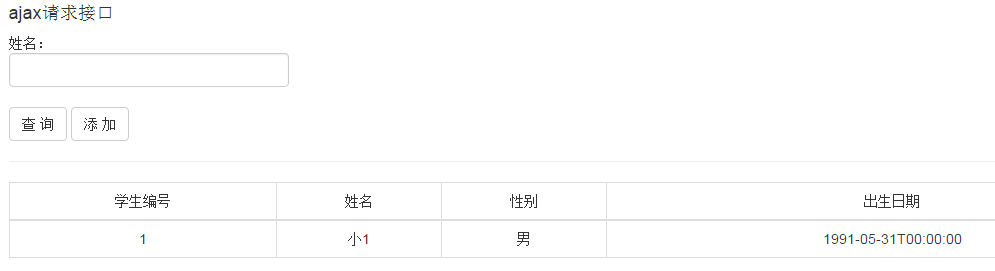
public string Name { get; set; }

}

然后我们改造下查询学生列表的ajax请求方法如：

1 $("#btnSearch").on("click", function () { 2 3 var tabObj = $("#tab tbody"); 4 tabObj.html(‘tr><td colspan="4">加载中...</td></tr>‘); 5 6 7 var url = "http://localhost:1001/s/all01\_3/1"; 8 var txtName = $("input[name=‘txtName‘]").val(); 9 if (txtName.length > 0) {10 url = url + "/" + txtName;11 }12 13 $.get(url, function (data) {14 15 console.log(data);16 17 var tabHtml = [];18 $.each(data, function (i, item) {19 20 tabHtml.push(‘<tr>‘);21 tabHtml.push("<td>" + item.Id + "</td>");22 tabHtml.push("<td>" + item.Name + "</td>");23 tabHtml.push("<td>" + (item.Sex ? "男" : "女") + "</td>");24 tabHtml.push("<td>" + item.Birthday + "</td>");25 tabHtml.push(‘</tr>‘);26 });27 if (tabHtml.length <= 0) { tabHtml.push(‘tr><td colspan="4">暂无数据</td></tr>‘); }28 29 tabObj.html(tabHtml.join(‘‘));30 });31 });

这里Id=1是固定了，直接查询id=1的学生信息，并且有一个可传递的参数学生名称txtName的值，如果用户没有输入学生名称，那么就不会加入到这个get请求中，下面来看下首先不输入学生姓名查询条件的效果：



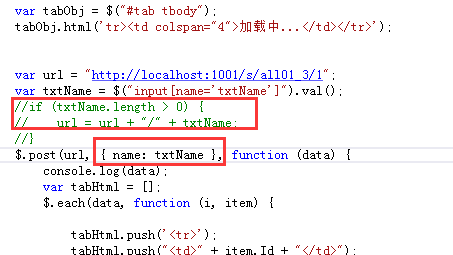
这里能看出来只有学生编号为1的学生被查出来了，下面我们再录入姓名查询条件，能看到结果如：



能看到此时查不到名称为“小2”的学生信息，咋们再改成“小1”试试，



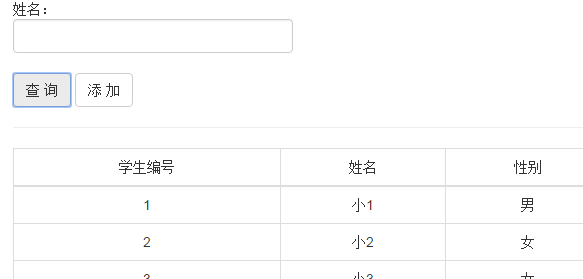
这个时候小1的学生能被查出来，这说明咋们的api的all01\_3能够通过 FromUri]MoSearch moSearch 获取到我们输入的参数；下面我们使用**FromBody**来演示她的效果，首先需要修改js请求webapi格式如：



然后把webapi的all01\_3修改为如下代码：

1 [Route(@"all01\_3/{name?}")] 2 [AcceptVerbs("POST", "GET")] 3 public HttpResponseMessage GetAllStudents01\_3([FromBody]MoSearch moSearch) 4 { 5 var students = db.GetAll().AsEnumerable(); 6 7 if (moSearch.Id > 0) 8 { 9 students = students.Where(b => b.Id == moSearch.Id);10 }11 12 if (!string.IsNullOrWhiteSpace(moSearch.Name))13 {14 students = students.Where(b => b.Name.Contains(moSearch.Name));15 }16 17 return Request.CreateResponse(HttpStatusCode.OK, students);18 }

主要区别吧FromUri换成了FromBody，最后咋们来一起看下没有录入学生名称查询条件的测试效果图：



录入学生查询条件后的效果图：



由上看出能正常查询出来数据，这说明 [FromBody]MoSearch moSearch 获取到了调用客户端的请求参数；好了FromBody和FromUri测试的例子就这些，这两者的区别这里总结下：

FromBody：主要用来获取post方式请求的参数

FromUri：主要获取url地址的请求参数（可以看成get方式）

这两者的效果对比我就不做了，有兴趣的朋友可以自己验证下；本次分享的内容就到这里吧，主要讲解了关于数据模型的一些知识，希望各位喜欢，多多点赞。

3.

这里想到，web api这一层，进来的参数可能很多，用FromUri或者FromBody，需要建立一个对应的参数模型类，而返回的结果，可以需要转化，所以也可能要建立一个output model类。

项目里面的代码就有这样的model类。

[Route("GetSubjectExpertList")]

[HttpPost]

public BaseApiModel<ApiPageListModel<SubjectExpertModel>> GetClinicSubjectExpertList([FromBody]SubjectExpertSearchModel search)

{

search.PageIndex = search.PageIndex - 1;

return \_clinicClient.Invoke(s => s.GetSubjectExpertList(search));

}

我自己也打算用FromBody了，因为请求分组名是guid，但一旦点击了之后，就把name属性的值传进来了。还不如直接就用model模型。

老杨表示这种方法比较麻烦，因为哪怕一个参数，也要写一个model类。

4.

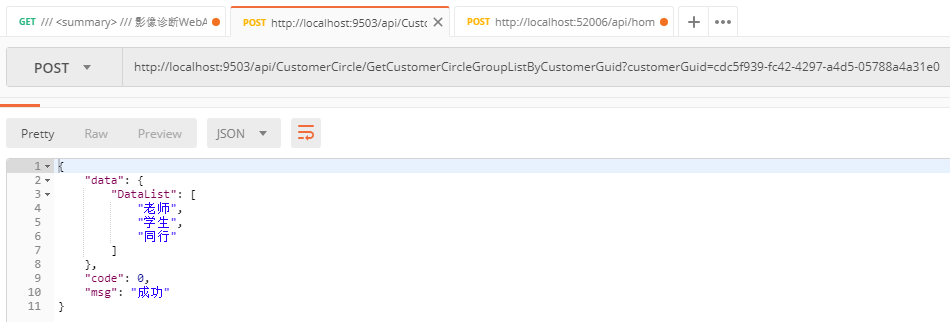
搞了半天，原来错误在于发送jason全部要双引号

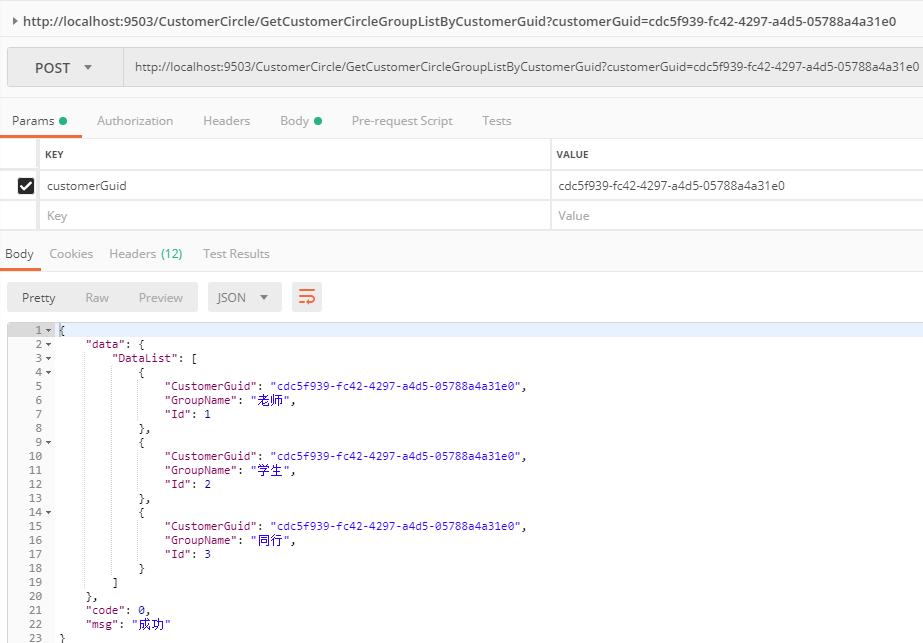


这样就正确了，囧。

现在把list<CustomerCircleGroup>变成了list<string>

结果成了：





一对比发现，不行的。少了字段了。所以还是要改回来，用什么outmap。

李大神的教育：

1.

医生的guid不是传进来的，而是默认当前登录的。

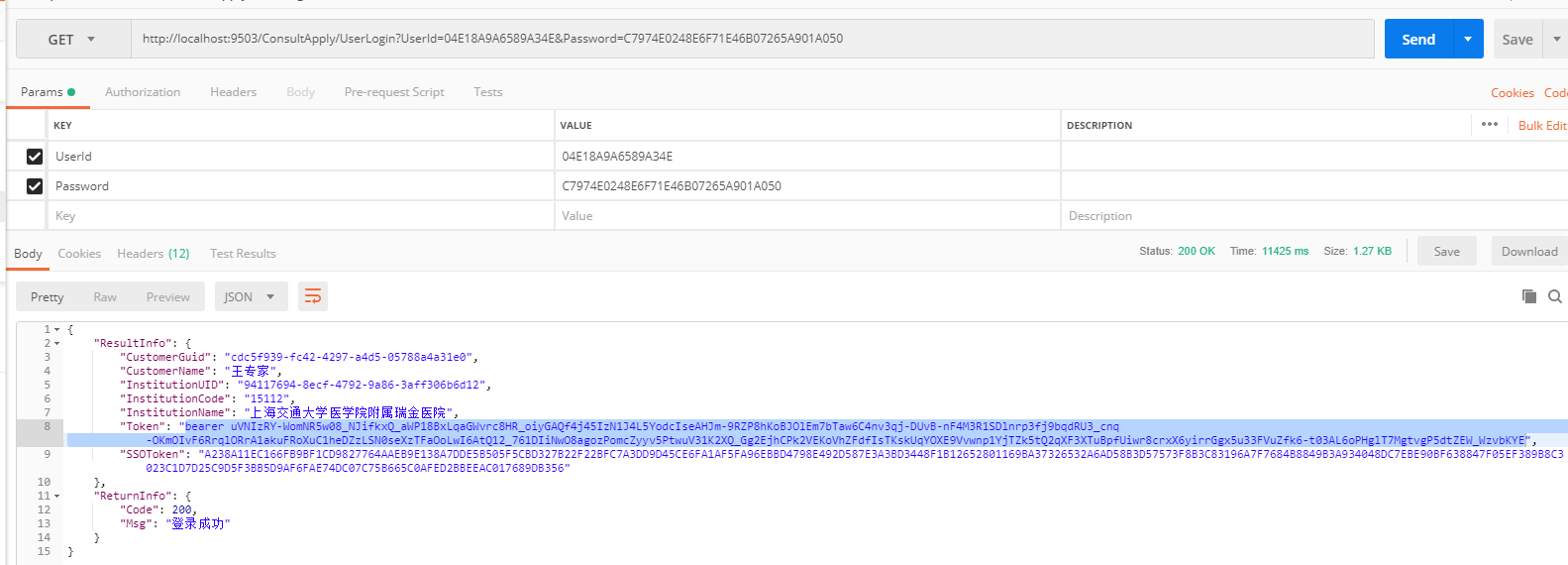
2.

mapto model的用法，以及mapto list<>的方法。

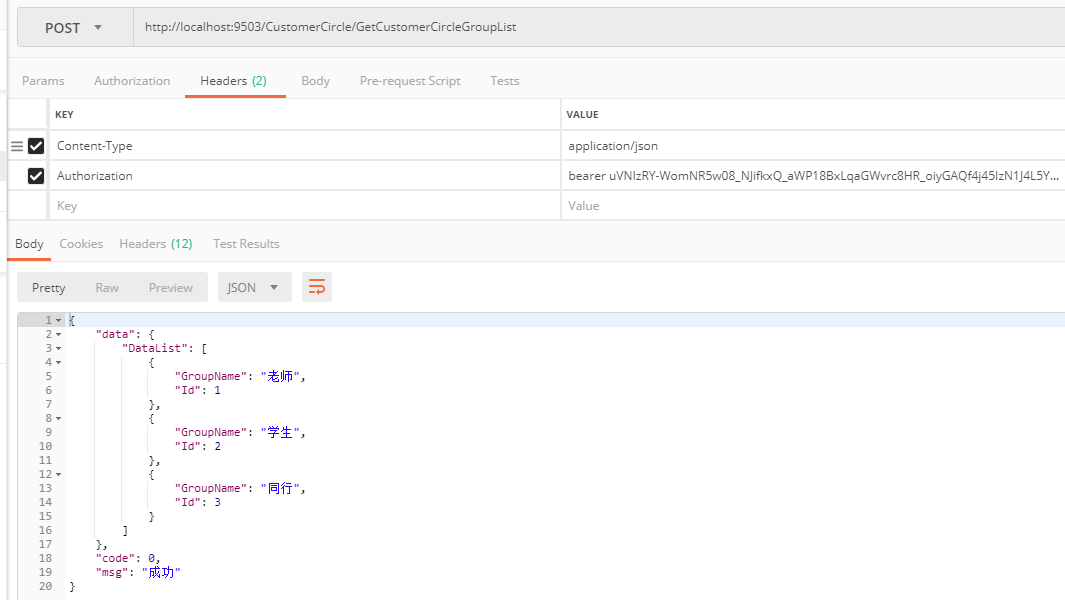
3.

步骤：

（1）先登录获取当前的token



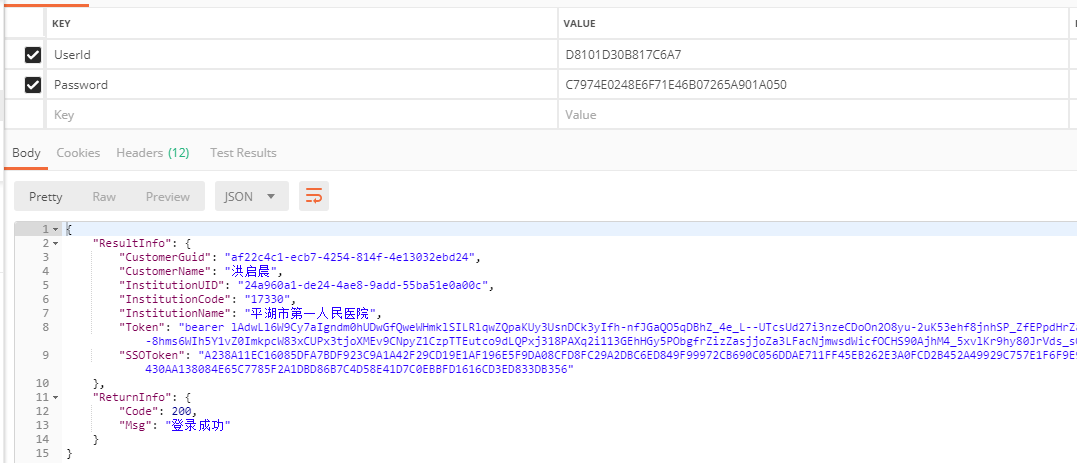
（2）把token复制到当前的接口中，请求分组信息



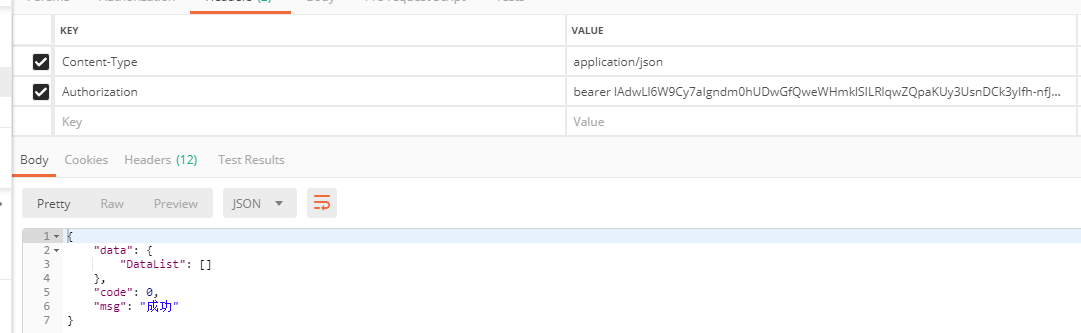
4.

换一个医生测试，看会不会自动生成三个默认分组

（1）登录洪启晨，获取token



获取分组信息

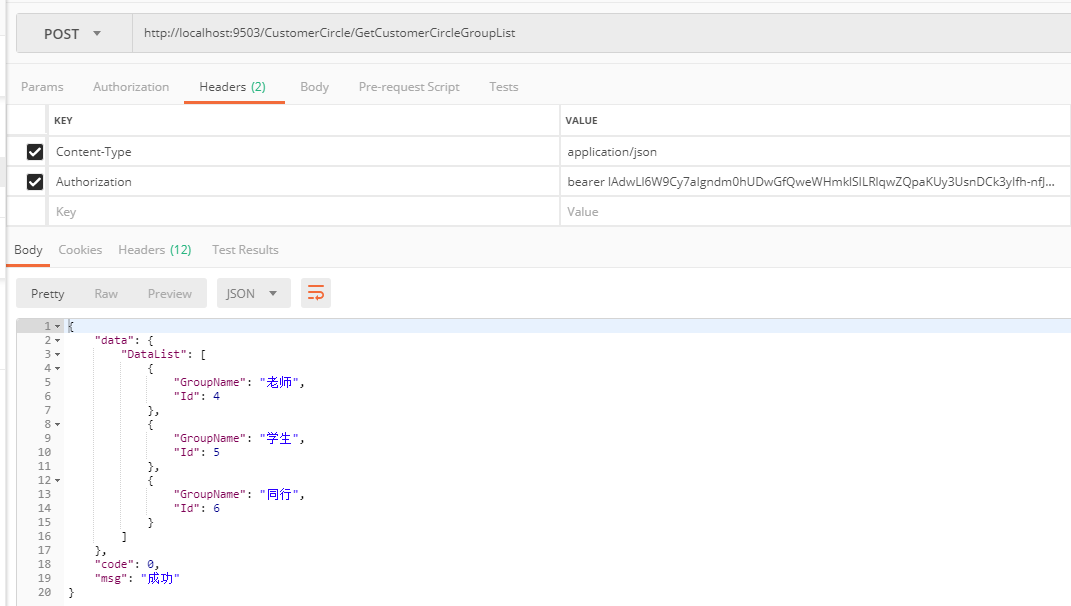


尼玛，没有自动创建啊

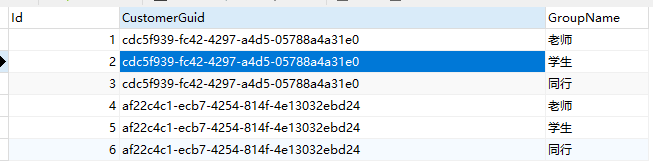
怀疑是延时加载的缘故，所以回去查看。

问了李彪大神，知道了怎么断点调试。把api设置为iis启动，然后断点调试，附加进程。

好了，成功！！！

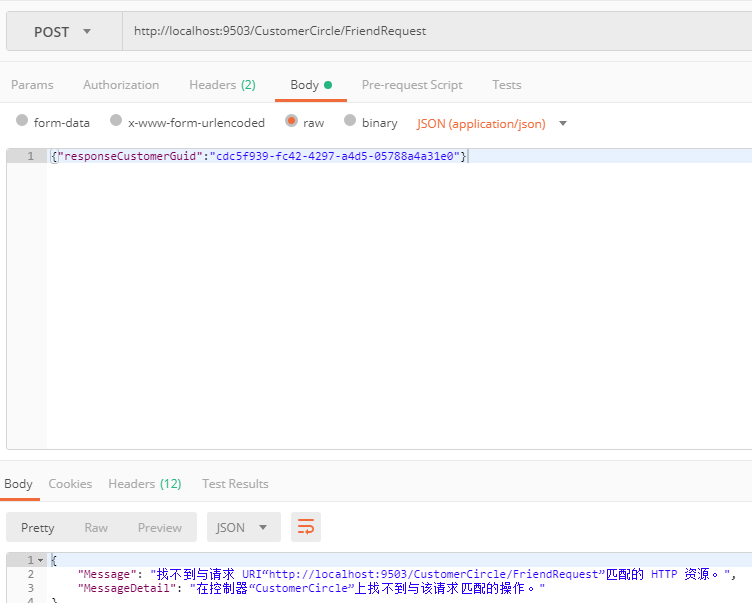


数据库中也进去了：

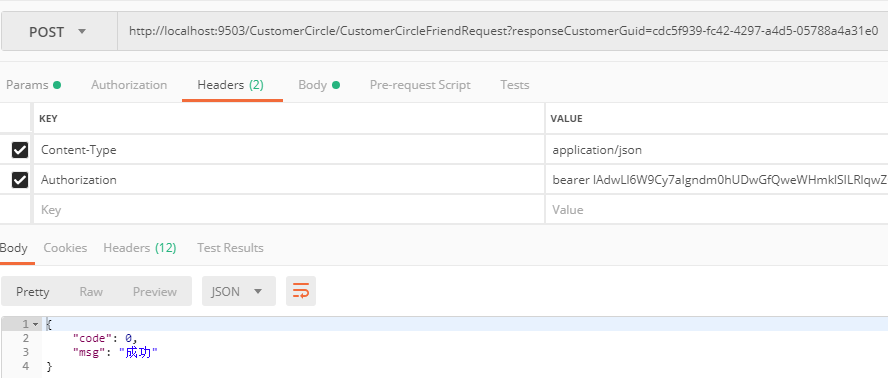


接下来测试，好友申请！

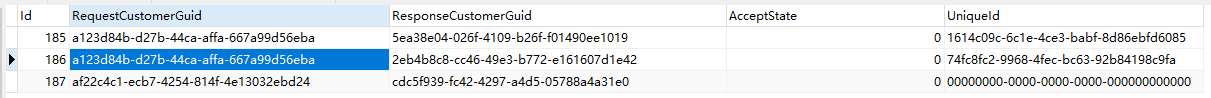
结果：



原来是方法名写得不对，以参数的形式，并且post提交，就ok了。！！

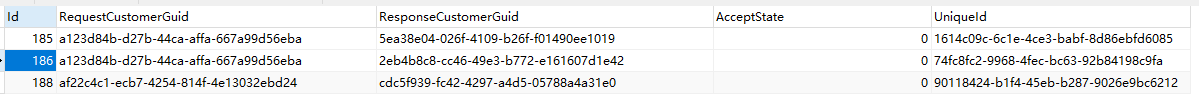


数据库中多了一条数据。



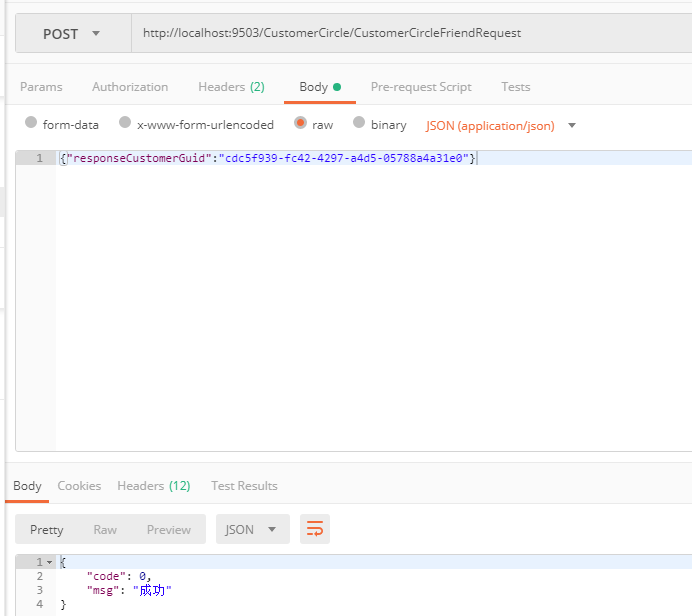
但是唯一guid坑爹了啊。

修改了guid的生成方法后就没有问题了。

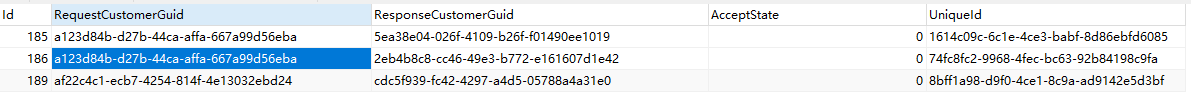


接下来把api这里的参数改为[FromBody]

ok！成功了！



数据库中有了这条记录：



5.