**[C#进阶系列——WebApi 接口参数不再困惑：传参详解](http://www.cnblogs.com/landeanfen/p/5337072.html)**

**阅读目录**

* [一、get请求](http://www.cnblogs.com/landeanfen/p/5337072.html#_label0)
  + [1、基础类型参数](http://www.cnblogs.com/landeanfen/p/5337072.html#_label0_0)
  + [2、实体作为参数](http://www.cnblogs.com/landeanfen/p/5337072.html#_label0_1)
  + [3、数组作为参数](http://www.cnblogs.com/landeanfen/p/5337072.html#_label0_2)
  + [4、“怪异”的get请求](http://www.cnblogs.com/landeanfen/p/5337072.html#_label0_3)
* [二、post请求](http://www.cnblogs.com/landeanfen/p/5337072.html#_label1)
  + [1、基础类型参数](http://www.cnblogs.com/landeanfen/p/5337072.html#_label1_0)
  + [2、实体作为参数](http://www.cnblogs.com/landeanfen/p/5337072.html#_label1_1)
  + [3、数组作为参数](http://www.cnblogs.com/landeanfen/p/5337072.html#_label1_2)
  + [4、后台发送请求参数的传递](http://www.cnblogs.com/landeanfen/p/5337072.html#_label1_3)
* [三、put请求](http://www.cnblogs.com/landeanfen/p/5337072.html#_label2)
  + [1、基础类型参数](http://www.cnblogs.com/landeanfen/p/5337072.html#_label2_0)
  + [2、实体作为参数](http://www.cnblogs.com/landeanfen/p/5337072.html#_label2_1)
  + [3、数组作为参数](http://www.cnblogs.com/landeanfen/p/5337072.html#_label2_2)
* [四、delete请求](http://www.cnblogs.com/landeanfen/p/5337072.html#_label3)
* [五、总结](http://www.cnblogs.com/landeanfen/p/5337072.html#_label4)

**正文**

前言：还记得刚使用WebApi那会儿，被它的传参机制折腾了好久，查阅了半天资料。如今，使用WebApi也有段时间了，今天就记录下API接口传参的一些方式方法，算是一个笔记，也希望能帮初学者少走弯路。本篇针对初初使用WebApi的同学们，比较基础，有兴趣的且看看。

WebApi系列文章

* [C#进阶系列——WebApi接口测试工具：WebApiTestClient](http://www.cnblogs.com/landeanfen/p/5210356.html)
* [C#进阶系列——WebApi 跨域问题解决方案：CORS](http://www.cnblogs.com/landeanfen/p/5177176.html)
* [C#进阶系列——WebApi身份认证解决方案：Basic基础认证](http://www.cnblogs.com/landeanfen/p/5287064.html)
* [C#进阶系列——WebApi接口传参不再困惑：传参详解](http://www.cnblogs.com/landeanfen/p/5337072.html)
* [C#进阶系列——WebApi接口返回值不困惑：返回值类型详解](http://www.cnblogs.com/landeanfen/p/5501487.html)
* [C#进阶系列——WebApi异常处理解决方案](http://www.cnblogs.com/landeanfen/p/5363846.html)
* C#进阶系列——WebApi区域Area使用小结

本篇打算通过get、post、put、delete四种请求方式分别谈谈基础类型（包括int/string/datetime等）、实体、数组等类型的参数如何传递。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

# 一、get请求

对于取数据，我们使用最多的应该就是get请求了吧。下面通过几个示例看看我们的get请求参数传递。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

## 1、基础类型参数

[HttpGet]

public string GetAllChargingData(int id, string name)

{

return "ChargingData" + id;

}

[复制代码](javascript:void(0);)

$.ajax({

type: "get",

url: "http://localhost:27221/api/Charging/GetAllChargingData",

data: { id: 1, name: "Jim", bir: "1988-09-11"},

success: function (data, status) {

if (status == "success") {

$("#div\_test").html(data);

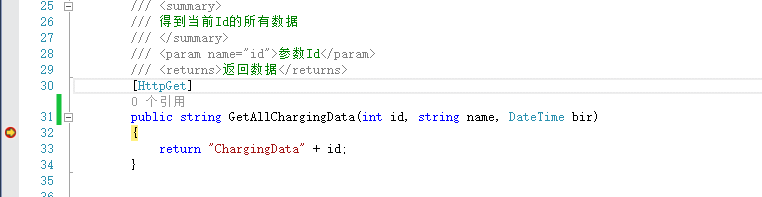
}

}

});

[复制代码](javascript:void(0);)

参数截图效果



 这是get请求最基础的参数传递方式，没什么特别好说的。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

## 2、实体作为参数

如果我们在get请求时想将实体对象做参数直接传递到后台，是否可行呢？我们来看看。

[复制代码](javascript:void(0);)

public class TB\_CHARGING

{

/// <summary>

/// 主键Id

/// </summary>

public string ID { get; set; }

/// <summary>

/// 充电设备名称

/// </summary>

public string NAME { get; set; }

/// <summary>

/// 充电设备描述

/// </summary>

public string DES { get; set; }

/// <summary>

/// 创建时间

/// </summary>

public DateTime CREATETIME { get; set; }

}

[复制代码](javascript:void(0);)

[HttpGet]

public string GetByModel(TB\_CHARGING oData)

{

return "ChargingData" + oData.ID;

}

[复制代码](javascript:void(0);)

　　$.ajax({

type: "get",

url: "http://localhost:27221/api/Charging/GetByModel",

contentType: "application/json",

data: { ID: "1", NAME: "Jim", CREATETIME: "1988-09-11" },

success: function (data, status) {

if (status == "success") {

$("#div\_test").html(data);

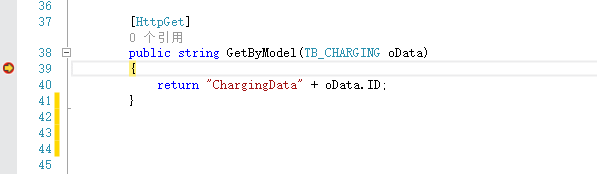
}

}

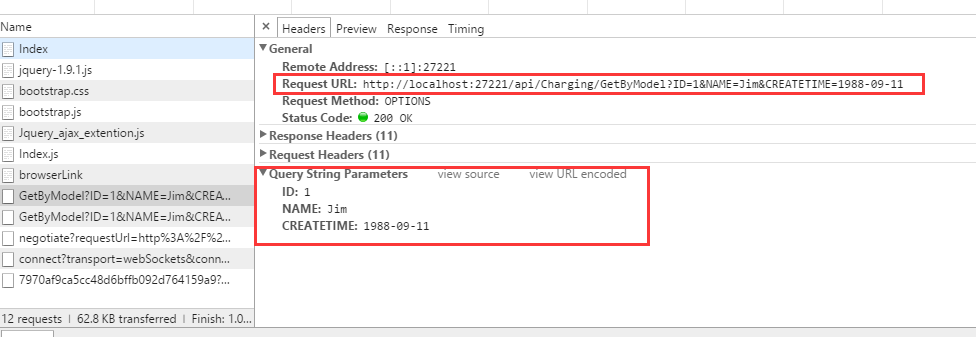
});

[复制代码](javascript:void(0);)

测试结果



由上图可知，在get请求时，我们直接将json对象当做实体传递后台，后台是接收不到的。这是为什么呢？我们来看看对应的http请求



原来，get请求的时候，默认是将参数全部放到了url里面直接以string的形式传递的，后台自然接不到了。

原因分析：**还记得有面试题问过get和post请求的区别吗？其中有一个区别就是get请求的数据会附在URL之后（就是把数据放置在HTTP协议头中），而post请求则是放在http协议包的包体中。**

根据园友们的提议，Get请求的时候可以在参数里面加上[FromUri]即可直接得到对象。还是贴上代码：

[复制代码](javascript:void(0);)

var postdata = { ID: "1", NAME: "Jim", CREATETIME: "1988-09-11" };

$.ajax({

type: "get",

url: "http://localhost:27221/api/Charging/GetAllChargingData",

data: postdata,

success: function (data, status) { }

});

[复制代码](javascript:void(0);)

[HttpGet]

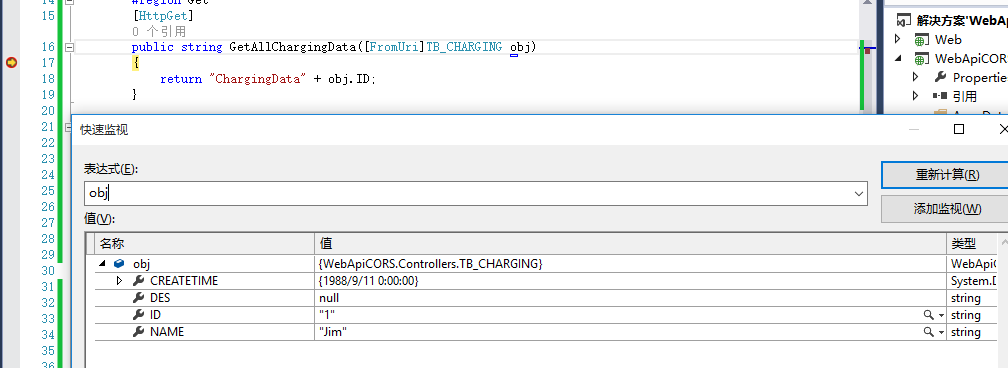
public string GetAllChargingData([FromUri]TB\_CHARGING obj)

{

return "ChargingData" + obj.ID;

}

得到结果：



如果你不想使用[FromUri]这些在参数里面加特性的这种“怪异”写法，也可以采用先序列化，再在后台反序列的方式。

[复制代码](javascript:void(0);)

　　$.ajax({

type: "get",

url: "http://localhost:27221/api/Charging/GetByModel",

contentType: "application/json",

data: { strQuery: JSON.stringify({ ID: "1", NAME: "Jim", CREATETIME: "1988-09-11" }) },

success: function (data, status) {

if (status == "success") {

$("#div\_test").html(data);

}

}

});

[复制代码](javascript:void(0);)

[HttpGet]

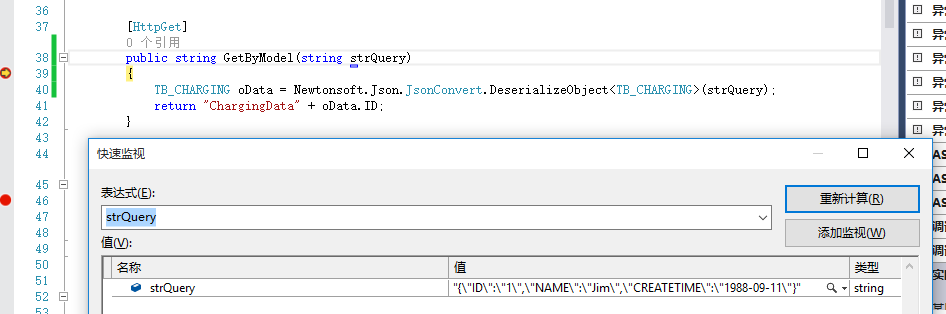
public string GetByModel(string strQuery)

{

TB\_CHARGING oData = Newtonsoft.Json.JsonConvert.DeserializeObject<TB\_CHARGING>(strQuery);

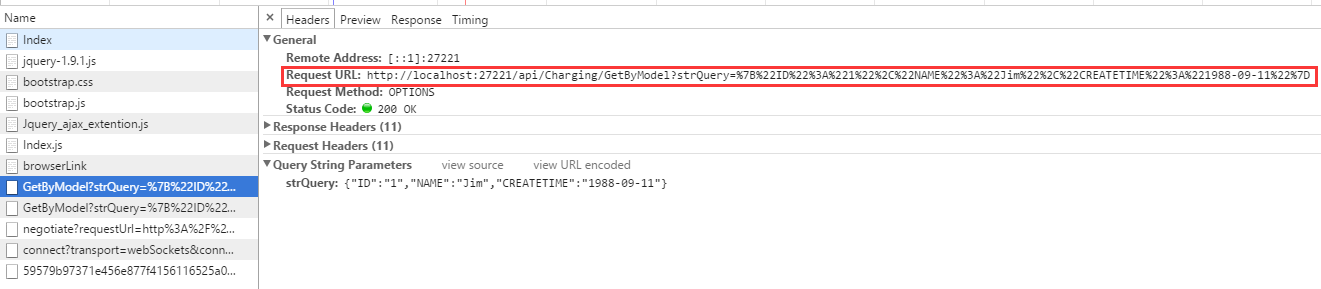
return "ChargingData" + oData.ID;

}



这样在后台得到我们序列化过的对象，再通过反序列化就能得到对象。

在url里面我们可以看到它自动给对象加了一个编码：



至于还有园友们提到<http://www.asp.net/web-api/overview/formats-and-model-binding/parameter-binding-in-aspnet-web-api>的model binder这种方式，博主看了下，觉得略复杂。有兴趣的也可以试试。至于用哪一种方式传递对象，园友们可以自行选择。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

## 3、数组作为参数

一般get请求不建议将数组作为参数，因为我们知道get请求传递参数的大小是有限制的，最大1024字节，数组里面内容较多时，将其作为参数传递可能会发生参数超限丢失的情况。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

## 4、“怪异”的get请求

为什么会说get请求“怪异”呢？我们先来看看下面的两种写法对比。

### （1）WebApi的方法名称以get开头

[复制代码](javascript:void(0);)

$.ajax({

type: "get",

url: "http://localhost:27221/api/Charging/GetByModel",

contentType: "application/json",

data: { strQuery: JSON.stringify({ ID: "1", NAME: "Jim", CREATETIME: "1988-09-11" }) },

success: function (data, status) {

if (status == "success") {

$("#div\_test").html(data);

}

}

});

[复制代码](javascript:void(0);)

[HttpGet]

public string GetByModel(string strQuery)

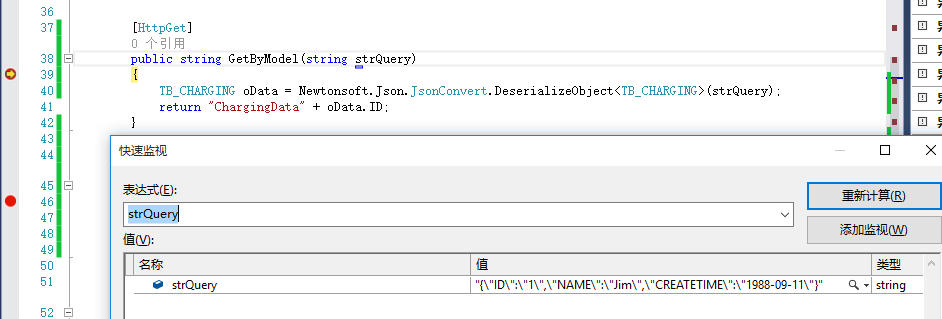
{

TB\_CHARGING oData = Newtonsoft.Json.JsonConvert.DeserializeObject<TB\_CHARGING>(strQuery);

return "ChargingData" + oData.ID;

}

这是标准写法，后台加**[HttpGet]**，参数正常得到：



为了对比，我将**[HttpGet]**去掉，然后再调用

//[HttpGet]

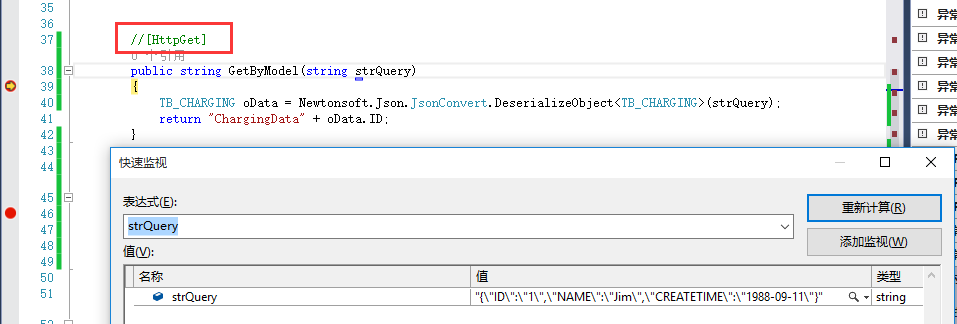
public string GetByModel(string strQuery)

{

TB\_CHARGING oData = Newtonsoft.Json.JsonConvert.DeserializeObject<TB\_CHARGING>(strQuery);

return "ChargingData" + oData.ID;

}



貌似没有任何问题！有人就想，那是否所有的get请求都可以省略掉[HttpGet]这个标注呢。我们试试便知。

### （2）WebApi的方法名称不以get开头

我们把之前的方法名由**GetByModel**改成**FindByModel**，这个再正常不过了，很多人查询就不想用Get开头，还有直接用Query开头的。这个有什么关系吗？有没有关系，我们以事实说话。

[复制代码](javascript:void(0);)

$.ajax({

type: "get",

url: "http://localhost:27221/api/Charging/FindByModel",

contentType: "application/json",

data: { strQuery: JSON.stringify({ ID: "1", NAME: "Jim", CREATETIME: "1988-09-11" }) },

success: function (data, status) {

if (status == "success") {

$("#div\_test").html(data);

}

}

});

[复制代码](javascript:void(0);)

[HttpGet]

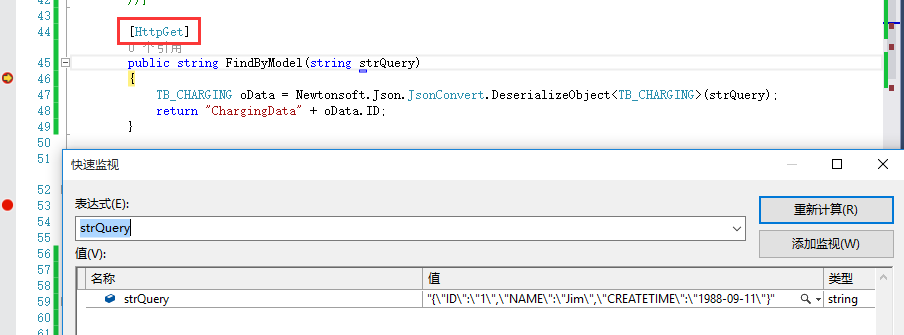
public string FindByModel(string strQuery)

{

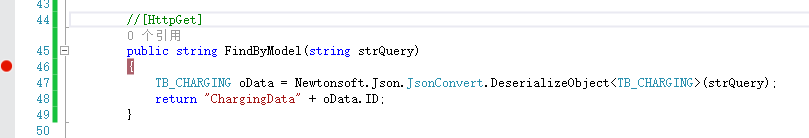
TB\_CHARGING oData = Newtonsoft.Json.JsonConvert.DeserializeObject<TB\_CHARGING>(strQuery);

return "ChargingData" + oData.ID;

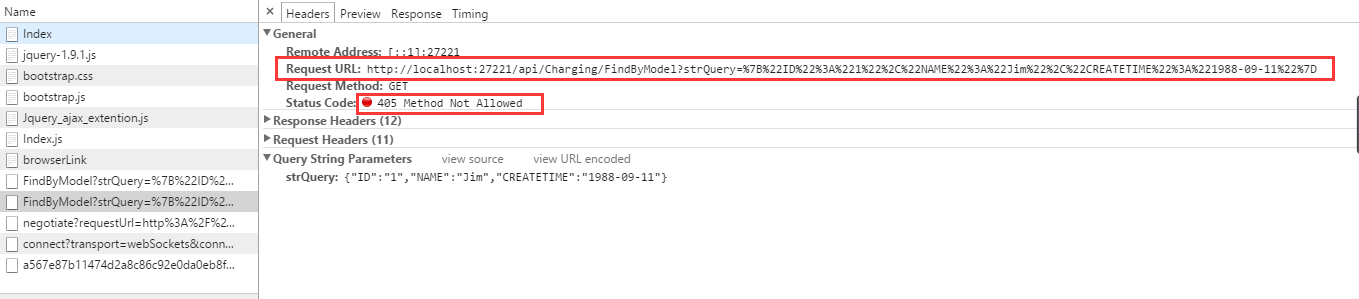
}



貌似又可行，没有任何问题啊。根据上面的推论，我们去掉**[HttpGet]**也是可行的，好，我们注释掉[HttpGet]，运行起来试试。



**结果是不进断点**，有些人不信，我们在浏览器里面看看http请求：



呵呵，这就奇怪了，就改了个方法名，至于这样么？还真至于！

博主的理解是：**方法名以Get开头，WebApi会自动默认这个请求就是get请求，而如果你以其他名称开头而又不标注方法的请求方式，那么这个时候服务器虽然找到了这个方法，但是由于请求方式不确定，所以直接返回给你405——方法不被允许的错误。**

最后结论：**所有的WebApi方法最好是加上请求的方式（[HttpGet]/[HttpPost]/[HttpPut]/[HttpDelete]），不要偷懒，这样既能防止类似的错误，也有利于方法的维护，别人一看就知道这个方法是什么请求。**

这也就是为什么很多人在园子里面问道为什么方法名不加[HttpGet]就调用不到的原因！

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

# 二、post请求

在WebApi的RESETful风格里面，API服务的增删改查，分别对应着http的post/delete/put/get请求。我们下面就来说说post请求参数的传递方式。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

## 1、基础类型参数

 post请求的基础类型的参数和get请求有点不一样，我们知道get请求的参数是通过url来传递的，而post请求则是通过http的请求体中传过来的，WebApi的post请求也需要从http的请求体里面去取参数。

### （1）错误的写法

[复制代码](javascript:void(0);)

$.ajax({

type: "post",

url: "http://localhost:27221/api/Charging/SaveData",

data: { NAME: "Jim" },

success: function (data, status) {

if (status == "success") {

$("#div\_test").html(data);

}

}

});

[复制代码](javascript:void(0);)

[HttpPost]

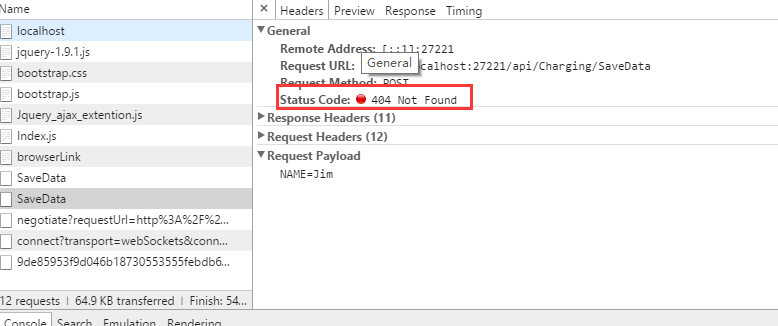
public bool SaveData(string NAME)

{

return true;

}

这是一种看上去非常正确的写法，可是实际情况是：



### （2）正确的用法

$.ajax({

type: "post",

url: "http://localhost:27221/api/Charging/SaveData",

data: { "": "Jim" },

success: function (data, status) {}

});

[HttpPost]

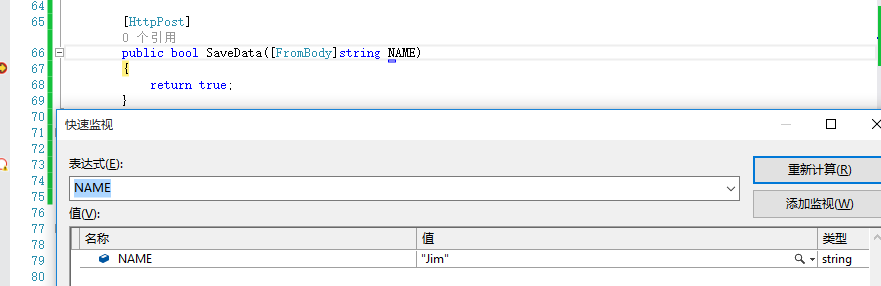
public bool SaveData([FromBody]string NAME)

{

return true;

}

这是一种另许多人头痛的写法，但是没办法，这样确实能得到我们的结果：



我们一般的通过url取参数的机制是键值对，即某一个key等于某一个value，而这里的FromBody和我们一般通过url取参数的机制则不同，它的机制是=value，没有key的概念，并且如果你写了key(比如你的ajax参数写的{NAME:"Jim"})，后台反而得到的NAME等于null。不信你可以试试。

上面讲的都是传递一个基础类型参数的情况，那么如果我们需要传递多个基础类型呢？按照上面的推论，是否可以([FromBody]string NAME, [FromBody]string DES)这样写呢。试试便知。

### （1）错误写法

$.ajax({

type: "post",

url: "http://localhost:27221/api/Charging/SaveData",

data: { "": "Jim","":"备注" },

success: function (data, status) {}

});

[HttpPost]

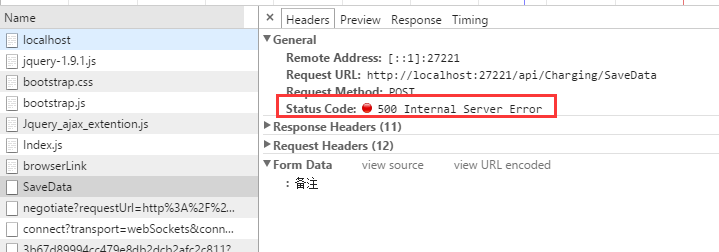
public bool SaveData([FromBody]string NAME, [FromBody] string DES)

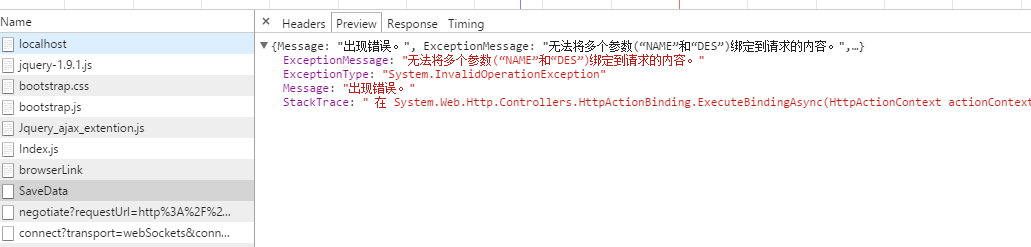
{

return true;

}

得到结果





这说明我们没办法通过多个[FromBody]里面取值，此法失败。

### （2）正确用法

既然上面的办法行不通，那我们如何传递多个基础类型的数据呢？很多的解决办法是新建一个类去包含传递的参数，博主觉得这样不够灵活，因为如果我们前后台每次传递多个参数的post请求都去新建一个类的话，我们系统到时候会有多少个这种参数类？维护起来那是相当的麻烦的一件事！所以博主觉得使用dynamic是一个很不错的选择。我们来试试。

[复制代码](javascript:void(0);)

$.ajax({

type: "post",

url: "http://localhost:27221/api/Charging/SaveData",

contentType: 'application/json',

data: JSON.stringify({ NAME: "Jim",DES:"备注" }),

success: function (data, status) {}

});

[复制代码](javascript:void(0);)

[HttpPost]

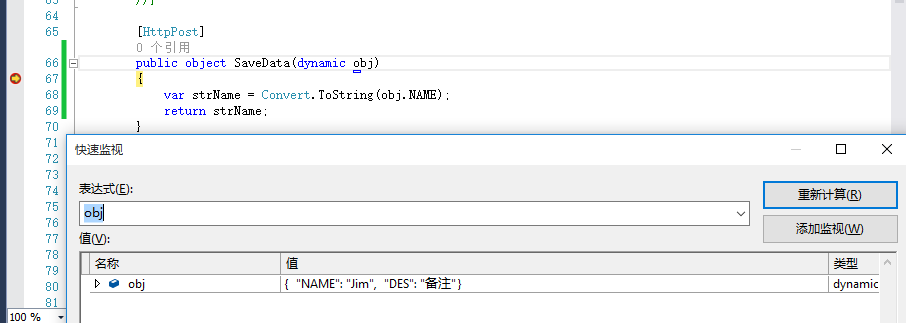
public object SaveData(dynamic obj)

{

var strName = Convert.ToString(obj.NAME);

return strName;

}



**通过dynamic动态类型能顺利得到多个参数，省掉了[FromBody]这个累赘，并且ajax参数的传递不用使用"无厘头"的{"":"value"}这种写法，有没有一种小清新的感觉~~有一点需要注意的是这里在ajax的请求里面需要加上参数类型为Json，即 contentType: 'application/json', 这个属性。**

### ****（3）推荐用法****

**通过上文post请求基础类型参数的传递，我们了解到了dynamic的方便之处，为了避免[FromBody]这个累赘和{"":"value"}这种"无厘头"的写法。博主推荐所有基础类型使用dynamic来传递，方便解决了基础类型一个或多个参数的传递，示例如上文。如果园友们有更好的办法，欢迎讨论。**

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

## 2、实体作为参数

### （1）单个实体作为参数

上面我们通过dynamic类型解决了post请求基础类型数据的传递问题，那么当我们需要传递一个实体作为参数该怎么解决呢？我们来看下面的代码便知：

$.ajax({

type: "post",

url: "http://localhost:27221/api/Charging/SaveData",

data: { ID: "1", NAME: "Jim", CREATETIME: "1988-09-11" },

success: function (data, status) {}

});

[HttpPost]

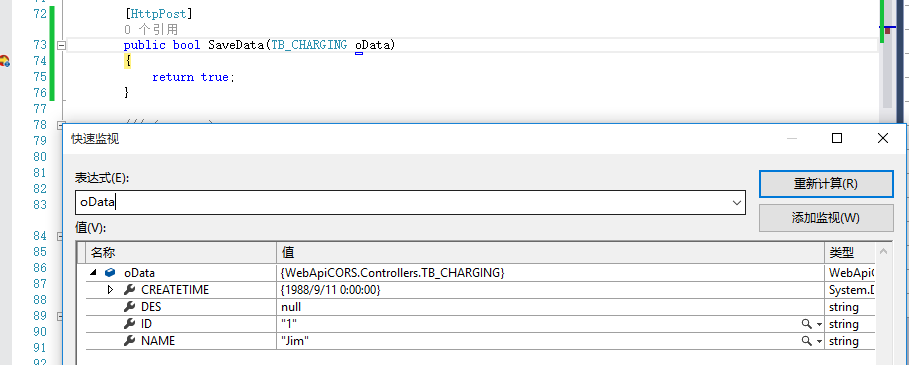
public bool SaveData(TB\_CHARGING oData)

{

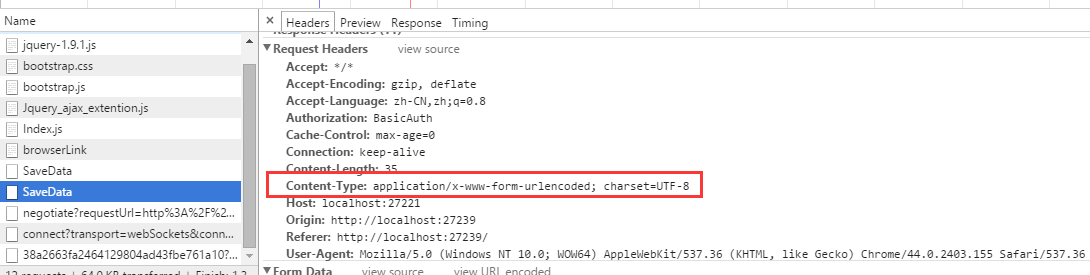
return true;

}

得到结果



原理解释：**使用实体作为参数的时候，前端直接传递普通json，后台直接使用对应的类型去接收即可，不用FromBody。但是这里需要注意的一点就是，这里不能指定contentType为appplication/json，否则，参数无法传递到后台。我们来看看它默认的contentType是什么：**



为了弄清楚原因，博主查了下http的Content-Type的类型。看到如下说明：

* application/x-www-form-urlencoded ： <form encType=””>中默认的encType，form表单数据被编码为key/value格式发送到服务器（表单默认的提交数据的格式）；
* application/json    ： JSON数据格式

**也就是说post请求默认是将表单里面的数据的key/value形式发送到服务，而我们的服务器只需要有对应的key/value属性值的对象就可以接收到。而如果使用application/json，则表示将前端的数据以序列化过的json传递到后端，后端要把它变成实体对象，还需要一个反序列化的过程。按照这个逻辑，那我们如果指定contentType为application/json，然后传递序列化过的对象应该也是可以的啊。博主好奇心重，还是打算一试到底，于是就有了下面的代码：**

[复制代码](javascript:void(0);)

var postdata = { ID: "1", NAME: "Jim", CREATETIME: "1988-09-11" };

$.ajax({

type: "post",

url: "http://localhost:27221/api/Charging/SaveData",

contentType: 'application/json',

data: JSON.stringify(postdata),

success: function (data, status) {}

});

[复制代码](javascript:void(0);)

[HttpPost]

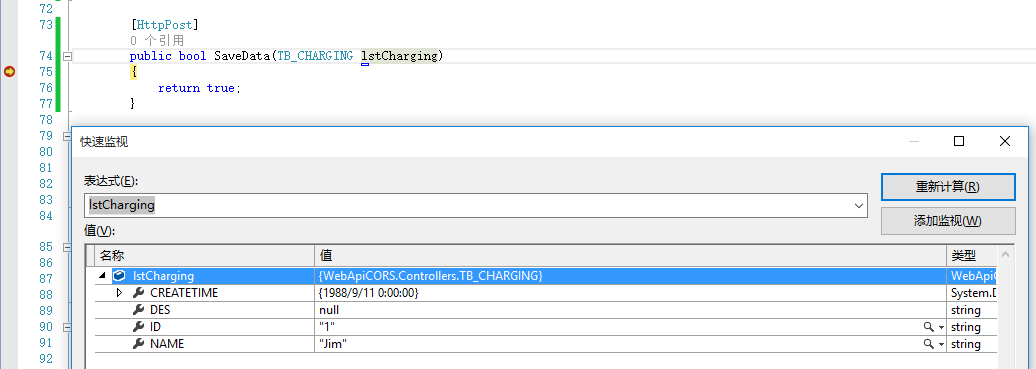
public bool SaveData(TB\_CHARGING lstCharging)

{

return true;

}

得到结果：



**尝试成功，也就是说，两种写法都是可行的。如果你指定了contentType为application/json，则必须要传递序列化过的对象；如果使用post请求的默认参数类型，则前端直接传递json类型的对象即可。**

### （2）实体和基础类型一起作为参数传递

有些时候，我们需要将基础类型和实体一起传递到后台，这个时候，我们神奇的dynamic又派上用场了。

[复制代码](javascript:void(0);)

var postdata = { ID: "1", NAME: "Jim", CREATETIME: "1988-09-11" };

$.ajax({

type: "post",

url: "http://localhost:27221/api/Charging/SaveData",

contentType: 'application/json',

data: JSON.stringify({ NAME:"Lilei", Charging:postdata }),

success: function (data, status) {}

});

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

[HttpPost]

public object SaveData(dynamic obj)

{

var strName = Convert.ToString(obj.NAME);

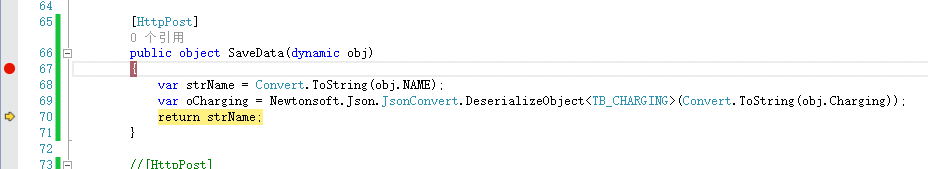
var oCharging = Newtonsoft.Json.JsonConvert.DeserializeObject<TB\_CHARGING>(Convert.ToString(obj.Charging));

return strName;

}

[复制代码](javascript:void(0);)

得到结果：



原理也不用多说，同上。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

## 3、数组作为参数

### （1）基础类型数组

[复制代码](javascript:void(0);)

var arr = ["1", "2", "3", "4"];

$.ajax({

type: "post",

url: "http://localhost:27221/api/Charging/SaveData",

contentType: 'application/json',

data: JSON.stringify(arr),

success: function (data, status) { }

});

[复制代码](javascript:void(0);)

[HttpPost]

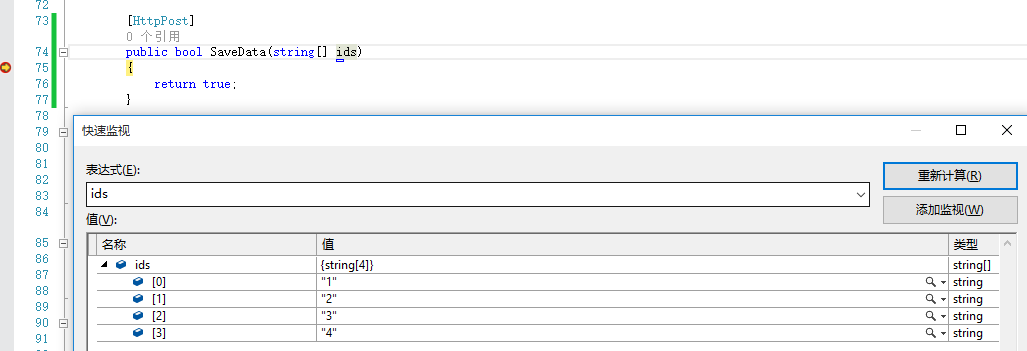
public bool SaveData(string[] ids)

{

return true;

}

得到结果：



### （2）实体集合

[复制代码](javascript:void(0);)

var arr = [

{ ID: "1", NAME: "Jim", CREATETIME: "1988-09-11" },

{ ID: "2", NAME: "Lilei", CREATETIME: "1990-12-11" },

{ ID: "3", NAME: "Lucy", CREATETIME: "1986-01-10" }

];

$.ajax({

type: "post",

url: "http://localhost:27221/api/Charging/SaveData",

contentType: 'application/json',

data: JSON.stringify(arr),

success: function (data, status) {}

});

[复制代码](javascript:void(0);)

[HttpPost]

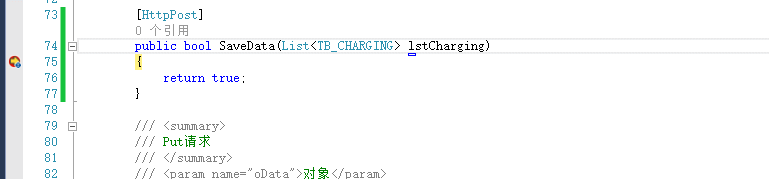
public bool SaveData(List<TB\_CHARGING> lstCharging)

{

return true;

}

得到结果：



[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

## 4、后台发送请求参数的传递

上面写了那么多，都是通过前端的ajax请求去做的，我们知道，如果调用方不是web项目，比如Android客户端，可能需要从后台发送http请求来调用我们的接口方法，如果我们通过后台去发送请求是否也是可行的呢？我们以实体对象作为参数来传递写写代码试一把。

[复制代码](javascript:void(0);)

　　　　public void TestReques()

　　　　{

　　　　 //请求路径

string url = "http://localhost:27221/api/Charging/SaveData";

//定义request并设置request的路径

WebRequest request = WebRequest.Create(url);

request.Method = "post";

//初始化request参数

string postData = "{ ID: \"1\", NAME: \"Jim\", CREATETIME: \"1988-09-11\" }";

//设置参数的编码格式，解决中文乱码

byte[] byteArray = Encoding.UTF8.GetBytes(postData);

//设置request的MIME类型及内容长度

request.ContentType = "application/json";

request.ContentLength = byteArray.Length;

//打开request字符流

Stream dataStream = request.GetRequestStream();

dataStream.Write(byteArray, 0, byteArray.Length);

dataStream.Close();

//定义response为前面的request响应

WebResponse response = request.GetResponse();

//获取相应的状态代码

Console.WriteLine(((HttpWebResponse)response).StatusDescription);

//定义response字符流

dataStream = response.GetResponseStream();

StreamReader reader = new StreamReader(dataStream);

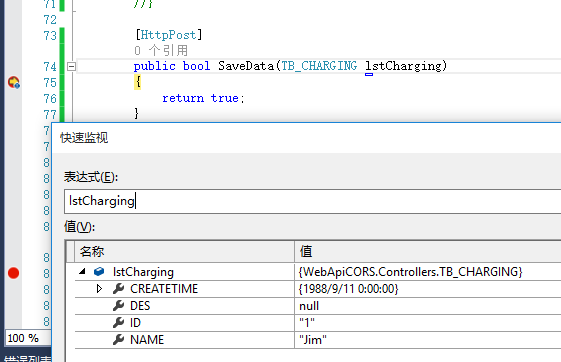
string responseFromServer = reader.ReadToEnd();//读取所有

Console.WriteLine(responseFromServer);

　　　　}

[复制代码](javascript:void(0);)

当代码运行到request.GetResponse()这一句的时候，API里面进入断点



尝试成功。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

# 三、put请求

WebApi里面put请求一般用于对象的更新。它和用法和post请求基本相同。同样支持[FromBody]，同样可以使用dynamic。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

## 1、基础类型参数

[复制代码](javascript:void(0);)

$.ajax({

type: "put",

url: "http://localhost:27221/api/Charging/Update",

contentType: 'application/json',

data: JSON.stringify({ ID: "1" }),

success: function (data, status) {}

});

[复制代码](javascript:void(0);)

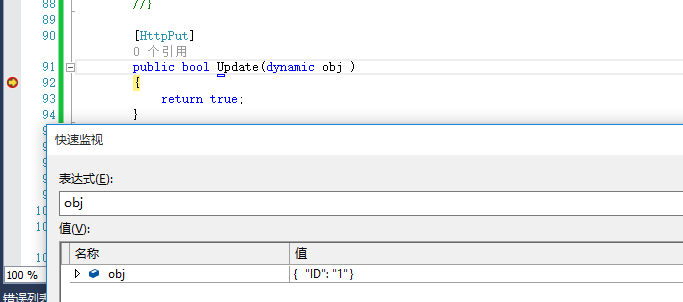
[HttpPut]

public bool Update(dynamic obj )

{

return true;

}



[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

## 2、实体作为参数

和post请求相同。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

## 3、数组作为参数

和post请求相同。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

# 四、delete请求

顾名思义，delete请求肯定是用于删除操作的。参数传递机制和post也是基本相同。下面简单给出一个例子，其他情况参考post请求。

[复制代码](javascript:void(0);)

var arr = [

{ ID: "1", NAME: "Jim", CREATETIME: "1988-09-11" },

{ ID: "2", NAME: "Lilei", CREATETIME: "1990-12-11" },

{ ID: "3", NAME: "Lucy", CREATETIME: "1986-01-10" }

];

$.ajax({

type: "delete",

url: "http://localhost:27221/api/Charging/OptDelete",

contentType: 'application/json',

data: JSON.stringify(arr),

success: function (data, status) {}

});

[复制代码](javascript:void(0);)

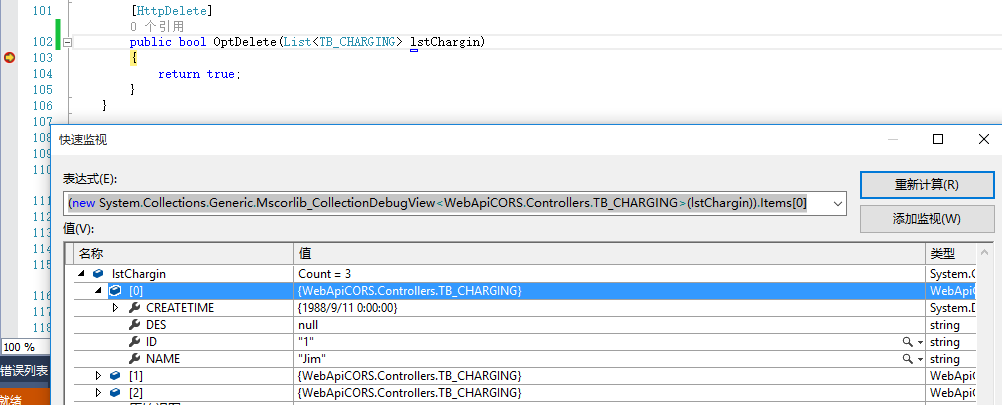
[HttpDelete]

public bool OptDelete(List<TB\_CHARGING> lstChargin)

{

return true;

}



[回到顶部](http://www.cnblogs.com/landeanfen/p/5337072.html#_labelTop)

# 五、总结

以上比较详细的总结了WebApi各种请求的各种参数传递。每种情况都是博主实际代码测试过的，内容不难，但如果刚接触这么些东西还是需要一点时间去熟悉的，在此做个总结，希望能帮到刚刚接触WebApi的园友们。如果本文能帮到你，不妨**推荐**下，您的推荐是博主继续总结的动力！

最近打算做点自己的东西出来，将博客里面的一些好的技术融合进去。有项目合作的小伙伴赶快联系博主吧！

本文原创出处：<http://www.cnblogs.com/landeanfen/>

欢迎各位转载，但是未经作者本人同意，转载文章之后**必须在文章页面明显位置给出作者和原文连接**，否则保留追究法律责任的权利！