**一:简单介绍什么是Web api**

API（Application Programming Interface,应用程序编程接口）

是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件得以

访问一组列程的能力，而又无需访问源码，或理解内部工作机制的细节。

REST属于一种设计风格，REST 中的 POST（新增数据），GET（取得数据），PUT（更新数据），DELETE（删除数据）

来进行数据库的增删改查，而如果开发人员的应用程式符合REST原则，则它的服务为“REST风格Web服务“也称的RESRful Web API”。

微软的web api是在vs2012上的mvc4项目绑定发行的，它提出的web api是完全基于RESTful标准的，完全不同于之前的

（同是SOAP协议的）wcf和webService，它是简单，代码可读性强的，上手快的，如果要拿它和web服务相比，我会说，

它的接口更标准，更清晰，没有混乱的方法名称，有的只有几种标准的请求，如get,post,put,delete等，

它们分别对应的几个操作，下面讲一下：

GET：生到数据列表（默认），或者得到一条实体数据

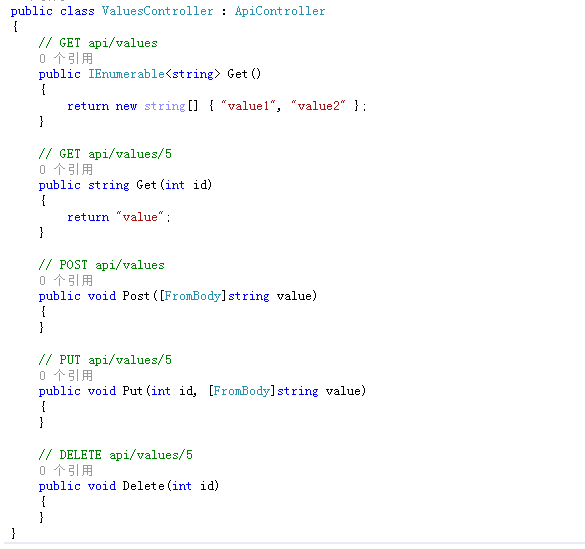
POST：添加服务端添加一条记录，记录实体为Form对象

PUT：添加或修改服务端的一条记录，记录实体的Form对象，记录主键以GET方式进行传输

DELETE：删除 服务端的一条记录

**二:怎么定义的 Post Get Put 和 Delete**

2.1 结合WebApi控制器来分析



从演示的列子,我们可以看到在Action 中没有使用[HttpGet]、[HttpPost] 等修饰,那究竟它是如何运作的呢

Action 皆以HTTP 动词开头Get、Post、Put、Delete ,这个也是刚好符合 webapi的约定的,什么约定呢?

你调用什么类型的方法 ,例如 post 方法,那么他就去 你的所有的 action 里面 去找 以 post 开头的方法 ,

名字可以随便叫,例如 postToDataBase 等等,只要开头匹配 就可以了

打个比喻，假设今天服务端收到了一个GET 请求时，会去查找对应的Controller 并且Action 以"Get..." 开头的方法，

举个例子：GetMembers、GetTime，以此类推，如果我们从jQuery Ajax 发出了一个POST 请求，也会自动对应到以"Post..."

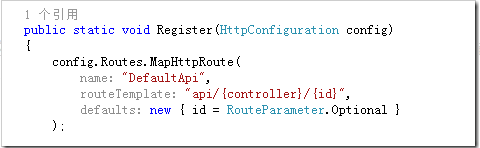
开头的Action 内，**也就是说实际呼叫哪个Controller 的Action 不是利用网址(/控制器名/Action名)来决定，而是依照HTTP 所送出的请求来决定，**

这也就是非常典型的REST风格，而在Web API 中也处理了回传的数据，让我们看看Get() 这个方法，

回传IEnumerable<T> 的方法，等于我们拥有了强类型。

**2.2我们再来看看默认的 api 路由表**

这里写图片描述



这里，只注册到了controller，没有到action，因为api的action名称是有约定的。

webapi大约有这样的约定：

action名称中有get的，0参数，匹配路由到/控制器

action名称中有get的，1参数，匹配路由到 /控制器/id

action名称中有post的，0参数，匹配路由到post方式的/控制器

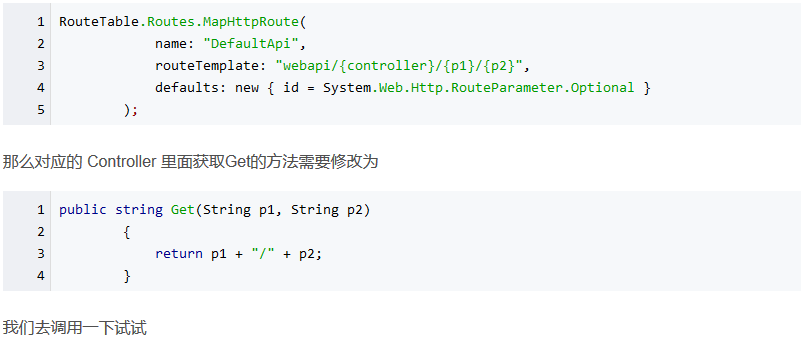
action名称中有post的，1参数，匹配路由到post方式的/控制器/id

(案例演示1)

2.3 另外我们可以注意到在Post Put的方法参数有一个关键字[ FromBody ]，而Get、Delete则没有。，事实上没有加[ FromBody ]就默认为[ FromUri ].

[ FromBody ]表示由请求文件本体中取得资料，就像一般表单Post Submit一样，取得资料的来源是由请求本体中取得，而[ FromUri ]则表示由URI中取得资料，就像在网址列中的所夹带的参数

在webapi的示范代码里，Get方法很简单只有一个id参数的传入并且是简单的int型，因此我们可以用 [http://localhost/api/Values/1](http://localhost/api/Values/1" \t "_blank) 这样的请求执行Get（int id）方法，但事实上并非每个请求都只用一个简单的参数就可以能搞定，有时我们可能需要2个或以上的参数才能传递或者是获取到数据，我们可以到路由里面改为接受两个参数，分别为{ p1 } & { p2 }

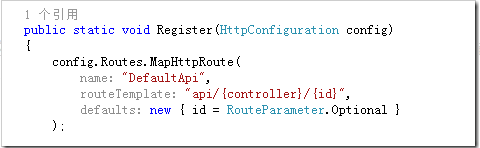


这种方法,虽然比较简单,但是如果我们的参数更多的时候,还是这样去修改 路由也是很麻烦的,我们就采用下面的方法,直接传递类到后台

2.4 第3方测试工具https://www.getpostman.com/

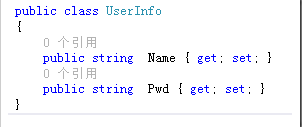
**三:复杂传值,直接从前台传递一个类到后台接收**

我们还是把路由表恢复成以下

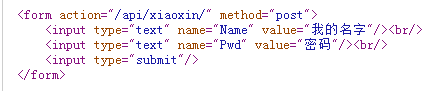


3.1 我们先在前台页面,用post方式,来传递一个类的2个值到后台

定义一个简单的类

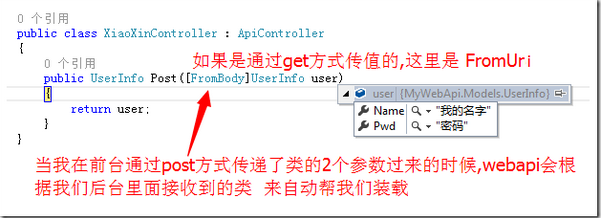


我们在前台提交的页面设置 方式为 post 提交 ,地址就指向我们的 webapi地址



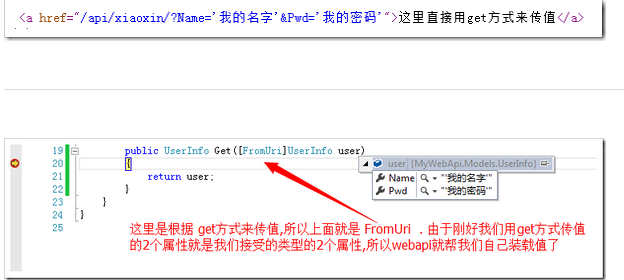
由于我们是通过 Post方式提交的数据,那么后台接收的时候,就是用 FromBody 来进行接收,由于刚好我们传递的前台数据就是类的2个字段,那么后台接收的时候,也可以直接用类来接收,webapi会根据类型和字段来帮我们自动加载数据,获取到值.

如果你这里是用 get 方式进行传值的,那么这里的 FromBody 就应该换成 FromUri



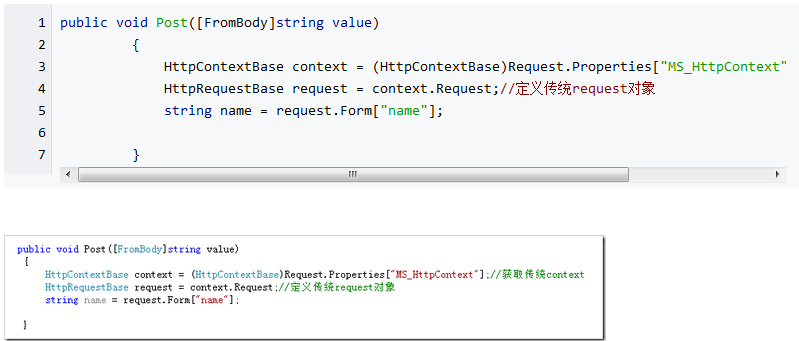
3.2 用get方式来获取数据

我直接在前台,用一个超链接,里面 指向我们的webapi 并且传递2个值,刚好是我们的UserInfo类的2个属性



3.4 最传统 通过 request.form 和 request.querystring 的方式的获取值

我们传统的在 aspx或者是 一般处理程序里面获取值是通过 request.querystring和request.form 来获取到.那么在 webapi里面,则是有些改变



可以通过上面这种强转的方式获取,也可以直接

var context = HttpContext.Current.Request;

WEBAPI中的Request是HttpRequestMessage类型，不能像Web传统那样有querystring和from 方法接收参数，而传统的HttpReqest的基类是HttpReqestBase. 所以这里我们就直接使用(HttpContextBase)Request.Properties[“MS\_HttpContext”]

# 四: 前台调用, ajax 来调用



# 五 C#后台调用

// GET提交 返回*字符串*

HttpClient hc = new HttpClient();

hc.BaseAddress = new Uri("http://localhost:5762/");//api所在的服务器

HttpResponseMessage hp= await hc.GetAsync("api/Values/"+id); //定位调用方法

string b= await hp.Content. ReadAsStringAsync ();//获取结果

注意：导入using System.Net.Http.Formatting

//GET提交 返回class

HttpClient hc = new HttpClient();

hc.BaseAddress = new Uri("http://localhost:5762/");//api所在的服务器

HttpResponseMessage hp= await hc.GetAsync("api/Values/"+id);//定位调用方法

Book b= await hp.Content.ReadAsAsync<Book>();//获取结果

//GET提交 返回List<class>

HttpClient hc = new HttpClient();

hc.BaseAddress = new Uri("http://localhost:5762/");//api所在的服务器

Task<HttpResponseMessage> ta = hc.GetAsync("api/Values");//定位调用方法

List<Book> list = ta.Result.Content.ReadAsAsync<List<Book>>().Result//获取结果

//put 提交 先创建一个和webapi对应的类

  HttpClient hc = new HttpClient();

hc.BaseAddress = new Uri("http://localhost:5762/");//api所在的服务器

HttpResponseMessage hp= await hc.PutAsJsonAsync("api/Values/", book); //定位调用方法

int result=await hp.Content.ReadAsAsync<int>();//获取结果

//post 提交 先创建一个和webapi对应的类

HttpClient hc = new HttpClient();

hc.BaseAddress = new Uri("http://localhost:5762/");//api所在的服务器

HttpResponseMessage hp=await hc.PostAsJsonAsync("api/Values",b); //定位调用方法

int i=await hp.Content.ReadAsAsync<int>();//获取结果

//delete 提交

HttpClient hc = new HttpClient();

hc.BaseAddress = new Uri("http://localhost:5762/");//api所在的服务器

HttpResponseMessage hp=await hc.DeleteAsync("api/values/" + id); //定位调用方法

int result=await hp.Content.ReadAsAsync<int>();//获取结果