

WebApi接口返回值不困惑：返回值类型详解 - 文章 - 伯乐在线



前言：已经有一个月没写点什么了，感觉心里空落落的。今天再来篇干货，想要学习Webapi的园友们速动起来，跟着博主一起来学习吧。作为程序猿，我们都知道参数和返回值是编程领域不可分割的两大块，此前分享了下WebApi的传参机制，今天再来看看WebApi里面另一个重要而又基础的知识点：返回值。

使用过Webapi的园友应该都知道，Webapi的接口返回值主要有四种类型

- void无返回值
- IHttpActionResult
- HttpResponseMessage
- 自定义类型

此篇就围绕这四块分别来看看它们的使用。

一、void无返回值

void关键字我们都不陌生，它申明方法没有返回值。它的使用也很简单，我们来看一个示例就能明白。

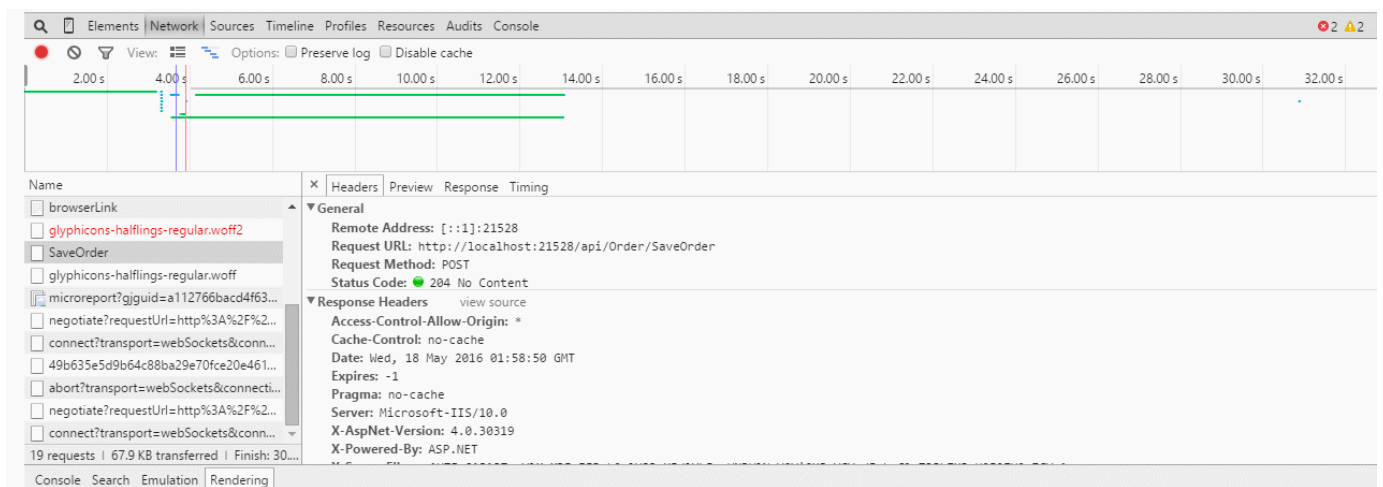
C#

```
public class ORDER
{
    public string ID { get; set; }
    public string NO { get; set; }
    public string NAME { get; set; }
    public string DESC { get; set; }
}

public class OrderController : ApiController
{
    [HttpPost]
    public void SaveOrder(ORDER name)
    {
        //处理业务逻辑
    }
}
```

```
$(function () {  
    $.ajax({  
        type: 'post',  
        url: 'http://localhost:21528/api/Order/SaveOrder',  
        data: { ID: "aaa", NAME: "test" },  
        success: function (data, status) {  
            alert(data);  
        }  
    });  
});
```

得到结果



可以看到，使用void声明的方法，在success方法里面得不到返回值，并且会返回http状态码204，告诉客户端此请求没有返回值。

二、 IHttpActionResult

IHttpActionResult类型是WebApi里面非常重要的一种返回值类型。下面博主就根据平时在项目里面使用最多的几种方式来讲解下这种类型的返回值的一些用法。

1、Json(T content)

使用MVC开发过的朋友一定记得，在MVC里面，请求数据的接口的返回值类型大部分使用的是 JsonResult，在MVC里面你一定也写过类似这样的接口：

C#

```
public JsonResult GetResult()  
{  
    return Json(new { }, JsonRequestBehavior.AllowGet);  
}
```

那么，在WebAPI里面是否也存在类似的用法呢。呵呵，在这点上面，微软总是贴心的。在WebApi的 ApiController这个抽象类里面，为我们封装了 Json(T content)这个方法，它的用法和MVC里面的 JsonResult基本类似。我们通过一个例子来说明它的用法：

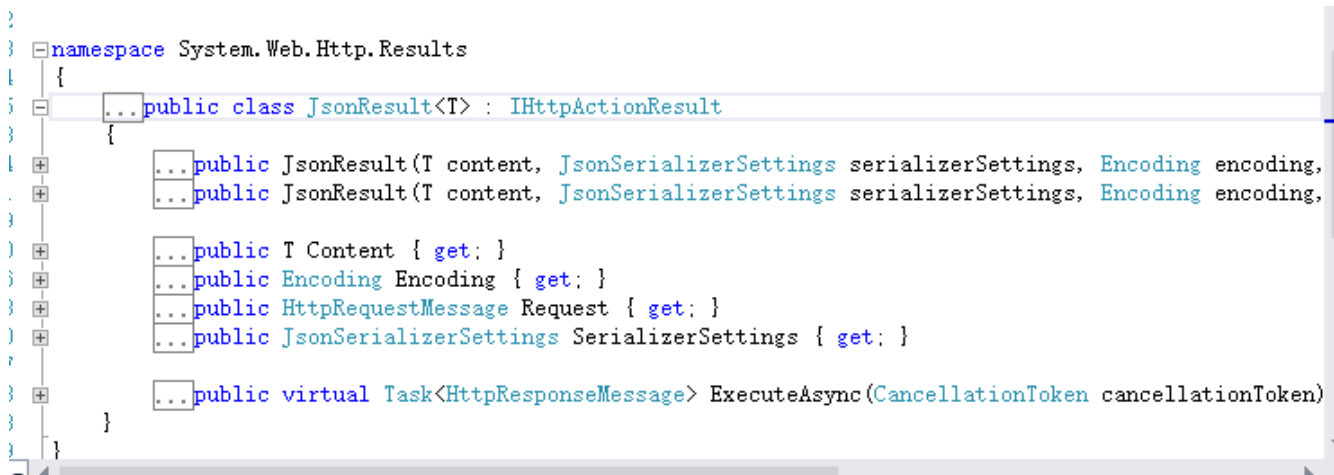
C#

```
[HttpGet]
public IHttpActionResult GetOrder()
{
    var lstRes = new List();
    //实际项目中，通过后台取到集合赋值给lstRes变量。这里只是测试。
    lstRes.Add(new ORDER() { ID = "aaaa", NO = "111", NAME = "111",
DESC = "1111" });
    lstRes.Add(new ORDER() { ID = "bbbb", NO = "222", NAME = "222",
DESC = "2222" });
    return Json>(lstRes);
}
```

看到这个代码，有人就疑惑了，我们定义的返回值类型是IHttpActionResult类型，直接返回Json(T content)这样可行么？我们将Json转到定义看看：

C#

我们继续将JsonResult转到定义



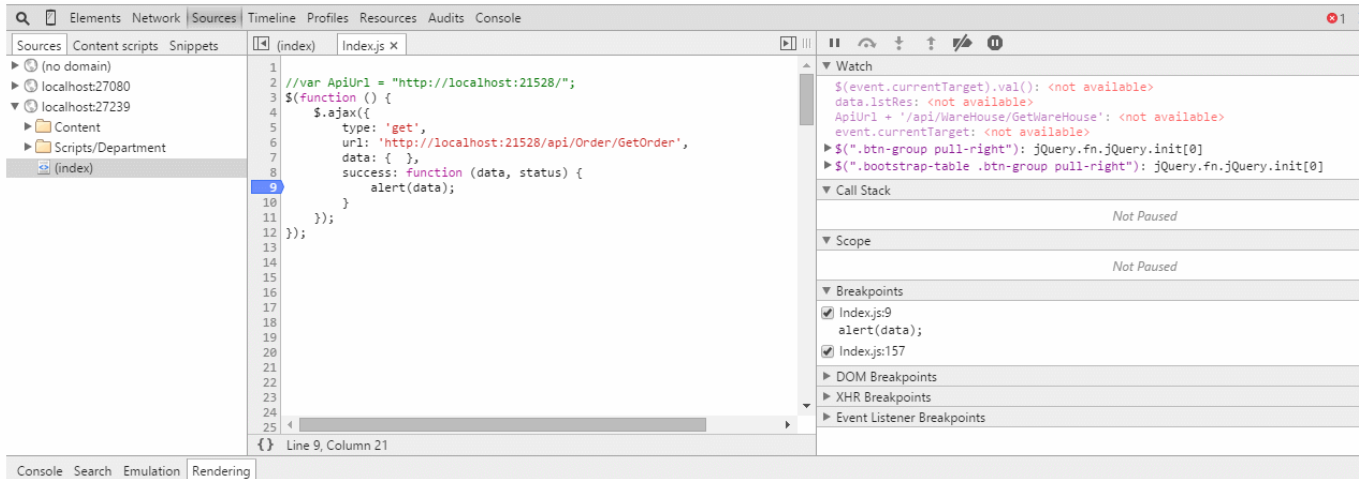
```
namespace System.Web.Http.Results
{
    public class JsonResult<T> : IHttpActionResult
    {
        public JsonResult(T content, JsonSerializerSettings serializerSettings, Encoding encoding,
        public JsonResult(T content, JsonSerializerSettings serializerSettings, Encoding encoding,
        public T Content { get; }
        public Encoding Encoding { get; }
        public HttpRequestMessage Request { get; }
        public JsonSerializerSettings SerializerSettings { get; }
        public virtual Task<HttpResponseMessage> ExecuteAsync(CancellationToken cancellationToken)
    }
}
```

原来JsonResult是实现了IHttpActionResult接口的，难怪可以直接返回呢。

知道了这个，我们直接在Web里面通过ajax请求来调用：

C#

来看结果：

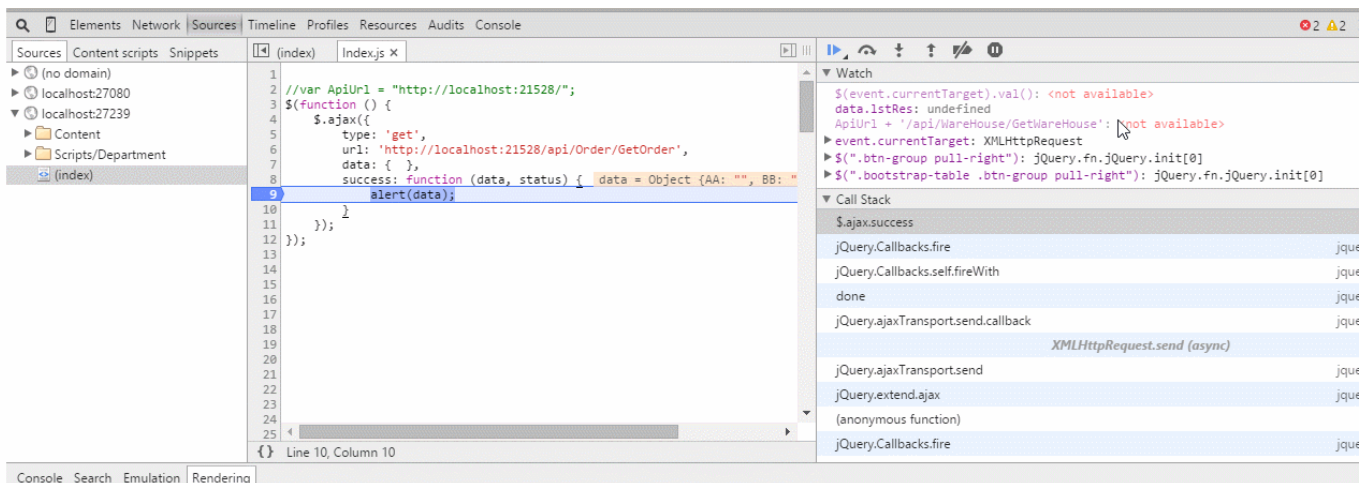


既然实体类可以直接这样传递，那么如果我们想要传递一些匿名类型呢，因为很多情况下，我们需要返回到前端的对象都没有对应的实体来对应，如果我们想要返回匿名对象怎么办呢？我们知道，这里的 `Json(T content)` 必须要传一个对应的泛型类型，如果是匿名类型这里肯定不好传。还好有我们的 `object` 类型，当然你可以使用 `dynamic`，我们来试一把。

C#

```
[HttpGet]
public IHttpActionResult GetOrder()
{
    return Json(dynamic>(new { AA = "", BB = "cc" }));
}
```

同样的来看测试结果：



2、Ok()、Ok(T content)

除了 `Json(T content)`，在 `ApiController` 里面还有另外一个比较常用的方法：`Ok()`。同样，我们将 `Ok()` 转到定义

C#

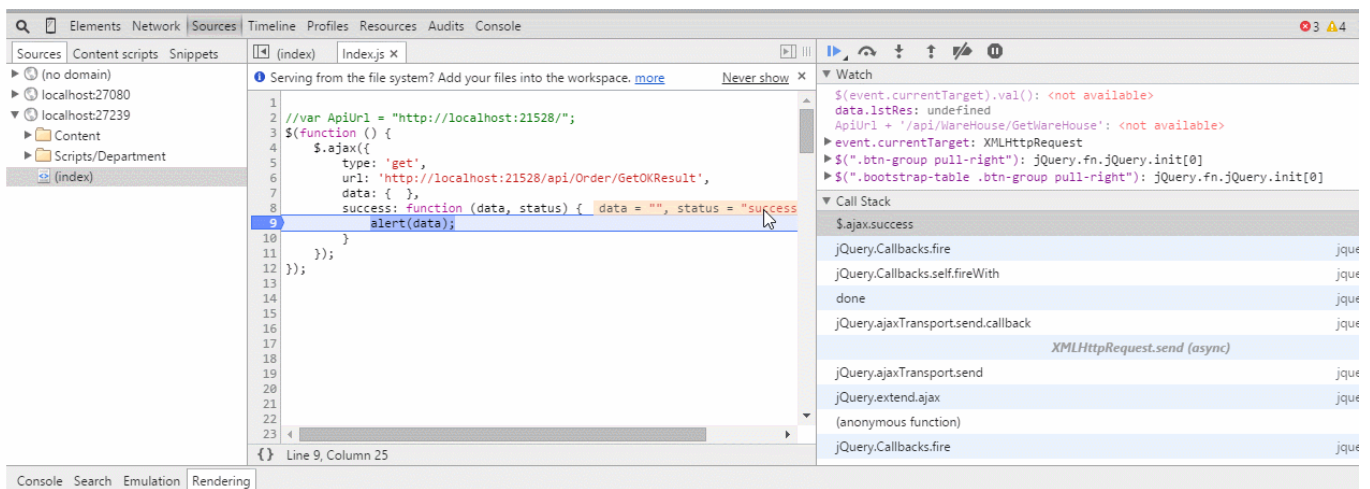
`OkResult` 转到定义

```
using System.Web.Http;  
namespace System.Web.Http.Results  
{  
    public class OkResult : IHttpActionResult  
    {  
        public OkResult(ApiController controller);  
        public OkResult(HttpRequestMessage request);  
        public HttpRequestMessage Request { get; }  
        public virtual Task<HttpResponseMessage> ExecuteAsync(CancellationToken cancellationToken)  
    }  
}
```

有了这个作为基础，我们就可以放心大胆的使用了。

C#

得到结果

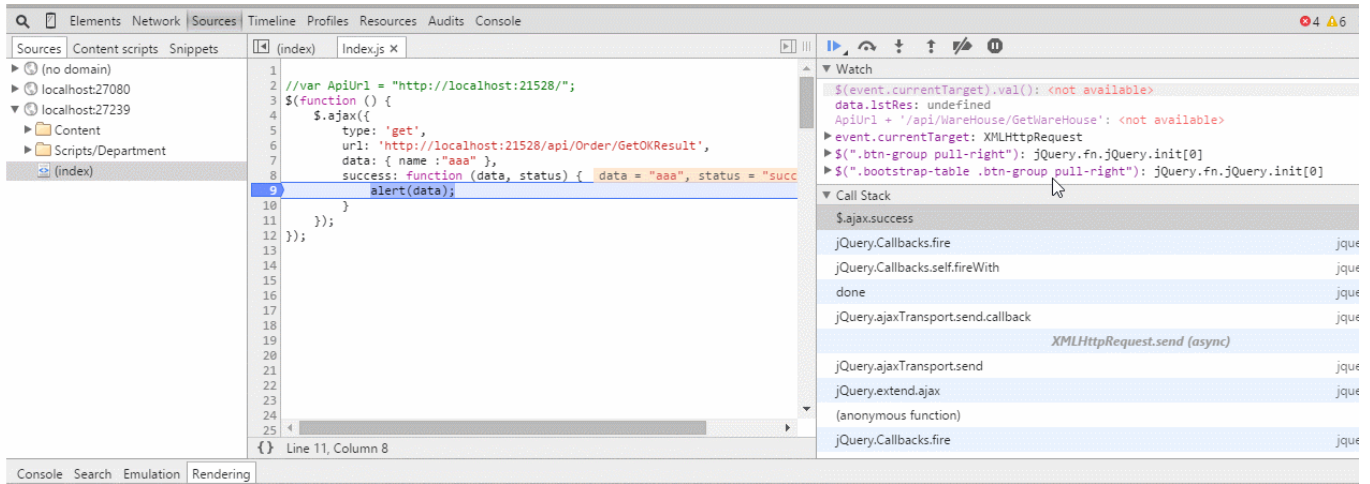


如果返回Ok(), 就表示不向客户端返回任何信息, 只告诉客户端请求成功。

除了Ok() 之外, 还有另外一个重载Ok(T content)。

C#

```
[HttpGet]  
public IHttpActionResult GetOKResult(string name)  
{  
    return Okstring>(name);  
}
```



这种用法和Json(T content)比较类似，如果你非要问这两者有什么区别，或者说怎么选择两者。那么我的理解是如果是返回实体或者实体集合，建议使用Json(T content)，如果是返回基础类型（如int、string等），使用Ok(T content)。

3、NotFound()

当需要向客户端返回找不到记录时，有时需要用到NotFound()方法。

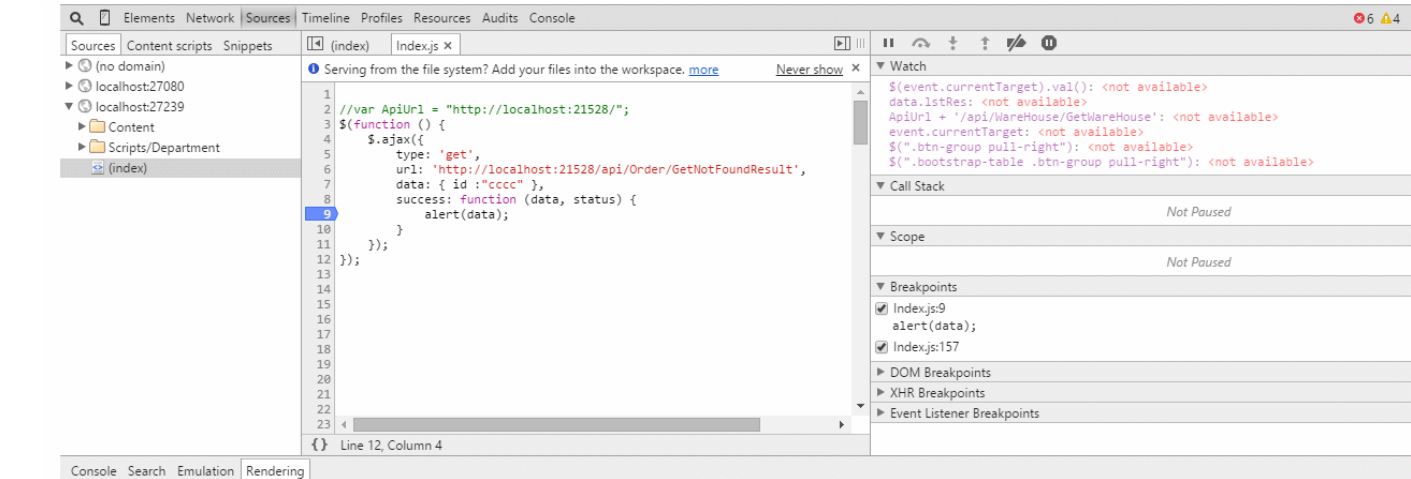
C#

来看看它的使用场景

C#

```
[HttpGet]
public IHttpActionResult GetNotFoundResult(string id)
{
    var lstRes = new List();
    //实际项目中，通过后台取到集合赋值给lstRes变量。这里只是测试。
    lstRes.Add(new ORDER() { ID = "aaaa", NO = "111", NAME = "111",
DESC = "1111" });
    lstRes.Add(new ORDER() { ID = "bbbb", NO = "222", NAME = "222",
DESC = "2222" });
    var oFind = lstRes.FirstOrDefault(x => x.ID == id) ;
    if (oFind == null)
    {
        return NotFound();
    }
    else
    {
        return Json(oFind);
    }
}
```

得到结果



NotFound() 方法会返回一个404的错误到客户端。

4、其他

其他还有一些方法，都有它特定的用途。在此贴出来。

```
[HttpGet]
public IHttpActionResult GetContentResult()
{
    return Contentstring>(HttpStatusCode.OK, "OK");
}
```

向客户端返回值和http状态码。

```
[HttpGet]
public IHttpActionResult GetBadRequest(ORDER order)
{
    if (string.IsNullOrEmpty(order.ID))
        return BadRequest();
    return Ok();
}
```

向客户端返回400的http错误。

将请求重定向到其他地方。

5、自定义IHttpActionResult接口的实现

上面介绍了一些系统内置的常用的实现IHttpActionResult接口的方法。如果我们需要自定义IHttpActionResult的返回呢？

在介绍之前，我们有必要先来看看IHttpActionResult类型的定义，将IHttpActionResult转到定义可以看到：

C#

```
namespace System.Web.Http
{
    // 摘要:
    //      Defines a command that asynchronously creates an
    System.Net.Http.HttpResponseMessage.
    public interface IHttpActionResult
    {
        // 摘要:
        //      Creates an System.Net.Http.HttpResponseMessage asynchronously.
        //
        // 参数:
        //      cancellationToken:
        //          The token to monitor for cancellation requests.
        //
        // 返回结果:
        //      A task that, when completed, contains the
    System.Net.Http.HttpResponseMessage.
        Task ExecuteAsync(CancellationToken cancellationToken);
    }
}
```

这个接口包含唯一的一个方法ExecuteAsync(), 此方法将以异步方式创建一个HttpResponseMessage实例返回给客户端。

有了这个作为基础, 下面, 我们自定义一个bootstrapTable服务端分页的子类去展示自定义IHttpActionResult的用法。

首先, 自定义一个实现类

C#

```
public class PageResult : IHttpActionResult
{
    object _value;
    HttpRequestMessage _request;
    public PageResult(object value, HttpRequestMessage request)
    {
        _value = value;
        _request = request;
    }
    public Task ExecuteAsync(System.Threading.CancellationToken
cancellationToken)
    {
        var response = new HttpResponseMessage()
        {
```



```

        Content = new ObjectContent(typeof(object), _value, new
JsonMediaTypeFormatter()),

        RequestMessage = _request
    };

    return Task.FromResult(response);
}
}

```

C#

```

[HttpGet]

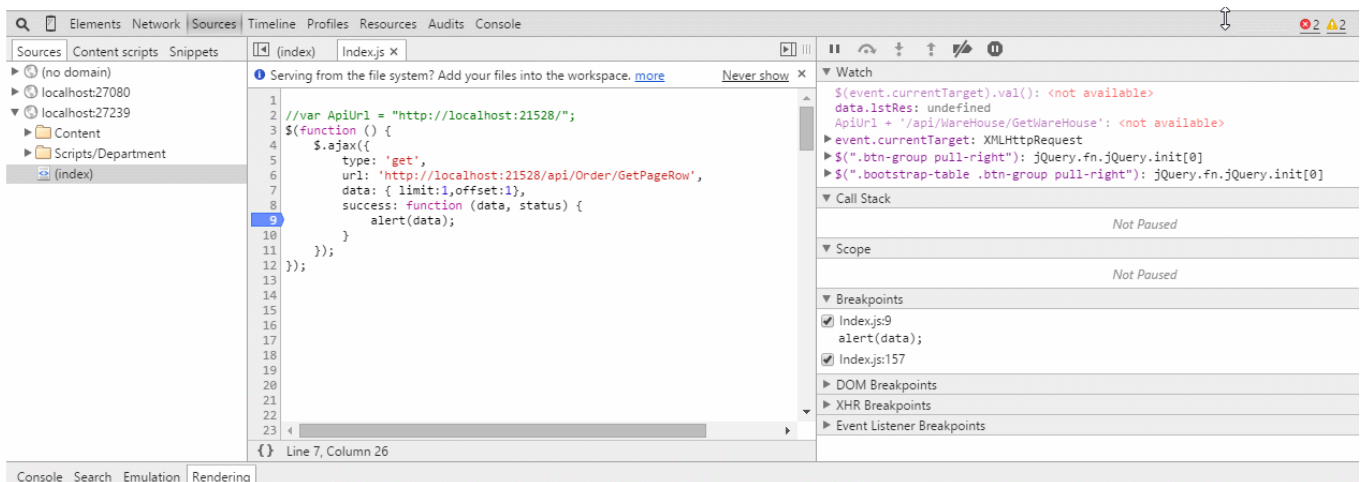
public IHttpActionResult GetPageRow(int limit, int offset)
{
    var lstRes = new List();
    //实际项目中, 通过后台取到集合赋值给lstRes变量。这里只是测试。
    lstRes.Add(new ORDER() { ID = "aaaa", NO = "111", NAME = "111",
DESC = "1111" });
    lstRes.Add(new ORDER() { ID = "bbbb", NO = "222", NAME = "222",
DESC = "2222" });

    var oData = new { total = lstRes.Count, rows =
lstRes.Skip(offset).Take(limit).ToList() };
    return new PageResult(oData, Request);
}

$(function () {
    $.ajax({
        type: 'get',
        url: 'http://localhost:21528/api/Order/GetPageRow',
        data: { limit:1,offset:1},
        success: function (data, status) {
            alert(data);
        }
    });
});

```

得到结果



三、HttpResponseMessage

在上文自定义IHttpRequestResult返回类型的时候，提到过HttpResponseMessage这个对象。它表示向客户端返回一个http响应的消息对象（包含http状态码和需要返回客户端的消息）。这个对象也有它独特的使用场景：需要向客户端返回HttpResponse时就要用到这个对象。以导出为例，由于需要将导出的Excel文件输出到客户端浏览器，Webapi的服务端需要向Web的客户端输出文件流，这个时候一般的IHttpRequestResult对象不方便解决这个问题，于是HttpReponseMessage派上了用场。我们来看看它的使用示例。

```
C#

public HttpResponseMessage Export()
{
    //取数据
    var lstRes = OrderBLL.Export();
    //向Excel里面填充数据
    HSSFWorkbook workbook = new HSSFWorkbook();
    CreateAndFillSheet(workbook, lstRes);
    //保存到服务
    var fileName = "Excel" + DateTime.Now.ToString("yyyyMMddHHmmss") +
        ".xls";

    var strPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
        @"Data" + fileName);

    using (FileStream fs = new FileStream(strPath, FileMode.Create))
    {
        workbook.Write(fs);
        using (MemoryStream ms = new MemoryStream())
        {
            workbook.Write(ms);
        }
    }
    //输出到浏览器
    try
    {
        var stream = new FileStream(strPath, FileMode.Open);
        HttpResponseMessage response = new
        HttpResponseMessage(HttpStatusCode.OK);
        response.Content = new StreamContent(stream);
        response.Content.Headers.ContentType = new
        MediaTypeHeaderValue("application/octet-stream");
        response.Content.Headers.ContentDisposition = new
        ContentDispositionHeaderValue("attachment")
    {
```

```
        FileName = fileName
    };
    return response;
}
catch
{
    return new HttpResponseMessage(HttpStatusCode.NoContent);
}
```

将文件流保存在StreamContent对象里面，然后输出到浏览器。在浏览器端即可将Excel输出。

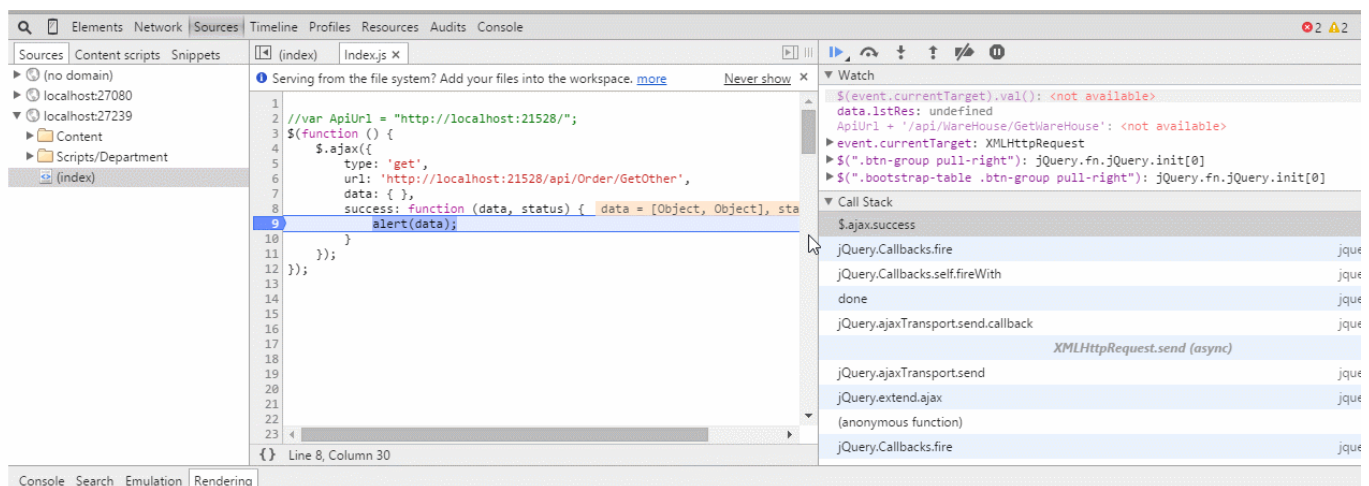
四、自定义类型

以上几种返回值类型能解决我们大部分返回值的问题，当然，你也可以将webapi的接口和普通方法一样，返回任意的类型，WebApi会自动序列化你自定义任何返回类型，然后将序列化的值写到响应正文里，状态码统一返回200。比如：

C#

```
[HttpGet]
public object GetOther()
{
    var lstRes = new List();
    //实际项目中，通过后台取到集合赋值给lstRes变量。这里只是测试。
    lstRes.Add(new ORDER() { ID = "aaaa", NO = "111", NAME = "111",
DESC = "1111" });
    lstRes.Add(new ORDER() { ID = "bbbb", NO = "222", NAME = "222",
DESC = "2222" });
    return lstRes;
}
```

得到结果



和上面的Json、Ok等用法在效果上面没有太大区别。

五、总结

以上通过四个方面详细分享了下WebApi里面返回值的常见用法，不能说哪种方式最好，因为每种方式都有其特定的使用场景。博主觉得为了规范WebApi接口，对于一般接口的返回值，尽量使用 IHttpActionResult 类型作为返回值，毕竟是微软内置的东西，可能为我们考虑了很多我们考虑不到的东西。当然，你可能会觉得麻烦，你可能会说直接和普通方法一样来使用不是更爽，博主当初也有这种想法，可是学习微软的东西多了之后发现很多东西还是遵守一定的标准比较好，至少维护起来方便。这就像博主最近正在努力学习的WebApi+oData一样，为什么要搞这么一套标准性的东西，还不是为了更加方便地规范Restful风格。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！

加入伯乐在线专栏作者。扩大知名度，还能得赞赏！详见《[招募专栏作者](#)》

1 赞 收藏 [评论](#)



合作联系

Email: bd@jobbole.com

QQ: 2302462408 （加好友请注明来意）

更多频道

- [小组](#) - 好的话题、有启发的回复、值得信赖的圈子
- [头条](#) - 分享和发现有价值的内容与观点
- [相亲](#) - 为IT单身男女服务的征婚传播平台
- [资源](#) - 优秀的工具资源导航
- [翻译](#) - 翻译传播优秀的外文文章
- [文章](#) - 国内外的精选文章
- [设计](#) - UI, 网页, 交互和用户体验
- [iOS](#) - 专注iOS技术分享
- [安卓](#) - 专注Android技术分享
- [前端](#) - JavaScript, HTML5, CSS
- [Java](#) - 专注Java技术分享
- [Python](#) - 专注Python技术分享