**[C#进阶系列——WebApi 异常处理解决方案](http://www.cnblogs.com/landeanfen/p/5363846.html)**

**阅读目录**

* [一、使用异常筛选器捕获所有异常](http://www.cnblogs.com/landeanfen/p/5363846.html#_label0)
* [二、HttpResponseException自定义异常信息](http://www.cnblogs.com/landeanfen/p/5363846.html#_label1)
* [三、返回HttpError](http://www.cnblogs.com/landeanfen/p/5363846.html#_label2)
* [四、总结](http://www.cnblogs.com/landeanfen/p/5363846.html#_label3)

**正文**

前言：上篇[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)介绍了WebApi参数的传递，这篇来看看WebApi里面异常的处理。关于异常处理，作为程序员的我们肯定不陌生，记得在介绍 [AOP](http://www.cnblogs.com/landeanfen/p/4782370.html) 的时候，我们讲过通过AOP可以统一截获异常。那么在我们的WebApi里面一般是怎么处理异常的呢，今天这一篇，博主带着大家一起来实践下WebApi的异常处理。

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
* C#进阶系列——WebApi区域Area使用小结

为什么说是实践？因为在[http://www.asp.net](http://www.asp.net/)里面已经明确给出WebApi的异常处理机制。光有理论还不够，今天我们还是来试一把。通过实践，我们可能发现一些更详尽的用法。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5363846.html#_labelTop)

# 一、使用异常筛选器捕获所有异常

我们知道，一般情况下，WebApi作为服务使用，每次客户端发送http请求到我们的WebApi服务里面，服务端得到结果输出response到客户端。这个过程中，一旦服务端发生异常，会统一向客户端返回500的错误。

[HttpGet]

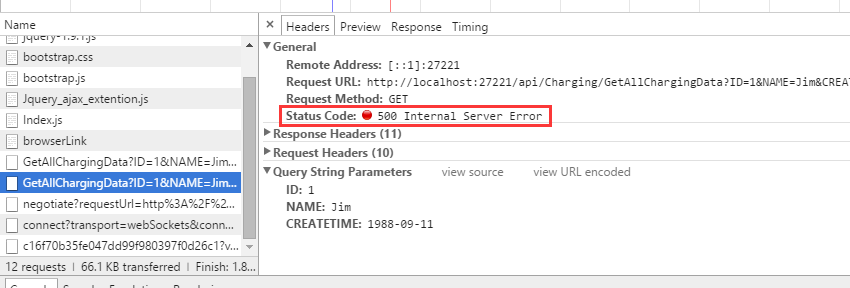
public string GetAllChargingData([FromUri]TB\_CHARGING obj)

{

throw new NotImplementedException("方法不被支持");

}

我们来看看http请求



而有些时候，我们客户端需要得到更加精确的错误码来判断异常类型，怎么办呢？

记得在介绍AOP的时候，我们介绍过MVC里面的IExceptionFilter接口，这个接口用于定义异常筛选器所需的方法，在WebApi里面，也有这么一个异常筛选器，下面我们通过一个实例来看看具体如何实现。

首先在App\_Start里面新建一个类WebApiExceptionFilterAttribute.cs，继承ExceptionFilterAttribute，重写OnException方法

[复制代码](javascript:void(0);)

public class WebApiExceptionFilterAttribute : ExceptionFilterAttribute

{

//重写基类的异常处理方法

public override void OnException(HttpActionExecutedContext actionExecutedContext)

{

//1.异常日志记录（正式项目里面一般是用log4net记录异常日志）

Console.WriteLine(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss") + "——" +

actionExecutedContext.Exception.GetType().ToString() + "：" + actionExecutedContext.Exception.Message + "——堆栈信息：" +

actionExecutedContext.Exception.StackTrace);

//2.返回调用方具体的异常信息

if (actionExecutedContext.Exception is NotImplementedException)

{

actionExecutedContext.Response = new HttpResponseMessage(HttpStatusCode.NotImplemented);

}

else if (actionExecutedContext.Exception is TimeoutException)

{

actionExecutedContext.Response = new HttpResponseMessage(HttpStatusCode.RequestTimeout);

}

//.....这里可以根据项目需要返回到客户端特定的状态码。如果找不到相应的异常，统一返回服务端错误500

else

{

actionExecutedContext.Response = new HttpResponseMessage(HttpStatusCode.InternalServerError);

}

base.OnException(actionExecutedContext);

}

}

[复制代码](javascript:void(0);)

 代码解析：**通过判断异常的具体类型，向客户端返回不同的http状态码，示例里面写了两个，可以根据项目的实际情况加一些特定的我们想要捕获的异常，然后将对应的状态码写入http请求的response里面，对于一些我们无法判断类型的异常，统一返回服务端错误500。**关于http的状态码，framework里面定义了一些常见的类型，我们大概看看：

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif Http状态码

定义好了异常处理方法，剩下的就是如何使用了。可以根据实际情况，在不同级别使用统一的异常处理机制。

### 1、接口级别

[WebApiExceptionFilter]

[HttpGet]

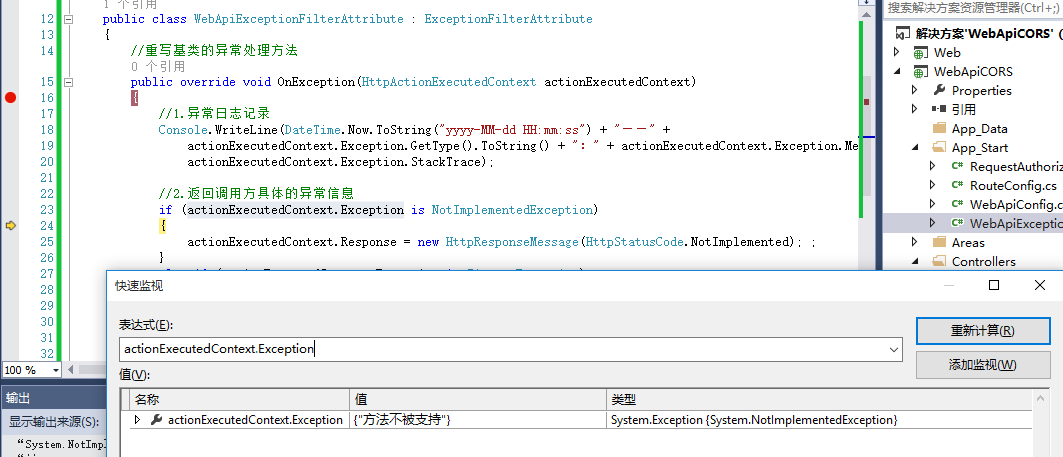
public string GetAllChargingData([FromUri]TB\_CHARGING obj)

{

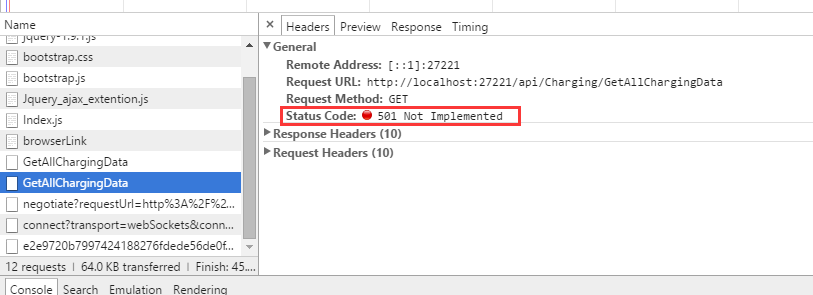
throw new NotImplementedException("方法不被支持");

}

执行到异常后，会先进到OnException方法：



执行完成之后浏览器查看：



**如果需要，甚至可以向Status Code里面写入自定义的描述信息，并且还可以向我们的Response的Content里面写入我们想要的信息**。我们稍微改下OnException方法：

[复制代码](javascript:void(0);)

　　　　　　　if (actionExecutedContext.Exception is NotImplementedException)

{

var oResponse = new HttpResponseMessage(HttpStatusCode.NotImplemented);

oResponse.Content = new StringContent("方法不被支持");

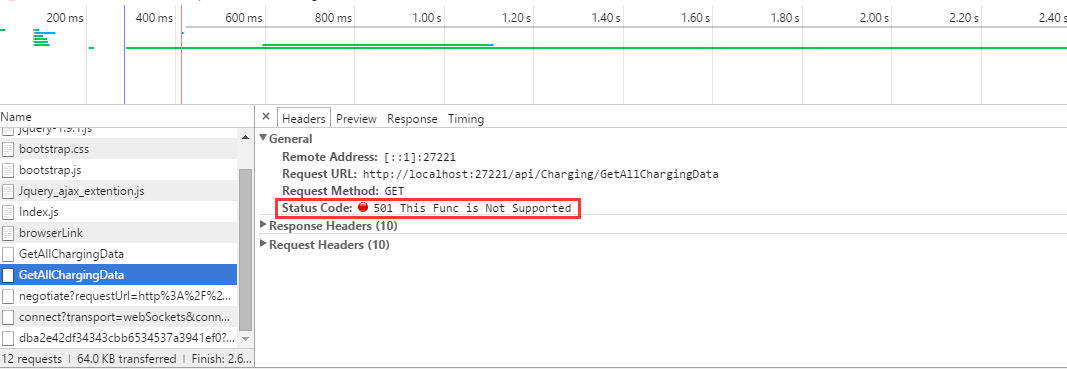
oResponse.ReasonPhrase = "This Func is Not Supported";

actionExecutedContext.Response = oResponse;

}

[复制代码](javascript:void(0);)

看看ReasonPhrase描述信息



看看Response的描述信息



### 2、控制器级别

如果想要某一个或者多个控制器里面的所有接口都使用异常过滤，直接在控制器上面标注特性即可。

* 某一个控制器上面启用异常过滤

[复制代码](javascript:void(0);)

　　[WebApiExceptionFilter]

public class ChargingController : BaseApiController

{

#region Get

[HttpGet]

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

{

throw new NotImplementedException("方法不被支持");

}

}

[复制代码](javascript:void(0);)

* 多个控制器上面同时启用异常过滤

[WebApiExceptionFilter]

public class BaseApiController : ApiController

{

}

[复制代码](javascript:void(0);)

public class ChargingController : BaseApiController

{

#region Get

[HttpGet]

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

{

throw new NotImplementedException("方法不被支持");

}

}

[复制代码](javascript:void(0);)

这样，所有继承BaseApiController的子类都会启用异常过滤。

### 3、全局配置

如果需要对整个应用程序都启用异常过滤，则需要做如下两步：

　　1、在Global.asax全局配置里面添加 GlobalConfiguration.Configuration.Filters.Add(new WebApiExceptionFilterAttribute()); 这一句，如下：

[复制代码](javascript:void(0);)

void Application\_Start(object sender, EventArgs e)

{

// 在应用程序启动时运行的代码

AreaRegistration.RegisterAllAreas();

GlobalConfiguration.Configure(WebApiConfig.Register);

RouteConfig.RegisterRoutes(RouteTable.Routes);

GlobalConfiguration.Configuration.Filters.Add(new WebApiExceptionFilterAttribute());

}

[复制代码](javascript:void(0);)

　　2、在WebApiConfig.cs文件的Register方法里面添加  config.Filters.Add(new WebApiExceptionFilterAttribute()); 这一句，如下：

[复制代码](javascript:void(0);)

public static void Register(HttpConfiguration config)

{

//跨域配置

config.EnableCors(new EnableCorsAttribute("\*", "\*", "\*"));

// Web API 路由

config.MapHttpAttributeRoutes();

RouteTable.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{action}/{id}",

defaults: new { id = RouteParameter.Optional }

).RouteHandler = new SessionControllerRouteHandler();

config.Filters.Add(new WebApiExceptionFilterAttribute());

}

[复制代码](javascript:void(0);)

[回到顶部](http://www.cnblogs.com/landeanfen/p/5363846.html#_labelTop)

# 二、HttpResponseException自定义异常信息

上面说的是全局的异常捕获以及处理方式，在某些情况下，我们希望以异常的方式向客户端发送相关信息，可能就需要用到我们的HttpResponseException。比如：

[复制代码](javascript:void(0);)

[HttpGet]

public TB\_CHARGING GetById(string id)

{

//从后台查询实体

var oModel = server.Find(id);

if (oModel == null)

{

var resp = new HttpResponseMessage(HttpStatusCode.NotFound)

{

Content = new StringContent(string.Format("没有找到id={0}的对象", id)),

ReasonPhrase = "object is not found"

};

throw new HttpResponseException(resp);

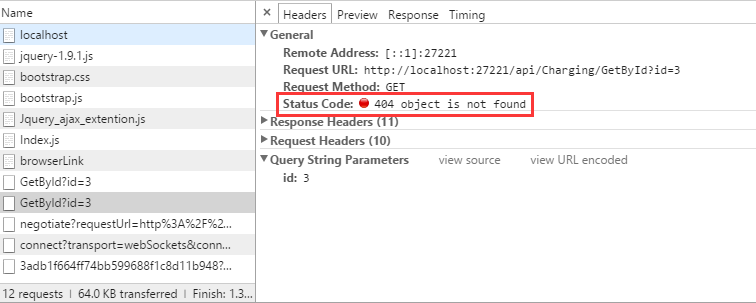
}

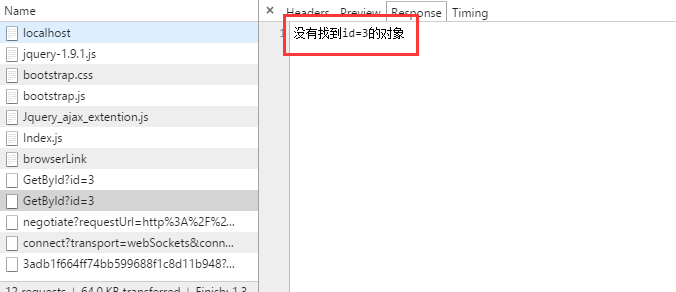
return oModel;

}

[复制代码](javascript:void(0);)

执行之后浏览器里面查看结果：





代码释疑：**细心的朋友可能，发现了，这里既使用了HttpResponseMessage，又使用了HttpResponseException，那么，像这种可控的异常，我们是否可以直接以HttpResponseMessage的形式直接返回到客户端而不用抛出异常呢？这里就要谈谈这两个对象的区别了，博主的理解是HttpResonseMessage对象用来响应讯息并包含状态码及数据内容，HttpResponseException对象用来向客户端返回包含错误讯息的异常。**

**在网上看到一篇**[**文章**](http://www.it165.net/pro/html/201305/6019.html)**这样描述两者的区别：当呼叫 Web API 服务时发生了与预期上不同的错误时，理当应该中止程序返回错误讯息，这时对于错误的返回就该使用 HttpResponseException，而使用 HttpResponseMessage 则是代表着当客户端发送了一个工作请求而 Web API 正确的完成了这个工作，就能够使用 HttpResponseMessage 返回一个 201 的讯息，所以 HttpResponseMessage 与 HttpResponseException 在使用上根本的目标就是不同的，用 HttpResponseMessage 去返回一个例外错误也会让程序结构难以辨别且不够清晰。**

[回到顶部](http://www.cnblogs.com/landeanfen/p/5363846.html#_labelTop)

# 三、返回HttpError

HttpError对象提供一致的方法来响应正文中返回错误的信息。准确来说，HttpError并不是一个异常，只是用来包装错误信息的一个对象。**其实在某一定的程度上，HttpError和HttpResponseMessage使用比较相似，二者都可以向客户端返回http状态码和错误讯息，并且都可以包含在HttpResponseException对象中发回到客户端。但是，一般情况下，HttpError只有在向客户端返回错误讯息的时候才会使用，而HttpResponseMessage对象既可以返回错误讯息，也可返回请求正确的消息**。其实关于HttpError没什么特别好讲的，我们来看一个例子就能明白：

[复制代码](javascript:void(0);)

　　　　public HttpResponseMessage Update(dynamic obj)

{

TB\_Product oModel = null;

try

{

var id = Convert.ToString(obj.id);

oModel = Newtonsoft.Json.JsonConvert.DeserializeObject<TB\_Product>(Convert.ToString(obj.dataModel));

//...复杂的业务逻辑

}

catch(Exception ex)

{

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex.Message);

}

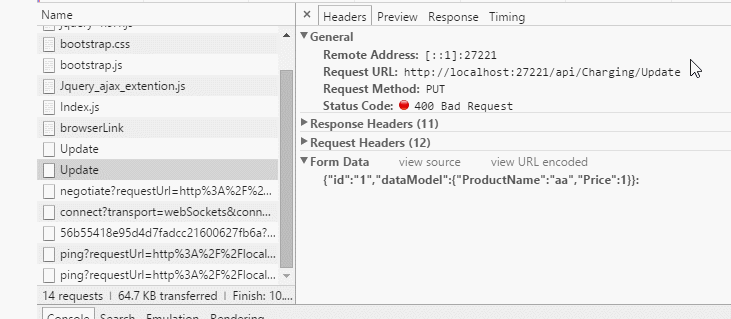
return Request.CreateResponse<TB\_Product>(HttpStatusCode.OK, oModel);

}

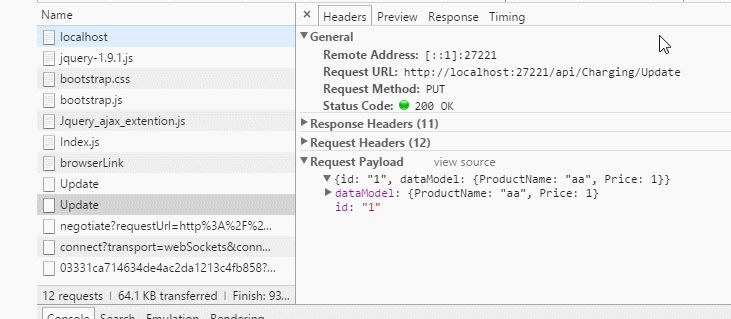
[复制代码](javascript:void(0);)

假如现在在执行try里面复杂业务逻辑的时候发生了异常，我们捕获到了异常然后向客户端返回HttpError对象，这个对象里面包含我们自定义的错误讯息，如果正常则返回HttpResponseMessage对象。

如果请求异常：



如果请求正常



[回到顶部](http://www.cnblogs.com/landeanfen/p/5363846.html#_labelTop)

# 四、总结

 以上三种异常的处理方法，可以根据不同的场景选择使用。

* 如果项目对异常处理要求并不高，只需要记录好异常日志即可，那么使用异常筛选器就能够搞定
* 如果项目需要对不同的异常，客户端做不同的处理。而这个时候使用异常筛选器不能详尽所有的异常，可能使用HttpResponseException对象是更好的选择，定义更加精细的异常和异常描述。
* 对于何时使用HttpError，又何时使用HttpResponseMessage，可以参考上文三里面用法。
* 当然实际项目中很可能以上两种或者三种同时使用。

上文通过一些简单的示例介绍了下WebApi里面异常的处理机制，可能不够深入，但对于一般项目的异常处理基本够用。其实有一点博主还没有想明白，对于构造函数里面的异常该如何统一捕获呢？通过异常筛选器是捕获不到的，不知道园友们有没有什么更好的办法，不吝赐教，感谢感谢！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！