What is the difference between HttpResponseMessage and HttpResponseException

Ask Question

I tried to understand both and write sample code:

```
public HttpResponseMessage Get()
{
    var response = ControllerContext.Request
                        .CreateResponse(HttpStatusCode.BadRequest, "abc");

    throw new HttpResponseException(response);
}
```

And:

```
public HttpResponseMessage Get()
{
    return ControllerContext.Request
                       .CreateResponse(HttpStatusCode.BadRequest, "abc");
}
```

From Fiddle, I really didn't see any differences between them, so what is the purpose of using `HttpResponseException` ?

c# asp.net-web-api

edited Mar 22 '13 at 4:37

asked May 18 '12 at 22:56
Cuong Le
52k 21 106 159

possible duplicate of Throw HttpResponseException or return Request.CreateErrorResponse? – Robert MacLean Mar 8 '15 at 21:36

5 Answers

The main difference between the two is this. The exception is useful to immediately stop processing and exit. For example assume I have the following code

```
public class CustomerController : ApiController {
  private ICustomerContext repo;

  public CustomerController(ICustomerContext repo) {
    this.repo = repo;
  }

  public Customer Get(int id) {
    var customer = repo.Customers.SingleOrDefault(c=>c.CustomerID == id);
    if (customer == null) {
      throw new HttpResponseException(new HttpResponseMessage(HttpStatusCode.NotFound));
    }
    return customer;
  }
}
```

If this code runs and I pass an id that is not present, it will immediately stop processing and return a status code of 404.

If instead I return HttpResponseMessage, the request will happily continue the rest of its processing and return a 404. The main difference being end the request or not.

As Darrel said the exception is useful in cases where in some cases I want processing to continue (as in when customer is found) and in others I don't.

The place where you might want to use something like HttpResponseMessage is in an Http POST to return a status code of 201 and set the location header. In that case I do want processing to continue. That would would do with this code.*

```
   public HttpResponseMessage Post(Customer customer) {
     repo.Add(customer);
     repo.SaveChanges();
     var response = Request.CreateResponse(HttpStatusCode.Created, customer);
     response.Headers.Location = new Uri(Request.RequestUri, string.format("customer/{0}",
 customer.id));
     return response;
   }
 }
```

*note: If you are using the beta bits you would create a new HttpResponseMessage. I am using the later bits however which require you to use the CreateResponse extension method off of the Request.

Above, I am creating a response which sets the status code to 201, passes in the customer, and then sets the location header.

The response is then returned and the request continues processing.

Hope this helps

edited Mar 1 '16 at 19:01
Jon Davis
4,531 3 30 56

answered May 24 '12 at 9:18
Glenn Block
8,183 1 26 32

3 I would add that exception-handling/logging is another reason. You may want to throw to log a 404 but not throw and not log a 412. – Luke Puplett Jul 4 '12 at 12:25

5 I'm really confused by what you mean with "the request continues processing". Isn't the request finished the moment you `return` ? – Stijn May 22 '13 at 14:32

1 @Stijn - there's some comments on Glenn's blog (goo.gl/ErSG9h) that make it a little bit clearer. It sounds like Glenn is referring to processing that Web.Api does after the controller returns such as Message Handlers. However, Steven Solomon questions whether that's the case or not. – Iain Aug 1 '13 at 9:58

@Iain thanks for the link, it's an interesting comment. It's not that I don't believe Glenn, but it'd be great if he could demonstrate the difference. – Stijn Aug 1 '13 at 10:12

1 @GlennBlock I am not sure this is the right answer. This answer seems to differ – Richard Tozier Nov 8 '13 at 16:35

HttpResponseException is useful when your Controller Action signature looks like

```
  Foo Get(int id)
```

In this case, you cannot easily return status code like 400.

Be aware that `HttpResponseMessage<T>` is going away in the next release of Web API.

edited May 24 '12 at 13:49

answered May 19 '12 at 3:41
Darrel Miller
106k 27 158 220

Thanks for your answer, it makes sense to me – Cuong Le May 22 '12 at 3:04

4 Only the generic HttpResponseMessage<T> is going away – ozba May 24 '12 at 12:17

@ozba Thanks, I hadn't noticed that the markup syntax prevented the angle brackets from appearing. – Darrel Miller May 24 '12 at 13:49

This is the real correct answer. Useful. – GazTheDestroyer Oct 29 '12 at 12:17

This is also the case referred to in `Professional ASP.Net MVC 4` , wrox.com/WileyCDA/WroxTitle/…. When your API is based around returning domain types like `Foo` , you can't return an `HttpResponseMessage` in the case of an error but you can return/throw `HttpResponseException` and thus provide status codes etc to the consumer. Why would you want to return domain types? Well in contrast to some other answers here, they say it simplifies UT dealing with domain objects rather than low level framework objects like `HttpResponseMessage` . – rism Dec 1 '15 at 3:20

Assuming you want to unit test the responses, doesn't it make sense to always return an

In a non-Web API class that fetched a Customer, you'd likely return null, with your calling code checking for a null response:

```
public Customer GetCustomer(int id)
{
    return db.Customers.Find(id);
}
```

But in Web API, you're not going to return null, you have to return something, even if that something is created after you throw an HttpResponseException. In that case, to ease testing, why not just always return an HttpResponseMessage, and make that your signature?

```
public HttpResponseMessage GetCustomer(int id)
{
    var customer = db.Customers.Find(id);
    if (customer == null)
    {
        return Request.CreateResponse(HttpStatusCode.NotFound);
    }

    return Request.CreateResponse(HttpStatusCode.OK, customer);
}
```

answered Jun 7 '12 at 14:52
Brian Vallelunga
5,881 8 43 65

1 I agree, Brian. This also makes for a better learning path for returning objects and status codes. When the template sets a return type as a business object, it makes you wonder what the proper way is to deal with other responses. – Luke Puplett Jul 4 '12 at 12:19

5 @Brian it is a style thing. Some people want to get knee deep in HTTP and feel like it is part of their application design. Others feel HTTP is an implementation detail and would rather abstract away the HTTPness using ActionFilters and HttpMessageHandlers. Neither approach is wrong if you know what you are doing. I tend to take your approach because I feel hiding the HTTP makes things look too much like RPC. – Darrel Miller Oct 29 '12 at 13:04

This is the way I naturally fell into doing it. It called WebAPI - so to me it makes perfect scene that it should always return a valid HTTP Response - and that response tells the caller what happened. – Morvael Oct 20 '17 at 15:19

`HttpResponseException` derives from `Exception` and embeds `HttpResponseMessage`. Since it derives from `Exception` it can be useful in `try` - `catch` scenarios.

Default status code returned by `HttpResponseException` is `HttpStatusCode.InternalServerError`.

edited Jun 26 '12 at 13:05
EBarr
8,855 5 48 77

answered May 19 '12 at 3:27
labilbe
2,421 2 17 30

As the original questions states, there is no real difference in the returned Response.

The real purpose of HttpResponseException is to allow sub methods to create and 'throw' their own HttpResponseMessages that flow back to the up the call stack and are returned to the client.

```
public class CustomerController : ApiController {
  private ICustomerContext repo;
  public CustomerController(ICustomerContext repo) {
    this.repo = repo;
  }

  public HttpResponseMessage Get(int id) {

    Customer customer = getCustomer(id);

    return Request.CreateResponse(customer);
  }

  private Customer getCustomer(int id){
    .....do some work
    .....we have a problem so throw exception
    throw new HttpResponseException(Request.CreateResponse(HttpStatusCode.BadRequest, "Id
out of range");
    return repo.Customers.SingleOrDefault(c=>c.CustomerID == id)
}
```

Forgive any mistakes, code written on the fly. The thrown HttpResponseException bubbles up thru the actions call stack, does not get caught by normal Exception handlers and returns its

812 1 7 20