Content Negotiation in ASP.NET Web API

05/20/2012   •   4 minutes to read   •   Contributors 🧑‍🦰 🧑 🧑 🧑 🐕 all

In this article

Serialization

How Content Negotiation Works

Default Content Negotiator

Selecting a Character Encoding

by [Mike Wasson](#)

This article describes how ASP.NET Web API implements content negotiation.

The HTTP specification (RFC 2616) defines content negotiation as "the process of selecting the best representation for a given response when there are multiple representations available." The primary mechanism for content negotiation in HTTP are these request headers:

- **Accept:** Which media types are acceptable for the response, such as "application/json," "application/xml," or a custom media type such as "application/vnd.example+xml"
- **Accept-Charset:** Which character sets are acceptable, such as UTF-8 or ISO 8859-1.
- **Accept-Encoding:** Which content encodings are acceptable, such as gzip.
- **Accept-Language:** The preferred natural language, such as "en-us".

The server can also look at other portions of the HTTP request. For example, if the request contains an X-Requested-With header, indicating an AJAX request, the server might default to JSON if there is no Accept header.

In this article, we'll look at how Web API uses the Accept and Accept-Charset headers. (At this time, there is no built-in support for Accept-Encoding or Accept-Language.)

Serialization

If a Web API controller returns a resource as CLR type, the pipeline serializes the return value and writes it into the HTTP response body.

For example, consider the following controller action:

C#   ⎘ Copy

```csharp
public Product GetProduct(int id)
{
    var item = _products.FirstOrDefault(p => p.ID == id);
    if (item == null)
    {
        throw new HttpResponseException(HttpStatusCode.NotFound);
    }
    return item;
}
```

A client might send this HTTP request:

console   ⎘ Copy

```console
GET http://localhost.:21069/api/products/1 HTTP/1.1
Host: localhost.:21069
Accept: application/json, text/javascript, */*; q=0.01
```

In response, the server might send:

console   ⎘ Copy

```console
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 57
Connection: Close

{"Id":1,"Name":"Gizmo","Category":"Widgets","Price":1.99}
```

In this example, the client requested either JSON, Javascript, or "anything" (*/*). The server responsed with a JSON representation of the `Product` object. Notice that the Content-Type header in the response is set to "application/json".

A controller can also return an **HttpResponseMessage** object. To specify a CLR object for the response body, call the **CreateResponse** extension method:

C#   ⎘ Copy

```csharp
public HttpResponseMessage GetProduct(int id)
{
    var item = _products.FirstOrDefault(p => p.ID == id);
```

```
    if (item == null)
    {
        throw new HttpResponseException(HttpStatusCode.NotFound);
    }
    return Request.CreateResponse(HttpStatusCode.OK, product);
}
```

This option gives you more control over the details of the response. You can set the status code, add HTTP headers, and so forth.

The object that serializes the resource is called a *media formatter*. Media formatters derive from the **MediaTypeFormatter** class. Web API provides media formatters for XML and JSON, and you can create custom formatters to support other media types. For information about writing a custom formatter, see [Media Formatters](#).

How Content Negotiation Works

First, the pipeline gets the **IContentNegotiator** service from the **HttpConfiguration** object. It also gets the list of media formatters from the **HttpConfiguration.Formatters** collection.

Next, the pipeline calls **IContentNegotiatior.Negotiate**, passing in:

- The type of object to serialize
- The collection of media formatters
- The HTTP request

The **Negotiate** method returns two pieces of information:

- Which formatter to use
- The media type for the response

If no formatter is found, the **Negotiate** method returns **null**, and the client recevies HTTP error 406 (Not Acceptable).

The following code shows how a controller can directly invoke content negotiation:

```csharp
C#                                                                    Copy

public HttpResponseMessage GetProduct(int id)
{
    var product = new Product()
        { Id = id, Name = "Gizmo", Category = "Widgets", Price = 1.99M };

    IContentNegotiator negotiator =
```

```
this.Configuration.Services.GetContentNegotiator();

    ContentNegotiationResult result = negotiator.Negotiate(
        typeof(Product), this.Request, this.Configuration.Formatters);
    if (result == null)
    {
        var response = new HttpResponseMessage(HttpStatusCode.NotAcceptable);
        throw new HttpResponseException(response));
    }

    return new HttpResponseMessage()
    {
        Content = new ObjectContent<Product>(
            product,                 // What we are serializing
            result.Formatter,        // The media formatter
            result.MediaType.MediaType  // The MIME type
        )
    };
}
```

This code is equivalent to the what the pipeline does automatically.

Default Content Negotiator

The **DefaultContentNegotiator** class provides the default implementation of **IContentNegotiator**. It uses several criteria to select a formatter.

First, the formatter must be able to serialize the type. This is verified by calling **MediaTypeFormatter.CanWriteType**.

Next, the content negotiator looks at each formatter and evaluates how well it matches the HTTP request. To evaluate the match, the content negotiator looks at two things on the formatter:

- The **SupportedMediaTypes** collection, which contains a list of supported media types. The content negotiator tries to match this list against the request Accept header. Note that the Accept header can include ranges. For example, "text/plain" is a match for text/* or */*.
- The **MediaTypeMappings** collection, which contains a list of **MediaTypeMapping** objects. The **MediaTypeMapping** class provides a generic way to match HTTP requests with media types. For example, it could map a custom HTTP header to a particular media type.

If there are multiple matches, the match with the highest quality factor wins. For example:

console	📋 Copy

```
Accept: application/json, application/xml; q=0.9, */*; q=0.1
```

In this example, application/json has an implied quality factor of 1.0, so it is preferred over application/xml.

If no matches are found, the content negotiator tries to match on the media type of the request body, if any. For example, if the request contains JSON data, the content negotiator looks for a JSON formatter.

If there are still no matches, the content negotiator simply picks the first formatter that can serialize the type.

Selecting a Character Encoding

After a formatter is selected, the content negotiator chooses the best character encoding by looking at the **SupportedEncodings** property on the formatter, and matching it against the Accept-Charset header in the request (if any).