Getting started with the IIS CORS Module

Thursday, March 1, 2018

IIS News Item (/iisteam/Tags/IIS%20News%20Item)

Browsers usually apply same-origin restrictions to network requests. These restrictions would prevent a malicious page from making a cross origin request initiated from within a script. As an example, this means ordinarily a script served from `https://foo.com` cannot make a request to `https://bar.com`. However, there are instances in which you may want to allow sites to make these requests. For example, it's a common practice the split the web frontend (`https://contoso.com`) from the service hosting your API (`https://api.contoso.com`). For such scenarios to work, you will need to configure your API to reply with appropriate CORS headers. The IIS CORS module provides a way for web administrators and web site authors to easily support the CORS protocol by delegating all CORS protocol handling to the module.

What is CORS?

Cross Origin Resource Sharing (CORS) is a W3C standard that allows an user agent to gain permission to request a resource by a mechanism that uses additional HTTP headers.

The CORS specification makes the distinction between **Simple** and **Preflighted** CORS requests and the IIS CORS module can help you with both.

Simple Requests

Simple requests meet **ALL THREE** of the following criteria:

- The HTTP method is either a HEAD/GET/POST
- Apart from the headers set by the user agent, the only additional headers allowed are those defined in the Fetch spec as CORS-safelisted-request-header (https://fetch.spec.whatwg.org/#cors-safelisted-request-header).
- The only allowed valued for the ***Content-Type*** header are ***application/x-www-form-urlencoded***, ***multipart/form-data***, and ***text/plain***.

Here's an example of a simple request:

```
GET http://bar.com/data.json HTTP/1.1
Host: bar.com
Connection: keep-alive
Origin: http://foo.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like G
Accept: */*
Referer: http://foo.com/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

The main header of interest is the ***Origin*** header which shows the origin of the request is from the domain `http://foo.com`.

Here's the response from the server to that simple request:

```
HTTP/1.1 200 OK
Content-Type: application/json
Last-Modified: Thu, 01 Feb 2018 21:51:05 GMT
Accept-Ranges: bytes
ETag: "1f116fc2a69bd31:0"
Access-Control-Allow-Origin: http://foo.com
Date: Thu, 01 Feb 2018 22:19:13 GMT
Content-Length: 38

{

"backgroundColor": "#ff0000"

}
```

The header of interest here is the ***Access-Control-Allow-Origin*** header which the server sets to `http://foo.com`. Since that matches origin header in the request, the XMLHttpRequest succeeds. If the server did not indicate that via the Access-Control headers, the browser would fail the request in a manner indistinguishable from a network error.

Preflighted Requests

For any cross-origin requests that don't meet all three of the above criteria, the browser will send a preflight request with the ***OPTIONS*** HTTP method and will only proceed to send the actual request if indicated by the server in it's response to the pre-flight request.

Here's an example of a preflighted request sent (in our simple example, it only differs from the simple request due to the inclusion of an additional header ***ADDITIONAL-HEADER***):

```
OPTIONS http://bar.com/data.json HTTP/1.1
Host: bar.com
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Access-Control-Request-Method: GET
Origin: http://foo.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like G
Access-Control-Request-Headers: additional-header
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
```

In addition to ***Origin*** header that I highlighted in the previous example, the browser adds two additional headers of interest: ***Access-Control-Request-Method*** and ***Access-Control-Request-Headers***. These are used to indicate the HTTP Method of the actual request and any additional headers that the client intends to send that aren't part of the fetch spec.

Here's the response from the server to that preflight request:

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: http://foo.com
Access-Control-Allow-Headers: additional-header
Date: Fri, 02 Feb 2018 22:50:58 GMT
```

In this case, based on the response headers, the browser has made the determination that it's okay to send the actual request which it then proceeds to send:

```
GET http://bar.com/data.json HTTP/1.1
Host: bar.com
Connection: keep-alive
Origin: http://foo.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like G
ADDITIONAL-HEADER: addtional-header-value
Accept: */*
Referer: http://foo.com/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
```

Look at the presence of the ***ADDITIONAL-HEADER*** that the browser had indicated it would be sending in it's preflight request.

On receiving the real request, the server responds with the expected response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Last-Modified: Thu, 01 Feb 2018 21:51:05 GMT
Accept-Ranges: bytes
ETag: "1f116fc2a69bd31:0"
Server: Microsoft-IIS/10.0
Access-Control-Allow-Origin: *
X-Powered-By: ASP.NET
Date: Sat, 03 Feb 2018 01:20:09 GMT
Content-Length: 38

{

"backgroundColor": "#ff0000"

}
```

What HTTP request headers can the browser send?

Besides the ***Origin*** header which is always set, there are two additional headers that sent as part of the pre-flight request

- ***Access-Control-Request-Method***: This header is used by the browser to indicate the HTTP method of the actual request
- ***Access-Control-Request-Headers***: This header is used by the browser to indicate the any additional headers that may be sent as part of actual request that aren't a part of the fetch specification.

What HTTP response headers can the server send?

- ***Access-Control-Allow-Origin***
- ***Access-Control-Expose-Headers***
- ***Access-Control-Max-Age***
- ***Access-Control-Allow-Credentials***
- ***Access-Control-Allow-Methods***
- ***Access-Control-Allow-Headers***

Configuring IIS CORS module

The IIS CORS module is configured via the ***<cors>*** element as part of the ***<system.webServer>*** section. The section can be configured at the server, site, or application level.

```
<?xml version="1.0"?>
<configuration>
  <system.webServer>
    <cors enabled="true">
      <add origin="*" />
    </cors>
  </system.webServer>
</configuration>
```

In this simplest example, the CORS module module will allow requests from all origins. All other settings like what are the permissible methods and and headers are keyed of the origin. Let's look at another example on how you might use that.

```
<?xml version="1.0"?>
<configuration>
  <system.webServer>
    <cors enabled="true">
      <add origin="https://readonlyservice.constoso.com">
        <allowMethods>
            <add method="GET" />
            <add method="HEAD" />
        </allowMethods>
      </add>
      <add origin="https://readwriteservice.constoso.com">
        <allowMethods>
            <add method="GET" />
            <add method="HEAD" />
            <add method="POST" />
            <add method="PUT" />
            <add method="DELETE" />
        </allowMethods>
      </add>
    </cors>
  </system.webServer>
</configuration>
```

The detailed IIS CORS Configuration reference is available at the IIS CORS module Configuration Reference (https://docs.microsoft.com/en-us/iis/extensions/cors-module/cors-module-configuration-reference).

Configuring IIS CORS to send additional CORS headers

All other CORS headers are keyed off the origin. You can add multiple origin by specifying the ***origin*** attribute of the child element collection of the ***<cors>*** element. The origin attribute supports wildcard matching via the ***** character. In the event that multiple rules match, the best match will win. In the example below, if the origin is `https://api.contoso.com` the ***Access-Control-Allow-Credentials*** header will be set.

```
<?xml version="1.0"?>
<configuration>
  <system.webServer>
    <cors enabled="true">
      <add origin="*" allowed="false"/>
      <add origin="https://*.contoso.com" allowCredentials="false" />
      <add origin="https://api.contoso.com" allowCredentials="true" />
    </cors>
  </system.webServer>
</configuration>
```

Additionally, you can specify force an HTTP 403 response for origins not specified in the collection by setting the `failUnlistedOrigins` attribute of the ***<cors>*** element to `true` .

```
<?xml version="1.0"?>
<configuration>
  <system.webServer>
    <cors enabled="true">
      <add origin="https://api.contoso.com" allowCredentials="true" maxAge="120"/>
    </cors>
  </system.webServer>
</configuration>
```

The ***Access-Control-Allow-Credentials*** and ***Access-Control-Max-Age*** headers are controlled by the `allowCredentials` and `maxAge` attributes respectively of the child collection of the ***<cors>*** element. One thing to note here is that the CORS spec does not allow credentials to be sent when just * is specified as the origin.

```
<?xml version="1.0"?>
<configuration>
  <system.webServer>
    <cors enabled="true">
      <add origin="https://api.contoso.com" allowCredentials="true" maxAge="120"/>
    </cors>
  </system.webServer>
</configuration>
```

The ***Access-Control-Expose-Headers***, ***Access-Control-Allow-Methods***, and ***Access-Control-Allow-Headers*** and controlled via child collections of each child element of the ***<cors>*** element. The ***<allowHeaders>*** collection also has an `allowAllRequestedHeaders` attribute that allow you to accept all requested headers.

```
<?xml version="1.0"?>
<configuration>
  <system.webServer>
    <cors enabled="true">
      <add origin="https://api.contoso.com">
        <allowHeaders allowAllRequestedHeaders="true" />
        <allowMethods>
            <add method="GET" />
        </allowMethods>
        <exposeHeaders>
            <add header="header1" />
            <add header="header2" />
        </exposeHeaders>
      </add>
    </cors>
  </system.webServer>
</configuration>
```

Working with Windows Authentication

While this is by no means the only scenario solved by the CORS module, it was important enough to warrant calling out. Previously, if you tried to make a cross-domain request to an application that used Windows Authentication, your preflight request would fail since the browser did not send credentials with the preflight request. There was no way to work around this without enabling anonymous authentication in your application. Since the CORS module kicks in before authentication, it makes it possible to handle a pre-flight request without compromising on the security model of your application. Here's an example of what your **web.config** might look like.

```
<?xml version="1.0"?>
<configuration>
    <system.web>
        <authentication mode="Windows"/>
         <authorization>
            <allow roles="DOMAIN\SuperSecretGroup" />
            <deny  users="*" />
        </authorization>
    </system.web>
    <system.webServer>
        <cors enabled="true">
            <add origin="https://supersecretservice" allowCredentials="true" />
        </cors>
    </system.webServer>
</configuration>
```

No Comments