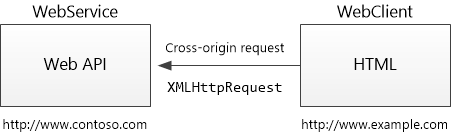
**Asp.net Web API 解决跨域详解**

浏览器安全防止web页面发出AJAX请求到另一个领域。这种限制称为同源策略,这是为了防止恶意网站读取敏感数据。然而,有时候。您可能想要让其他网站调用您的web API。

Cross Origin Resource Sharing（CORS）是一种W3C标准,允许服务器放松同源策略。CROS,服务器可以允许一些跨域源而拒绝其他域的请求。CORS比之前JSONP等技术更安全、更灵活。本教程展示了如何在Web API的应用程序中启用CROS。

**介绍**

本教程演示了ASP.NET Web API.中使用 CORS。我们将首先创建两个ASP.NET 项目。一个包含Web API控制器的“WebService”,另外一个其他“WebClient”,它调用WebService的接口。因为两个应用程序在不同的领域,一个AJAX请求从WebClient到WebService是一个跨源的要求。



**什么是同源**

如果两个URL他们有相同的域名,端口号，这两个URL就是有相同的源.即：同源

下面是同源的两个URL

1 http://example.com/foo.htm

2 http://example.com/bar.html

下面几个URL相比上面两个URL是不同源的

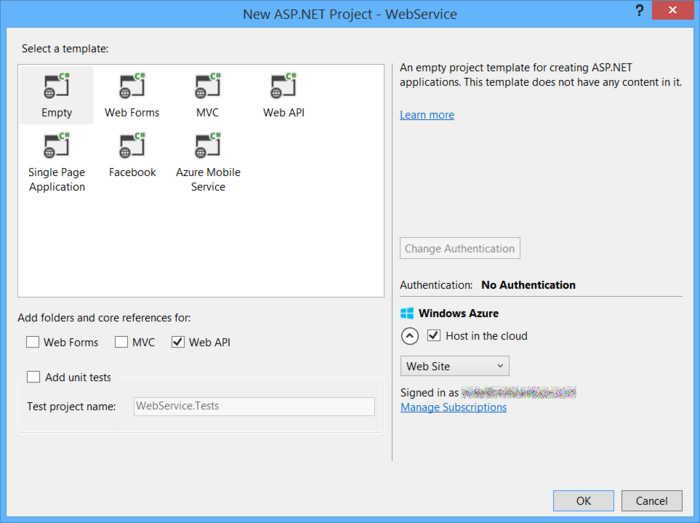
1 http://example.net

2 http://example.com:9000/foo.html

3 https://example.com/foo.html

4 https://www.example.com/foo.html

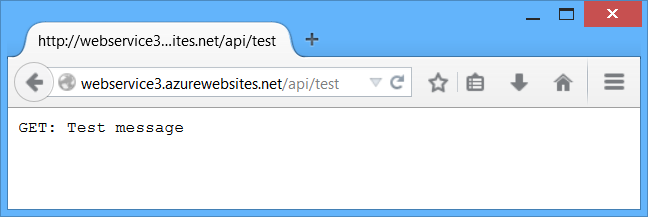
**创建WebService 项目**



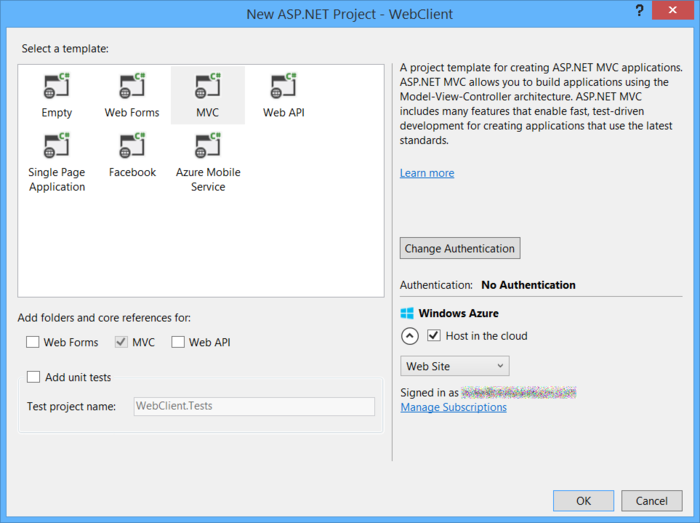
添加一个 名为 TestControllerWeb API 控制器

|  |
| --- |
| using System.Net.Http; using System.Web.Http; namespace WebService.Controllers { public class TestController : ApiController { public HttpResponseMessage Get() { return new HttpResponseMessage() { Content = new StringContent("GET: Test message") }; } public HttpResponseMessage Post() { return new HttpResponseMessage() { Content = new StringContent("POST: Test message") }; } public HttpResponseMessage Put() { return new HttpResponseMessage() { Content = new StringContent("PUT: Test message") }; } } } |

你可以在本地运行应用程序或部署到Azure。(本教程中的截图,我Web应用程序部署到Azure应用服务。)验证web API是否启动成功,导航到http://hostname/api/test/,主机名是署应用程序时使用的域名。您应该看到响应报文,“GET: Test Message”。



**创建WebClient 项目**

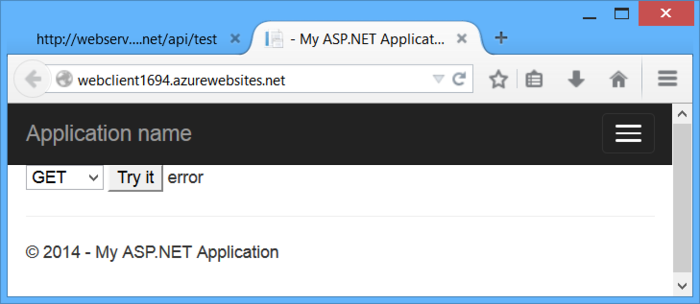


在解决方案资源管理器,打开文件/ Home / Index.cshtml 。用以下代码替换该文件中的代码:

|  |
| --- |
| <div> <select id="method"> <option value="get">GET</option> <option value="post">POST</option> <option value="put">PUT</option> </select> <input type="button" value="Try it" onclick="sendRequest()" /> <span id='value1'>(Result)</span> </div> @section scripts { <script> // TODO: Replace with the URL of your WebService app var serviceUrl = 'http://mywebservice/api/test';  function sendRequest() { var method = $('#method').val(); $.ajax({ type: method, url: serviceUrl }).done(function (data) { $('#value1').text(data); }).error(function (jqXHR, textStatus, errorThrown) { $('#value1').text(jqXHR.responseText || textStatus); }); } </script> } |

备注：serviceUrl变量,使用WebService 项目的URI。现在在本地运行WebClient应用程序或发布到另一个网站。

点击"Try It”按钮提交一个AJAX请求到WebService应用程序,使用下拉框中列出的HTTP方法(GET、POST、或者put)。这让我们检查不同跨源请求。现在, WebService应用程序不支持CORS,所以如果你单击按钮,你会得到一个错误。



**允许CORS**

现在让我们在WebService应用CORS。首先,添加CORSNuGet包。在Visual Studio中,从“工具”菜单上,选择库软件包管理器,然后选择包管理器控制台。在包管理器控制台窗口中,键入以下命令:

nstall-Package Microsoft.AspNet.WebApi.Cors

这个命令安装最新的包和更新所有依赖项,包括核心Web API库。User version标志针对一个特定的版本。 CORS包需要Web API 2.0或更高版本。

打开文件App\_Start / WebApiConfig.cs。将下面的代码添加到WebApiConfig.Register方法。

|  |
| --- |
| using System.Web.Http; namespace WebService { public static class WebApiConfig { public static void Register(HttpConfiguration config) { // New code config.EnableCors(); config.Routes.MapHttpRoute( name: "DefaultApi", routeTemplate: "api/{controller}/{id}", defaults: new { id = RouteParameter.Optional } ); } } } |

接下来,TestController类添加EnableCors属性:

|  |
| --- |
| using System.Net.Http; using System.Web.Http; using System.Web.Http.Cors; namespace WebService.Controllers { [EnableCors(origins: "http://mywebclient.azurewebsites.net", headers: "\*", methods: "\*")] public class TestController : ApiController { // Controller methods not shown... } } |

origins参数,使用WebClient应用程序部署时用用的的URI，这允许跨域源来自WebClient的请求,同时还禁止所有其他跨域请求。之后,我将详细描述[EnableCors]的参数。

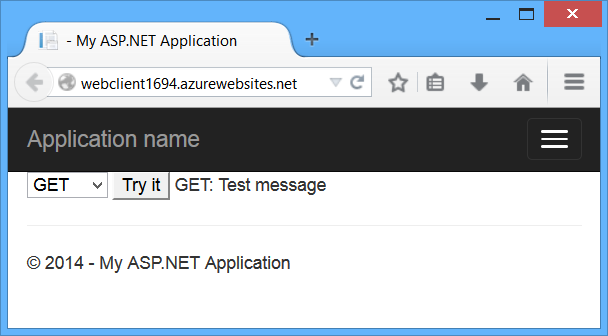
备注：URI与URL不同，URI ：Uniform Resource Identifier，统一资源标识符

URL：Uniform Resource Locator，统一资源定位符

URI以scheme和冒号开头。Scheme用大写/小写字母开头，后面为空或者跟着更多的大写/小写字母、数字、加号、减号和点号。冒号把scheme与scheme-specific-part分开了，并且scheme-specific-part的语法和语义（意思）由URI的名字空间决定。如下面的例子：

http://域名，其中http是scheme，//域名 是scheme-specific-part，并且它的scheme与scheme-specific-part被冒号分开了。

重新部署更新WebService 的应用程序。你不需要更新WebClient。现在WebClient的AJAX请求应该成功。GET、PUT和POST方法都是允许的。

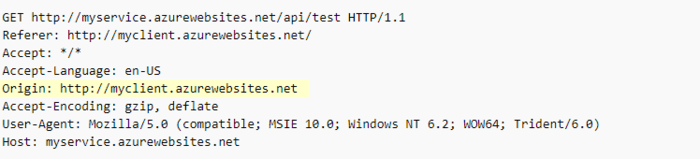


**CORS工作原理**

本节描述了在Http协议标准上http跨域请求中究竟发生了什么。重要的是要理解CORS是如何工作的,这样你就可以正确配置[EnableCors]属性,和如果CORS不像您预期的那样工作怎样排除错误。

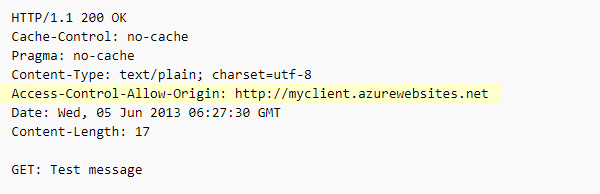
CORS为了允许使跨源请求引入了几个新的HTTP头。如果浏览器支持CORS,它自动设置这些请求头，你不需要在你的JavaScript代码做任何修改。

这是一个跨域请求的例子。Origin请求头提供了产生这个跨域请求的网站域名。



请求报文

如果服务器允许这个跨域请求，响应报文中自动设置Access-Control-Allow-Origin头。这个头的值匹配请求报文中Origin头的值,或者是通配符“\*”,这意味着任何起源是被允许的。



响应报文

如果响应不包括Access-Control-Allow-Origin头,这是AJAX请求失败。具体来说是浏览器不允许请求。即使服务器返回一个成功的响应,浏览器响应的结果不可用于客户端应用程序。

**CORS 预检请求preflight request**

对于一些CORS请求,浏览器会发送一个额外的请求,称为预检请求“preflight request”,在发送的实际请求的资源之前。

浏览器可以跳过preflight request如果下列条件属实:

1 请求的方法是GET, HEAD, or POST,等

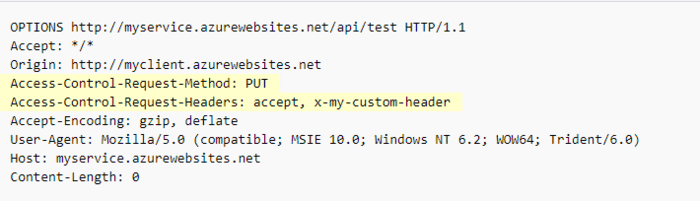
2 应用程序不设置任何请求头除了Accept, Accept-Language, Content-Language, Content-Type, or Last-Event-ID等

3 content - type报头(如果设置)是下列之一:

|  |
| --- |
| application/x-www-form-urlencoded multipart/form-data text/plain |

这个请求头的规范适用于当应用程序调用setRequestHeade XMLHttpRequest对象时发起的请求头。 规范并不适用于浏览器的请求头可以设置,如用户代理,主机,或内容长度。

下面是preflight request的一个例子

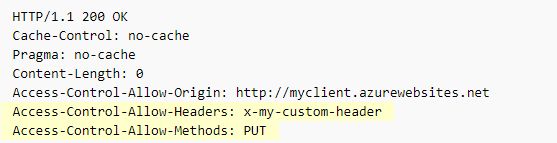


pre-flight请求使用HTTP OPTIONS方法。它包括两个特殊的请求头:

Access-Control-Request-Method:HTTP方法将被用于实际的请求。

Access-Control-Request-Headers:应用程序设置的实际的请求头的列表。(同样,这并不包括浏览器设置的请求头。)

这里有一个响应报文例子,假设服务器允许请求:



响应包含一个Access-Control-Allow-Methods列出允许的方法、和可选一个Access-Control-Allow-Headers头列表允许的头。如果preflight请求成功,浏览器发送实际的请求,如前所述。

**[EnableCors]设置**

您可以启用 CORS在每一个 action,controller,或Web API全局控制器中。

1 action设置

在action上允许跨域，设置[EnableCors]属性的action方法。下面的例子使GetItemmethod 单独允许跨域

|  |
| --- |
| public class ItemsController : ApiController { public HttpResponseMessage GetAll() { ... } [EnableCors(origins: "http://www.example.com", headers: "\*", methods: "\*")] public HttpResponseMessage GetItem(int id) { ... } public HttpResponseMessage Post() { ... } public HttpResponseMessage PutItem(int id) { ... } } |

2 controller设置

如果您设置EnableCors在控制器,它适用于该控制器上的所有的action。如果想对某一个action禁用跨域，请使用[DisableCors]特性。下面的例子除了PutItem action 其他action都支持跨域。

|  |
| --- |
| [EnableCors(origins: "http://www.example.com", headers: "\*", methods: "\*")] public class ItemsController : ApiController { public HttpResponseMessage GetAll() { ... } public HttpResponseMessage GetItem(int id) { ... } public HttpResponseMessage Post() { ... } [DisableCors] public HttpResponseMessage PutItem(int id) { ... } } |

3 全局设置

在应用程序中为所有Web API 控制器允许跨域,将一个EnableCorsAttribute实例传递给EnableCors方法:

|  |
| --- |
| public static class WebApiConfig { public static void Register(HttpConfiguration config) { var cors = new EnableCorsAttribute("www.example.com", "\*", "\*"); config.EnableCors(cors); // ... } } |

如果你在多个范围内多个设置[EnableCors]、优先顺序是:

|  |
| --- |
| 1Action 2Controller 3Global |

**[EnableCors]参数origin介绍**

[EnableCors]的origins 参数指定了哪一个请求起源是允许访问的。允许的值之间是一个以逗号分隔的。

|  |
| --- |
| [EnableCors(origins: "http://www.contoso.com,http://www.example.com",  headers: "\*", methods: "\*")] |

**[EnableCors]参数methods介绍**

[EnableCors]特性的methods，指定了哪一个HTTP方法可以访问资源。为了使所有方法都可以访问，使用通配符“ \* ”。下面是一个只允许GET和POST方法的请求示例：

|  |
| --- |
| [EnableCors(origins: "http://www.example.com", headers: "\*", methods: "get,post")] public class TestController : ApiController { public HttpResponseMessage Get() { ... } public HttpResponseMessage Post() { ... } public HttpResponseMessage Put() { ... }  } |

**[EnableCors]参数headers介绍**

[EnableCors]特性的headers，指定了哪一个HTTP请求头可以访问资源。为了使任何请求头都可以访问，使用通配符“ \* ”，多个允许的headers之间使用一个逗号来分隔。

|  |
| --- |
| [EnableCors(origins: "http://example.com",  headers: "accept,content-type,origin,x-my-header", methods: "\*")] |

**设置允许响应标头**

默认情况下,浏览器不公开所有的应用程序响应标头。可用的响应头默认情况下是:

|  |
| --- |
| Cache-Control Content-Language Content-Type Expires Last-Modified Pragma |

CORS 规定了调用这些简单的响应头。于应用程序中使用其他头文件,请设置[EnableCors]的exposedHeaders参数

在接下来的例子中,控制器的Get方法设置一个自定义标头命名为“X-Custom-Header”。默认情况下,浏览器不会在跨源的请求中暴露这个自定义标头。为了使自定义标头有效,使 exposedHeaders 参数的值为X-Custom-Header”。

|  |
| --- |
| [EnableCors(origins: "\*", headers: "\*", methods: "\*", exposedHeaders: "X-Custom-Header")] public class TestController : ApiController { public HttpResponseMessage Get() { var resp = new HttpResponseMessage() { Content = new StringContent("GET: Test message") }; resp.Headers.Add("X-Custom-Header", "hello"); return resp; } } |

**在跨域请求中通过证书请求**

跨域请求中使用证书需要特殊处理。默认情况下,浏览器不发送任何证书凭证与跨源的要求。凭证不但包括cookies还包括HTTP身份验证方案。为了在跨源请求发送凭证,客户端必须设置XMLHttpRequest.withCredentials为true。

|  |
| --- |
| var xhr = new XMLHttpRequest(); xhr.open('get', 'http://www.example.com/api/test'); xhr.withCredentials = true; |

此外,服务器必须允许凭据。在Web API,(EnableCors)特性的允许跨源凭证SupportsCredentials参数设置为true

|  |
| --- |
| $.ajax({ type: 'get', url: 'http://www.example.com/api/test', xhrFields: { withCredentials: true } |

如果这个属性是true,HTTP响应将包含Access-Control-Allow-Credentials头。这个头告诉浏览器跨源请求的服务器允许凭据。

如果浏览器发送证书,但是响应不包括一个有效的Access-Control-Allow-Credentials头,浏览器不会公开响应应用程序,并且AJAX请求失败。

务必小心将SupportsCredentials设置为true,因为这意味在一个网站在另一个域可以发送一个登录的用户的凭证代表用户的Web API,。CORS还规定,设置“\*”的Origin是无效的，在SupportsCredentials是true的情况下。

**自定义[EnableCors]特性**

[EnableCors]特性实现了ICorsPolicy

|  |
| --- |
| [EnableCors(origins: "http://myclient.azurewebsites.net", headers: "\*",  methods: "\*", SupportsCredentials = true)] |

Provider接口。您可以提供自己的实现通过创建一个类,它来继承Attribute和实现了ICorsProlicyProvider接口。

|  |
| --- |
| [AttributeUsage(AttributeTargets.Method | AttributeTargets.Class, AllowMultiple = false)] public class MyCorsPolicyAttribute : Attribute, ICorsPolicyProvider  { private CorsPolicy \_policy; public MyCorsPolicyAttribute() { // Create a CORS policy. \_policy = new CorsPolicy { AllowAnyMethod = true, AllowAnyHeader = true }; // Add allowed origins. \_policy.Origins.Add("http://myclient.azurewebsites.net"); \_policy.Origins.Add("http://www.contoso.com"); } public Task<CorsPolicy> GetCorsPolicyAsync(HttpRequestMessage request) { return Task.FromResult(\_policy); } } |

现在你可以在你想要允许跨域的任何地方使用你刚才自定义的[EnableCors].特性

|  |
| --- |
| [MyCorsPolicy] public class TestController : ApiController { .. // } |