18
12月 2011

利用CORS实现跨域请求

by liuyanghejerry

跨域请求一直是网页编程中的一个难题，在过去，绝大多数人都倾向于使用JSONP来解决这一问题。不过现在，我们可以考虑一下W3C中一项新的特性——CORS（Cross-Origin Resource Sharing）了。

> *本文的所有代码均来自http://www.html5rocks.com/en/tutorials/cors/，如果您对其中的任何技术细节存在疑问，请以原文为准。*

客户端

创建XmlHttpRequest对象

对于CORS，Chrome、FireFox以及Safari，需要使用XmlHttpRequest2对象；而对于IE，则需要使用XDomainRequest；Opera目前还不支持这一特性，但很快就会支持。

因此，在对象的创建上，我们不得不首先针对不同的浏览器而进行一下预处理：

```
function createCORSRequest(method, url) {
  var xhr = new XMLHttpRequest();
  if ("withCredentials" in xhr) {

    // "withCredentials"属性是XMLHTTPRequest2中独有的
    xhr.open(method, url, true);

  } else if (typeof XDomainRequest != "undefined") {

    // 检测是否XDomainRequest可用
    xhr = new XDomainRequest();
    xhr.open(method, url);

  } else {

    // 看起来CORS根本不被支持
    xhr = null;

  }
  return xhr;
}

var xhr = createCORSRequest('GET', url);
if (!xhr) {
  throw new Error('CORS not supported');
}
```

事件处理

原先的XmlHttpRequest对象仅仅只有一个事件——onreadystatechange，用以通知所有的事

近期文章

HTML Preload 文章两篇
V8即将启用新的引擎 – TurboFan & Ignition
TypedArray vs. DataView
Cache-Control新成员：immutable
有关Error.captureStackTrace
有关line-height
8款纯CSS模拟移动设备
CSS3 Transform + Animation 实现复古时钟
Beacon API 来了
把Modern UI风格带入Web – WinJS

Search

标签

3D animated buttons bootstrap canvas CSS CSS3 CSS3 animation css3 transform font-face hover effect HTML5 icon icon font javascript jquery jquery plugin node.js UI V8 V8 解析 Web Audio API web font 下载 免费 典范 创意 图标 字体 工具 手机网页 按钮 插件 效果 教程 浏览器 移动应用 网页 翻译 翻页 翻页效果 色彩 设计 资源 适应性设

件，而现在，我们除了这个事件之外又多了很多新的。

事件	说明
onloadstart*	当请求发生时触发
onprogress	读取及发送数据时触发
onabort*	当请求被中止时触发，如使用abort()方法
onerror	当请求失败时触发
onload	当请求成功时触发
ontimeout	当调用者设定的超时时间已过而仍未成功时触发
onloadend*	请求结束时触发（无论成功与否）

注：带星号的表示IE的XDomainRequest仍不支持。
数据来自http://www.w3.org/TR/XMLHttpRequest2/#events。

绝大多数情况下，我们只需要和onload及onerror打交道，就像下面这样：

```
xhr.onload = function() {
 var responseText = xhr.responseText;
 console.log(responseText);
 // 继续其它代码
};

xhr.onerror = function() {
  console.log('There was an error!');
};
```

这儿有一点小异样。尽管我们可以通过onerror得知请求发生了错误，但在事件处理时，我们无法从代码上获知失败的任何原因。比如，FireFox在失败时将responseText置空并返回一个0值作为状态，这当中并不包含任何错误的具体情况。

withCredentials

标准的CORS请求不对cookies做任何事情，既不发送也不改变。如果希望改变这一情况，就需要将withCredentials设置为true。

```
xhr.withCredentials = true;
```

另外，服务端在处理这一请求时，也需要将Access-Control-Allow-Credentials设置为true。这一点我们稍后来说。

withCredentials属性使得请求包含了远程域的所有cookies，但值得注意的是，这些cookies仍旧遵守“同域”的准则，因此从代码上你并不能从document.cookies或者回应HTTP头当中进行读取。

发送请求

请求通过一个简单的send()方法进行发送，如果请求当中需要包含任何内容，也只需要将其作为一个参数传递给send()即可。一旦服务端配置OK，那么你将只需要处理后续的onload事件，这正像我们平时所熟悉的XHR一样。

来看一段完整的小代码：

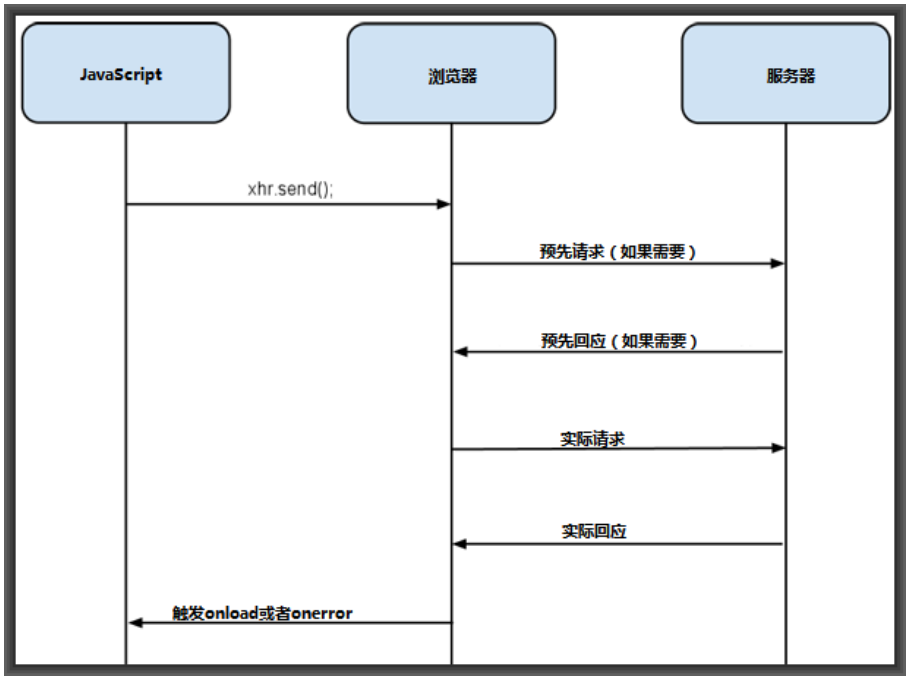
```
// 创建XHR对象
function createCORSRequest(method, url) {
  var xhr = new XMLHttpRequest();
```

```
  if ("withCredentials" in xhr) {
    // 针对Chrome/Safari/Firefox.
    xhr.open(method, url, true);
  } else if (typeof XDomainRequest != "undefined") {
    // 针对IE
    xhr = new XDomainRequest();
    xhr.open(method, url);
  } else {
    // 不支持CORS
    xhr = null;
  }
  return xhr;
}

// 辅助函数，用于解析返回的内容
function getTitle(text) {
  return text.match('')[1];
}

// 发送CORS请求
function makeCorsRequest() {
  // bibliographica.org是支持CORS的
  var url = 'http://bibliographica.org/';

  var xhr = createCORSRequest('GET', url);
  if (!xhr) {
    alert('CORS not supported');
    return;
  }

  // 回应处理
  xhr.onload = function() {
    var text = xhr.responseText;
    var title = getTitle(text);
    alert('Response from CORS request to ' + url + ': ' + title);
  };

  xhr.onerror = function() {
    alert('Woops, there was an error making the request.');
  };

  xhr.send();
}
```

服务端

一个CORS请求可能包含多个HTTP头，甚至有多个请求实际发送，这对于客户端的开发者来说通常是透明的。因为浏览器已经负责实现了CORS最关键的部分；但是服务端的后台脚本则需要我们自己进行处理，因此我们还需要了解到服务端到底从浏览器那里收到了怎样的内容。

先来看看流程图吧。



CORS分类

CORS可以分成两种：

- 简单请求
- 复杂请求

一个简单的请求大致如下：

- HTTP方法是下列之一
  - HEAD
  - GET
  - POST
- HTTP头包含
  - Accept
  - Accept-Language
  - Content-Language
  - Last-Event-ID
  - Content-Type，但仅能是下列之一
    - application/x-www-form-urlencoded
    - multipart/form-data
    - text/plain

任何一个不满足上述要求的请求，即被认为是复杂请求。一个复杂请求不仅有包含通信内容的请求，同时也包含预请求（preflight request）。

简单请求

为了搞清楚复杂请求与简单请求有何区别，我们首先来看看简单请求是怎样处理的。

JavaScript：

```
var url = 'http://alice.com/cors';
var xhr = createCORSRequest('GET', url);
xhr.send();
```

HTTP请求：

```
GET /cors HTTP/1.1
Origin: http://api.alice.com
Host: api.bob.com
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```

简单请求的发送从代码上来看和普通的XHR没太大区别，但是HTTP头当中要求总是包含一个域（Origin）的信息。该域包含协议名、地址以及一个可选的端口。不过这一项实际上由浏览器代为发送，并不是开发者代码可以触及到的。

HTTP回应：

```
Access-Control-Allow-Origin: http://api.bob.com
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: FooBar
Content-Type: text/html; charset=utf-8
```

在回应中，COR相关的项目全都是以“Access-Control-”作为前缀的，其意义分列如下：

- Access-Control-Allow-Origin（必含）- 不可省略，否则请求按失败处理。该项控制数据的可见范围，如果希望数据对任何人都可见，可以填写“*”。
- Access-Control-Allow-Credentials（可选） – 该项标志着请求当中是否包含cookies信息，只有一个可选值：true（必为小写）。如果不包含cookies，请略去该项，而不是填写false。这一项与XmlHttpRequest2对象当中的withCredentials属性应保持一致，即withCredentials为true时该项也为true；withCredentials为false时，省略该项不写。反之则导致请求失败。
- Access-Control-Expose-Headers（可选） – 该项确定XmlHttpRequest2对象当中getResponseHeader()方法所能获得的额外信息。通常情况下，getResponseHeader()方法只能获得如下的信息：
  - Cache-Control
  - Content-Language
  - Content-Type
  - Expires
  - Last-Modified
  - Pragma

  当你需要访问额外的信息时，就需要在这一项当中填写并以逗号进行分隔。不过目前浏览器对这一项的实现仍然有一些问题，具体请见文尾的BUG一节。
-

复杂请求

如果仅仅是简单请求，那么即便不用CORS也没有什么大不了，但CORS的复杂请求就令CORS显得更加有用了。简单来说，任何不满足上述简单请求要求的请求，都属于复杂请求。比如说你需要发送PUT、DELETE等HTTP动作，或者发送Content-Type: application/json的内容。

复杂请求表面上看起来和简单请求使用上差不多，但实际上浏览器发送了不止一个请求。其中最先发送的是一种“预请求”，此时作为服务端，也需要返回“预回应”作为响应。预请求实际上是对服务端的一种权限请求，只有当预请求成功返回，实际请求才开始执行。

JavaScript：

```
var url = 'http://alice.com/cors';
var xhr = createCORSRequest('PUT', url);
xhr.setRequestHeader(
   'X-Custom-Header', 'value');
xhr.send();
```

预请求：

```
OPTIONS /cors HTTP/1.1
Origin: http://api.alice.com
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: X-Custom-Header
Host: api.bob.com
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```

预请求以OPTIONS形式发送，当中同样包含域，并且还包含了两项CORS特有的内容：

- Access-Control-Request-Method – 该项内容是实际请求的种类，可以是GET、POST之类的简单请求，也可以是PUT、DELETE等等。
- Access-Control-Request-Headers – 该项是一个以逗号分隔的列表，当中是复杂请求所使用的头部。

显而易见，这个预请求实际上就是在为之后的实际请求发送一个权限请求，在预回应返回的内容当中，服务端应当对这两项进行回复，以让浏览器确定请求是否能够成功完成。例如，刚才的预请求可能获得服务端如下的回应：

```
Access-Control-Allow-Origin: http://api.bob.com
Access-Control-Allow-Methods: GET, POST, PUT
Access-Control-Allow-Headers: X-Custom-Header
Content-Type: text/html; charset=utf-8
```

来看看预回应当中可能的项目：

- Access-Control-Allow-Origin（必含） – 和简单请求一样的，必须包含一个域。
- Access-Control-Allow-Methods（必含） – 这是对预请求当中Access-Control-Request-Method的回复，这一回复将是一个以逗号分隔的列表。尽管客户端或许只请求某一方法，但服务端仍然可以返回所有允许的方法，以便客户端将其缓存。
- Access-Control-Allow-Headers（当预请求中包含Access-Control-Request-Headers时必须包含） – 这是对预请求当中Access-Control-Request-Headers的回复，和上面一样是以逗号分隔的列表，可以返回所有支持的头部。
- Access-Control-Allow-Credentials（可选） – 和简单请求当中作用相同。
- Access-Control-Max-Age（可选） – 以秒为单位的缓存时间。预请求的的发送并非免费午餐，允许时应当尽可能缓存。

一旦预回应如期而至，所请求的权限也都已满足，则实际请求开始发送。

实际请求：

```
PUT /cors HTTP/1.1
Origin: http://api.alice.com
Host: api.bob.com
X-Custom-Header: value
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```

实际回应：

```
Access-Control-Allow-Origin: http://api.bob.com
Content-Type: text/html; charset=utf-8
```

如果预请求所要求的权限服务端不允许，那么服务端可以直接返回一个普通的HTTP回应，比

如：

```
// ERROR - No CORS headers, this is an invalid request!
Content-Type: text/html; charset=utf-8
```

这样的返回因为不符合客户端的需求，因而客户端会直接将请求以失败计，虽然不是很美气，不过正符合我们的实际。此时如果客户端的onerror事件有监听函数，那么将会触发，而浏览器的console窗口也会输出：

```
XMLHttpRequest cannot load http://api.alice.com. Origin http://api.bob.com is not allowed by Access-Control-Allow-Origin.
```

不过很可惜，浏览器并不会给出详细的错误情况，仅仅是告知我们出错而已。

安全问题

跨域请求始终是网页安全中一个比较头疼的问题，CORS提供了一种跨域请求方案，但没有为安全访问提供足够的保障机制，如果你需要信息的绝对安全，不要依赖CORS当中的权限制度，应当使用更多其它的措施来保障，比如OAuth2。

已知问题

CORS是W3C中一项较"新"的方案，以至于各大网页解析引擎还没有对其进行完美的实现。下面是截至2011年11月13日时的已知问题：

- getAllResponseHeaders()方法无法获取Access-Control-Expose-Headers当中要求的信息。在Chrome/Safari当中，仅仅只有简单的头部能够读取，其他无法获取；在FireFox当中，无法获得任何信息。（FireFox Bugzilla/Webkit Bugzilla）
- 在Safari当中，使用GET、POST方法的复杂请求发送时没有发送预请求的环节。
- onerror触发时statusText获取不到任何内容。
- Opera截至11.60仍旧不支持CORS，但在12当中会支持（Opera Core Concerns – CORS goes mainline）。

阅读更多

- CORS的W3C规范
- 为CORS而配置服务器

Posted in: JavaScirpt, 教程 · Tagged: CORS, javascript, XDomainRequest, XmlHttpRequest2, 教程, 翻译, 跨域请求

About liuyanghejerry

富有激情的前端工程师，专注GUI开发。

View all posts by liuyanghejerry →

4 Thoughts on "利用CORS实现跨域请求"

Pingback: jquery 是如何实现ajax跨域请求的 - 编程问答 - 开发者第1632892个问答

Pingback: 利用CORS实现跨域请求 | Do you know me?

Pingback: CORS跨域请求，以及cookie操作 | Passion and Dream

Pingback: cors跨域 | otarim

← Previous Post

Next Post →