

前端跨域请求原理及实践

2017-05-24 前端大全

(点击上方公众号，可快速关注)

作者：高鹏

tingandpeng.com/2016/09/05/前端跨域请求原理及实践/

[如有好文章投稿，请点击 → 这里了解详情](#)

一、跨域请求的含义

浏览器的同源策略，出于防范跨站脚本的攻击，禁止客户端脚本（如 JavaScript）对不同域的服务进行跨站调用。

一般的，只要网站的 协议名protocol、主机host、端口号port 这三个中的任意一个不同，网站间的数据请求与传输便构成了跨域调用。这也是我们下面实践的理论基础。我们利用 NodeJs 创建了两个服务器，分别监听 3000、3001 端口（下面简称 服务器3000 与 服务器3001），由于端口号不一样，这两个服务器以及服务器上页面通信构成了跨域请求。

在服务器3000 上有如下的页面：

Ajax 跨域测试页面

tingandpeng.com

名称

ID

提交

服务器3000 上的请求页面中包含如下 JavaScript 代码：

```
$(function() {  
  $("#submit").click(function() {  
    var data = {  
      name: $("#name").val(),  
      id: $("#id").val()  
    };  
    $.ajax({  
      type: 'POST',  
      data: data,  
      url: 'http://localhost:3000/ajax/deal',  
      dataType: 'json',  
      cache: false,  
      timeout: 5000,  
      success: function(data) {  
        console.log(data)  
      },  
      error: function(jqXHR, textStatus, errorThrown) {  
        console.log('error ' + textStatus + ' ' + errorThrown);  
      }  
    });  
  });  
});
```

服务器3000 对应的处理函数为

```
pp.post('/ajax/deal', function(req, res) {  
  console.log("server accept: ", req.body.name, req.body.id)  
  var data = {  
    name: req.body.name + ' - server 3000 process',  
    id: req.body.id + ' - server 3000 process'  
  }  
  res.send(data)  
  res.end()  
})
```

请求页面返回结果：

```
▼ Object {name: "chiaki - server 3000 process", id: "3000 - server 3000 process"}  
  id: "3000 - server 3000 process"  
  name: "chiaki - server 3000 process"  
  ► __proto__: Object
```

此处数据处理成功。

由于数据请求一般都是由页面发送数据字段，服务器根据这些字段作相应的处理，如数据库查询，字符串操作等等。所以我们这里简单的处理数据（在数据后面加上字符串 'server 3000 process' ），并且返回给浏览器，表示数据经过服务器端处理。

如果让 服务器3000 上的页面向 服务器 3001 发起请求会怎样呢？

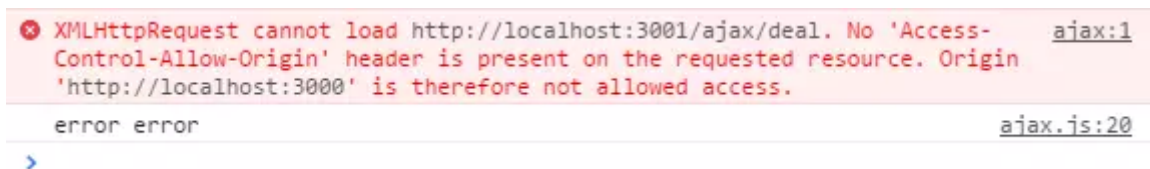
将请求页面中的 ajax 请求路径改为：

```
$.ajax({  
  ...  
  url: 'http://localhost:3001/ajax/deal',  
  ...  
});
```

服务器3001 对应的处理函数与 服务器3000 类似：

```
app.post('/ajax/deal', function(req, res) {  
  console.log("server accept: ", req.body.name, req.body.id)  
  var data = {  
    name: req.body.name + ' - server 3001 process',  
    id: req.body.id + ' - server 3001 process'  
  }  
  res.send(data)  
  res.end()  
})
```

结果如下：



结果证明了我们上面所说的端口号不同，发生了跨域请求的调用。

需要注意的是，服务器 3001 控制台有输出：

```
server accept: chiaki 3001
```

这说明跨域请求并非是浏览器限制了发起跨站请求，而是请求可以正常发起，到达服务器端，但是服务器返回的结果会被浏览器拦截。

二、利用 JSONP 实现跨域调用

说道跨域调用，可能大家首先想到的或者听说过的就是 JSONP 了。

2.1 什么是JSONP

JSONP (JSON with Padding or JSON-P) is a JSON extension used by web developers to overcome the cross-domain restrictions imposed by browsers' same-origin policy that limits access to resources retrieved from origins other than the one the page was served by. In layman's terms, one website cannot just simply access the data from another website.

It was developed because handling a browsers' same origin policy can be difficult, so using JSONP abstracts the difficulties and makes it easier.

JSON stands for "JavaScript Object Notation", a format by which object fields are represented as key-value pairs which is used to represent data.

JSONP 是 JSON 的一种使用模式，可以解决主流浏览器的跨域数据访问问题。其原理是根据 XMLHttpRequest 对象受到同源策略的影响，而 <script> 标签元素却不受同源策略影响，可以加载跨域服务器上的脚本，网页可以从其他来源动态产生 JSON 资料。用 JSONP 获取的不是 JSON 数据，而是可以直接运行的 JavaScript 语句。

2.2 使用 jQuery 集成的 \$.ajax 实现 JSONP 跨域调用

我们先从简单的实现开始，利用 jQuery 中的 \$.ajax 来实现上述的跨域调用。

依然是上面的例子，我们将 服务器 3000 上的请求页面的 JavaScript 代码改为：

```
// 回调函数
function jsonpCallback(data) {
  console.log("jsonpCallback: " + data.name)
}
$("#submit").click(function() {
  var data = {
    name: $("#name").val(),
    id: $("#id").val()
  };
  $.ajax({
    url: 'http://localhost:3001/ajax/deal',
```

```

data: data,
dataType: 'jsonp',
cache: false,
timeout: 5000,
// jsonp 字段含义为服务器通过什么字段获取回调函数的名称
jsonp: 'callback',
// 声明本地回调函数的名称, jquery 默认随机生成一个函数名称
jsonpCallback: 'jsonpCallback',
success: function(data) {
    console.log("ajax success callback: " + data.name)
},
error: function(jqXHR, textStatus, errorThrown) {
    console.log(textStatus + ' ' + errorThrown);
}
});
});

```

服务器 3001 上对应的处理函数为：

```

app.get('/ajax/deal', function(req, res) {
    console.log("server accept: ", req.query.name, req.query.id)
    var data = "{" + "name:" + req.query.name + " - server 3001 process'," + "id:" + req.query.id + " - server 3001 process" + "}"
    var callback = req.query.callback
    var jsonp = callback + '(' + data + ')'
    console.log(jsonp)
    res.send(jsonp)
    res.end()
})

```

这里一定要注意 data 中字符串拼接，不能直接将 JSON 格式的 data 直接传给回调函数，否则会发生编译错误： `parsererror Error: jsonpCallback was not called.`

其实脑海里应该有一个概念：利用 JSONP 格式返回的值一段要立即执行的 JavaScript 代码，所以不会像 ajax 的 XMLHttpRequest 那样可以监听不同事件对数据进行不同处理。

处理结果如下所示：

```

jsonpCallback: chiaki - server 3001 process                                ajax.js:2
ajax success callback: chiaki - server 3001 process                        ajax.js:21
>

```

2.3 使用 <script> 标签原生实现 JSONP

经过上面的事件，你是不是觉得 JSONP 的实现和 Ajax 大同小异？

其实，由于实现的原理不同，由 JSONP 实现的跨域调用不是通过 XMLHttpRequest 对象，而是通过 script 标签，所以在实现原理上，JSONP 和 Ajax 已经一点关系都没有了。看上去形式相似只是由于 jQuery 对 JSONP 做了封装和转换。

比如在上面的例子中，我们假设要传输的数据 data 格式如下：

```
{
  name: "chiaki",
  id: "3001"
}
```

那么数据是如何传输的呢？HTTP 请求头的第一行如下：

```
GET /ajax/deal?callback=jsonpCallback&name=chiaki&id=3001&_=1473164876032 HTTP/1.1
```

可见，即使形式上是用 POST 传输一个 JSON 格式的数据，其实发送请求时还是转换成 GET 请求。

其实如果理解 JSONP 的原理的话就不难理解为什么只能使用 GET 请求方法了。由于是通过 script 标签进行请求，所以上述传输过程根本上是以下的形式：

```
<script src = 'http://localhost:3001/ajax/deal?callback=jsonpCallback&name=chiaki&id=3001&_=1473164876032'>
</script>
```

这样从服务器返回的代码就可以直接在这个 script 标签中运行了。下面我们自己实现一个 JSONP：

服务器 3000 请求页面的 JavaScript 代码中，只有回调函数 jsonpCallback：

```
function jsonpCallback(data) {
  console.log("jsonpCallback: "+data.name)
}
```

服务器 3000 请求页面还包含一个 script 标签：

```
<script src = 'http://localhost:3001/jsonServerResponse?jsonp=jsonpCallback'></script>
```

服务器 3001 上对应的处理函数：

```
app.get('/jsonServerResponse', function(req, res) {
  var cb = req.query.jsonp
  console.log(cb)
  var data = 'var data = {' + 'name: $("#name").val() + " - server 3001 jsonp process",' + 'id: $("#id").val() + " - server 3001 jsonp process"' + '};'
  var debug = 'console.log(data);'
  var callback = $('#submit').click(function() {' + data + cb + '(data);' + debug + '});'
  res.send(callback)
  res.end()
})
```

与上面一样，我们在所获取的参数后面加上 “ - server 3001 jsonp process” 代表服务器对数据的操作。从代码中我么可以看到，处理函数除了根据参数做相应的处理，更多的也是进行字符串的拼接。

最终的结果为：



```
jsonpCallback: chiaki - server 3001 jsonp process    ajax.js:57
jsonServerResponse?jsonp=jsonpCallback:1
Object {name: "chiaki - server 3001 jsonp process", id: "3001 - server 3001 jsonp process"}
```

2.4 JSONP 总结

至此，我们了解了 JSONP 的原理以及实现方式，它帮我们实现前端跨域请求，但是在实践的过程中，我们还是可以发现它的不足：

只能使用 GET 方法发起请求，这是由于 script 标签自身的限制决定的。

不能很好的发现错误，并进行处理。与 Ajax 对比，由于不是通过 XMLHttpRequest 进行传输，所以不能注册 success、error 等事件监听函数。

三、使用 CORS 实现跨域调用

3.1 什么是 CORS?

Cross-Origin Resource Sharing (CORS) 跨域资源共享是一份浏览器技术的规范，提供了 Web 服务从不同域传来沙盒脚本的方法，以避开浏览器的同源策略，是 JSONP 模式的现代版。与 JSONP 不同，CORS 除了 GET 要求方法以外也支持其他的 HTTP 要求。用 CORS 可以让网页设

计师用一般的 XMLHttpRequest，这种方式的错误处理比 JSONP 要来的好。另一方面，JSONP 可以在不支持 CORS 的老旧浏览器上运作。现代的浏览器都支持 CORS。

3.2 CORS 的实现

还是以 服务器 3000 上的请求页面向 服务器 3001 发送请求为例。

服务器 3000 上的请求页面 JavaScript 不变，如下：

```
$(function() {  
  $("#submit").click(function() {  
    var data = {  
      name: $("#name").val(),  
      id: $("#id").val()  
    };  
    $.ajax({  
      type: 'POST',  
      data: data,  
      url: 'http://localhost:3001/cors',  
      dataType: 'json',  
      cache: false,  
      timeout: 5000,  
      success: function(data) {  
        console.log(data)  
      },  
      error: function(jqXHR, textStatus, errorThrown) {  
        console.log('error ' + textStatus + ' ' + errorThrown);  
      }  
    });  
  });  
});
```

服务器 3001 上对应的处理函数：

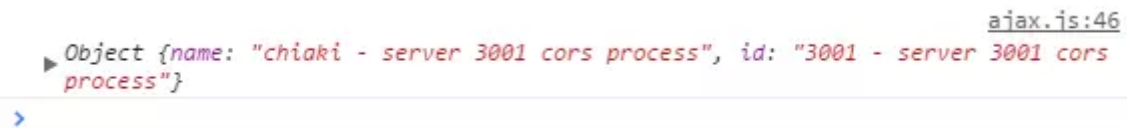
```
app.post('/cors', function(req, res) {  
  res.header("Access-Control-Allow-Origin", "*");  
  res.header("Access-Control-Allow-Headers", "X-Requested-With");  
  res.header("Access-Control-Allow-Methods", "PUT,POST,GET,DELETE,OPTIONS");  
  res.header("X-Powered-By", ' 3.2.1')  
  res.header("Content-Type", "application/json;charset=utf-8");  
  var data = {
```



```
    name: req.body.name + ' - server 3001 cors process',  
    id: req.body.id + ' - server 3001 cors process'  
  }  
  console.log(data)  
  res.send(data)  
  res.end()  
})
```

在服务器中对返回信息的请求头进行了设置。

最终的结果为：



```
Object {name: "chiaki - server 3001 cors process", id: "3001 - server 3001 cors process"}  
ajax.js:46
```

3.3 CORS 中属性的分析

1. Access-Control-Allow-Origin

The origin parameter specifies a URI that may access the resource. The browser must enforce this. For requests without credentials, the server may specify "*" as a wildcard, thereby allowing any origin to access the resource.

2. Access-Control-Allow-Methods

Specifies the method or methods allowed when accessing the resource. This is used in response to a preflight request. The conditions under which a request is preflighted are discussed above.

3. Access-Control-Allow-Headers

Used in response to a preflight request to indicate which HTTP headers can be used when making the actual request.

3.4 CORS 与 JSONP 的对比

1. CORS 除了 GET 方法外，也支持其它的 HTTP 请求方法如 POST、PUT 等。
2. CORS 可以使用 XMLHttpRequest 进行传输，所以它的错误处理方式比 JSONP 好。
3. JSONP 可以在不支持 CORS 的老旧浏览器上运作。

四、一些其它的跨域调用方式

4.1 window.name

window对象有个name属性，该属性有个特征：即在一个窗口 (window) 的生命周期内，窗口载入的所有的页面都是共享一个 window.name 的，每个页面对 window.name 都有读写的权限，window.name 是持久存在一个窗口载入过的所有页面中的，并不会因新页面的载入而进行重置。

4.2 window.postMessage()

这个方法是 HTML5 的一个新特性，可以用来向其他所有的 window 对象发送消息。需要注意的是我们必须要保证所有的脚本执行完才发送 MessageEvent，如果在函数执行的过程中调用了他，就会让后面的函数超时无法执行。

参考：<https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>

觉得本文对你有帮助？请分享给更多人
关注「前端大全」，提升前端技能

前端大全

分享前端相关技术干货 · 资讯 · 高薪职位 · 教程



微信号：FrontDev



长按识别二维码关注

伯乐在线 旗下微信公众号

商务合作QQ：2302462408