# ajax跨域问题以及解决方案

在工作中，大家应该都遇到过ajax跨域问题，浏览器的错误如下：

XMLHttpRequest cannot load http://目标地址 No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://当前页面地址' is therefore not allowed access.

## 为什么会出现跨域问题

跨域，指的是浏览器不能执行其他网站的脚本。它是由浏览器的同源策略造成的，是浏览器对[JavaScript](http://lib.csdn.net/base/javascript" \o "JavaScript知识库" \t "http://blog.csdn.net/lambert310/article/details/_blank)施加的安全限制。

在此说明一下，所谓的同源，指的是域名、协议、端口均相等。举例如下：

http://www.abc.com/a/b 调用 http://www.abc.com/d/c （非跨域）

http://www.abc.com/a/b 调用 http://www.def.com/d/c （跨域：域名不一致）

http://www.abc.com:81/a/b 调用 http://www.abc.com:82/d/c （跨域：端口不一致）

http://www.abc.com/a/b 调用 https://www.abc.com/d/c （跨域：协议不同）

请注意：localhost和127.0.0.1虽然都指向本机，但也属于跨域。

在一个http请求中，http头部Referer或Origin字段标识了当前域名，Host字段标识了此时请求的域名。

故，如果我们在当前的js页面，通过ajax请求第三方的数据，就会出现浏览器的跨域问题。

## 解决跨域问题

解决跨域问题，有如下三种方式：

1. 使用jsonp
2. 服务器代理
3. 在服务端设置response header中Access-Control-Allow-Origin字段。

### 使用jsonp

jsonp解决跨域问题的原理是，浏览器的script标签是不受同源策略限制的，我们可以在script标签中访问任何域名下的资源文件。利用这一特性，用script标签从服务器中请求数据，同时服务器返回一个带有方法和数据的js代码，请求完成，调用本地的js方法，来完成数据的处理。

前端实现，以Jquery的ajax方法为例：

$.ajax({

url:"",

dataType:'jsonp',

data:'',

jsonp:'callback', //传递给请求处理程序或页面的，用以获得jsonp回调函数名的参数名(默认为:callback)

success:function(result) {

//成功的处理

},

error:function(){

//错误处理

}

});

服务端此时返回的不能是普通的json字符串，而是一段可以被前端js执行的一段js代码。

比较一下json与jsonp格式的区别：

json格式：

{

"message":"获取成功",

"state":"1",

"result":{"name":"工作组1","id":1,"description":"11"}

}

jsonp格式：

callback({

"message":"获取成功",

"state":"1",

"result":{"name":"工作组1","id":1,"description":"11"}

})

从格式来看，jsonp是在json的基础上包装了一个方法名，此方法名是前端请求传过来的，如请求地址为：http://localhost:9999/tookApp/tbk/getItem?callback=JSONP\_CALLBACK，那么方法名就是JSONP\_CALLBACK。

下面提供一段java代码，对象转jsonp的工具类：

package com.tooklili.app.web.util;

import javax.servlet.http.HttpServletRequest;

import org.apache.commons.lang.StringUtils;

import org.springframework.web.context.request.RequestContextHolder;

import org.springframework.web.context.request.ServletRequestAttributes;

import com.fasterxml.jackson.databind.util.JSONPObject;

/\*\*

\*

\* @author ding.shuai

\* @date 2016年8月15日上午9:47:02

\*/

public class AppUtil {

/\*\*

\* 判断json字符串是否需要转化成jsonp格式

\* @param request

\* @param result

\* @return

\*/

public static Object conversionJsonp(Object result){

HttpServletRequest request = ((ServletRequestAttributes)RequestContextHolder.getRequestAttributes()).getRequest();

return conversionJsonp(request, result);

}

public static Object conversionJsonp(HttpServletRequest request,Object result){

String callback = request.getParameter("callback");

if(StringUtils.isNotEmpty(callback)){

return new JSONPObject(callback, result);

}

return result;

}

}

jsonp的缺点：

1. JSONP是一种非官方的方法，而且这种方法只支持GET方法，不如POST方法安全。（从实现机制就可明白）。
2. JSONP的实现需要服务器配合，如果是访问的是第三方的服务器，我们没有修改服务器的权限，那么这种方式是不可行的。

### 服务器代理

这种方式运用的就是服务器的反向代理技术，控制客户端和服务器的访问都从代理服务器经过，比如用nginx作为服务器代理，在nginx上配置客户端和第三方服务的反向代理，这样就可保证客户端、第三方是同源的了，同一个源，都来自代理服务器。

关于nginx的反向代理配置，可访问我的这篇博客：

<http://blog.csdn.net/csdn_ds/article/details/58605591>

服务器代理的缺点：

开发比较麻烦，对开发环境比较严格，需要在本机上配置代理服务器。

优点:

完美解决使用jsonp，第三方服务没有修改权限的问题。程序的代码侵入性小，代码级别不需要考虑跨域问题。

### 在服务端设置response header中Access-Control-Allow-Origin字段

在被请求的Response Header中加入如下代码：

// 指定允许其他域名访问

response.setHeader("Access-Control-Allow-Origin", "\*");

// 响应类型

response.setHeader("Access-Control-Allow-Methods", "POST");

// 响应头设置

response.setHeader("Access-Control-Allow-Headers", "x-requested-with,content-type");

如果所有请求都想让其他域名的服务通过浏览器ajax请求到，可以通过Filter统一设置response header。

package com.tooklili.app.web.filter;

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* 设置公共属性的过滤器

\* @author shuai.ding

\*

\* @date 2017年6月21日上午11:02:27

\*/

public class CommonSetFilter implements Filter{

@Override

public void init(FilterConfig filterConfig) throws ServletException {

}

@Override

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

throws IOException, ServletException {

//解决跨域问题

HttpServletResponse httpServletResponse =(HttpServletResponse)response;

// 指定允许其他域名访问

httpServletResponse.setHeader("Access-Control-Allow-Origin", "\*");

// 响应类型

httpServletResponse.setHeader("Access-Control-Allow-Methods", "POST");

// 响应头设置

httpServletResponse.setHeader("Access-Control-Allow-Headers", "x-requested-with,content-type");

chain.doFilter(request, response);

}

@Override

public void destroy() {

}

}

此处说明一下，笔者亲测：只设置Access-Control-Allow-Origin属性也是可以的。

Access-Control-Allow-Origin:\* 表示允许任何域名跨域访问

如果需要指定某域名才允许跨域访问，只需把Access-Control-Allow-Origin:\*改为Access-Control-Allow-Origin:允许的域名

例如：response.setHeader(“Access-Control-Allow-Origin”,”http://www.client.com”);

缺点：

1. 此种解决跨域方案，需要浏览器支持H5，因为这是HTML5解决跨域的方式，如果产品面向的是PC端，这种方式可能就不是一个好的解决方案，如果面向的是手机端，此方法不为一个简单、粗暴的好方式。
2. 设置\*，存在安全隐患。

## 总结

综上三种解决跨域的方案，个人感觉使用服务代理最好，没有破坏浏览器的安全策略，但这个对开发环境要高一点。设置response header的方式，根据具体情况分析，要考虑清楚产品面向的用户。对于jsonp这种方式，虽然没有破坏浏览器的安全策略，但只支持get方式的请求，有点不能接受，因为get传输有参数长度的限制，同时又要考虑传输中文的乱码问题，但如果项目中只是简单的查询、展示，这种方式还是可以考虑的。

其他好文推荐：

Access-Control-Allow-Origin 跨域设置多域名：

<http://www.jianshu.com/p/b587dd1b7086>