ajax跨域，这应该是最全的解决方案了

2017-03-22 | 分类 blog | 标签 ajax 前端 跨域 | 浏览 2027

前言

从刚接触前端开发起，`跨域`这个词就一直以很高的频率在身边重复出现，一直到现在，已经调试过N个跨域相关的问题了，16年时也整理过一篇相关文章，但是感觉还是差了点什么，于是现在重新梳理了一下。

个人见识有限，如有差错，请多多见谅，欢迎提出issue，另外看到这个标题，请勿喷~

题纲

关于跨域，有N种类型，本文只专注于`ajax请求跨域`(,ajax跨域只是属于浏览器"同源策略"中的一部分,其它的还有Cookie跨域iframe跨域,LocalStorage跨域等这里不做介绍)，内容大概如下:

- 什么是ajax跨域
  - 原理
  - 表现(整理了一些遇到的问题以及解决方案)
- 如何解决ajax跨域
  - JSONP方式
  - CORS方式
  - 代理请求方式

- 如何分析ajax跨域

  - http抓包的分析
  - 一些示例

什么是ajax跨域

ajax跨域的原理

ajax出现请求跨域错误问题,主要原因就是因为浏览器的“同源策略”,可以参考

浏览器同源政策及其规避方法(阮一峰)

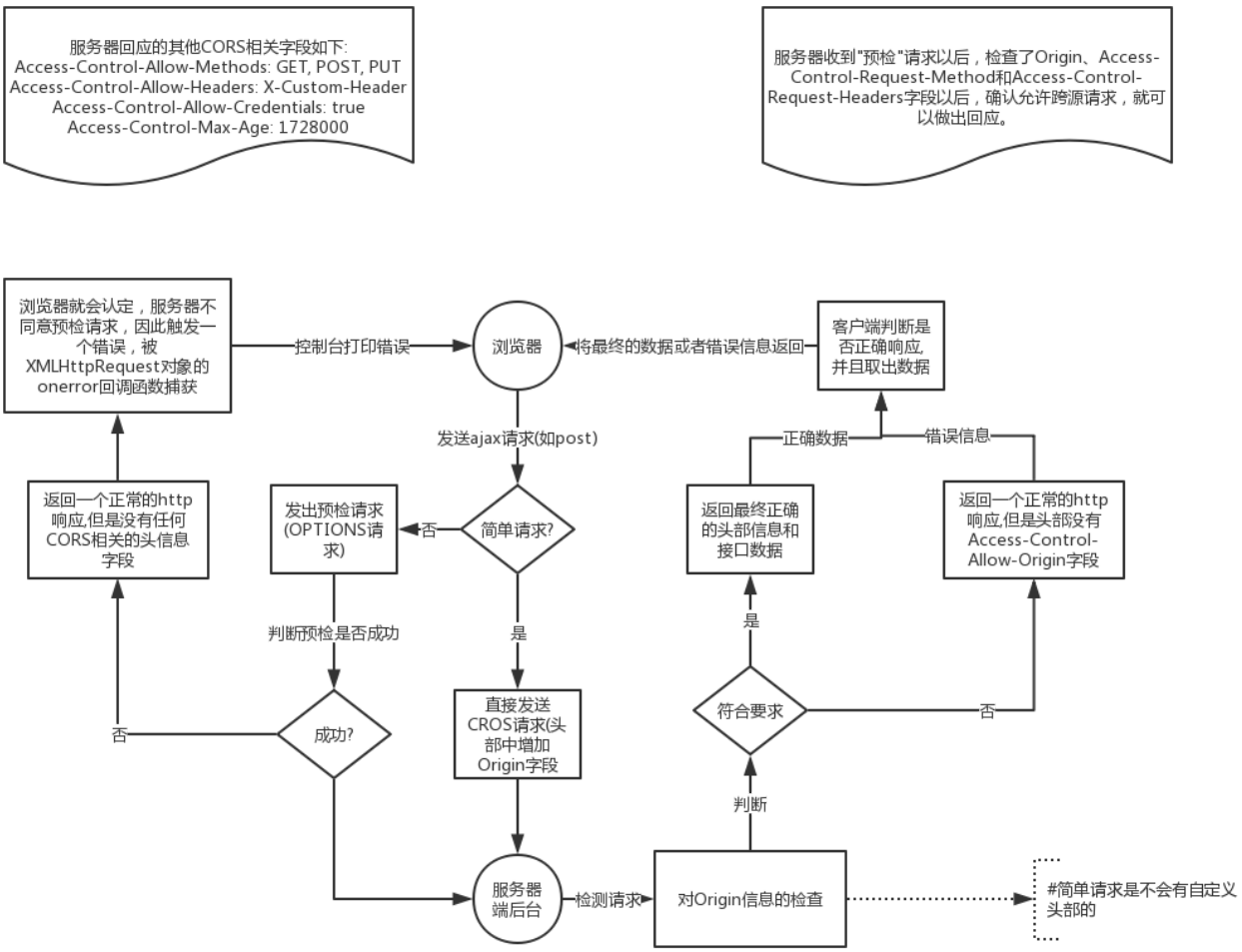
CORS请求原理

CORS是一个W3C标准，全称是”跨域资源共享”（Cross-origin resource sharing）。它允许浏览器向跨源服务器，发出XMLHttpRequest请求，从而克服了AJAX只能同源使用的限制。

基本上目前所有的浏览器都实现了CORS标准,其实目前几乎所有的浏览器ajax请求都是基于CORS机制的,只不过可能平时前端开发人员并不关心而已(所以说其实现在CORS解决方案主要是考虑后台该如何实现的问题)。

关于CORS，强烈推荐阅读 跨域资源共享 CORS 详解(阮一峰)

另外，这里也整理了一个实现原理图(简化版):



如何判断是否是简单请求?

浏览器将CORS请求分成两类：简单请求（simple request）和非简单请求（not-so-simple request）。只要同时满足以下两大条件，就属于简单请求。

- 请求方法是以下三种方法之一：HEAD,GET,POST
- HTTP的头信息不超出以下几种字段:
  - Accept
  - Accept-Language
  - Content-Language
  - Last-Event-ID
  - Content-Type(只限于三个值application/x-www-form-urlencoded、 multipart/form-data、text/plain)

凡是不同时满足上面两个条件，就属于非简单请求。

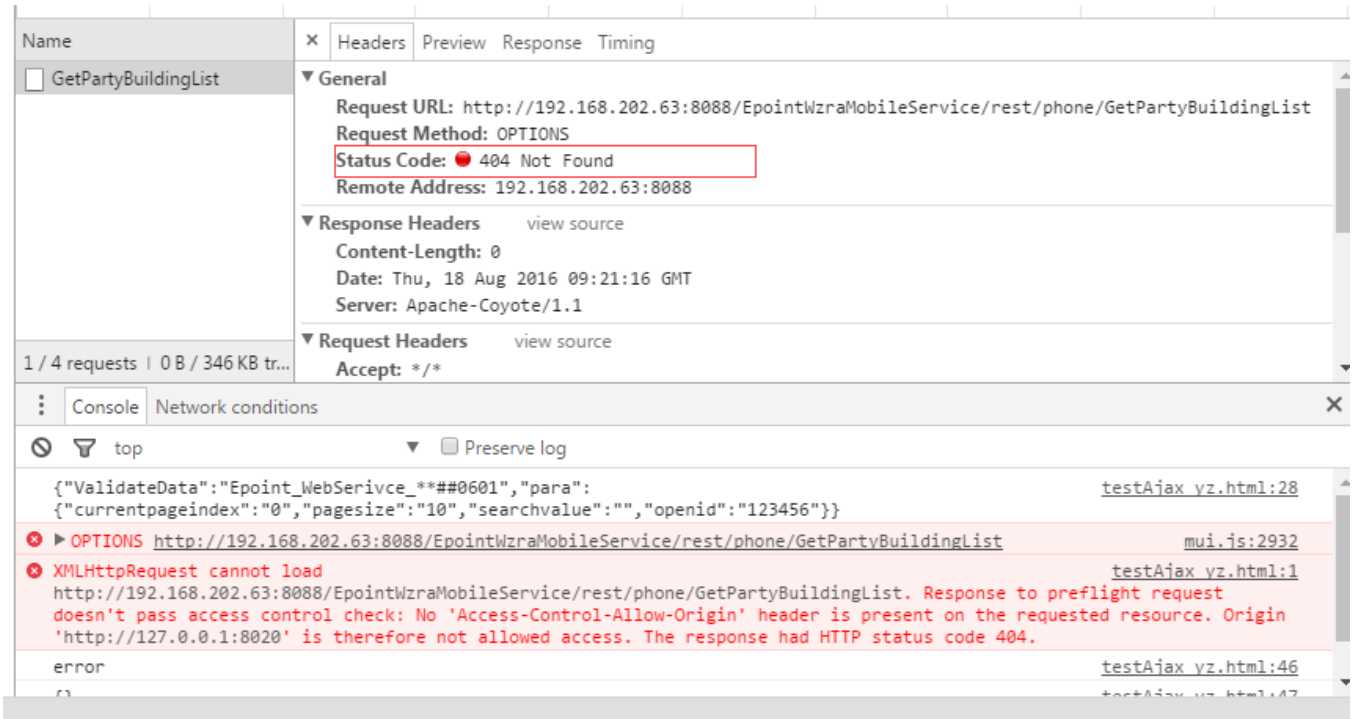
ajax跨域的表现

说实话，当初整理过一篇文章，然后作为了一个解决方案，但是后来发现仍然有很多人还是不会。无奈只能耗时又耗力的调试。然而就算是我来分析，也只会根据对应的表现来判断是否是跨域，因此这一点是很重要的。

ajax请求时,如果存在跨域现象,并且没有进行解决,会有如下表现:(注意，是ajax请求，请不要说为什么http请求可以，而ajax不行，因为ajax是伴随着跨域的，所以仅仅是http请求ok是不行的)

注意:具体的后端跨域配置请看题纲位置。

第一种现象: `No 'Access-Control-Allow-Origin' header is present on the requested resource` **,并且** `The response had HTTP status code 404`

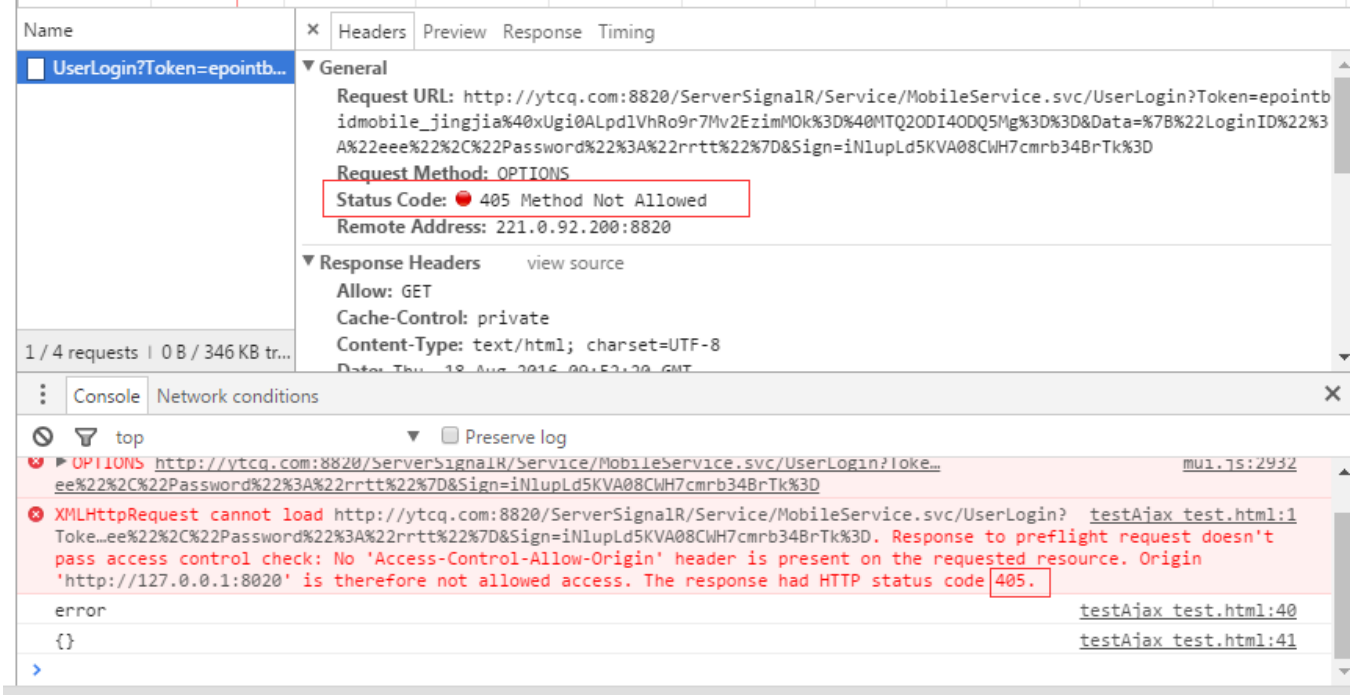


出现这种情况的原因如下:

- 本次ajax请求是“非简单请求”,所以请求前会发送一次预检请求(OPTIONS)
- 服务器端后台接口没有允许OPTIONS请求,导致无法找到对应接口地址

解决方案: 后端允许options请求

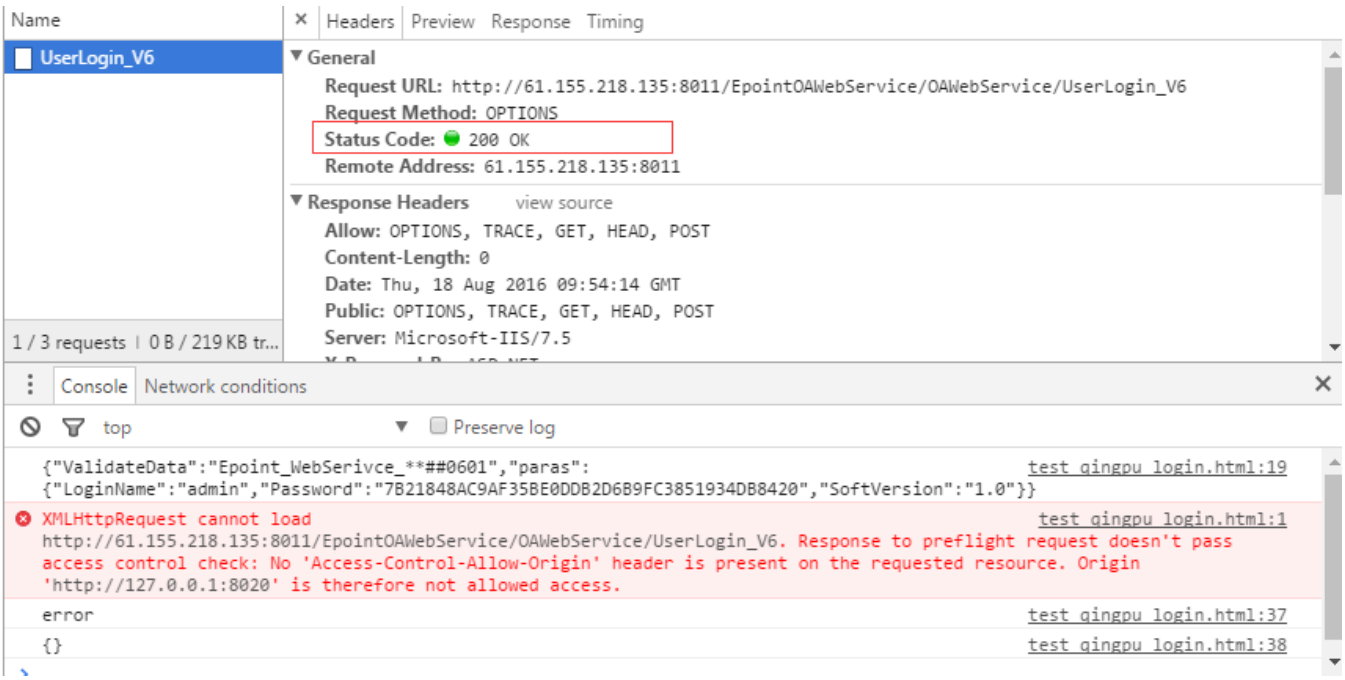
第二种现象: `No 'Access-Control-Allow-Origin' header is present on the requested resource` **,并且** `The response had HTTP status code 405`



这种现象和第一种有区别,这种情况下，后台方法允许OPTIONS请求,但是一些配置文件中(如`安全配置`),阻止了OPTIONS请求,才会导致这个现象

解决方案: 后端关闭对应的安全配置

第三种现象:`No 'Access-Control-Allow-Origin' header is present on the requested resource`**,并且**`status 200`

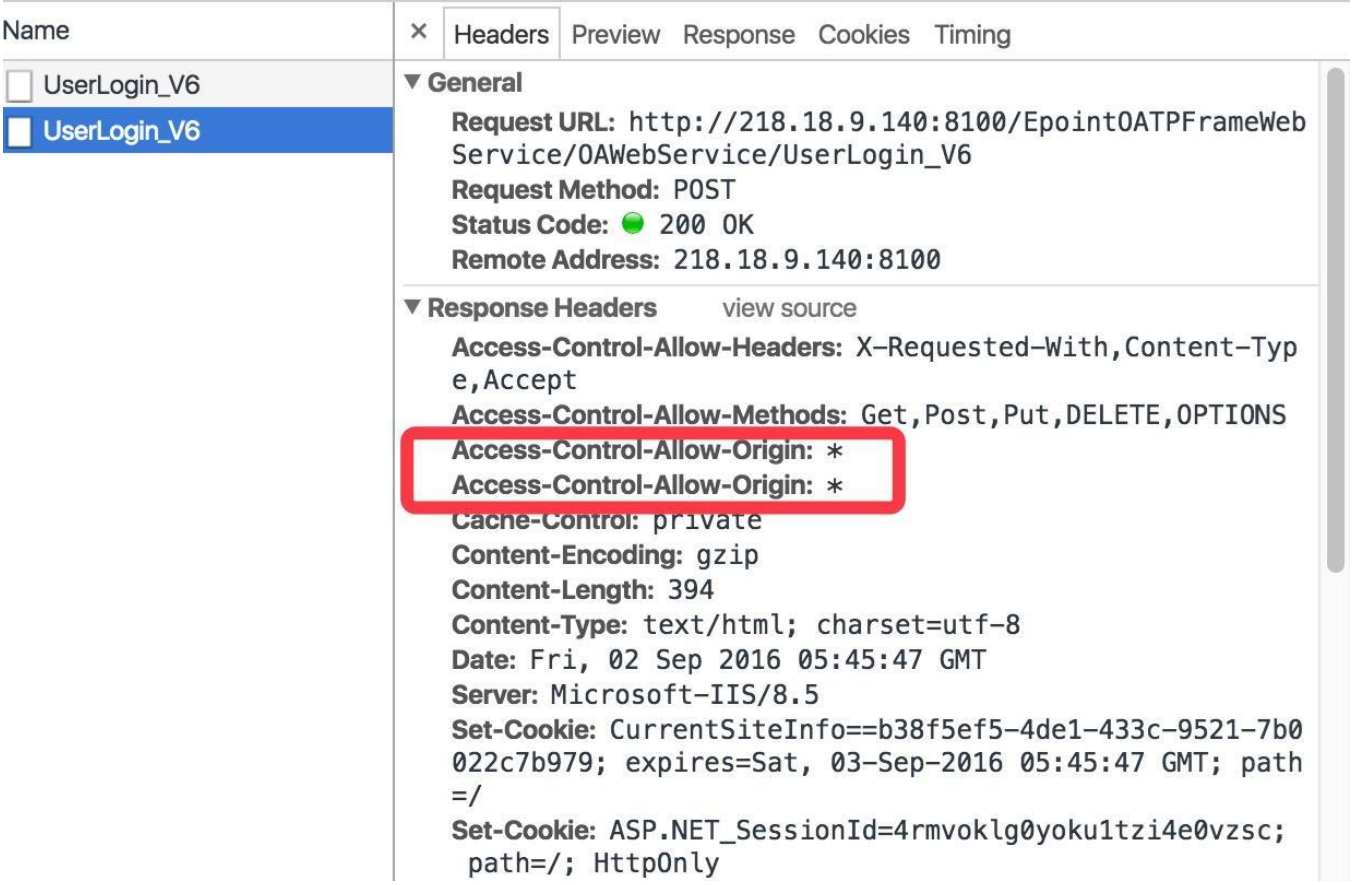


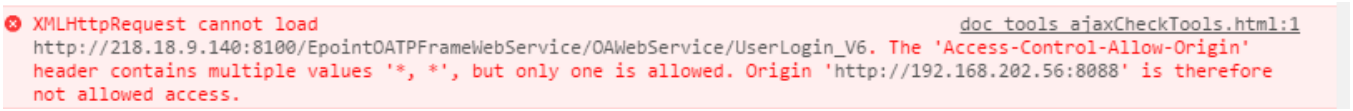
这种现象和第一种和第二种有区别,这种情况下，服务器端后台允许OPTIONS请求,并且接口也允许OPTIONS请求,但是头部匹配时出现不匹配现象

比如origin头部检查不匹配,比如少了一些头部的支持(如常见的X-Requested-With头部),然后服务端就会将response返回给前端,前端检测到这个后就触发XHR.onerror,导致前端控制台报错

解决方案: 后端增加对应的头部支持

第四种现象:`heade contains multiple values '*,*'`





表现现象是，后台响应的http头部信息有两个`Access-Control-Allow-Origin:*`

说实话，这种问题出现的主要原因就是进行跨域配置的人不了解原理，导致了重复配置，如:

- 常见于.net后台(一般在web.config中配置了一次origin,然后代码中又手动添加了一次origin(比如代码手动设置了返回*))
- 常见于.net后台(在IIS和项目的webconfig中同时设置Origin:*)

解决方案(一一对应):

- 建议删除代码中手动添加的*，只用项目配置中的即可
- 建议删除IIS下的配置*，只用项目配置中的即可

如何解决ajax跨域

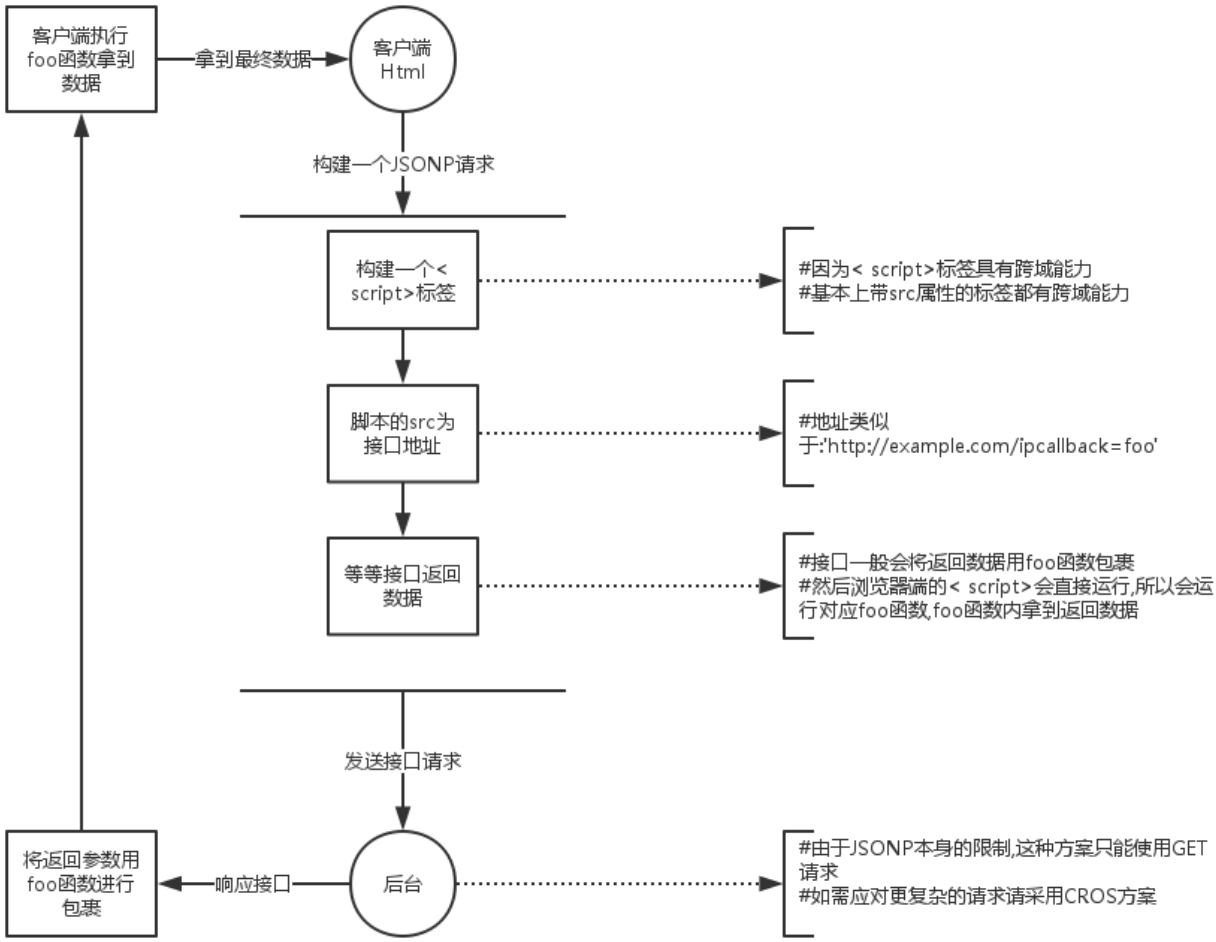
一般ajax跨域解决就是通过JSONP解决或者CORS解决,如以下:(注意，现在已经几乎不会再使用JSONP了，所以JSONP了解下即可)

JSONP方式解决跨域问题

jsonp解决跨域问题是一个比较古老的方案(实际中不推荐使用),这里做简单介绍(实际项目中如果要使用JSONP,一般会使用JQ等对JSONP进行了封装的类库来进行ajax请求)

实现原理

JSONP之所以能够用来解决跨域方案,主要是因为 `<script>` 脚本拥有跨域能力,而JSONP正是利用这一点来实现。具体原理如图



实现流程

JSONP的实现步骤大致如下(参考了来源中的文章)

- 客户端网页网页通过添加一个 `<script>` 元素，向服务器请求JSON数据，这种做法不受同源政策限制

```
function addScriptTag(src) {
  var script = document.createElement('script');
  script.setAttribute("type","text/javascript");
  script.src = src;
  document.body.appendChild(script);
}
```

```
window.onload = function () {
  addScriptTag('http://example.com/ip?callback=foo');
}

function foo(data) {
  console.log('response data: ' + JSON.stringify(data));
};
```

请求时,接口地址是作为构建出的脚本标签的src的,这样,当脚本标签构建出来时,最终的src是接口返回的内容

- 服务端对应的接口在返回参数外面添加函数包裹层

```
foo({
  "test": "testData"
});
```

- 由于`<script>`元素请求的脚本，直接作为代码运行。这时，只要浏览器定义了foo函数，该函数就会立即调用。作为参数的JSON数据被视为JavaScript对象，而不是字符串，因此避免了使用JSON.parse的步骤。

注意,一般的JSONP接口和普通接口返回数据是有区别的,所以接口如果要做JSONO兼容,需要进行判断是否有对应callback关键字参数,如果有则是JSONP请求,返回JSONP数据,否则返回普通数据

使用注意

基于JSONP的实现原理,所以JSONP只能是“GET”请求,不能进行较为复杂的POST和其它请求,所以遇到那种情况,就得参考下面的CORS解决跨域了(所以如今它也基本被淘汰了)

CORS解决跨域问题

CORS的原理上文中已经介绍了，这里主要介绍的是，实际项目中，后端应该如何配置以解决问题(因为大量项目实践都是由后端进行解决的)，这里整理了一些常见的后端解决方案:

PHP后台配置

PHP后台得配置几乎是所有后台中最为简单的,遵循如下步骤即可:

- 第一步:配置Php 后台允许跨域

```
<?php
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, A
//主要为跨域CORS配置的两大基本信息,Origin和headers
```

- 第二步:配置Apache web服务器跨域(httpd.conf中)

原始代码

```
<Directory />
    AllowOverride none
    Require all denied
</Directory>
```

改为以下代码

```
<Directory />
    Options FollowSymLinks
    AllowOverride none
    Order deny,allow
    Allow from all
</Directory>
```

Node.js后台配置(express框架)

Node.js的后台也相对来说比较简单就可以进行配置。只需用express如下配置:

```
app.all('*', function(req, res, next) {
    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')
        //这段仅仅为了方便返回json而已
    res.header("Content-Type", "application/json;charset=utf-8");
    if(req.method == 'OPTIONS') {
        //让options请求快速返回
        res.sendStatus(200);
    } else {
        next();
    }
});
```

JAVA后台配置

JAVA后台配置只需要遵循如下步骤即可:

- 第一步:获取依赖jar包

  下载 cors-filter-1.7.jar, java-property-utils-1.9.jar 这两个库文件放到lib目录下。(放到对应项目的webcontent/WEB-INF/lib/下)

- 第二步:如果项目用了Maven构建的,请添加如下依赖到pom.xml中:(非maven请忽视)

```
<dependency>
    <groupId>com.thetransactioncompany</groupId>
    <artifactId>cors-filter</artifactId>
    <version>[ version ]</version>
</dependency>
```

其中版本应该是最新的稳定版本,CORS过滤器

- 第三步:添加CORS配置到项目的Web.xml中(App/WEB-INF/web.xml)

```
<!-- 跨域配置-->
<filter>
        <!-- The CORS filter with parameters -->
        <filter-name>CORS</filter-name>
        <filter-class>com.thetransactioncompany.cors.CORSFilter</filter-class>

        <!-- Note: All parameters are options, if omitted the CORS
             Filter will fall back to the respective default values.
          -->
        <init-param>
            <param-name>cors.allowGenericHttpRequests</param-name>
            <param-value>true</param-value>
        </init-param>

        <init-param>
            <param-name>cors.allowOrigin</param-name>
            <param-value>*</param-value>
        </init-param>

        <init-param>
            <param-name>cors.allowSubdomains</param-name>
            <param-value>false</param-value>
        </init-param>

        <init-param>
            <param-name>cors.supportedMethods</param-name>
            <param-value>GET, HEAD, POST, OPTIONS</param-value>
        </init-param>
```

```
        <init-param>
            <param-name>cors.supportedHeaders</param-name>
            <param-value>Accept, Origin, X-Requested-With, Content-Type, Last-M
        </init-param>

        <init-param>
            <param-name>cors.exposedHeaders</param-name>
            <!--这里可以添加一些自己的暴露Headers   -->
            <param-value>X-Test-1, X-Test-2</param-value>
        </init-param>

        <init-param>
            <param-name>cors.supportsCredentials</param-name>
            <param-value>true</param-value>
        </init-param>

        <init-param>
            <param-name>cors.maxAge</param-name>
            <param-value>3600</param-value>
        </init-param>

    </filter>

    <filter-mapping>
        <!-- CORS Filter mapping -->
        <filter-name>CORS</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

请注意,以上配置文件请放到web.xml的前面,作为第一个filter存在(可以有多个filter的)

- 第四步:可能的安全模块配置错误(注意，某些框架中-譬如公司私人框架，有安全模块的,有时候这些安全模块配置会影响跨域配置，这时候可以先尝试关闭它们)

JAVA Spring Boot配置

20171230补充

仅列举简单的全局配置

```
 @Configuration
public class CorsConfig {

    private CorsConfiguration buildConfig() {
        CorsConfiguration corsConfiguration = new CorsConfiguration();

        // 可以自行筛选
        corsConfiguration.addAllowedOrigin("*");
        corsConfiguration.addAllowedHeader("*");
        corsConfiguration.addAllowedMethod("*");
```

```
        return corsConfiguration;
    }

    @Bean
    public CorsFilter corsFilter() {
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfiguration$

        source.registerCorsConfiguration("/**", buildConfig());

        return new CorsFilter(source);
    }
}
```

新建配置，然后添加Configuration注解即可配置成功

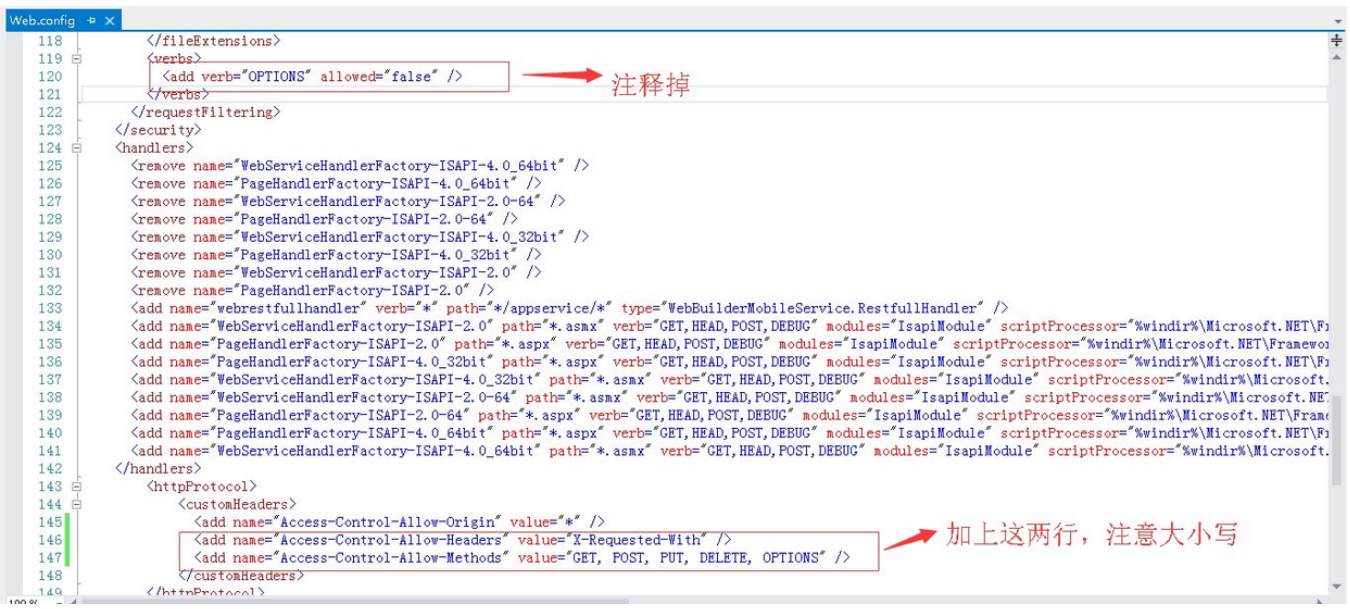
PS：这一部分方法是收录的，没有亲身实践过，但根据反馈，理论上可行

NET后台配置

.NET后台配置可以参考如下步骤:

- 第一步:网站配置

打开控制面板，选择管理工具,选择iis;右键单击自己的网站，选择浏览;打开网站所在目录,用记事本打开web.config文件添加下述配置信息,重启网站



请注意,以上截图较老,如果配置仍然出问题,可以考虑增加更多的headers允许,比如:

```
"Access-Control-Allow-Headers":"X-Requested-With,Content-Type,Accept,Origin"
```

- 第二步:其它更多配置,如果第一步进行了后,仍然有跨域问题，可能是:
    - 接口中有限制死一些请求类型(比如写死了POST等)，这时候请去除限 制
    - 接口中，重复配置了`Origin:*`，请去除即可
    - IIS服务器中，重复配置了`Origin:*`，请去除即可

代理请求方式解决接口跨域问题

注意，由于接口代理是有代价的，所以这个仅是开发过程中进行的。

与前面的方法不同，前面CORS是后端解决，而这个主要是前端对接口进行代理，也就是:

- 前端ajax请求的是本地接口
- 本地接口接收到请求后向实际的接口请求数据，然后再将信息返回给前端
- 一般用node.js即可代理

关于如何实现代理，这里就不重点描述了，方法和多，也不难，基本都是基于node.js的。

搜索关键字`node.js`,`代理请求`即可找到一大票的方案。

OPTIONS预检的优化

```
Access-Control-Max-Age:
```

这个头部加上后，可以缓存此次请求的秒数。

在这个时间范围内，所有同类型的请求都将不再发送预检请求而是直接使用此次返回的头作为判断依据。

非常有用，可以大幅优化请求次数

如何分析ajax跨域

上述已经介绍了跨域的原理以及如何解决，但实际过程中，发现仍然有很多人对照着类似的文档无法解决跨域问题，主要体现在，前端人员不知道什么时候是跨域问题造成的，什么时候不是，因此这里稍微介绍下如何分析一个请求是否跨域:

抓包请求数据

第一步当然是得知道我们的ajax请求发送了什么数据，接收了什么，做到这一步并不难，也不需要`fiddler`等工具，仅基于`Chrome`即可

- `Chrome`浏览器打开对应发生ajax的页面，`F12`打开`Dev Tools`
- 发送ajax请求
- 右侧面板->`NetWork`->`XHR`，然后找到刚才的ajax请求，点进去

示例一(正常的ajax请求)

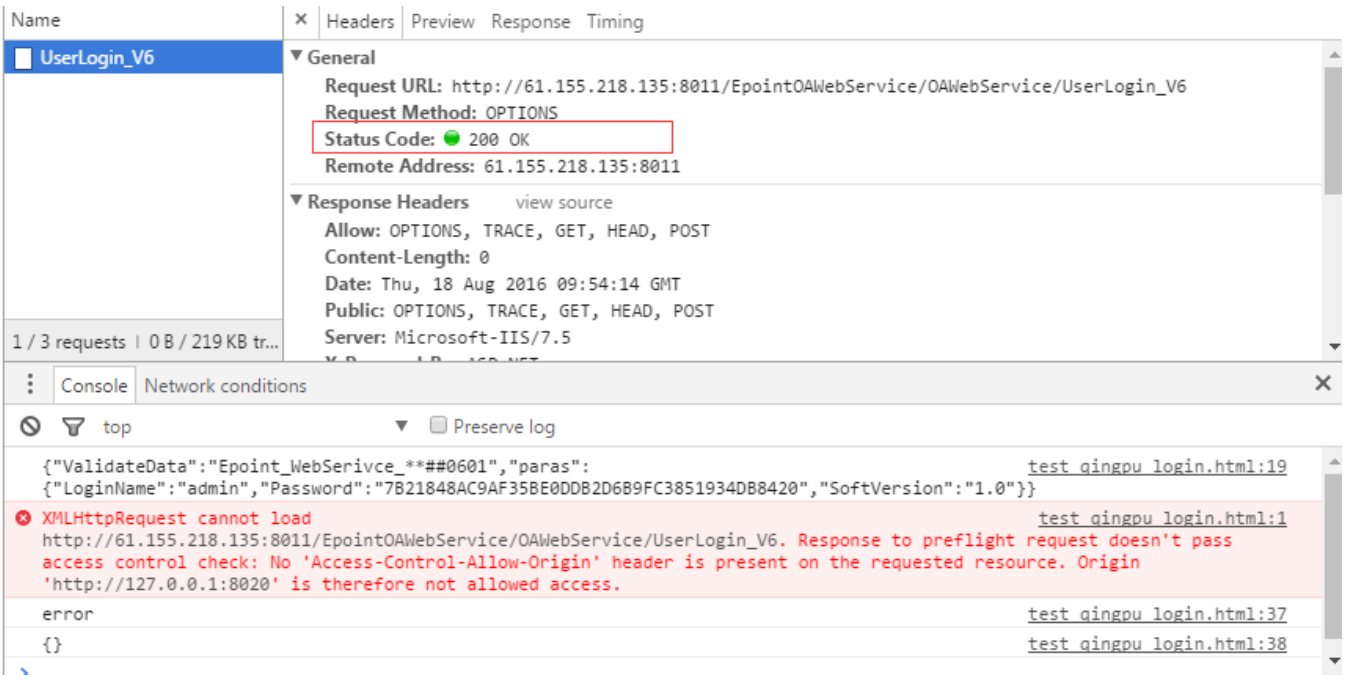
上述请求是一个正确的请求，为了方便，我把每一个头域的意思都表明了，我们可以清晰的看到，接口返回的响应头域中，包括了

```
 Access-Control-Allow-Headers: X-Requested-With,Content-Type,Accept
Access-Control-Allow-Methods: Get,Post,Put,OPTIONS
Access-Control-Allow-Origin: *
```

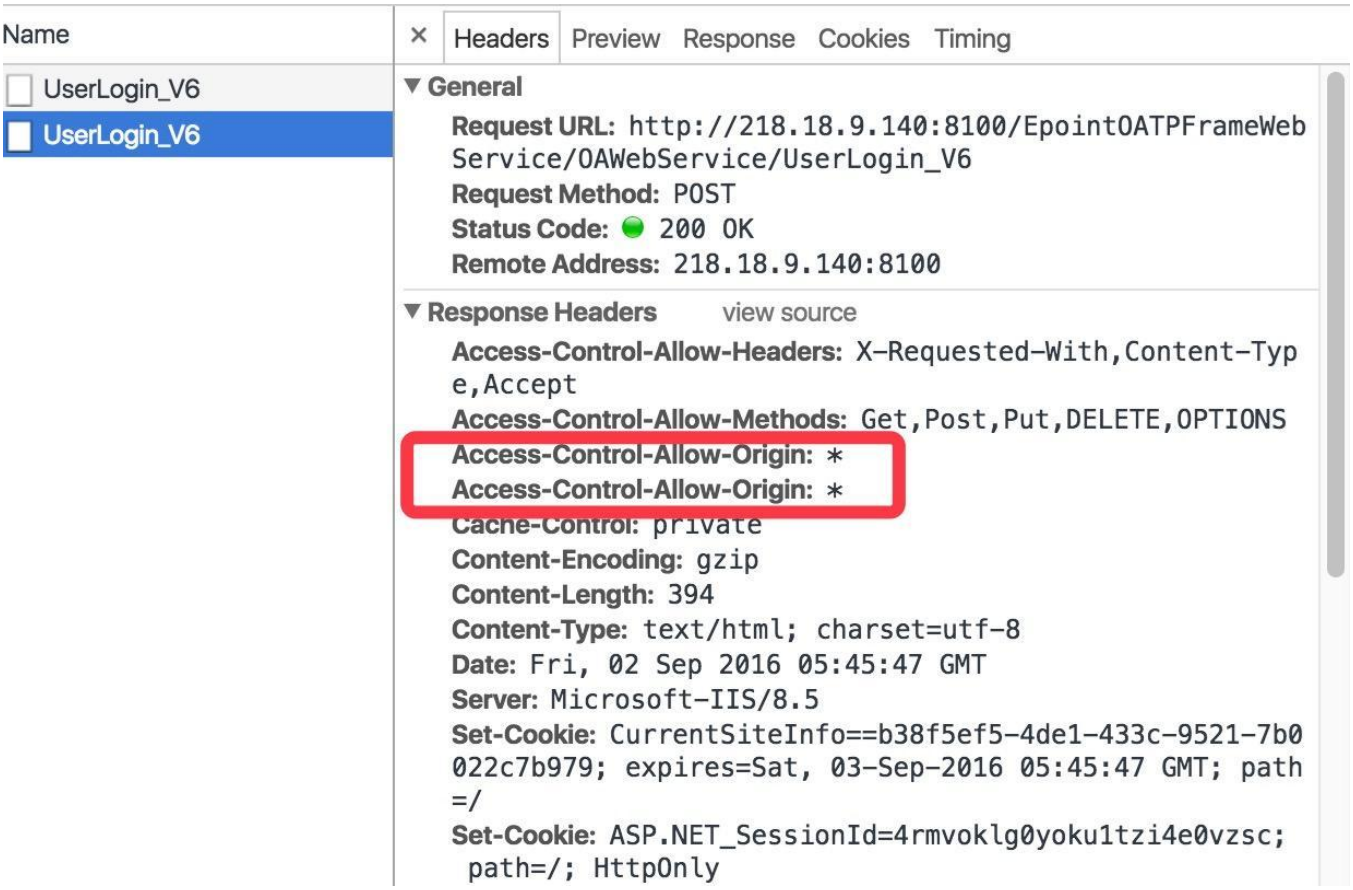
所以浏览器接收到响应时，判断的是正确的请求，自然不会报错，成功的拿到了响应数据。

示例二(跨域错误的ajax请求)

为了方便，我们仍然拿上面的错误表现示例举例。



这个请求中，接口Allow里面没有包括`OPTIONS`，所以请求出现了跨域、

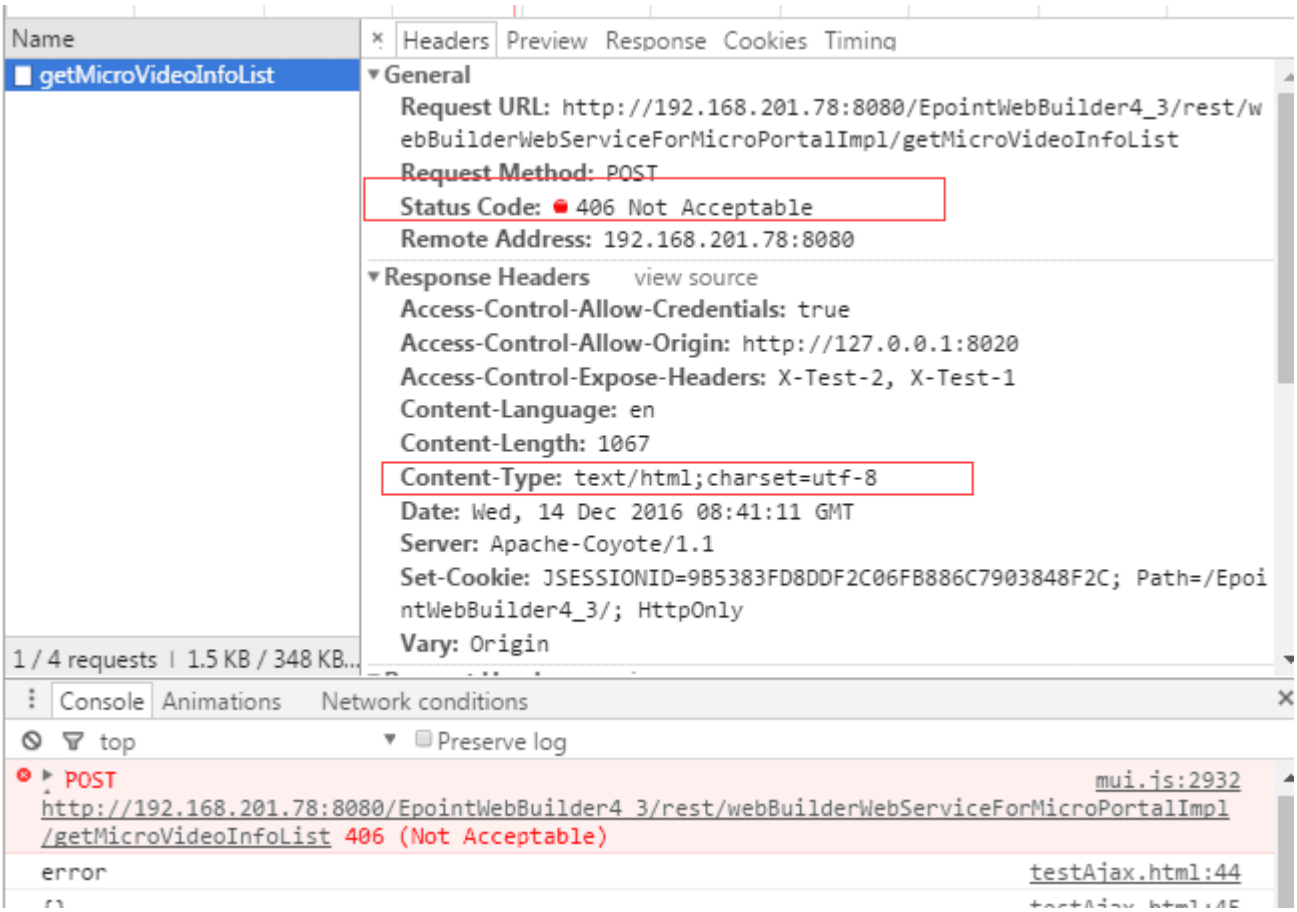


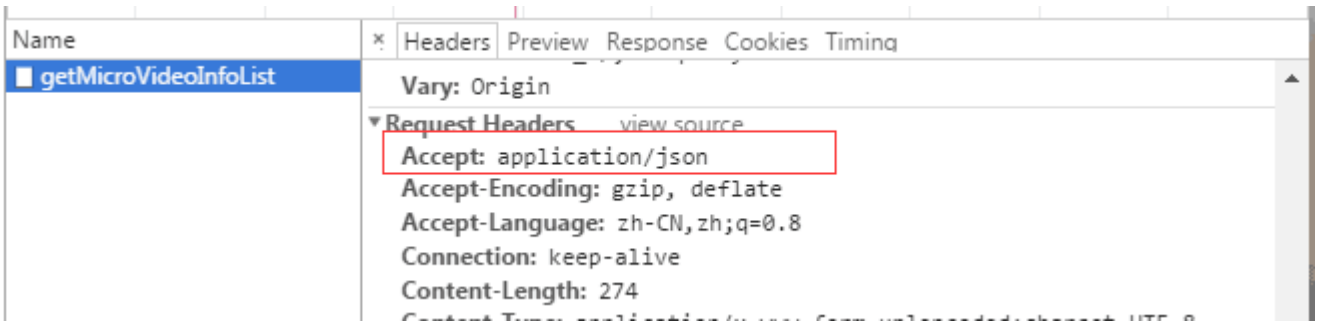
这个请求中，`Access-Control-Allow-Origin: *`出现了两次，导致了跨域配置没有正确配置，出现了错误。

更多跨域错误基本都是类似的，就是以上三样没有满足(Headers,Allow,Origin)，这里不再一一赘述。

示例三(与跨域无关的ajax请求)

当然，也并不是所有的ajax请求错误都与跨域有关，所以请不要混淆，比如以下:





比如这个请求，它的跨域配置没有一点问题，它出错仅仅是因为request的`Accept`和response的`Content-Type`不匹配而已。

更多

基本上都是这样去分析一个ajax请求，通过`Chrome`就可以知道了发送了什么数据，收到了什么数据，然后再一一比对就知道问题何在了。

写在最后的话

跨域是一个老生常谈的话题，网上也有大量跨域的资料，并且有不少精品(比如阮一峰前辈的)，但是身为一个前端人员不应该浅尝而止，故而才有了本文。

漫漫前端路，望与诸君共勉之！

附录

参考资料

- 浏览器同源政策及其规避方法(阮一峰)
- 跨域资源共享 CORS 详解(阮一峰)
- 本人之前在cnblog上的文章

博客

初次发布`2017.03.22`于个人博客

- http://www.dailichun.com/2017/03/22/ajaxCrossDomainSolution.html

上一篇　下一篇