浏览器中的HTTP请求

XMLHttpRequest

XHR对象用于与服务器交互。通过`XMLHttpRequest`可以在不刷新页面的情况下请求特定URL，获取数据。
`XMLHttpRequest`在`AJAX`编程中被大量使用。

MDN文档

Fetch

Fetch API提供了一个获取资源的接口（包括跨域请求）

MDN文档

AJAX

Asynchronous JavaScript And XML，是一种使用`XMLHttpRequest`技术构建更复杂，动态的网页的编程实践。大部分的ajax其实就是对`XMLHttpRequest`的相关API进行封装，使其使用起来更加方便。

MDN文档

跨域

跨域，顾名思义，跨越区域。大概意思为访问的网站请求非同源资源。

当前页面URL	被请求资源URL	跨域	原因
http://www.test.com	http://www.test.com/api/users	否	同源(协议 域名 端口号相同)
http://www.test.com	https://www.test.com/api/users	是	协议不同(http/https)
http://www.test.com	http://www.baidu.com/api/users	是	主域名不同(test/baidu)
http://www.test.com	http://blog.test.com/api/users	是	子域名不同(www/blog)
http://www.test.com:8080	http://www.test.com:7070/api/user	是	端口号不同(8080/7070)

为什么会有跨域

遇事先问为什么。所以，浏览器为什么要设置跨域的限制，然后我们后面还要费心费力地消除跨域的限制。

为了web生态的安全。看一个例子：

假设浏览器里面的代码可以随意访问第三方的数据（非同源），那么你可以让你的代码定时轮询访问一个非同源的网页，假设某个时刻恰好有10万人在访问你的网页，那这个第三方的网页每秒就要承受10万的并发量，这样网络中大量的带宽就会被这样白白的浪费掉，整个web生态将会混乱无比。

当访问的网站需要请求非同源资源时，浏览器将拒绝这些非同源请求。在这种情况下，我们需要解决浏览器跨域时拒绝请求非同源资源的限制。

当浏览器出现跨域时，那就不可避免的引出两个关键的概念了。**简单请求**和**非简单请求**。

当跨域产生时，**非简单请求**会在真正向服务端发送请求前进行**预检请求**（OPTIONS）。

简单请求

1、条件定义：若请求满足以下**所有的条件**，则请求可视为**简单请求**。

- 使用下列方法之一：

1. GET
2. HEAD
3. POST

- 请求首部字段不得超出以下集合

1. Accept
2. Accept-Language
3. Content-Language
4. Conent-Type：text/plain || multipart/form-data || application/x-www-form-urlencoded
5. DPR
6. Downlink
7. Save-Data
8. Viewport-Width
9. Width

- 请求中的任意XMLHttpRequestUpload 对象均没有注册任何事件监听器
- 请求中没有使用 ReadableStream 对象

非简单请求

1、条件定义：若请求满足下列任一条件时，即应首先发送预检请求（options）。

- 使用了下面的任一方法：

1. PUT
2. DELETE
3. CONNECT
4. OPTIONS
5. TRACE
6. PATCH

- 设置了额外的请求首部字段（除去以下集合中的）

1. Accept
2. Accept-Language
3. Content-Language
4. Conent-Type：text/plain || multipart/form-data || application/x-www-form-urlencoded
5. DPR
6. Downlink
7. Save-Data

8. Viewport-Width
9. Width

- 请求中的XMLHttpRequestUpload 对象注册了任意多个事件监听器
- 请求中使用了ReadableStream对象

解决跨域的方案

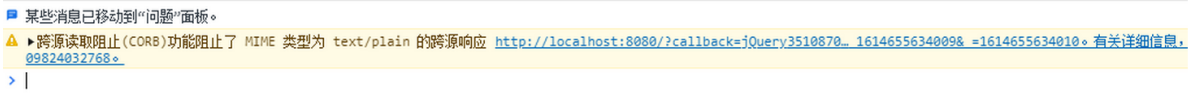
jsonp

JSON with Padding，是JSON的一种使用模式，可以让网页从别的网域获取资料。由于同源策略，一般来说位于server1.example.com的网页无法与不是server1.example.com的服务器沟通，**而HTML的元素是一个例外。**利用元素的这个开放策略，网页就可以实现跨域获取后端接口数据。

由于使用script标签的src属性，**因此只支持get方法。**

当使用JSONP这种方案时，前后端都要有相对应的写法。
大致流程就是，前端通过标签的src属性向后台接口发起请求（只支持GET请求），并且传递参数`callback='response'`，与此同时，前端必须定义函数`response(responseData)`，这是用来处理接口返回数据后一些操作。
当接口收到请求，返回数据格式为`response(responseData)`。这样，当前端接受到数据`response(responseData)`，就刚好执行了我们已经定义好的`response(...)`

当报错如下时：



原因是：后端接口没有返回`callback(...)`

维基百科

JSONP的原理和实现

CORS

Cross Origin Resource Sharing，跨域资源共享，由一系列传输的HTTP头组成，这些HTTP头决定浏览器是否阻止前端JavaScript代码获取跨域请求的响应。

MDN文档

1、Access-Control-Allow-Origin：指示请求的资源能共享给哪些域

2、Access-Control-Allow-Credentials：指示当请求的凭证标记为`true`时，是否响应该请求

3、Access-Control-Allow-Headers：用在对预请求的响应中，哪些HTTP方法允许访问请求的资源

4、Access-Control-Expose-Headers：指示哪些HTTP头的名称能在响应中列出

5、Access-Control-Max-Age：指示预请求的结果能被缓存多久

6、Access-Control-Request-Headers：用于发起一个预请求，告知服务器正式请求会使用哪些HTTP头

7、Access-Control-Request-Method：用于发起一个预请求，告知服务器正式请求会使用哪一种HTTP请求方法

8、Origin：指示获取资源的请求是从什么域发起的

koa2中接口允许跨域响应，响应头部字段设置如下：

```
ctx.set('Access-Control-Allow-Origin', '*');
ctx.set('Access-Control-Allow-Methods', 'POST, GET, OPTIONS, DELETE, PUT');
ctx.set('Access-Control-Allow-Headers', 'X-Requested-With, User-Agent, Referer,
Content-Type, Cache-Control,accesstoken');
ctx.set('Access-Control-Max-Age', '86400');
ctx.set('Access-Control-Allow-Credentials', 'true');
```

注意事项:
若添加了自定义的Header字段，必须将这个字段名添加到服务端响应头部Access-Control-Allow-Headers中，不然会报错。

项目踩坑:
在接口响应中添加了以上允许跨域响应的头部字段，但是在开发中还报了跨域的错误（`Response to preflight request doesn't pass access control check: Redirect is not allowed for a preflight request`）,报错的大致意思是**预检请求禁止重定向**。经过排查，发现是服务端nginx做了HTTP到HTTPS的重定向设置，而我恰好是以**http+ip地址**的形式发起请求的，那么请求就被重定向到**https**了，然而，浏览器发起的预检请求是禁止重定向的，因此报错了。解决方案就是将请求地址改为https+域名的形式，这样预检请求就不会重定向了。