Ajax

环境：Node.js

学习前提：js、jses-6 、jquery

Ajax

1.原生AJAX

- 1.1 AJAX简介
- 1.2 XML简介
- 1.3 AJAX的特点
  - 1.3.1 AJAX的优点
  - 1.3.2 AJAX的缺点
- 1.4 AJAX-HTTP协议请求报文与响应文本结构
  - HTTP
  - 请求报文
  - 响应报文
- 1.5Chrome网络控制台查看通信报文
- 1.6NodeJS的安装
- 1.7express框架介绍和基本使用
  - 1.7.1安装
  - 1.7.2使用
- 1.8AJAX发送SET请求
  - 1.8.1案例准备
  - 关闭端口
- 1.9AJAX请求的基本操作
  - 1.创建 XMLHttpRequest 对象
  - 2.向服务器发送请求
  - 3.onreadystatechange 事件
  - 案例继续：
- 1.9AJAX设置请求参数
- 1.10AJAX发送POST请求
- 1.11AJAX设置请求头信息
  - **解决自定义头报错问题：**
- 1.12AJAX服务端响应JSON数据
- 1.13nodemon自动重启工具安装
- 1.14AJAX-IE缓存问题解决
- 1.15AJAX请求超时与网络异常处理
- 1.16AJAX取消请求
- 1.17AJAX请求重复发送问题

2.jQuery中的AJAX

- 2.1 get请求
- 2.2 post请求
- 2.3jQuery通用方法发送AJAX请求

3.Axios发送AJAX请求

- 3.1GET请求
- 3.2POST请求
- 3.3Axios函数请求

4.fetch发送AJAX请求

5.跨域

- 5.1同源策略
- 5.2如何解决跨域
  - 5.2.1 JSONP
- 5.3JSONP案例

5.4jquery发送jsonp请求
5.5CORS

1.原生AJAX

1.1 AJAX简介

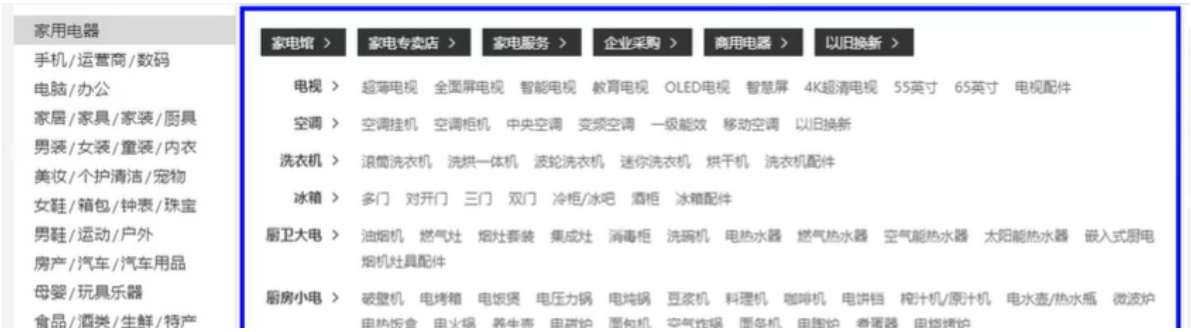
AJAX全称为Asynchronous JavaScript And XML，就是异步的JS和XML.

通过AJAX可以在浏览器中向服务器发送异步请求，**最大的优势:无刷新获取数据。**

AJAX不是新的编程语言，而是"一种将现有的标准组合在一起使用的新方式"。

如输入关键字出现相关关键字

又如二级标题内容展开



1.2 XML简介

是什么？ XML是可扩展标记语言

设计目的？ XML被设计用来传输和存储数据。

XML和HTML类似，不同的是HTML中都是预定义标签，而XML中没有预定义标签，全都是自定义标签，用来表示一些数据。

如：

有一个学生数据:

```
name = "孙悟空"; age= 18 ; gender="男";
```

用XML表示:

```
<student>
    <name>孙悟空</name>
    <age>18</age>
    <gender>男</gender>
</student>
```

但现在已经被JSON取代，JSON更简洁更灵活

用JSON表示:

```
{"name":"孙悟空","age":18,"gender":"男"}
```

1.3 AJAX的特点

1.3.1 AJAX的优点

1)可以无需刷新页面而与服务器端进行通信。

2)允许你根据用户事件(如鼠标、键盘等事件)来更新部分页面内容。

1.3.2 AJAX的缺点

1)没有浏览历史，不能回退

2)存在跨域问题(同源)（两个网页之间的服务器不能通信，后面会讲解如何解决跨域问题）

3) SEO（搜索引擎优化） 不友好（无法通过爬虫获取）

1.4 AJAX-HTTP协议请求报文与响应文本结构

HTTP

HTTP (hypertext transport protocol) 协议[**超文本传输协议**]，协议详细规定了浏览器和万维网服务器之间互相通信的规则。可以将协议简单理解为约定或规则。

主要掌握请求报文和响应报文这两个的格式和参数。

请求报文

由四部分组成：行、头、空行、体

```
行 包括3部分内容
    1.请求类型：GET/POST(这两种用的比较多) 2.url路径 3.HTTP版本（有1.0/2.0，用的最多的是1.1）
    POST /all?keyword=ajax HTTP/1.1
头   格式特点：key: value
    Host: atguigu.com
    Cookie: name=guigu
    Content-type: application/x-www-form-urlencoded //请求类型
    User-Agent: chrome 83
空行 是固定的，必须有
体   GET请求，则请求体为空；POST请求，请求体可以不为空
    username= admin&password=admin
```

响应报文

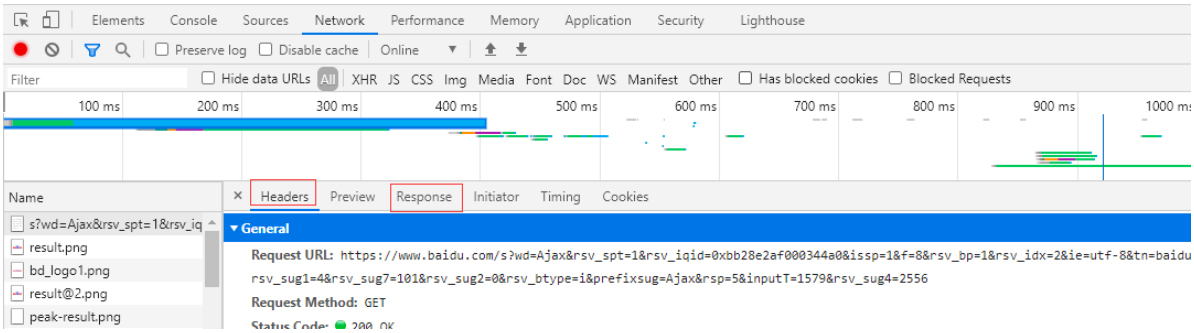
由四部分组成：行、头、空行、体

```
行  包括3部分：协议版本、响应状态码、响应状态字符串
    HTTP/1.1 200 OK      404 找不到 403 没有权限 401 未授权 200 OK等
头    格式与请求头一样
    Content-Type: text/html; charset=utf-8 //响应类型
    Content-length: 2048 //响应长度
    Content-encoding: gzip //响应编码方式
空行  是固定的，必须有
体  主要返回结果,html放在响应体里面
    <html>
        <head>
        </head>
        <body>
            <h1>尚硅谷</h1>
        </body>
    </html>
```

1.5Chrome网络控制台查看通信报文

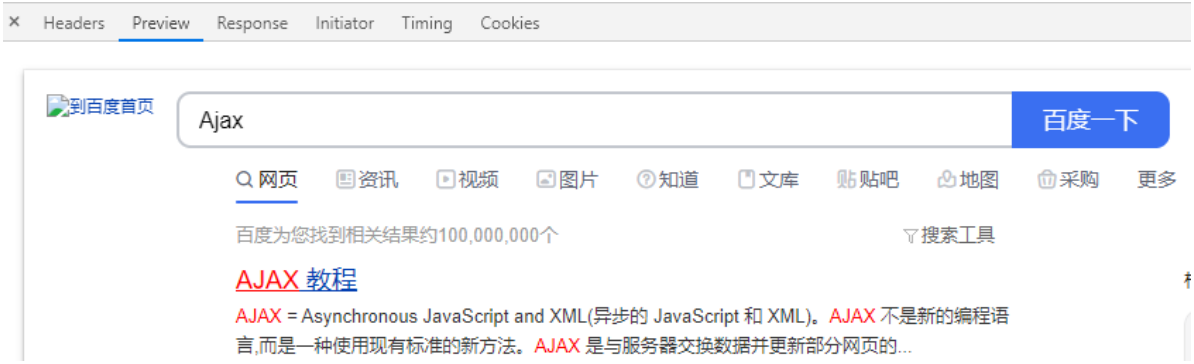
以百度搜索'AJAX'为例

F12后，点击'Network'后刷新页面，点击进入第一条内容



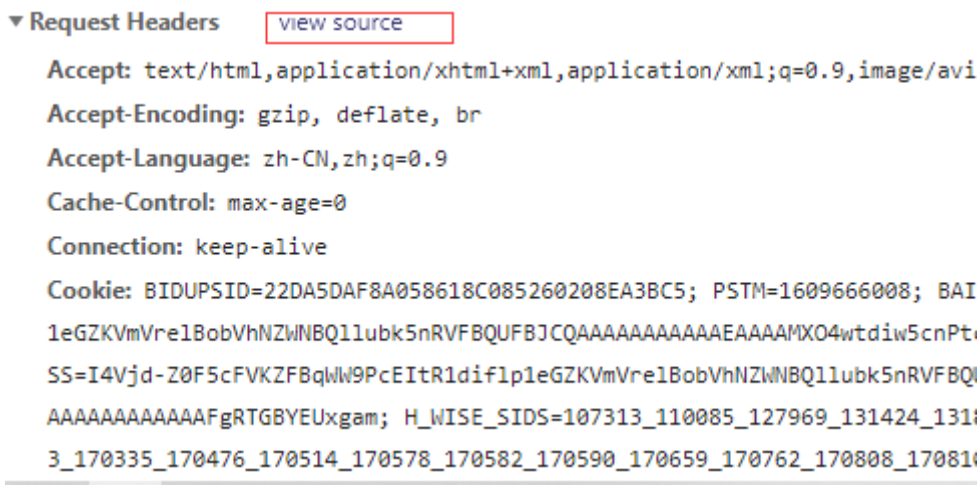
目前主要了解前三个：Headers、Preview、Response。

Preview是解析之后的预览。



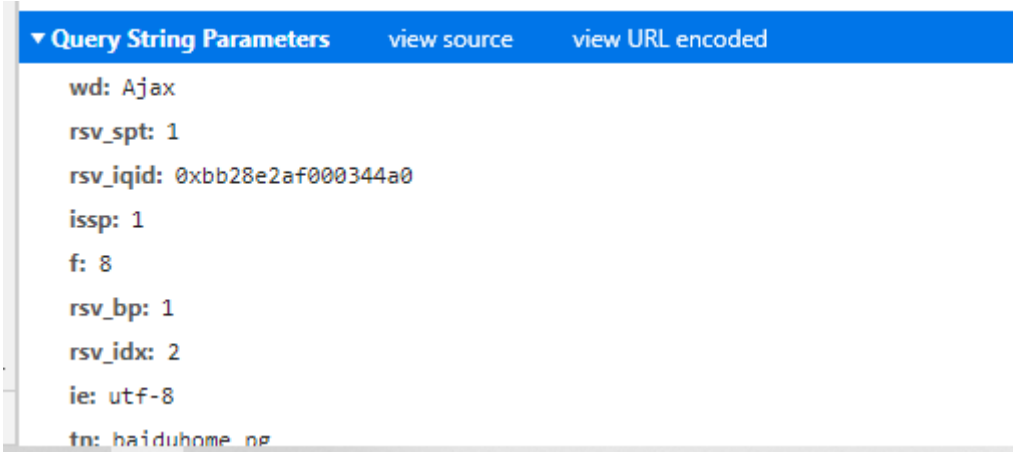
Headers主要由以下四部分组成：

1.进入**Request Headers请求头**看到：请求头的内容，格式都为 名字：内容



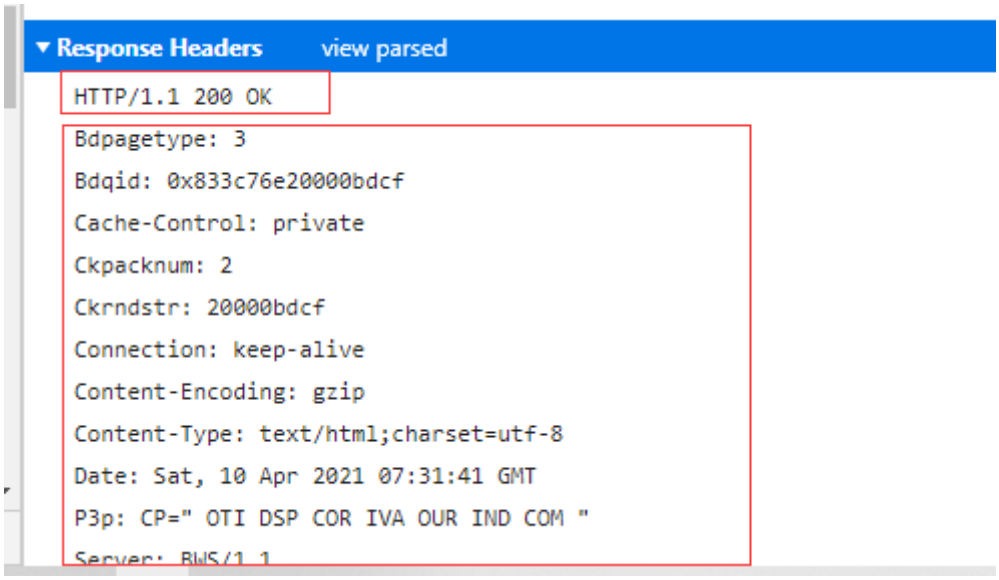
点击红色框的`view source`查看原始的请求报文信息，**请求行**的内容，如下图。因为是get请求，所以请求体为空。

2.进入**Query String Paramenters**：里面是对url（即上图里请求行里面的url）内容做的解析

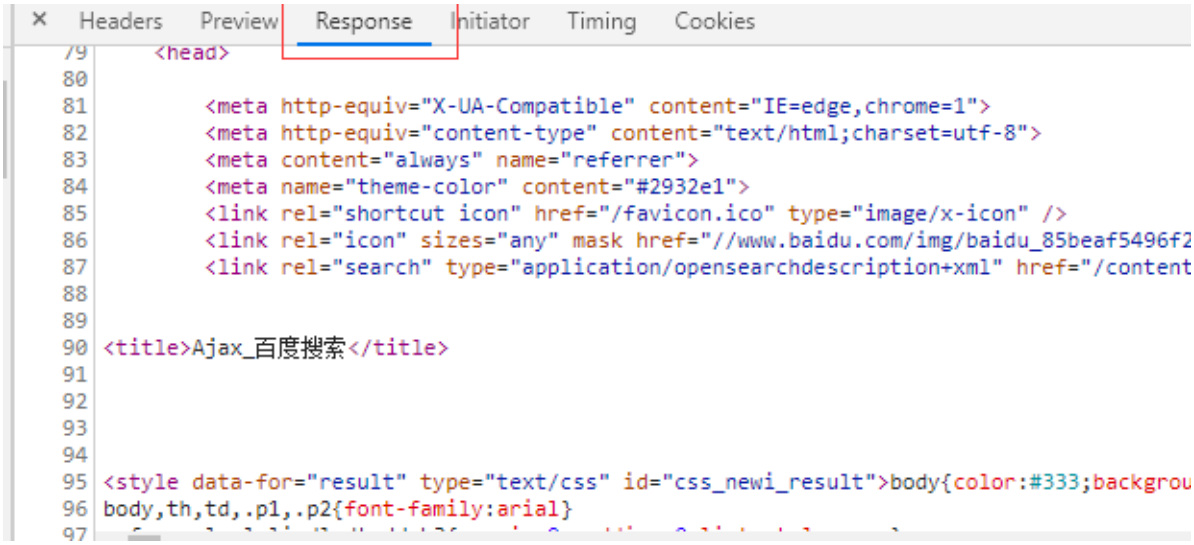


3.进入**Response Headers响应头**：响应头的内容，格式都为名字：内容

进入`view source`查看原始的响应报文，里面只有**响应行**和响应头，如下图



3.**响应体**在**Response**内

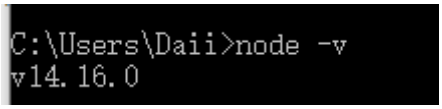


某个网站登录后，查看**请求体**内容From Data

1.6NodeJS的安装

官网安装：https://nodejs.org/en/

检验是否安装完成，命令行窗口输入`node -v`，出现数字版本号即说明安装完毕



1.7express框架介绍和基本使用

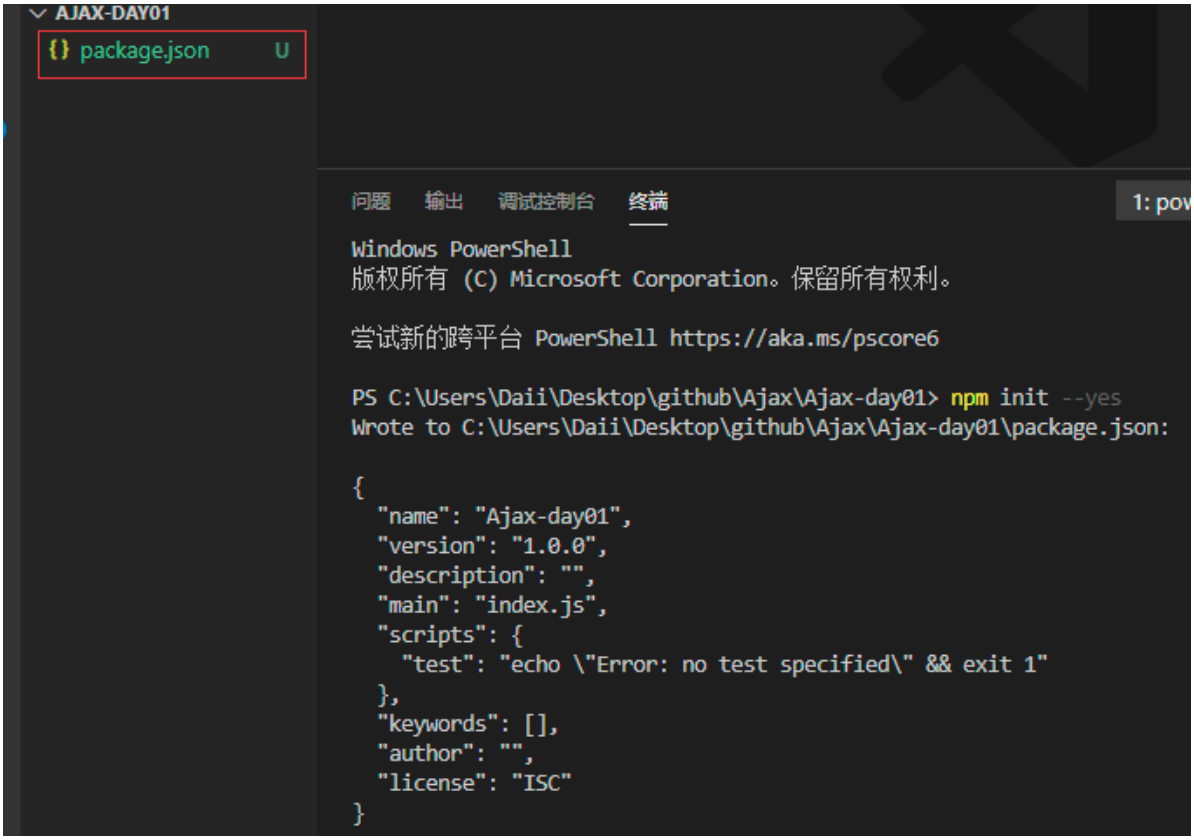
Express 是基于Node.js平台，快速、开放、极简的Web并发框架

1.7.1安装

安装方法：

1.VS中当前文件中ctrl+`打开终端

2.在终端内输入`npm init --yes`**进行初始化**，如下图，初始化后会生成一个左上角的文件



3.输入`npm i express`安装

```
PS C:\Users\Daii\Desktop\github\Ajax\Ajax-day01> npm i express

added 50 packages in 4s
```

1.7.2使用

1.新建一个js文件，在js文件内输入

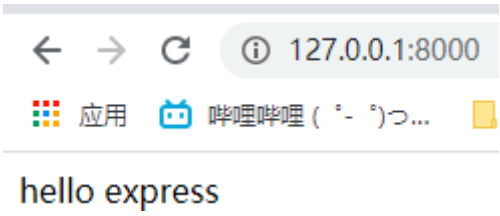
express使用的四个步骤：

```
// 1.引入express
const { response } = require('express')
const express = require('express')
// 2.创建应用对象
const app = express()
// 3.创建路由规则
// request是对请求报文的封装,request对象表示HTTP请求，包含了请求查询字符串、参数、内容、HTTP头部等属性
// response是对响应报文的封装，response对象表示 HTTP响应，即在接收到请求时向客户端发送的HTTP响应数据
app.get('/', (request, response) => {
    // 设置响应，传送HTTP响应
    response.send('hello express')
})
// 4.监听端口启动服务
app.listen(8000, () => {
    console.log('服务已经启动，8000端口监听中...');
})
```

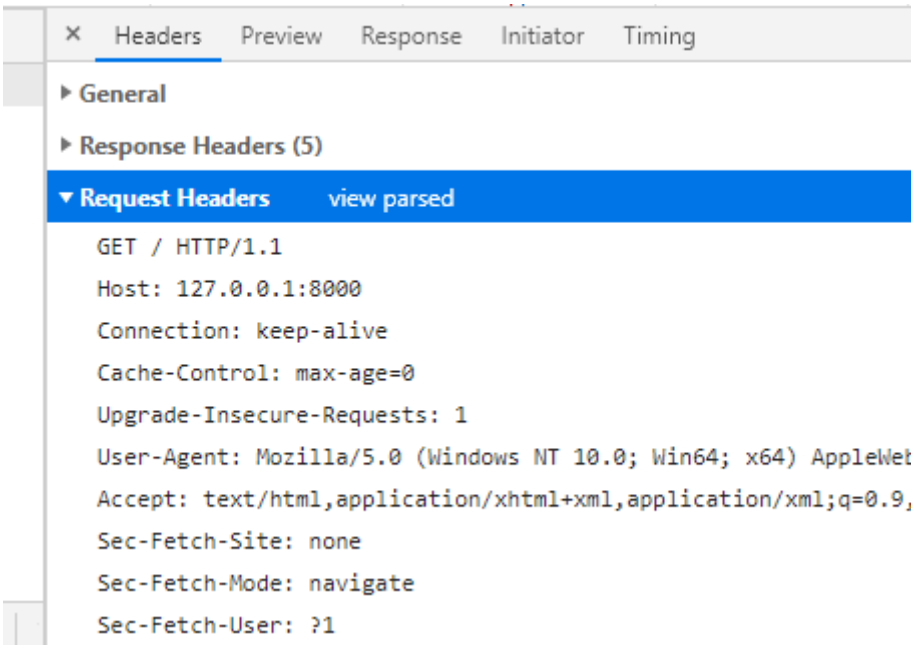
2.在VS终端内输入`node 脚本文件名`**执行以上代码**，如下图

```
added 50 packages in 4s
PS C:\Users\Daii\Desktop\github\Ajax\Ajax-day01> node 01-express的基本使用.js
服务已经启动，8000端口监听中...
```

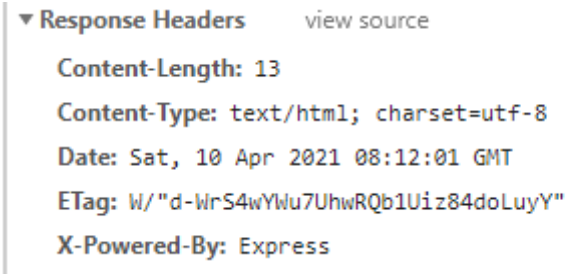
3.在浏览器输入`127.0.0.1:8000`，可以看到我们设置的响应内容



F12后查看请求头



响应头



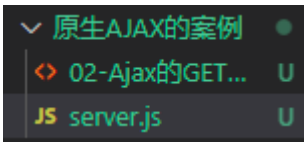
1.8AJAX发送SET请求

通过一个案例了解

案例要求：点击按钮，向服务器发送请求，将服务器响应体结果呈现在div里，不刷新页面

1.8.1案例准备

新建两个文件，一个是前端页面，一个是服务器



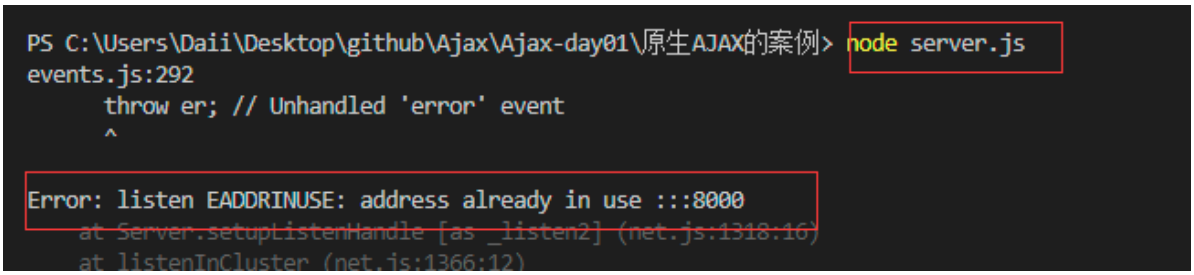
前端页面内放置按钮和div

服务器放上个express示例内容，对于路由规则做一定的修改

响应请求行的第二段内容（即url）的路径是/server则进行回调函数

```
// 3.创建路由规则
// request是对请求报文的封装
// response是对响应报文的封装
//  /server 页面响应
app.get('/server', (request, response) => {
    // 设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    // 设置响应体
    response.send('hello ajax')
})
```

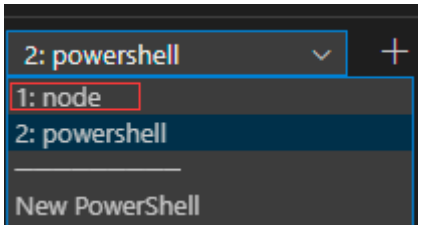
在当前页面打开终端，启动服务，结果如下图



原因：上个案例的8000端口没有关闭仍在使用中

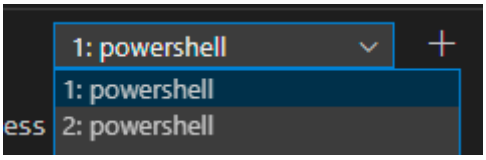
关闭端口

1.在终端的右上角，切换到node,node表示有个服务正在启动。



2.按**ctrl+c即可关闭端口**

关闭端口后可以发现，右上角没有node了



端口类似于一辆车，我占用了端口，把车开走了，别人就不能用，只能我释放了之后别人才可以用。

——

关闭端口后切换回原来的，启动服务

在浏览器中输入网址:`http://127.0.0.1:8000/server`

可以看到响应体

响应头内有我们定义的部分

1.9AJAX请求的基本操作

1.创建 XMLHttpRequest 对象

XMLHttpRequest 用于在后台与服务器交换数据

语法：

```
let xhr = new XMLHttpRequest()
```

2.向服务器发送请求

使用 XMLHttpRequest 对象的 **open() 和 send() 方法**将请求发送到服务器。

语法：

```
xhr.open("GET","url",true);
xhr.send();
```

参数说明：

1. `open(method,url,async)` 规定请求的类型、URL 以及是否异步处理请求。

`method`：请求的类型，GET 或 POST

`url`：文件在服务器上的位置，可以是任何类型的文件，如 .txt 和 .xml，或者服务器脚本文件，如 .asp 和 .php

`async`：true（异步）或 false（同步）

2. `send(string)` 将请求发送到服务器。

string：仅用于 POST 请求，格式很灵活，只要是字符串类型都可以，但更建议下面的第一种格式。

```
xhr.send('a=100&b=20&c=32') //更建议使用这种格式
// xhr.send('a:100&b:20&c:32')
// xhr.send('112318774')
```

3.onreadystatechange 事件

每当 readyState 改变时，就会触发 onreadystatechange 事件。

下面是 **XMLHttpRequest 对象的三个重要的属性**：

1. `onreadystatechange` **属性：存储函数**（或函数名），每当 readyState 属性改变时，就会调用该函数。

2. `readyState` **属性**存有 XMLHttpRequest 的状态信息，表示状态（共5个，从0到4）：

0：表示请求未初始化，也是默认值

1：表示open()调用完毕，服务器连接已建立

2：表示send()调用完毕，请求已接收

3：表示服务端返回部分结果，请求处理中

4：表示服务端返回所有结果，请求已完成

3.`status`**属性**：200: "OK" 404: 未找到页面 等　2xx都算成功

注意：

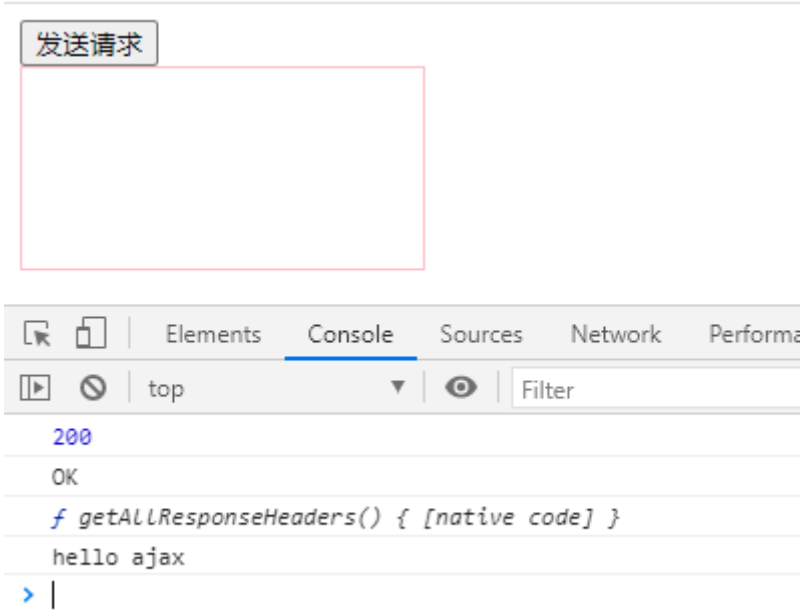
1.在 onreadystatechange 事件中，我们**规定当服务器响应已做好被处理的准备时（即readyState状态为4时）**所执行的任务。

2.onreadystatechange 事件被触发 **4 次**（0 - 4），分别是： 0-1、1-2、2-3、3-4，对应着 readyState 的每个变化。

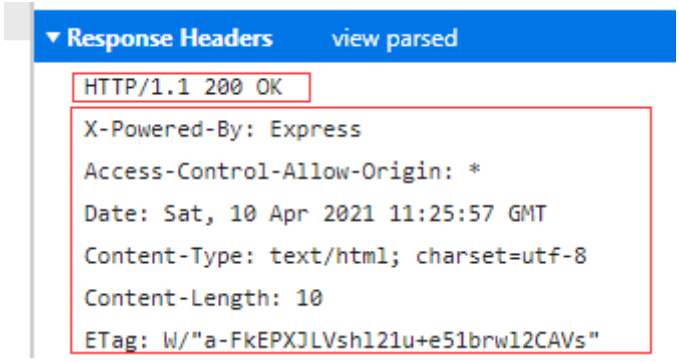
案例继续：

```
btn.onclick = function () {
            // AJAX请求的基本操作
            // 1.创建对象
            let xhr = new XMLHttpRequest()
            // 2.初始化 设置请求方法和url
            xhr.open('GET', 'http://127.0.0.1:8000/server')
            // 3.发送
            xhr.send()
            // 4.事件绑定 处理服务端返回的结果
            // onreadystatechange事件解释：
            // on 当...时
            // readystate 是xhr对象中的属性，表示状态（共5个）： 0(表示未初始化，也是默认值) 1(表示open()调用完毕) 2(表示send()调用完毕) 3(表示服务端返回部分结果) 4(表示服务端返回所有结果)
            // change 改变
            xhr.onreadystatechange = function () {
                // 服务端返回所有结果时才进行处理
                if (xhr.readyState === 4) {
                    // 再判断响应状态码，成功才处理 2xx都算成功
                    if (xhr.status >= 200 & xhr.status < 300) {
                        // 处理结果即响应报文，包括四部分：行 头 空行 体
                        // 响应行 3部分：协议版本 状态码 状态字符串
                        console.log(xhr.status); //状态码
                        console.log(xhr.statusText); //状态字符串
                        // 所有响应头
                        console.log(xhr.getAllResponseHeaders);
                        // 响应体
                        console.log(xhr.response);
                    }
                }
            }
        }
```

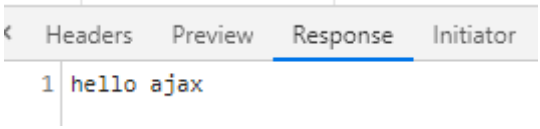
点击按钮后：



其响应头、行



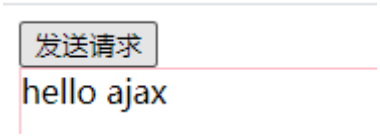
响应体：



将响应体结果放入div内

```
div.innerHTML = xhr.response
```

点击按钮后：



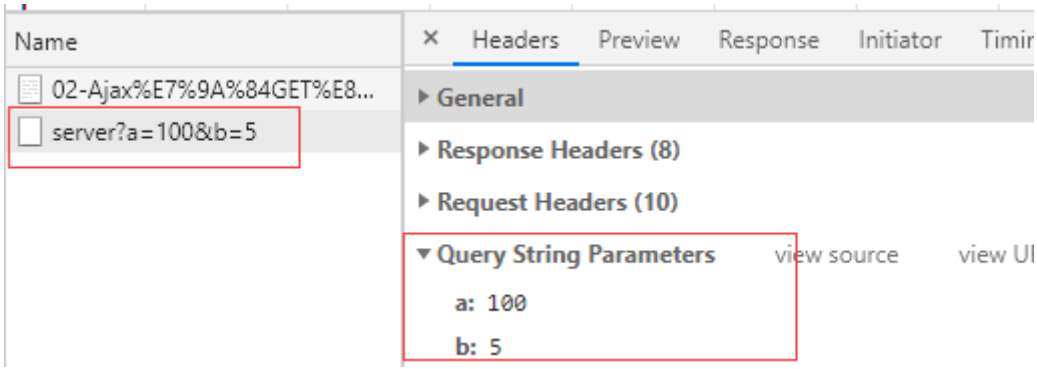
1.9AJAX设置请求参数

语法：用?分割，参数名=参数值，多个参数用&分割。

示例：

```
xhr.open('GET', 'http://127.0.0.1:8000/server?a=100&b=5')
```

可以发现参数的存在：



1.10AJAX发送POST请求

通过一个案例来了解

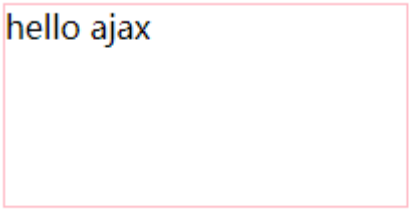
要求：鼠标经过div时，向服务端发送post请求，把响应体结果在div上呈现

与SET请求类似，也是4个AJAX请求的基本步骤。

但要**注意将服务端的路由规则修改**，SET请求对应get方法，POST请求对应post方法

注意修改后需要重启端口

```
// SET请求使用get方法
app.get('/server', (request, response) => {
    // 设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    // 设置响应体
    response.send('hello ajax')
})
// POST请求使用post方法
app.post('/server', (request, response) => {
    // 设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    // 设置响应体
    response.send('hello ajax')
})
```



1.11AJAX设置请求头信息

`setRequestHeader()` **设置请求头信息**

语法：

```
xhr.setRequestHeader("Content-type","application/x-www-form-urlencoded");
```

参数说明：header: 规定头的名称　value: 规定头的值

注意：添加自定义的头会报错，浏览器会有安全机制。上面写的头是预定义的。

解决自定义头报错问题：

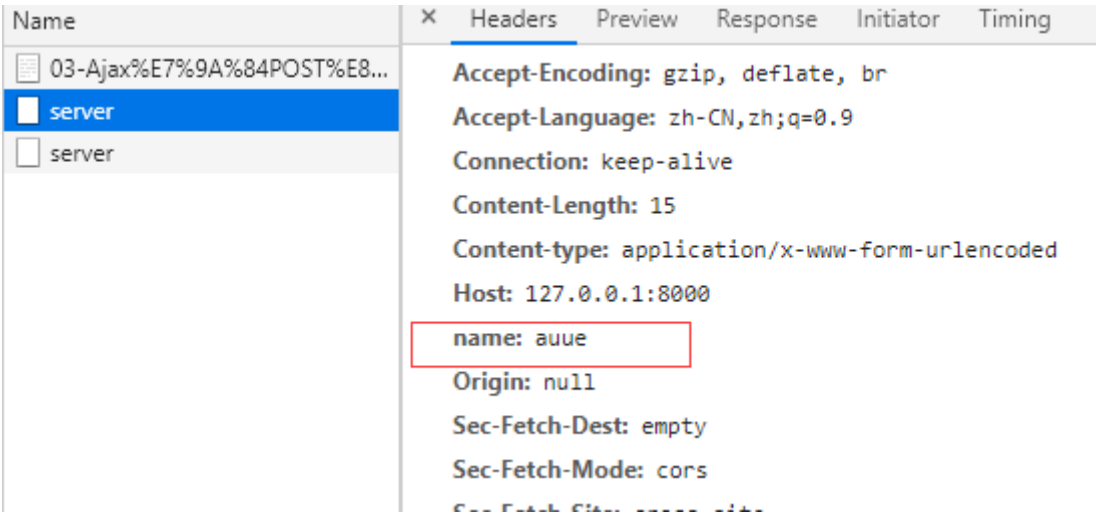
1.使用**all方法接收任意类型的请求**

2.添加`response.setHeader('Access-Control-Allow-Headers', '*')`，接收所有的响应头信息

```
// all可以接收任意类型的请求
app.all('/server', (request, response) => {
    // 设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    // 响应头，*表示接收所有的响应头信息
    response.setHeader('Access-Control-Allow-Headers', '*')
    // 设置响应体
    response.send('hello ajax')
})
```

添加自定义属性

```
xhr.setRequestHeader('name', 'auue')
```



1.12AJAX服务端响应JSON数据

通过一个案例了解

案例要求：键盘中按下任意键，向服务端发送请求，把响应体结果在div上呈现。响应体的结果为一个对象。

由于**响应体里的参数只能是字符串**类型，解决方法：

1.首先在服务端使用**JSON.stringify将其转为字符串** `response.send(JSON.stringify(data))`

2.接收端将响应体结果转换为对象类型，有以下两种方法：

1）手动转换

使用`JSON.parse`将字符型转为对象类型

```
// 1.手动转换
let data = JSON.parse(xhr.response)
div.innerHTML = data.name
console.log(data);
```

2）自动转换

设置响应体数据的类型

```
// 设置响应体数据的类型
xhr.responseType = 'json'

// 2.自动转换
div.innerHTML = xhr.response.name
console.log(xhr.response);
```

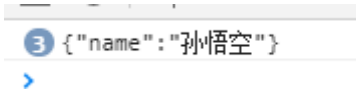
服务端：设置响应的数据

```
app.all('/json-server', (request, response) => {
    // 设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    // 响应头，*表示接收所有的响应头信息
    response.setHeader('Access-Control-Allow-Headers', '*')
    // 响应一个数据
    let data = {
        name: '孙悟空'
    }
    // 设置响应体
    // response.send('hello ajax')
    // 对对象进行字符串转换
    let str = JSON.stringify(data)
    response.send(str)
})
```

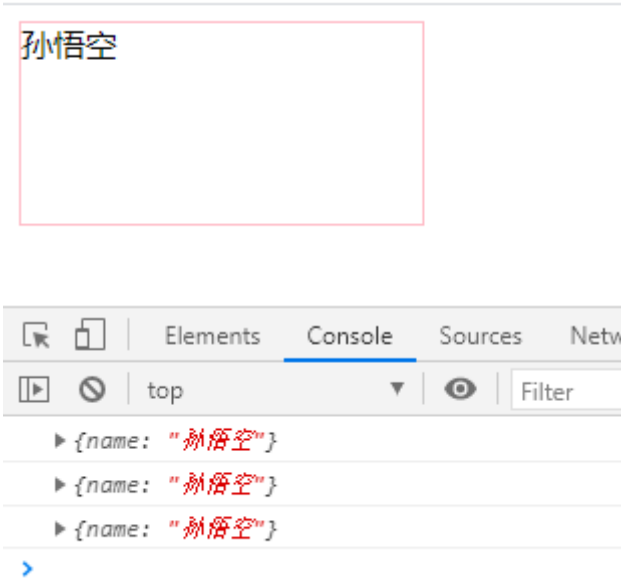
```
window.onkeydown = function () {
            let xhr = new XMLHttpRequest()
            // 设置响应体数据的类型
            xhr.responseType = 'json'
            xhr.open('GET', 'http://127.0.0.1:8000/json-server')
            xhr.send()
            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4) {
                    if (xhr.status >= 200 & xhr.status < 300) {
                        // 1.手动转换
                        /* let data = JSON.parse(xhr.response)
                        div.innerHTML = data.name
```

```
                    console.log(data); */
                    // 2.自动转换
                    div.innerHTML = xhr.response.name
                    console.log(xhr.response);
                }
            }
        }
    }
```

未转换数据类型，输出是字符串类型



转换数据类型后



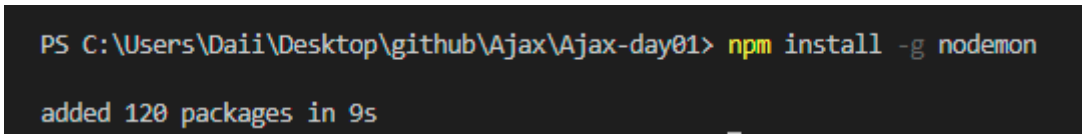
注意：发送请求的url链接修改，服务端方法里的链接页面参数也要修改。

1.13nodemon自动重启工具安装

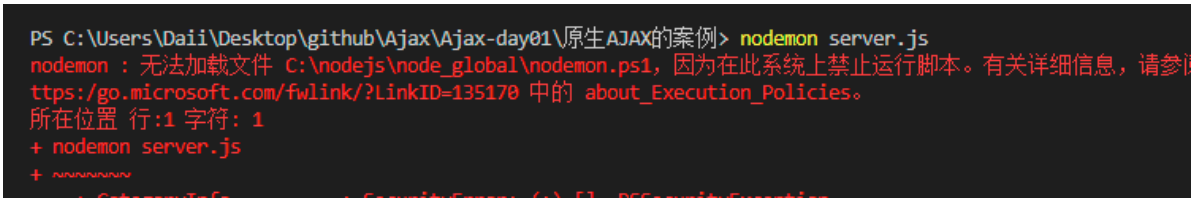
nodemon是一种工具，可以自动检测到目录中的文件更改时通过重新启动应用程序来调试基于node.js的应用程序。

安装方法：

在VS终端输入`npm install -g nodemon`

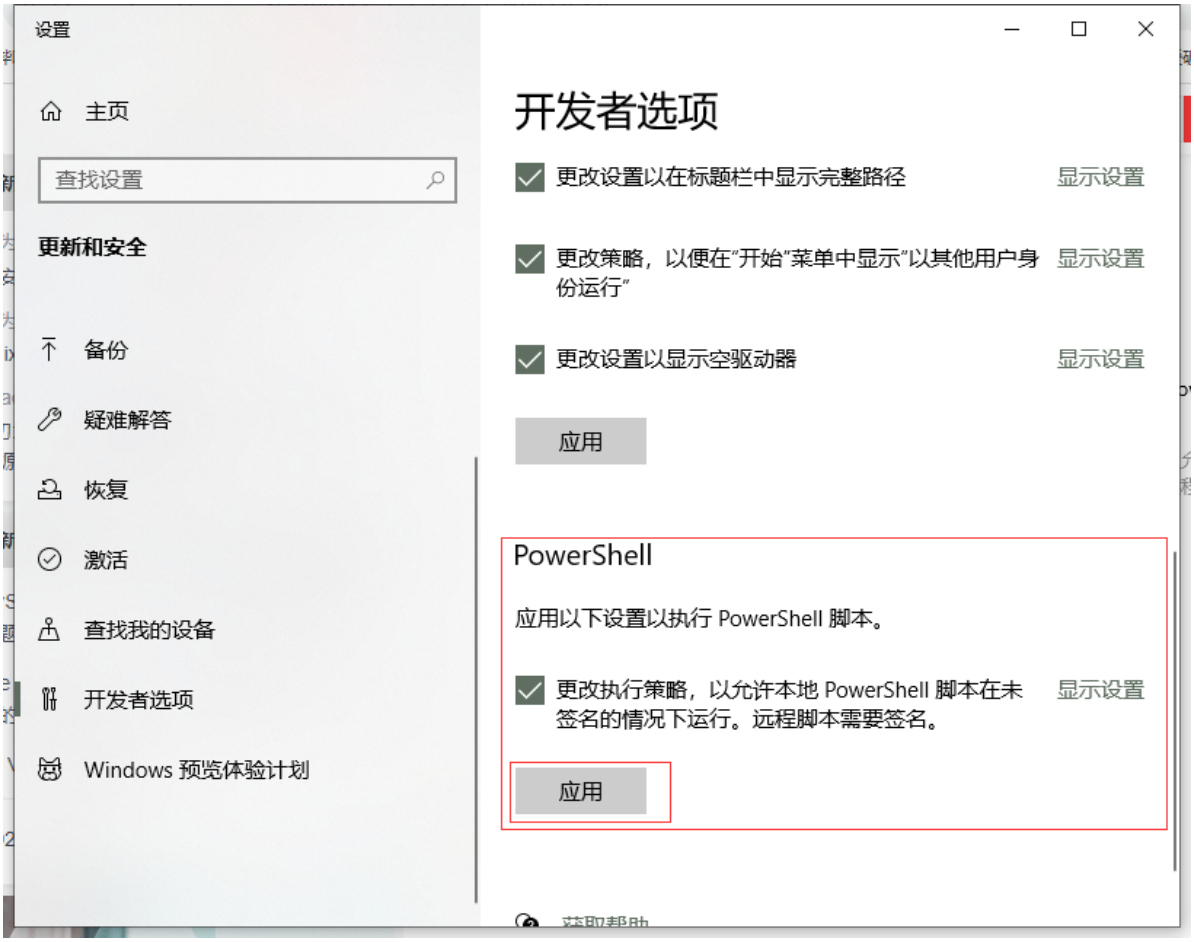


使用:输入`nodemon server.js`

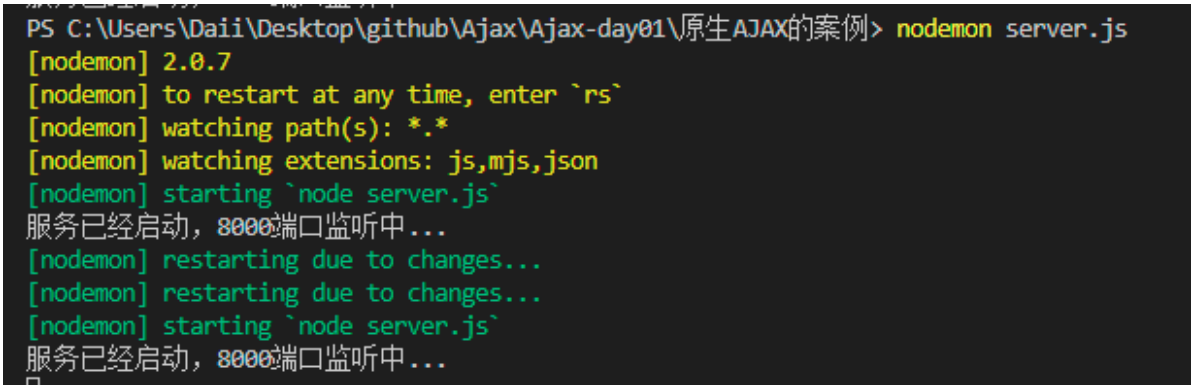


报错，提示信息：系统上禁止允许脚本。

解决方案：在系统设置—开发者选项—点击 PowerShell 下的应用



再次启动服务



这样修改服务端代码后不需要收到重启服务端了，更改代码保存后会自动重启。

1.14AJAX-IE缓存问题解决

对于时效性比较强的环境，AJAX缓存会影响我们的结果，返回结果不能及时返回给浏览器。

解决方法：在**请求时添加时间戳参数**（因为时间戳是唯一的，不可能重复的），如此浏览器会认为两次相近时间的请求是不同的请求，就能解决缓存问题了

具体代码如下：

```
xhr.open("GET", 'http://127.0.0.1 :8000/ie?t=' +Date.now());
```

1.15AJAX请求超时与网络异常处理

还是通过一个案例来了解

案例要求：点击按钮向服务端发送请求，

请求超时：如果2s后没有返回响应结果，则提示"网络异常，请稍后重试"；

网络异常处理：断网情况下2s后没有返回响应结果，则提示"你的网络除了一点问题"

首先，**服务端模拟延迟响应**

```
// 4）/delay页面 延迟响应模拟超时和网络异常
app.get('/delay', (request, response) => {
    // 设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    // 设置响应体  3s后再响应
    setTimeout(() => {
        response.send('延迟响应')
    }, 3000);
})
```

然后，进行**超时设置**

```
// 超时设置 2s设置
xhr.timeout = 2000
```

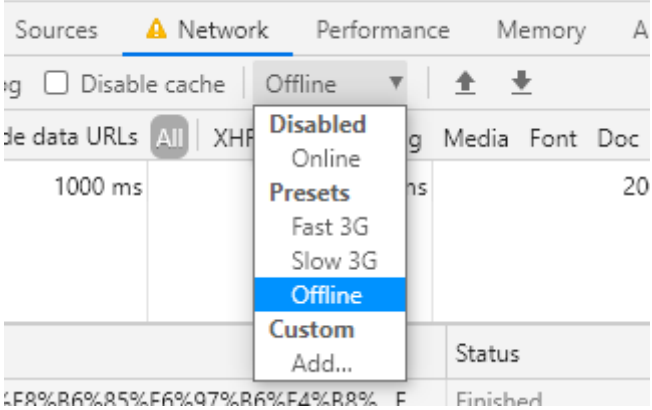
对于请求超时的处理：

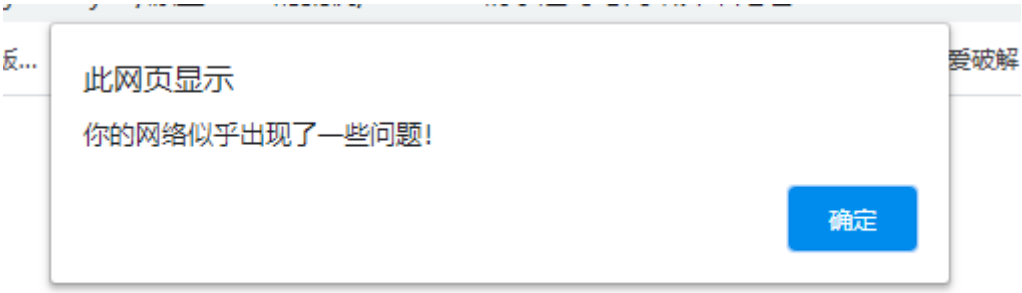
```
// 超时回调
            xhr.ontimeout = function () {
                alert('网络异常，请稍后重试')
            }
```

对于网络异常的处理：

```
// 网络异常回调
            xhr.onerror = function () {
                alert('你的网络似乎出现了一些问题！')
            }
```

通过将outline切换成offline模拟断网的情况



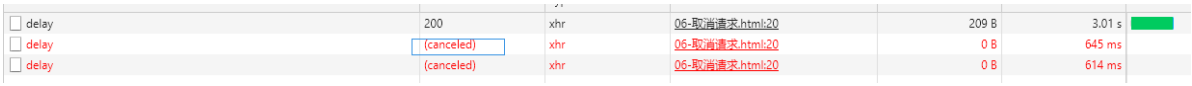


一般不会使用alert进行提示，而是用一个遮罩层或其他进行更友好的提示。

1.16AJAX取消请求

使用XMLHttpResquest对象的`abort()`方法取消请求。

```
let btn = document.querySelectorAll('button')
        let xhr = null; //由于发送和取消请求都需要使用xhr对象，那么设置为全局变量
        btn[0].onclick = function () {
            xhr = new XMLHttpRequest()
            xhr.open('GET', 'http://127.0.0.1:8000/delay')
            xhr.send()
        }
        btn[1].onclick = function () {
            // abort()取消请求
            xhr.abort()
        }
```



1.17AJAX请求重复发送问题

如果用户多次快速的进行发送相同的请求，那样服务器压力会很大。

解决方案：在**用户发送请求时判断之前是否有相同的请求操作**，若有则**取消前一次的请求操作**，保证请求数量只有一个，减小服务器的压力。**本质类似节流阀**

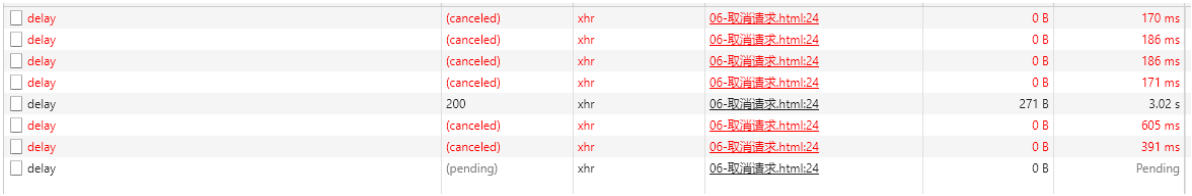
注意：

1.仍需要将xhr对象定义为全局变量，用来取消前一次的请求（尚未返回服务端所有结果的请求）

2.**注意修改标识变量的两个时机**，一个是正在发送请求时修改，一个是请求完毕即服务端返回所有结果的时候。

```
let btn = document.querySelector('button')
        let xhr = null //仍需要将xhr对象定义为全局变量
        // 使用节流阀解决重复请求问题，保证当前只有一个请求
        // 标识变量，默认处于没有在发送请求
        let isSending = false //是否正在发送AJAX请求
        btn.onclick = function () {
            // 如果正在发送AJAX请求，则取消该请求（前一次的请求），创建一个新的请求
            if (isSending) xhr.abort()
            xhr = new XMLHttpRequest()
            xhr.open('GET', 'http://127.0.0.1:8000/delay')
            xhr.send()
            // 目前正在发送请求，修改标识变量
            isSending = true
```

```
            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4) {
                    // 无论请求是否成功，都要修改标识变量
                    isSending = false
                }
            }
        }
```



delay	(canceled)	xhr	06-取消请求.html:24	0 B	170 ms
delay	(canceled)	xhr	06-取消请求.html:24	0 B	186 ms
delay	(canceled)	xhr	06-取消请求.html:24	0 B	186 ms
delay	(canceled)	xhr	06-取消请求.html:24	0 B	171 ms
delay	200	xhr	06-取消请求.html:24	271 B	3.02 s
delay	(canceled)	xhr	06-取消请求.html:24	0 B	605 ms
delay	(canceled)	xhr	06-取消请求.html:24	0 B	391 ms
delay	(pending)	xhr	06-取消请求.html:24	0 B	Pending

2.jQuery中的AJAX

2.1 get请求

```
$.get(url, [data], [callback], [type])
url:请求的URL地址。
data:请求携带的参数。
callback:载入成功时回调函数，参数data是响应体
type:设置返回内容格式，xml, html, script, json, text,. _default.
```

注意参数的数据格式为对象

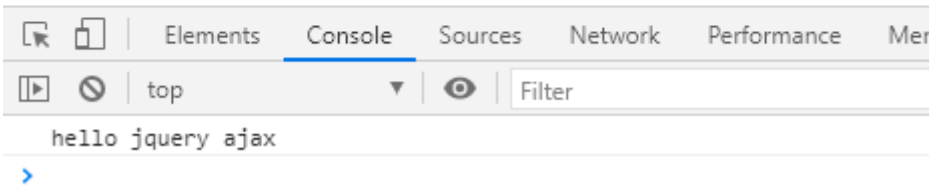
示例:

```
 $('button').eq(0).click(function () {
            $.get('http://127.0.0.1:8000/jquery-server', { a: 100, b: 20 },
function (data) {
                console.log(data);
            })
        })
```

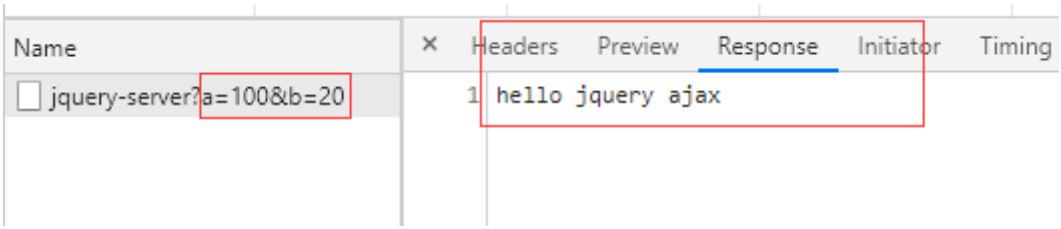
服务端

```
// 5)/jquery-server页面 jquery服务
app.get('/jquery-server', (request, response) => {
    // 设置响应头 设置允许跨域
    response.setHeader('Access-Control-Allow-Origin', '*')
    // 设置响应体
    response.send('hello jquery ajax')
})
```

控制台输出结果

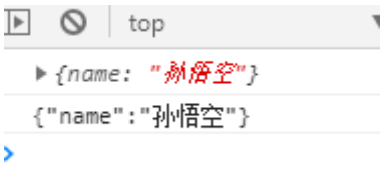


参数和响应体



设置内容格式为json，可以自动转换成对象类型数据，比之前手动/自动转换数据类型更方便。

```
$.get('http://127.0.0.1:8000/jquery-server', { a: 100, b: 20 }, function (data)
{
          console.log(data);
       }, 'json')
```



2.2 post请求

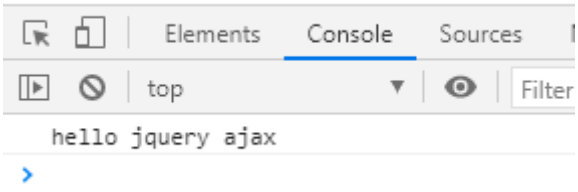
```
$.post(url, [data], [callback], [type])
url:请求的URL地址。
data:请求携带的参数。
callback:载入成功时回调函数，参数data是响应体
type:设置返回内容格式，xml，html，script，json，text，. _default.
```

post请求和get请求格式类似

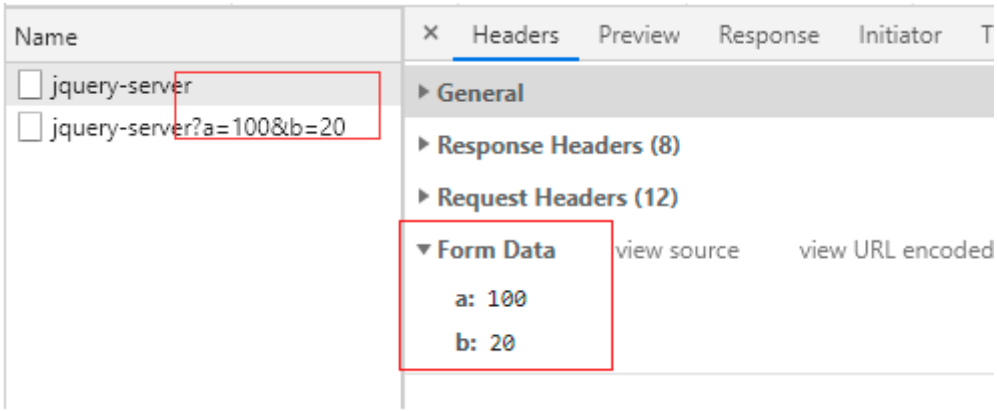
示例:

```
// post请求
        $('button').eq(1).click(function () {
            $.post('http://127.0.0.1:8000/jquery-server', { a: 100, b: 20 },
function (data) {
                console.log(data);
            })
        })
```

将服务端的get方法改为all



注意参数放置的位置，如下图



2.3jQuery通用方法发送AJAX请求

语法：

```
$.ajax(url,[settings]) Object
```

url:一个用来包含发送请求的URL字符串。

settings:AJAX 请求设置。所有选项都是可选的。

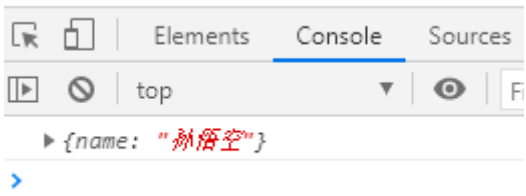
注意里面的参数都是用对象形式来写。

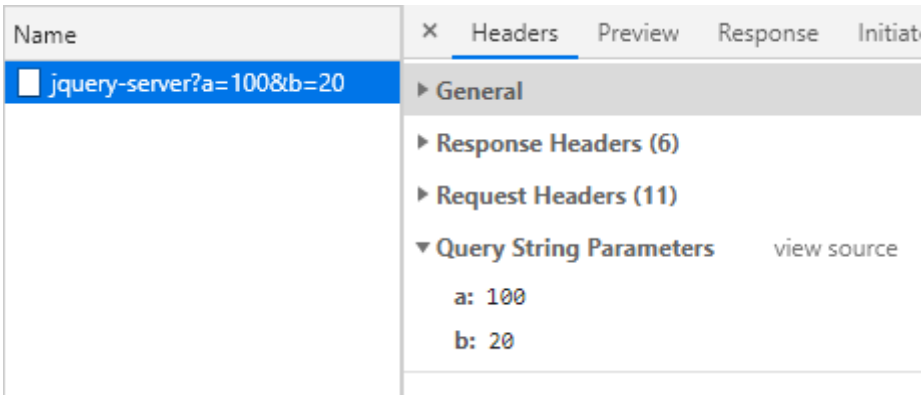
示例：主要记住下面几个参数即可

更多参数见https://jquery.cuishifeng.cn/jQuery.Ajax.html

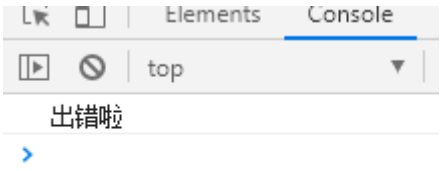
```
// jQuery通用方法发送AJAX请求
        $('button').eq(2).click(function () {
            $.ajax({
                // url
                url: 'http://127.0.0.1:8000/jquery-server',
                // 参数
                data: { a: 100, b: 20 },
                // 请求类型
                type: 'GET',
                // 响应体结果
                dataType: 'json',
                // 成功的回调
                success: function (data) {
                    console.log(data);
                },
                // 超时时间
                timeout: 2000,
                // 失败的回调
                error: function () {
```

```
                console.log('出错啦');
            },
            // 自定义头信息
            headers: {
                aaa: 111,
                ccc: 333
            }
        })
    })
```





失败的回调



三种请求的小结：第三种适合个性化的请求，一般使用POST/GET请求就可以了。

3.Axios发送AJAX请求

使用教程：https://github.com/axios/axios

首先引用axios

```
<script src="https://cdn.bootcdn.net/ajax/libs/axios/0.21.1/axios.min.js">
</script>
```

3.1GET请求

语法：

```
axios.get(url[, config])
```

示例：注意**响应结果返回的写法**

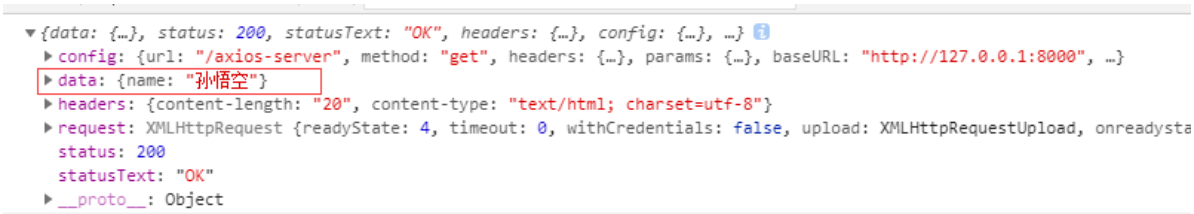
```
axios.get('/user', {
    params: {
      ID: 12345
    }
  })
  .then(function (response) {
    console.log(response);
  })
});
```

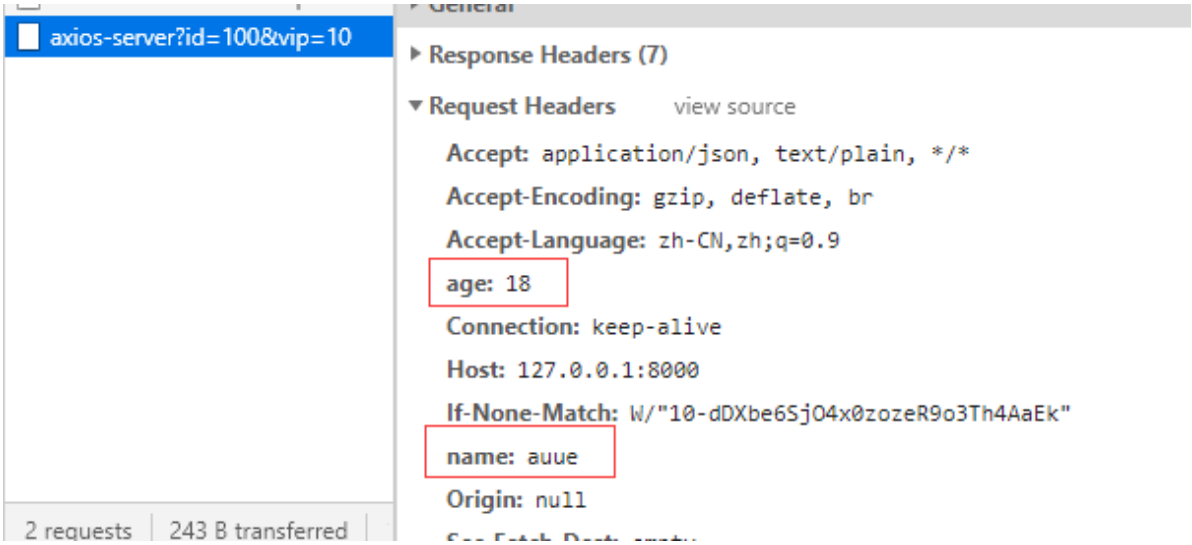
案例:

```
// 配置baseURL 自定义示例默认值
axios.defaults.baseURL = 'http://127.0.0.1:8000'
```

```
// 配置baseURL 自定义示例默认值
        axios.defaults.baseURL = 'http://127.0.0.1:8000'
        // GET请求
        btn[0].onclick = function () {
            axios.get('/axios-server', {
                // url 参数
                params: {
                    id: 100,
                    vip: 10
                },
                // 请求头信息,注意不能是中文
                headers: {
                    name: 'auue',
                    age: 18
                }
            }).then(value => {
                // 基于管理层返回响应体结果
                console.log(value);
            })
        }
```

将JSON格式的数据自动转换为对象





3.2POST请求

语法:

```
axios.post(url[, data[, config]])
```

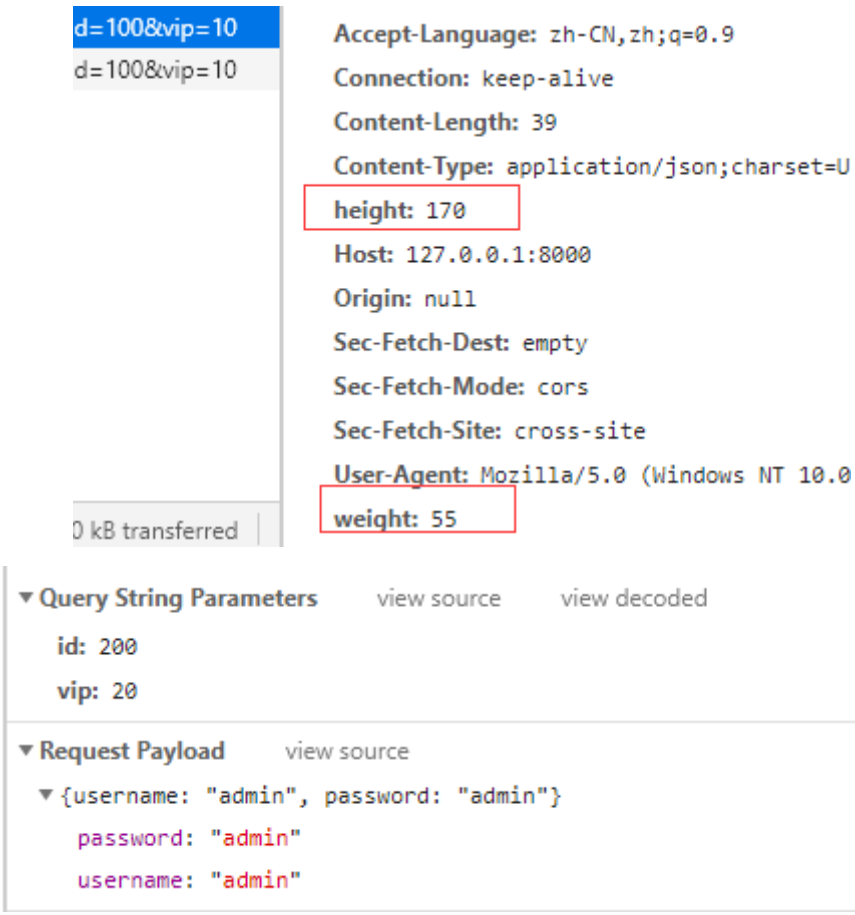
注意data请求体的位置

示例:

```
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

案例:

```
// POST请求
        btn[1].onclick = function () {
            axios.post('/axios-server', {
                username: 'admin',
                password: 'admin'
            }, {
                params: {
                    id: 200,
                    vip: 20
                },
                // 请求头参数
                headers: {
                    height: 170,
                    weight: 55
                }
            })
        }
```



3.3Axios函数请求

语法：`axios(config)`

示例：

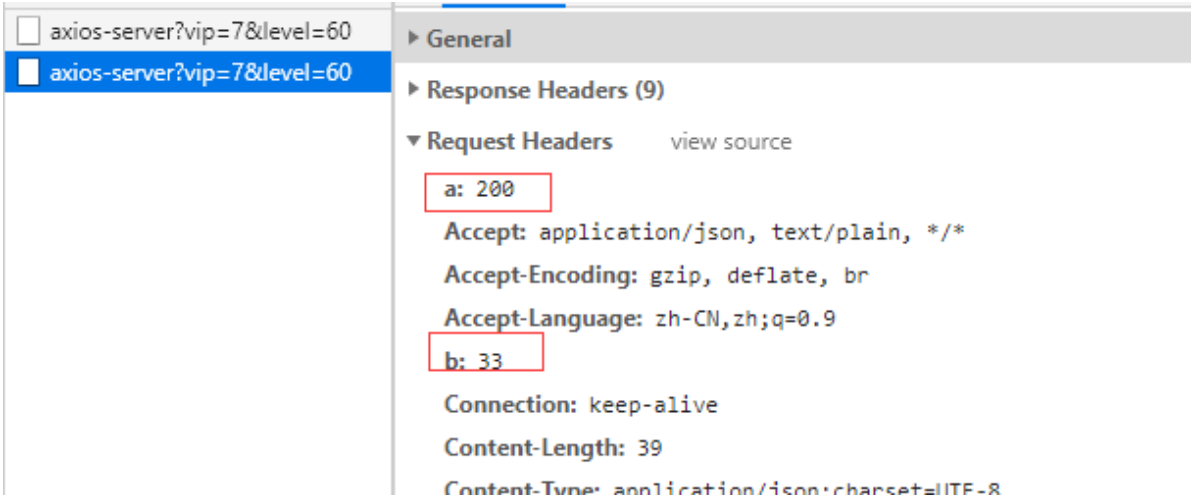
```
// Send a POST request
axios({
  method: 'post',
  url: '/user/12345',
  data: {
    firstName: 'Fred',
    lastName: 'Flintstone'
  }
});
```

案例：

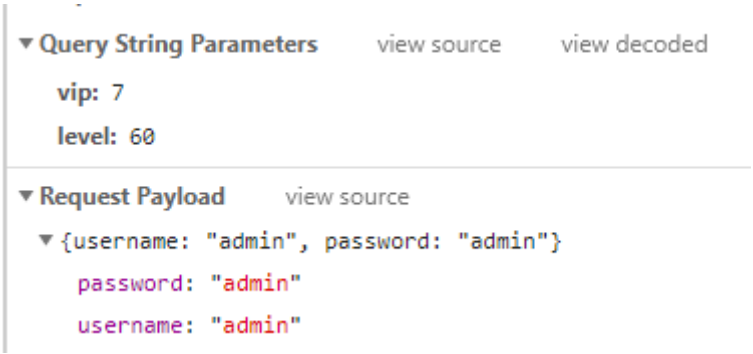
```
// Axios函数发送请求
        btn[2].onclick = function () {
            axios({
                // 请求方法
                method: 'post',
                // url
                url: '/axios-server',
                // url参数
                params: {
                    vip: 7,
                    level: 60
                },
                // 头信息
```

```
            headers: {
                a: 200,
                b: 33
            },
            // 请求体参数
            data: {
                username: 'admin',
                password: 'admin'
            }
        }).then(response => {
            console.log(response);
            // 响应状态码
            console.log(response.status);
            // 响应状态字符串
            console.log(response.statusText);
            // 响应头信息
            console.log(response.headers);
            // 响应体
            console.log(response.data);
        })
    }
```

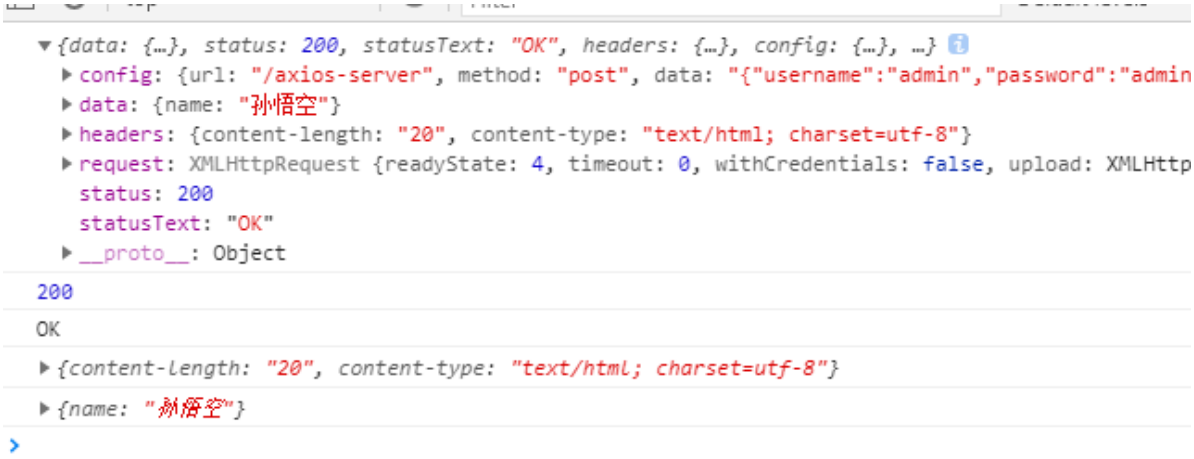
url参数和请求头信息：



url参数和请求体参数



响应结果：响应状态码+响应状态字符串+响应头信息+响应体



4.fetch发送AJAX请求

使用fetch发送请求使用次数较少

使用教程：https://developer.mozilla.org/zh-CN/docs/Web/API/WindowOrWorkerGlobalScope/fetch

语法：

```
fetch(input[, init]);
```

参数说明：

`input`:定义要获取的资源，可以是一个 USVString 字符串，包含要获取资源的 URL，也可以是Request对象。

`init`：可选。一个配置项对象，包括所有对请求的设置。可选的参数有：method、headers、body等

示例：

```
let btn = document.querySelector('button')
        btn.addEventListener('click', function () {
            fetch('http://127.0.0.1:8000/fetch-server', {
                // 请求方法
                method: 'POST',
                // 请求头
                headers: {
                    name: 'auue'
                },
                // 请求体
                body: 'username=admin&password=admin'
            }).then(response => {
                // console.log(response);
                // 获取响应体,使用里面的text方法,然后输出
                // return response.text()
                // 如果服务端返回结果是json数据类型，使用下面方法解析为js对象
                return response.json()
            }).then(response => {
                console.log(response);
            })
        })
```

完成的响应结果：

此时无法看到响应体的结果，使用里面的text()获取输出 `response.text()`

如果服务端返回结果是json数据类型，使用 `response.json()` 解析为js对象

5.跨域

5.1同源策略

同源策略(Same-Origin Policy)收早由Netscape 公司提出，是浏览器的一种安全策略。

同源:协议、域名、端口号必须完全相同。（浏览器与服务器同源）

违背同源策略就是跨域。

示例：

```
<h2>111</h2>
    <button>发送请求</button>
    <script>
        // 点击按钮获得响应结果
        let btn = document.querySelector('button')
        btn.addEventListener('click', function () {
            let xhr = new XMLHttpRequest()
            // 因为满足同源策略，url可以简写
            xhr.open('GET', '/data')
            xhr.send()
            xhr.onreadystatechange = function () {
```

```
                if (xhr.readyState == 4) {
                    if (xhr.status >= 200 & xhr.status < 300) {
                        console.log(xhr.response);
                    }
                }
            }
        })
    </script>
```

服务端

```
const { response, request } = require('express')
const express = require('express')
const app = express()
app.get('/home', (request, response) => {
    // 响应一个页面 绝对路径
    response.sendFile(__dirname + '/index.html')
})
app.get('/data', (request, response) => {
    response.send('用户数据')
})
app.listen(9000, () => {
    console.log('服务已经启动...');
})
```

进入9000端口，显示如下。点击按钮获取响应结果

此时浏览器和服务器同源，协议、域名、端口号相同，端口号都为9000

5.2如何解决跨域

先安装下`live server`插件

```
方案1JSONP

方案2CORS
```

5.2.1 JSONP

1) JSONP是什么

JSONP(JSON with Padding),是一个非官方的跨域解决方案，纯粹凭借程序员的聪明才智开发出来，只支持get请求。

2) JSONP怎么工作的?

在网页有一些标签天生具有跨域能力，比如: img link iframe script,（注意他们都有src）

JSONP就是利用script标签的跨域能力来发送请求的。

3) JSONP的使用

1.动态的创建一个script标签

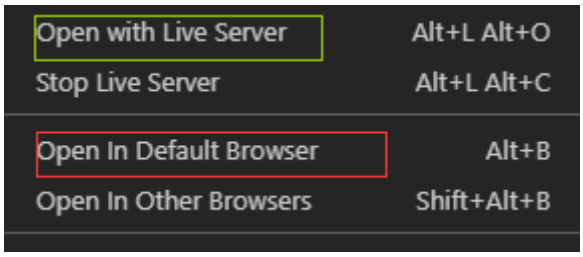
```
var script = document.createElement("script");
```

2.设置script的src,设置回调函数

```
script.src = http://localhost:3000/testAJAX?callback=abc";
```

4) JSONP的实现原理

先随便写个html文件、js文件并引入，

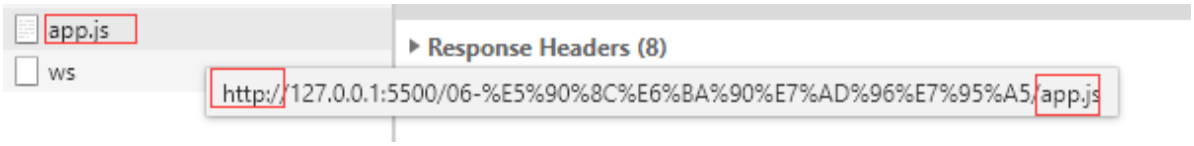


用红色方框的方法运行代码，可以看到浏览器的地址是

```
C:/Users/Daii/Desktop/github/Ajax/Ajax-day01/06-同源策略/2-JSONP原理.html
```

如果用live server运行代码，可以看到浏览器的地址是

里面的js资源访问也变成了http协议请求

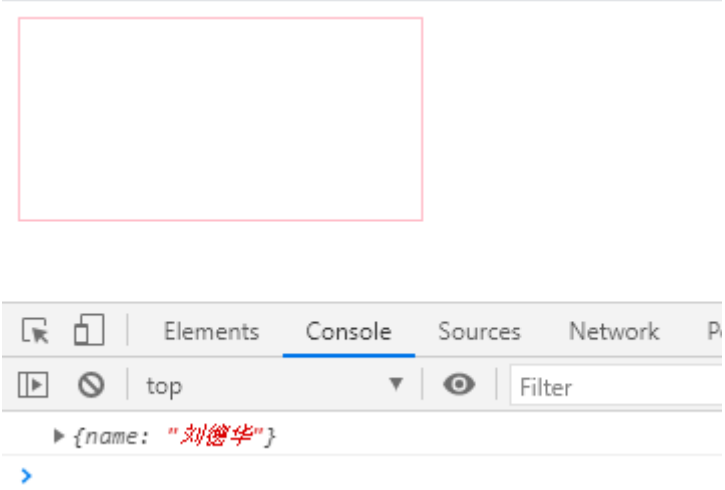


域名发生了改变!

双击点进去，复制链接，**使用该链接引入js**

```
<script src="http://127.0.0.1:5500/06-%E5%90%8C%E6%BA%90%E7%AD%96%E7%95%A5/app.js"></script>
```

仍可以正常显示页面



在html内定义函数，js文件中调用函数，可以成功调用函数

html内

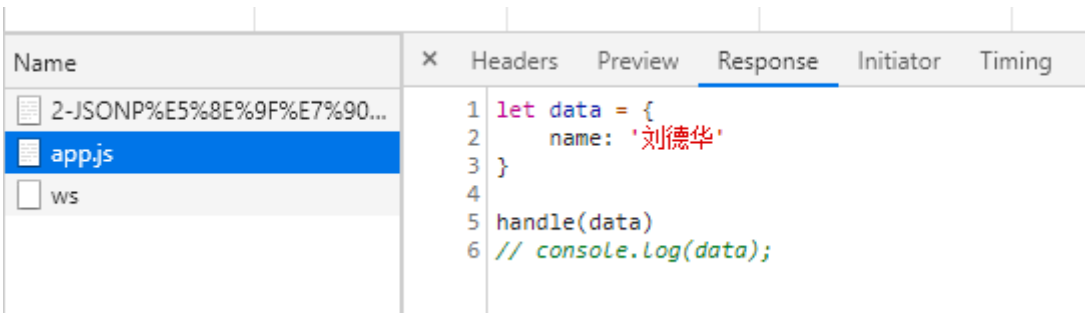
```
<script>
        // 让数据在div内显示
        function handle(data) {
            let div = document.querySelector('div')
            div.innerHTML = data.name
        }
    </script>
```

js内

```
let data = {
    name: '刘德华'
}

handle(data)
```

原理：`script`标签将服务端返回的响应结果（如下图）进行解析处理。因此需要将函数放在引用的script前面，否则解析出来会因handle函数未定义而报错。

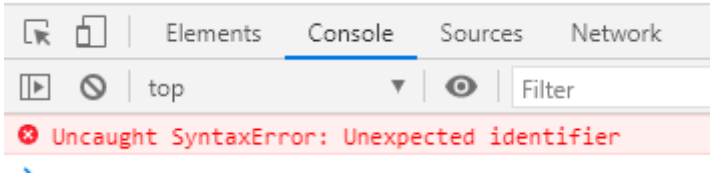


```
<script>
        // 让数据在div内显示
        function handle(data) {
            let div = document.querySelector('div')
            div.innerHTML = data.name
        }
    </script>
    <script src="http://127.0.0.1:5500/06-
%E5%90%8C%E6%BA%90%E7%AD%96%E7%95%A5/app.js"></script>
```

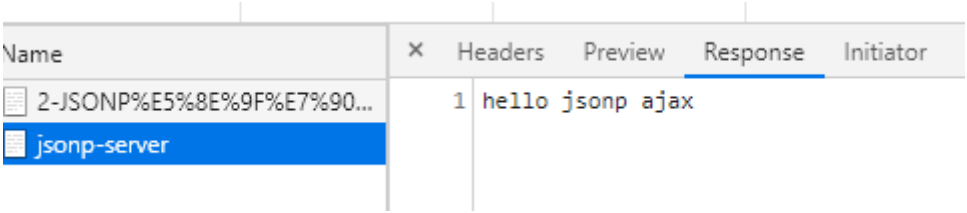
将src路径修改（此时不同源了，跨域了），如下

```
<script src="http://127.0.0.1:8000/jsonp-server"></script>
```

结果报错



但响应结果返回了

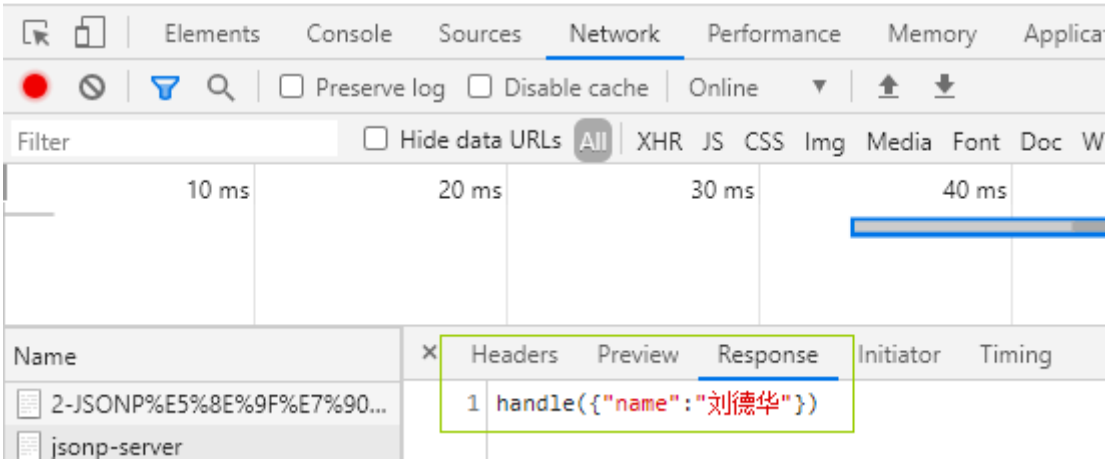


原因：**script标签**无法对该结果进行解析处理，**只能对js代码进行解析处理**

这就需要解决跨域了，如下

返回结果形式是参数调用（使用模板字符串调用函数），参数就是我们想要给服务端返回的结果数据，注意函数需要提前声明。

```
// 8）/jsonp-server页面 jsonp服务
app.all('/jsonp-server', (request, response) => {
    // response.send('hello jsonp ajax')
    let data = {
        name: '刘德华'
    }
    // 将数据转化为字符串
    let str = JSON.stringify(data)
    // 返回结果 使用模板字符串调用函数
    response.end(`handle(${str})`)
})
```



5.3JSONP案例

要求：表单输入用户名离开焦点后判断检测该用户名是否存在，存在则表示重名，表单边框变红色，且出现“用户名已存在”提示语

核心：离开焦点后发送请求，服务端返回响应结果，服务端直接不存在结果，而不直接进行比较（因为没有数据库）

注意：

1.同样通过调用handle回调函数来进行css样式设置，对返回结果数据进行处理

2.**向服务端发送请求的三个步骤：**

1）创建script标签

2）设置标签的src属性

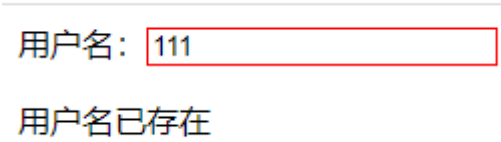
3）将script放入文件内

```
let ipt = document.querySelector('input')
        let p = document.querySelector('p')
        // 声明handle函数
        function handle(data) {
            // 边框变红色，出现提示语
            ipt.style.border = 'solid 1px red'
            p.innerHTML = data.msg
        }
        // 失去焦点触发事件
        ipt.onblur = function () {
            // 获取用户的输入值
            let username = this.value
            // 向服务端发送请求 检测用户名是否存在
            // 1.创建script标签
            let script = document.createElement('script')
            // 2.设置标签的src属性
```

```
            script.src = 'http://127.0.0.1:8000/check-username'
            // 3.将script放入文件内
            document.body.appendChild(script)
        }
```

服务端

```
// 9）/check-username页面 检测用户名是否存在
app.all('/check-username', (request, response) => {
    let data = {
        exist: 1,
        msg: '用户名已存在'
    }
    // 将数据转化为字符串
    let str = JSON.stringify(data)
    // 返回结果 使用模板字符串调用函数
    response.end(`handle(${str})`)
})
```

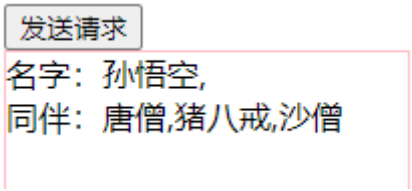


5.4jquery发送jsonp请求

语法格式与示例：

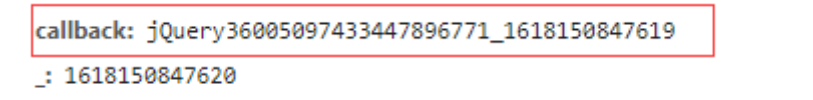
注意url后面的`?callback=?`，callback=?是固定格式

```
$('button').click(function () {
            $.getJSON('http://127.0.0.1:8000/jquery-jsonp-server?callback=?',
function (data) {
                $('div').html(`
                    名字：${data.name},<br>
                    同伴：${data.parter}
                `)
            })
        })
```



注意：

1.callback=? 在实际发送请求时，参数是有值的，自动生成的



2.服务端可以接收这个参数进行拼接字符串

```
let data = {
        name: '孙悟空',
        parter: ['唐僧', '猪八戒', '沙僧']
    }
    // 将数据转化为字符串
    let str = JSON.stringify(data)
    // 接收callback参数
    let cb = request.query.callback
    // 返回结果 使用模板字符串调用函数
    response.end(`${cb}(${str})`)
```

5.5CORS

推荐阅读：

- http://www.ruanyifeng.com/blog/2016/04/cors.html
- https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS

1. **CORS是什么?**

   CORS (Cross-Origin Resource Sharing), 跨域资源共享。CORS 是官方的跨域解决方案，它的特点是不需要在客户端做任何特殊的操作，完全在服务器中进行处理，支持 get 和 post 等请求。跨域资源共享标准新增了一组 HTTP 首部字段（响应头），允许服务器声明哪些源站通过浏览器有权限访问哪些资源

2. **CORS怎么工作的?**

   CORS 是通过设置一个响应头来告诉浏览器，该请求允许跨域，浏览器收到该响应以后就会对响应放行。

3. **CORS 的使用**

   主要是服务端的设置：

   ```
   rounter.get("/testAJAX",function(req, res){
    // 设置响应头 设置允许跨域 *表示所有页面，也可以写成指定页面才能跨域
       response.setHeader('Access-Control-Allow-Origin', '*')
   })
   ```