# AJAX简介

·AJAX 全称为 Asynchronous（异步的） JavaScript And 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

·在不重新加载整个页面的情况下，可以与服务器交换数据并更新部分网页内容

·AJAX发送的请求就是http请求

·每一次请求都可以在控制台的Network看到，还可以看到具体的数据交换

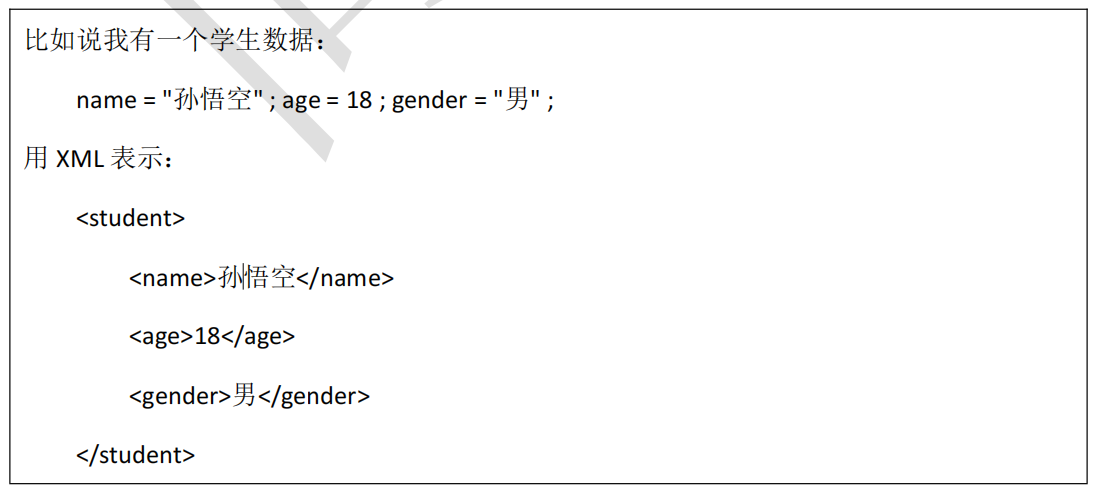
# XML简介

·可扩展标记语言。

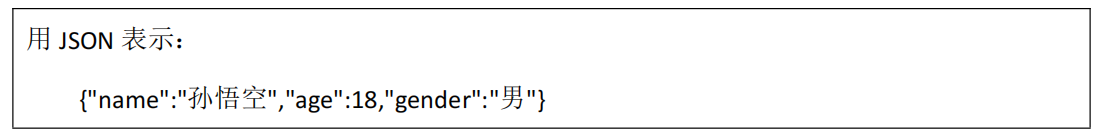
·XML被设计用来传输和存储数据。

·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。

·比如说我有一个学生数据：



·现在已经被 JSON 取代了



# AJAX的特点

## 优点

·可以无需刷新页面而与服务器端进行通信

·允许你根据用户事件（鼠标、键盘、表单、文档事件）来更新部分页面内容

## 缺点

·没有浏览历史，不能回退

·存在跨域问题(同源)

- 不允许向另一个服务器发送请求

·SEO（搜索引擎）不友好

# HTTP

·HTTP（Hyper Text Transfer Protocol）协议

- 超文本传输协议

·详细规定了浏览器和万维网服务器之间互相通信的规则

## 请求报文

### 请求行

·请求类型

- GET

- POST

- 等

·URL路径

- 如：/s?ie=UTF-8&wd=HTTP

·HTTP协议版本

- HTTP/1.1

·如 Post /s?ie=UTF-8&wd=HTTP HTTP/1.1

### 请求头

·Host:atguigu.com

·Cookie:name = guigu

·content-type:application/x-www-from-urlencoded

·User-Agent:chrome 83

### 空行

### 请求体

·如果是GET，请求体为空

·如果是Post请求体可以不为空

·username=admin&password=admin

## 响应报文

### 响应行

·HTTP协议版本

- HTTP/1.1

·状态码和状态字符串

- 200 OK

-- 表示从客户端发送给服务器的请求被正常处理并返回

- 404 Not Found

-- 表示服务器上无法找到请求的资源，除此之外，也可以在服务器拒绝请求但不想给拒绝原因时使用

- 403 Forbidden

-- 服务器拒绝该次访问（访问权限出现问题）

- 401 Unauthorized

-- 未经许可，需要通过HTTP认证

- 500 Inter Server Error

-- 表示服务器在执行请求时发生了错误，也有可能是web应用存在的bug或某些临时的错误时

### 响应头

·content-type:text/html;charset=utf-8

·content-length:2048

·content-encoding:gzip

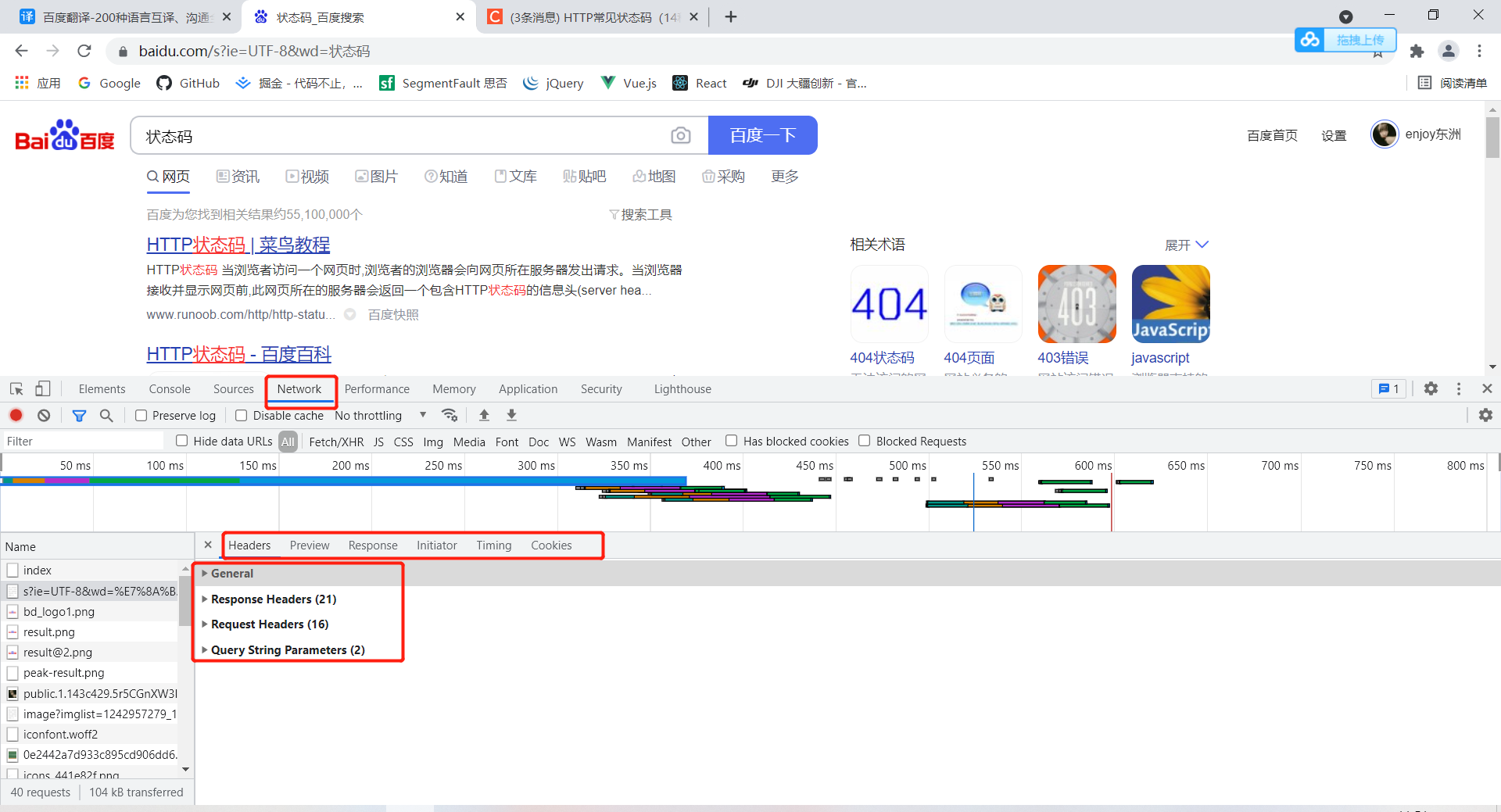
### 空行

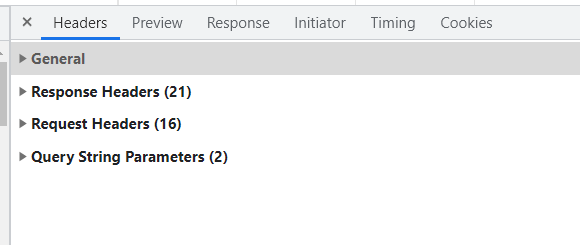
### 响应体

·主要的返回结果

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
</head>  
<body>  
 <h1>响应体</h1>  
</body>  
</html>

# Chrome网络控制台查看通信报文





## Request Headers

·请求头





·请求行

- GET/s?ie=UTF-8&wd=%E7%8A%B6%E6%80%81%E7%A0%81 HTTP/1.1

·请求头

- Host: [www.baidu.com](http://www.baidu.com)

- Connection: keep-alive

- sec-ch-ua: "Chromium" ;,v="92"," Not A;Brand" ; v="99"，"Google Chrome " ; v="92"

- sec-ch-ua-mobile: ?0

- Upgrade-Insecure-Requests: 1

- User-Agent: Wozilla/5.0(lkindows MT7 1.0;Wwin64; x64) ApplelebKit/537.36(KHTML，like Gecko) Chrome/92.0.4515.107 Safari/537.36

- Acept: text/html appication xhtml xmn. ,appication)oxml;g=0.9 mage/avif , image/ebp,image/apng " /\*,q=0.8,application/signed-exchange;v=b3,q=0.9

- Sec-Fetch-Site: same-origin

- Sec-Fetch-Mode: navigate

- Sec-Fetch-Dest: document

- Referer: https : / / www .baidu.com/

- Accept-Encoding: gzip, deflate， br

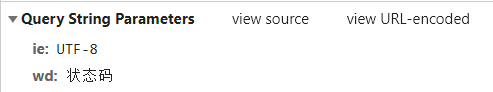
- Accept-Language: zh-CN,zh;q=0.9

- Cookie

## Query String Parameters

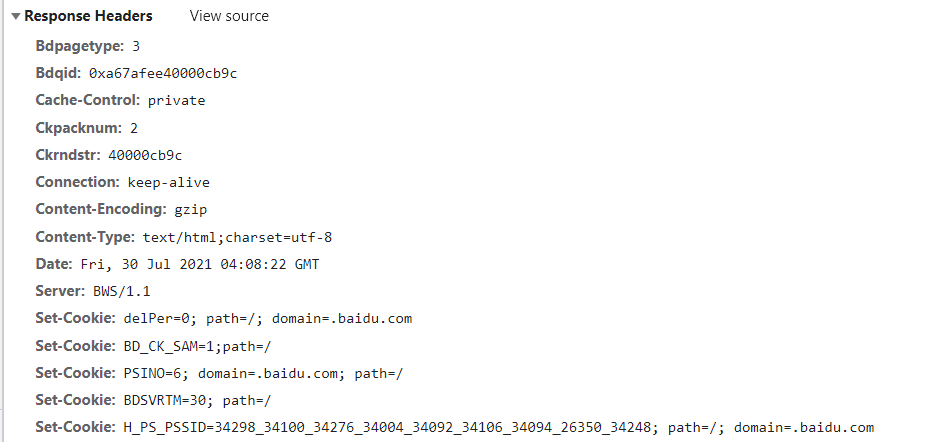
·查询字符串参数

·对请求行的参数做解析



Response Headers

·响应报文



·原始响应报文



·响应行

- HTTP/1.1 200 OK

·响应头

- Bdpagetype: 3

- Bdqid: oxa67afee40000cb9c

- Cache-Control: private

- Ckpacknum: 2

- Ckrndstr: 40000cb9c

- Connection: keep-alive

- Content-Encoding: gzip

- Content-Type: text/html; charset=utf-8Date: Fri，30 Jul 2021 04:08:22 GMT

- Server: BwS/1.1

- Set-Cookie: delPer=0; path=/ ; domain=.baidu.com

- Set-Cookie: BD\_CK\_SAM=1 ; path=/

- Set-Cookie: PSINO=6; domain=.baidu.com; path=/

- Set-Cookie: BDSVRTM=30; path=/

- Set-Cookie: H\_PS\_PSSI0=34298\_34108\_34276\_34004\_34092\_34106\_34094\_26350\_34248; path=/; domain=.baidu.com

- Strict-Transport-Security: max-age=172800

- Traceid: 1627618102027200359411996180812660788124

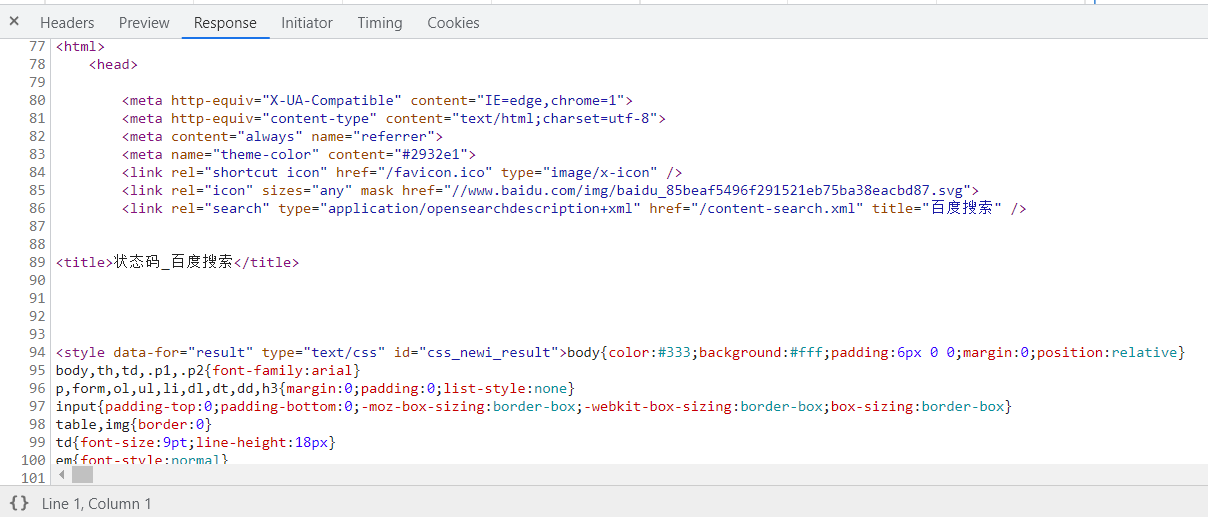
- Vary: Accept-Encoding

- X-Frame-Options: sameorigin

- X-Ua-Compatible: IE=Edge,chrome=1

- Transfer-Encoding: chunked

·响应体



# Node.js的安装

下载node.js

<https://nodejs.org/zh-cn/>

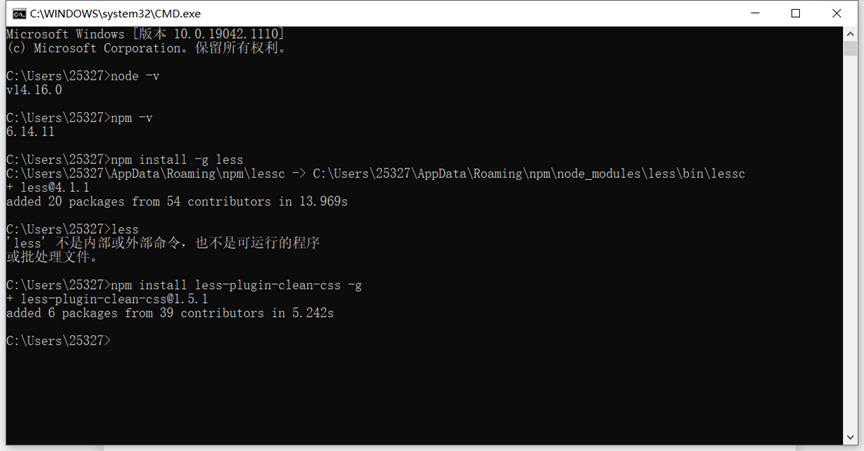
安装

<https://segmentfault.com/a/1190000023390756>

·查看是否安装成功

- node -v 查看 node 版本

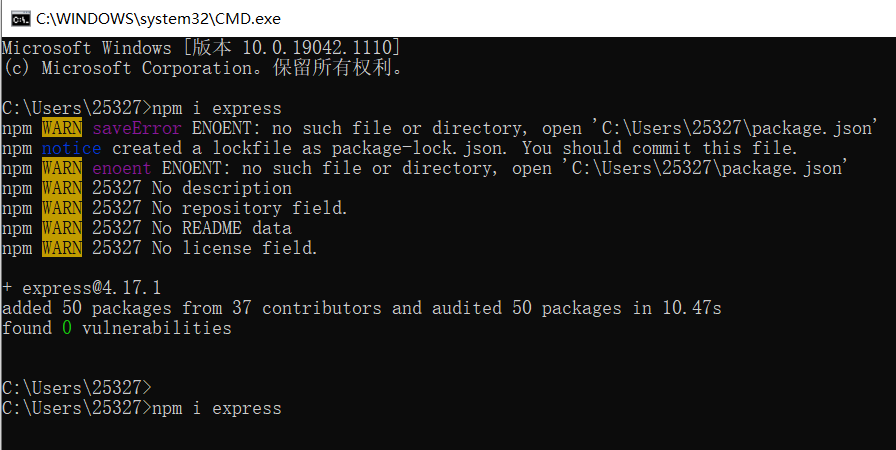
- npm -v 查看 npm 版本



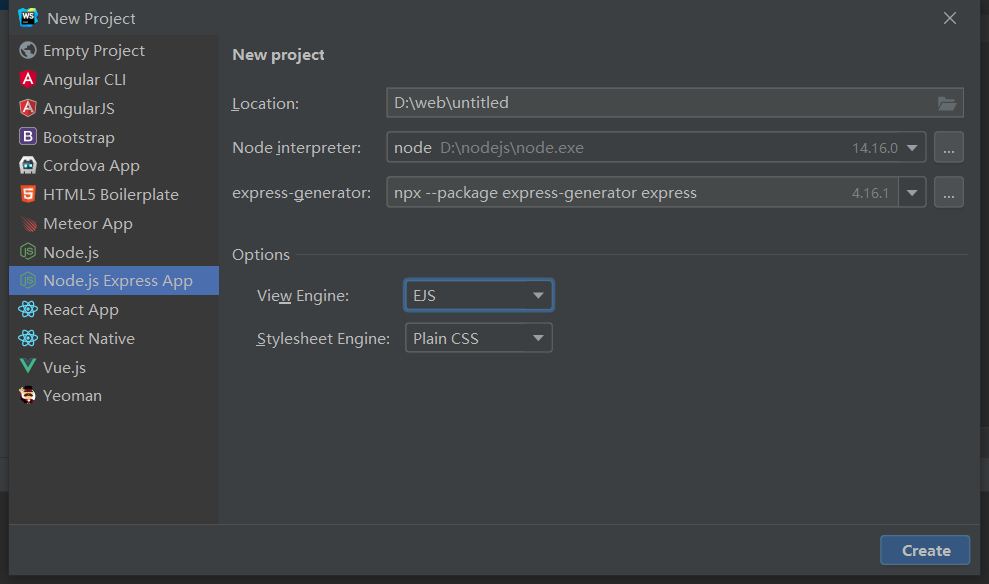
# express框架介绍与基本使用

## 安装

·npm i express



·在webstorm中创建express项目



·编写js代码

·()=>{

}

相当于

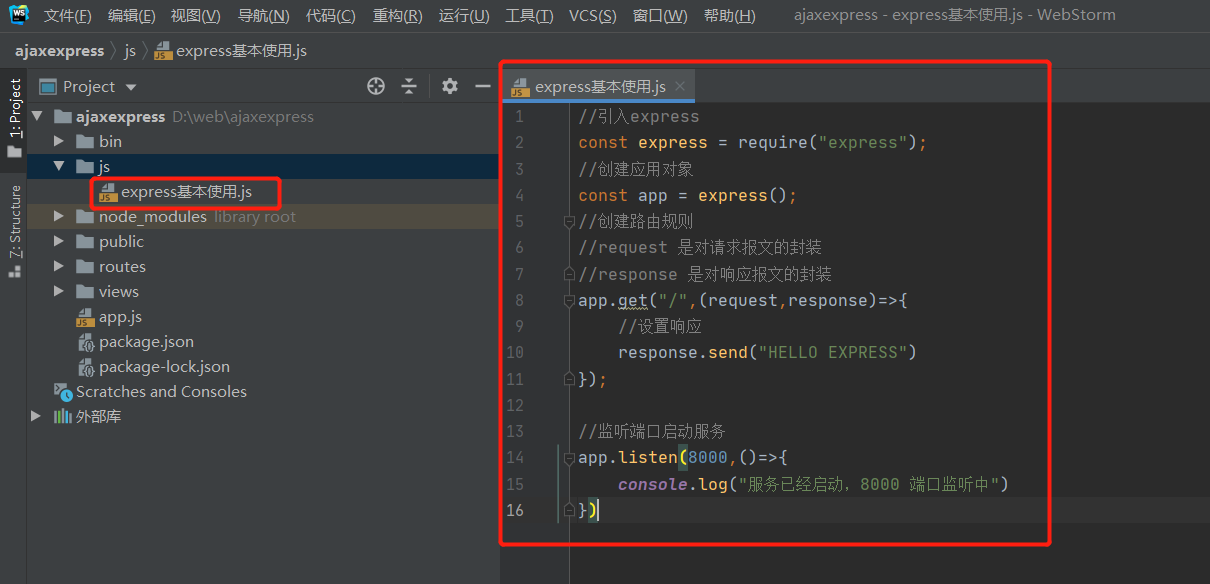
function () {

}

//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//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  
app.get("/",(request,response)=>{  
 //设置响应  
 response.send("HELLO EXPRESS")  
});  
  
//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")  
})

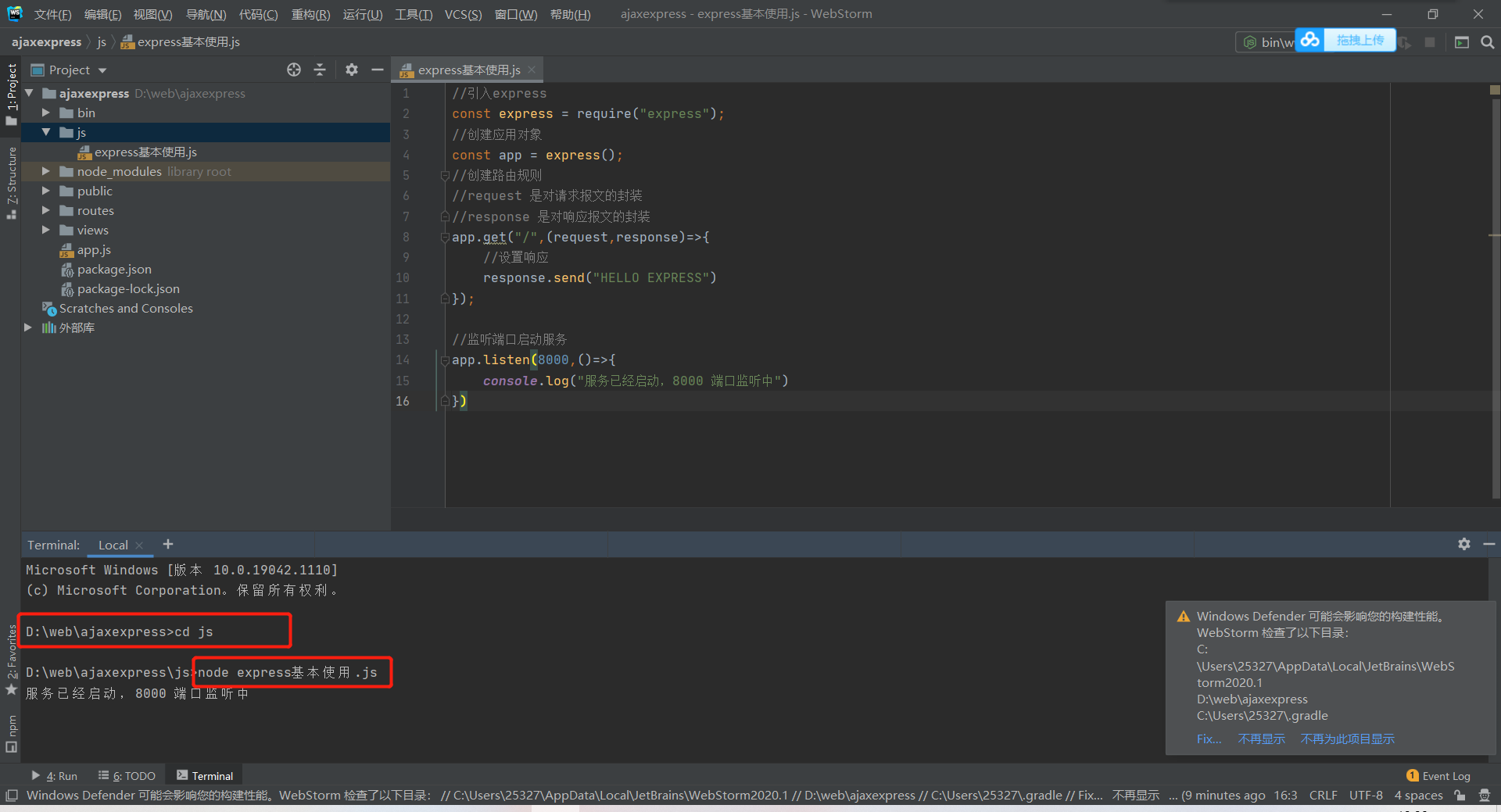
或者

//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//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  
//如果url路径是/server则执行以下代码  
//get请求  
  
app.get("/server",function (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("get请求");  
});  
  
//post请求  
app.post("/server",function (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("post请求");  
});  
  
//监听端口启动服务  
app.listen(8000,function () {  
 ***console***.log("服务已经启动，8000 端口监听中");  
})

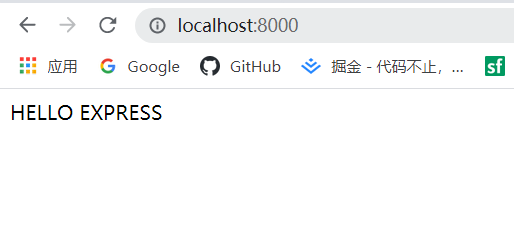


·启动服务

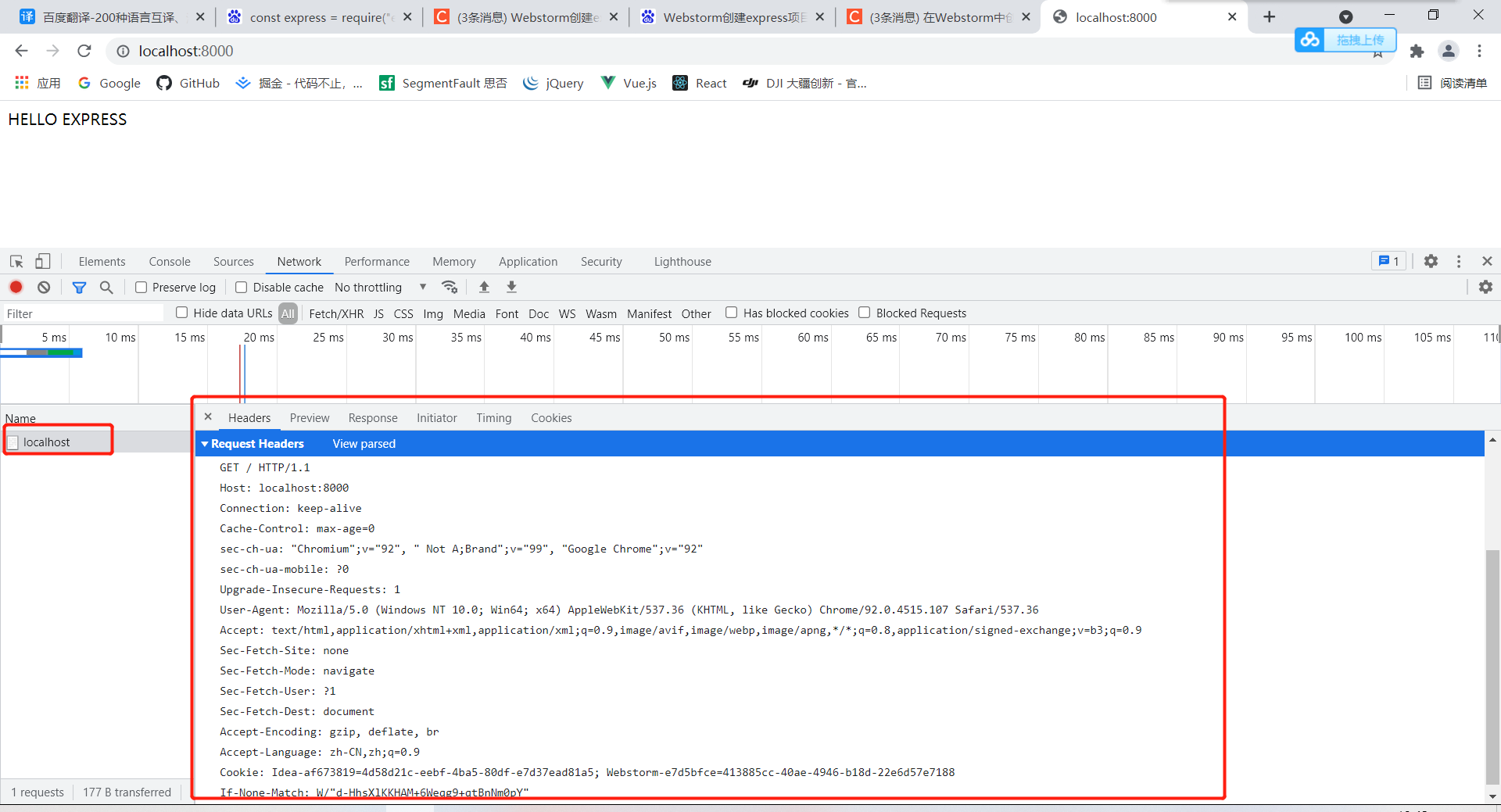
- node 文件名.js



·浏览器输入http://localhost:8000



·查看请求头和响应头等信息



# const、let、var的区别

·const定义的变量不可以修改，而且必须初始化。

·var定义的变量可以修改，如果不初始化会输出undefined，不会报错

·let是块级作用域，函数内部使用let定义后，对函数外部无影响

# AJAX的使用

·核心对象

- XMLHttpRequest，AJAX 的所有操作都是通过该对象进行的

# AJAX的GET请求

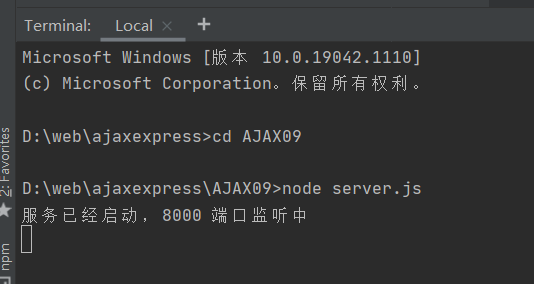
·前端html页面代码

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>AJAX GET请求</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
 <button>点击发送请求</button>  
 <div id="result"></div>  
</body>  
</html>

·server.js代码

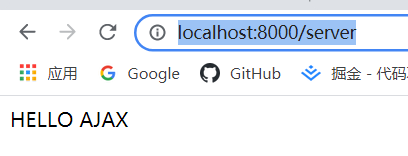
//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//创建路由规则  
//request 是对请求报文的封装  
//response 是对响应报文的封装  
//如果url路径是/server则执行以下代码  
app.get("/server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*")  
 //设置响应体  
 response.send("HELLO AJAX")  
});  
  
//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")  
})

·启动服务，需要把之前启动的8000端口服务终止

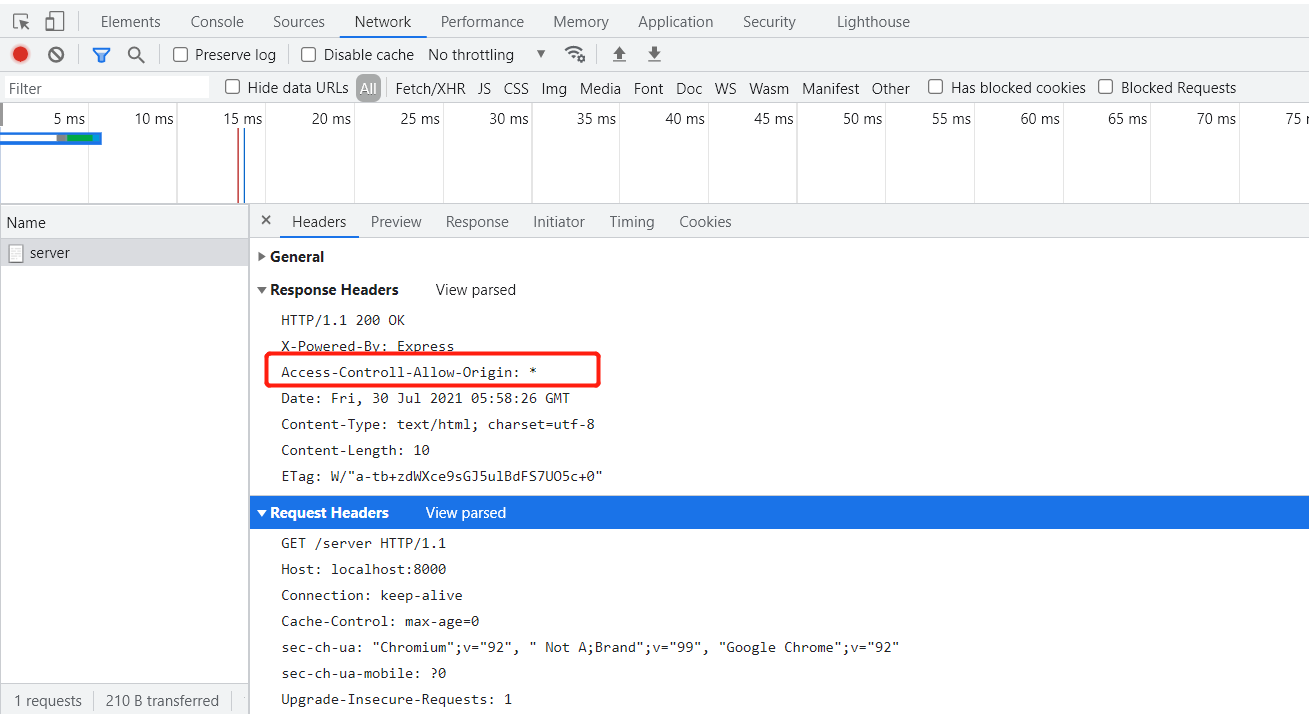


·浏览器输入

<http://localhost:8000/server>



·查看请求头和响应头等信息



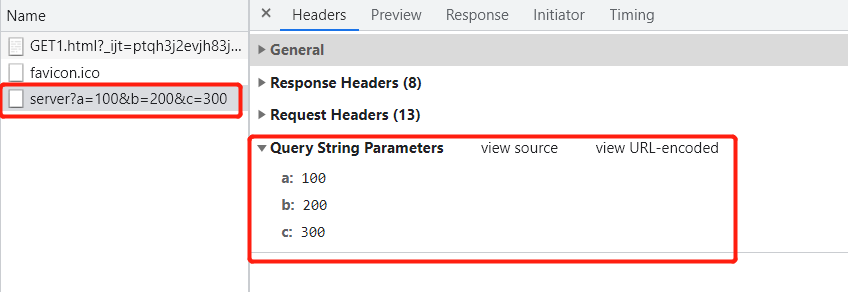
## 请求的基本操作

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>AJAX GET请求</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
 <button>点击发送请求</button>  
 <div id="result"></div>  
<script type="text/javascript">  
 //获取button元素  
 const ***btn*** = ***document***.getElementsByTagName("button")[0];  
 //绑定事件  
 ***btn***.onclick = function () {  
 //创建ajax对象  
 const xhr = new ***XMLHttpRequest***();  
 //获取响应结果div  
 const result = ***document***.getElementById("result");  
 //初始化 设置请求方法和url  
 xhr.open("GET","http://localhost:8000/server");  
 //发送  
 xhr.send();  
 //事件绑定 处理服务端返回的结果  
 //on这个单词有when的意思，当...时  
 //readystate（准备状态）是xhr对象中的属性，表示状态 0 （被初始化）1（open方法调用完毕） 2（send方法调用完毕） 3（服务端返回了部分结果） 4（服务端返回了所有结果）  
 //change（改变），当改变的时候触发  
 //会触发四次  
 xhr.onreadystatechange = function () {  
 //xhr.readyState获取返回状态  
 if (xhr.readyState === 4){//当服务端返回了所有结果时  
 //判断响应状态码 200 404 403 401 500  
 //响应状态码中2开头的表示成功，并非只有200  
 if (xhr.status >=200 && xhr.status < 300){//xhr.status获取状态码，获取结果成功时  
 //处理结果 行 头 空行 体  
 // console.log(xhr.status);//响应行的状态码  
 // console.log(xhr.statusText);//响应行的状态字符串  
 // console.log(xhr.getAllResponseHeaders())//所有响应头  
 // console.log(xhr.response);//响应体  
 //将响应体显示到div中  
 result.innerHTML = xhr.response;  
 }else {  
  
 }  
 }  
 }  
 }  
</script>  
</body>  
</html>

## 设置请求参数

//初始化 设置请求方法和url，并设置需要传递的参数  
xhr.open("GET","http://localhost:8000/server?a=100&b=200&c=300");

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>AJAX GET请求</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
 <button>点击发送请求</button>  
 <div id="result"></div>  
<script type="text/javascript">  
 //获取button元素  
 const ***btn*** = ***document***.getElementsByTagName("button")[0];  
 //绑定事件  
 ***btn***.onclick = function () {  
 //创建ajax对象  
 const xhr = new ***XMLHttpRequest***();  
 //获取响应结果div  
 const result = ***document***.getElementById("result");  
 //初始化 设置请求方法和url，并设置需要传递的参数  
 xhr.open("GET","http://localhost:8000/server?a=100&b=200&c=300");  
 //发送  
 xhr.send();  
 //事件绑定 处理服务端返回的结果  
 //on这个单词有when的意思，当...时  
 //readystate（准备状态）是xhr对象中的属性，表示状态 0 （被初始化）1（open方法调用完毕） 2（send方法调用完毕） 3（服务端返回了部分结果） 4（服务端返回了所有结果）  
 //change（改变），当改变的时候触发  
 //会触发四次  
 xhr.onreadystatechange = function () {  
 //xhr.readyState获取返回状态  
 if (xhr.readyState === 4){//当服务端返回了所有结果时  
 //判断响应状态码 200 404 403 401 500  
 //响应状态码中2开头的表示成功，并非只有200  
 if (xhr.status >=200 && xhr.status < 300){//xhr.status获取状态码，获取结果成功时  
 //处理结果 行 头 空行 体  
 // console.log(xhr.status);//响应行的状态码  
 // console.log(xhr.statusText);//响应行的状态字符串  
 // console.log(xhr.getAllResponseHeaders())//所有响应头  
 // console.log(xhr.response);//响应体  
 //将响应体显示到div中  
 result.innerHTML = xhr.response;  
 }else {  
  
 }  
 }  
 }  
 }  
</script>  
</body>  
</html>



# AJAX的POST请求

·编写服务端post请求的代码

//post请求  
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 POST");  
});

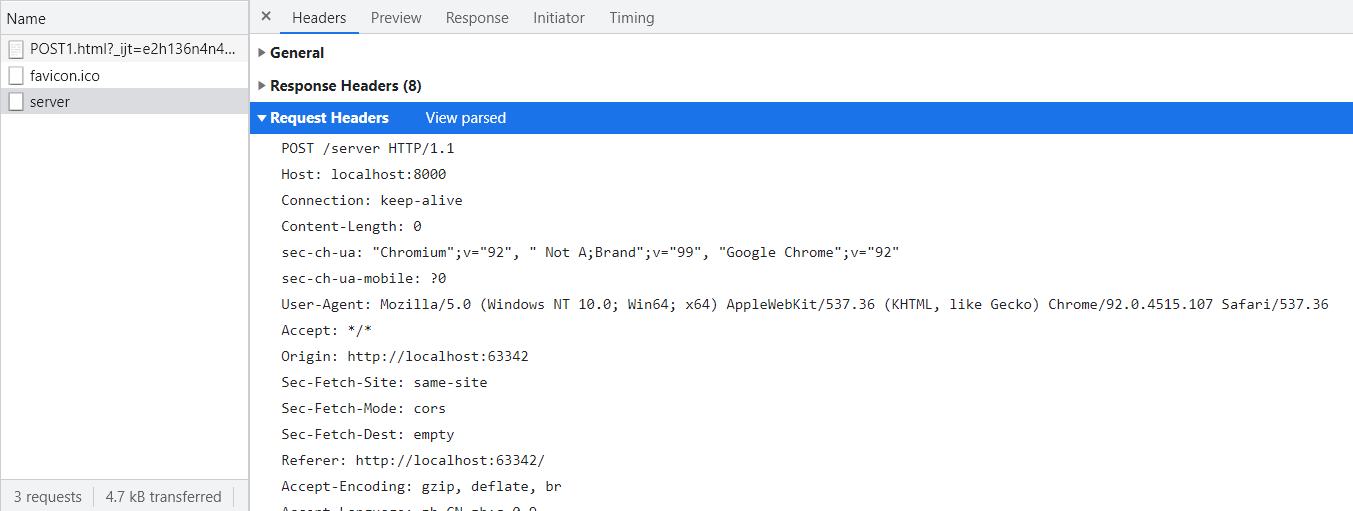
//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//创建路由规则  
//request 是对请求报文的封装  
//response 是对响应报文的封装  
//如果url路径是/server则执行以下代码  
app.get("/server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*")  
 //设置响应体  
 response.send("HELLO AJAX GET")  
});  
  
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 POST")  
});  
  
//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")  
})

·启动服务

·编写html代码

//初始化 设置请求方法和url  
xhr.open("POST","http://localhost:8000/server");

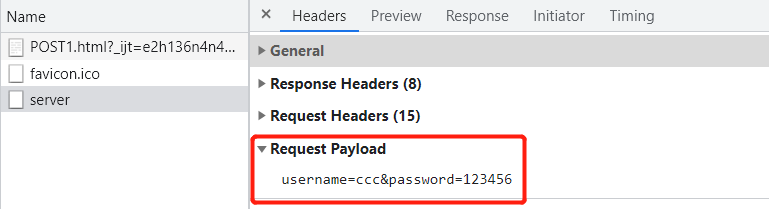
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>AJAX GET请求</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
<button>点击发送请求</button>  
<div id="result"></div>  
<script type="text/javascript">  
 //获取响应结果div  
 const ***result*** = ***document***.getElementById("result");  
 //绑定鼠标移入事件  
 ***result***.addEventListener("mouseover",function () {  
 //创建ajax对象  
 const xhr = new ***XMLHttpRequest***();  
 //初始化 设置请求方法和url  
 xhr.open("POST","http://localhost:8000/server");  
 //发送  
 xhr.send();  
 //事件绑定 处理服务端返回的结果  
 //on这个单词有when的意思，当...时  
 //readystate（准备状态）是xhr对象中的属性，表示状态 0 （被初始化）1（open方法调用完毕） 2（send方法调用完毕） 3（服务端返回了部分结果） 4（服务端返回了所有结果）  
 //change（改变），当改变的时候触发  
 //会触发四次  
 xhr.onreadystatechange = function () {  
 //xhr.readyState获取返回状态  
 if (xhr.readyState === 4) {//当服务端返回了所有结果时  
 //判断响应状态码 200 404 403 401 500  
 //响应状态码中2开头的表示成功，并非只有200  
 if (xhr.status >= 200 && xhr.status < 300) {//xhr.status获取状态码，获取结果成功时  
 //处理结果 行 头 空行 体  
 // console.log(xhr.status);//响应行的状态码  
 // console.log(xhr.statusText);//响应行的状态字符串  
 // console.log(xhr.getAllResponseHeaders())//所有响应头  
 // console.log(xhr.response);//响应体  
 //将响应体显示到div中  
 ***result***.innerHTML = xhr.response;  
 } else {  
  
 }  
 }  
 }  
 })  
</script>  
</body>  
</html>

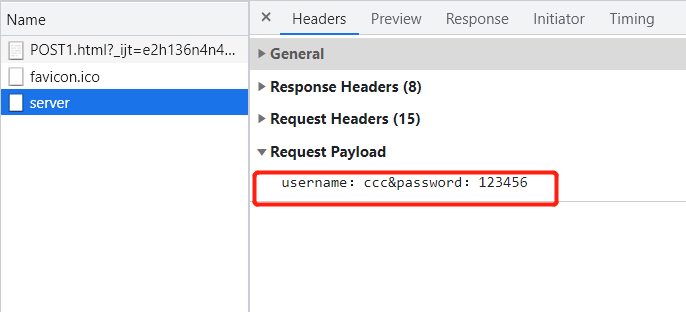


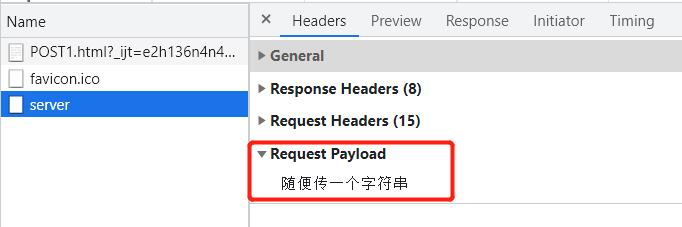
## POST设置请求体

//设置请求体并发送  
xhr.send("username=ccc&password=123456");  
// xhr.send("username：ccc&password：123456");  
// xhr.send("随便传一个字符串");

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>AJAX GET请求</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
<button>点击发送请求</button>  
<div id="result"></div>  
<script type="text/javascript">  
 //获取响应结果div  
 const ***result*** = ***document***.getElementById("result");  
 //绑定鼠标移入事件  
 ***result***.addEventListener("mouseover",function () {  
 //创建ajax对象  
 const xhr = new ***XMLHttpRequest***();  
 //初始化 设置请求方法和url  
 xhr.open("POST","http://localhost:8000/server");  
 //设置请求体并发送  
 // xhr.send("username=ccc&password=123456");  
 // xhr.send("username：ccc&password：123456");  
 xhr.send("随便传一个字符串");  
 //事件绑定 处理服务端返回的结果  
 //on这个单词有when的意思，当...时  
 //readystate（准备状态）是xhr对象中的属性，表示状态 0 （被初始化）1（open方法调用完毕） 2（send方法调用完毕） 3（服务端返回了部分结果） 4（服务端返回了所有结果）  
 //change（改变），当改变的时候触发  
 //会触发四次  
 xhr.onreadystatechange = function () {  
 //xhr.readyState获取返回状态  
 if (xhr.readyState === 4) {//当服务端返回了所有结果时  
 //判断响应状态码 200 404 403 401 500  
 //响应状态码中2开头的表示成功，并非只有200  
 if (xhr.status >= 200 && xhr.status < 300) {//xhr.status获取状态码，获取结果成功时  
 //处理结果 行 头 空行 体  
 // console.log(xhr.status);//响应行的状态码  
 // console.log(xhr.statusText);//响应行的状态字符串  
 // console.log(xhr.getAllResponseHeaders())//所有响应头  
 // console.log(xhr.response);//响应体  
 //将响应体显示到div中  
 ***result***.innerHTML = xhr.response;  
 } else {  
  
 }  
 }  
 }  
 })  
</script>  
</body>  
</html>





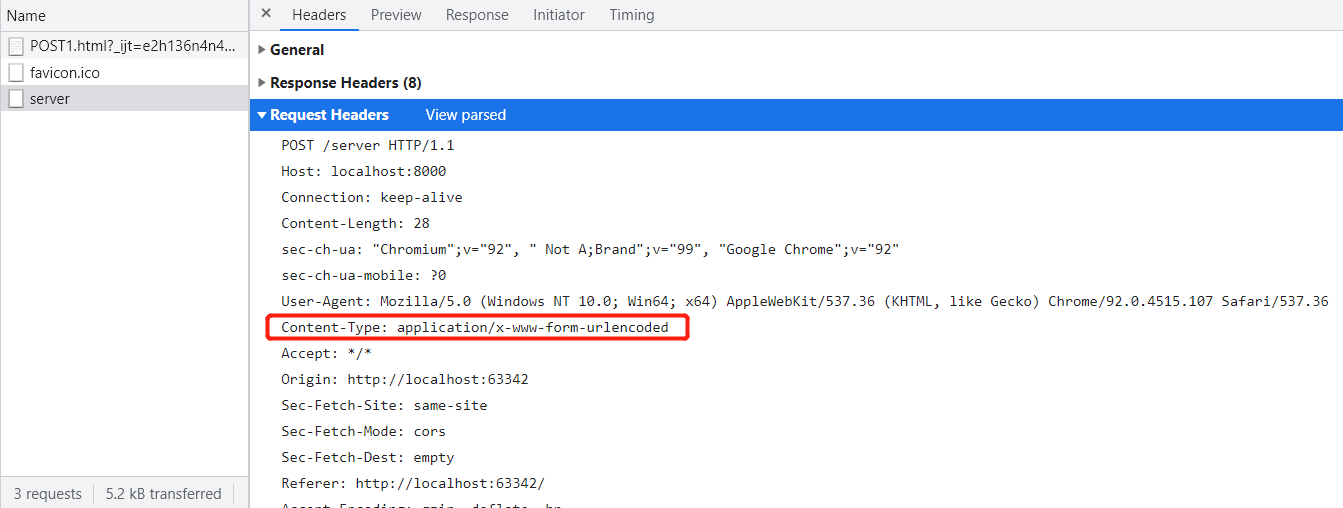


# 设置请求头信息

·请求头必须设置在open后，send前

//设置请求头信息  
//Content-Type是用来设置请求体类型的  
//application/x-www-form-urlencoded是参数查询字符串的类型  
//还可以设置其他请求头  
xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>AJAX GET请求</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
<div id="result"></div>  
<script type="text/javascript">  
 //获取响应结果div  
 const ***result*** = ***document***.getElementById("result");  
 //绑定鼠标移入事件  
 ***result***.addEventListener("mouseover",function () {  
 //创建ajax对象  
 const xhr = new ***XMLHttpRequest***();  
 //初始化 设置请求方法和url  
 xhr.open("POST","http://localhost:8000/server");  
 //设置请求头信息  
 //Content-Type是用来设置请求体类型的  
 //application/x-www-form-urlencoded是参数查询字符串的类型  
 //还可以设置其他请求头  
 xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");  
 //设置请求体并发送  
 xhr.send("username=ccc&password=123456");  
 // xhr.send("username：ccc&password：123456");  
 // xhr.send("随便传一个字符串");  
 //事件绑定 处理服务端返回的结果  
 //on这个单词有when的意思，当...时  
 //readystate（准备状态）是xhr对象中的属性，表示状态 0 （被初始化）1（open方法调用完毕） 2（send方法调用完毕） 3（服务端返回了部分结果） 4（服务端返回了所有结果）  
 //change（改变），当改变的时候触发  
 //会触发四次  
 xhr.onreadystatechange = function () {  
 //xhr.readyState获取返回状态  
 if (xhr.readyState === 4) {//当服务端返回了所有结果时  
 //判断响应状态码 200 404 403 401 500  
 //响应状态码中2开头的表示成功，并非只有200  
 if (xhr.status >= 200 && xhr.status < 300) {//xhr.status获取状态码，获取结果成功时  
 //处理结果 行 头 空行 体  
 // console.log(xhr.status);//响应行的状态码  
 // console.log(xhr.statusText);//响应行的状态字符串  
 // console.log(xhr.getAllResponseHeaders())//所有响应头  
 // console.log(xhr.response);//响应体  
 //将响应体显示到div中  
 ***result***.innerHTML = xhr.response;  
 } else {  
  
 }  
 }  
 }  
 })  
</script>  
</body>  
</html>



# 设置自定义请求头

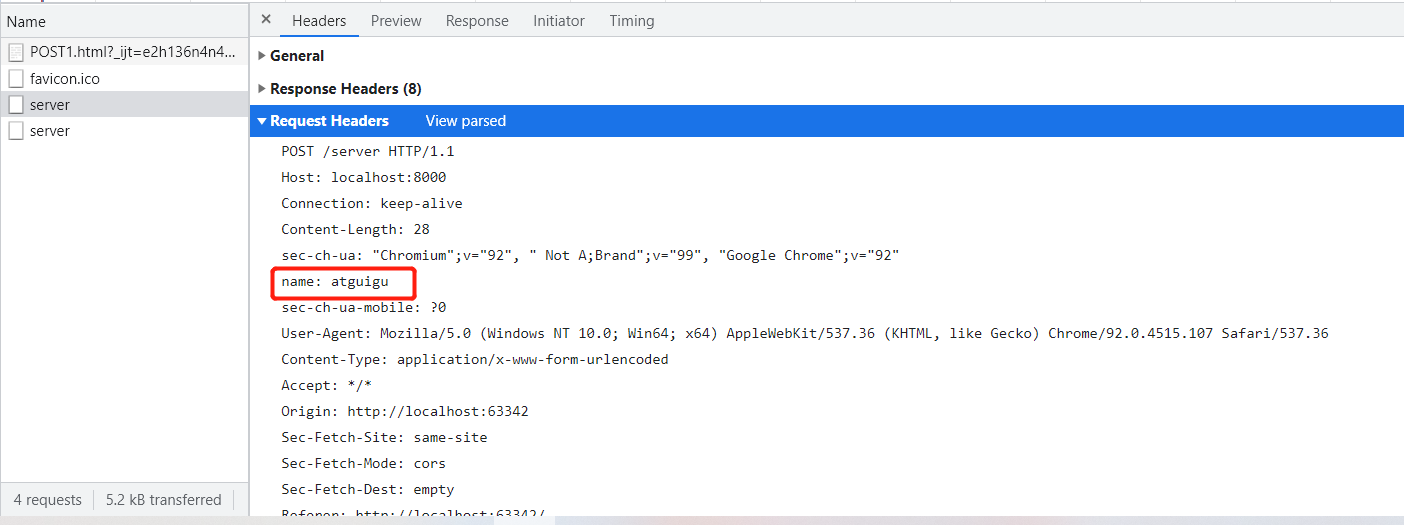
·GET也可以设置请求头和自定义请求头

//任意类型请求  
app.all("/server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //所有响应头信息都可以添加  
 response.setHeader("Access-Control-Allow-Headers","\*");  
 //设置响应体  
 response.send("HELLO AJAX POST");  
});

//自定义请求头信息  
xhr.setRequestHeader("name","atguigu");

//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//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  
//如果url路径是/server则执行以下代码  
//get请求  
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 GET");  
});  
  
//post请求  
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 POST");  
});  
  
//任意类型请求  
app.all("/server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //所有响应头信息都可以添加  
 response.setHeader("Access-Control-Allow-Headers","\*");  
 //设置响应体  
 response.send("HELLO AJAX POST");  
});  
  
//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");  
})

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>AJAX GET请求</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
<div id="result"></div>  
<script type="text/javascript">  
 //获取响应结果div  
 const ***result*** = ***document***.getElementById("result");  
 //绑定鼠标移入事件  
 ***result***.addEventListener("mouseover",function () {  
 //创建ajax对象  
 const xhr = new ***XMLHttpRequest***();  
 //初始化 设置请求方法和url  
 xhr.open("POST","http://localhost:8000/server");  
 //设置请求头信息  
 //Content-Type是用来设置请求体类型的  
 //application/x-www-form-urlencoded是参数查询字符串的类型  
 //还可以设置其他请求头  
 xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");  
 //自定义请求头信息  
 xhr.setRequestHeader("name","atguigu");  
 //设置请求体并发送  
 xhr.send("username=ccc&password=123456");  
 // xhr.send("username：ccc&password：123456");  
 // xhr.send("随便传一个字符串");  
 //事件绑定 处理服务端返回的结果  
 //on这个单词有when的意思，当...时  
 //readystate（准备状态）是xhr对象中的属性，表示状态 0 （被初始化）1（open方法调用完毕） 2（send方法调用完毕） 3（服务端返回了部分结果） 4（服务端返回了所有结果）  
 //change（改变），当改变的时候触发  
 //会触发四次  
 xhr.onreadystatechange = function () {  
 //xhr.readyState获取返回状态  
 if (xhr.readyState === 4) {//当服务端返回了所有结果时  
 //判断响应状态码 200 404 403 401 500  
 //响应状态码中2开头的表示成功，并非只有200  
 if (xhr.status >= 200 && xhr.status < 300) {//xhr.status获取状态码，获取结果成功时  
 //处理结果 行 头 空行 体  
 // console.log(xhr.status);//响应行的状态码  
 // console.log(xhr.statusText);//响应行的状态字符串  
 // console.log(xhr.getAllResponseHeaders())//所有响应头  
 // console.log(xhr.response);//响应体  
 //将响应体显示到div中  
 ***result***.innerHTML = xhr.response;  
 } else {  
  
 }  
 }  
 }  
 })  
</script>  
</body>  
</html>



# 服务端响应JSON数据

·编写server代码

//任意类型请求  
app.all("/json-server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //所有响应头信息都可以添加  
 response.setHeader("Access-Control-Allow-Headers","\*");  
 //响应一个数据的对象  
 const data = {  
 name: "atguigu",  
 age:18  
 }  
 //将数据转换成json字符串  
 let str = ***JSON***.stringify(data);  
 //设置响应体  
 response.send(str);  
});

//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//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  
//如果url路径是/server则执行以下代码  
//get请求  
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 GET");  
});  
  
//post请求  
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 POST");  
});  
  
//任意类型请求  
app.all("/json-server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //所有响应头信息都可以添加  
 response.setHeader("Access-Control-Allow-Headers","\*");  
 //响应一个数据的对象  
 const data = {  
 name: "atguigu",  
 age:18  
 }  
 //将数据转换成json字符串  
 let str = ***JSON***.stringify(data);  
 //设置响应体  
 response.send(str);  
});  
  
//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");  
})

·编写html代码

- 手动转换

//将获取到的JSON字符串手动转换为js对象  
let data = ***JSON***.parse(xhr.response);  
//将手动转换的数据显示到div中  
***result***.innerHTML = "name = " + data.name + "</br>" + "age = " + data.age;

- 自动转换

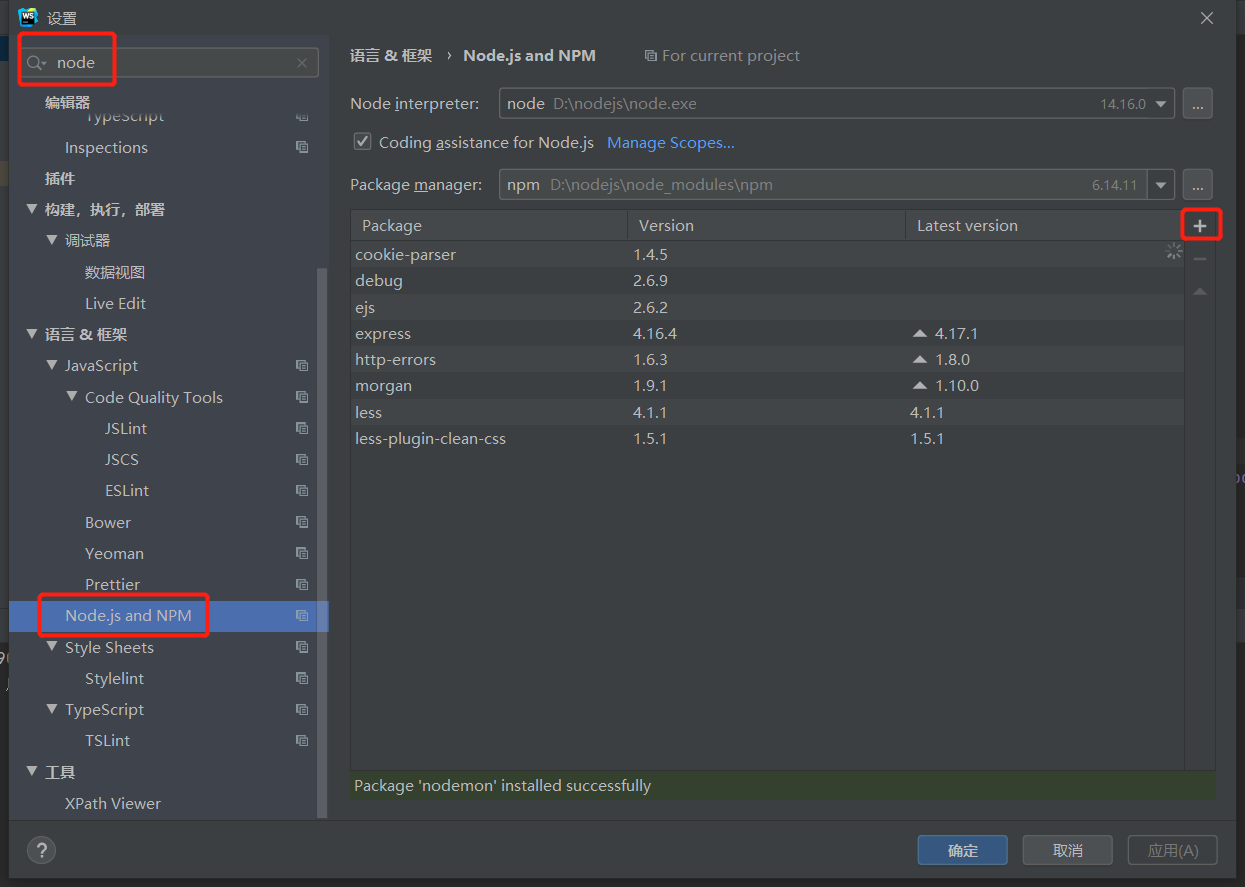
//设置响应体数据的类型，将获取到的JSON字符串手动转换为js对象  
xhr.responseType = "json";

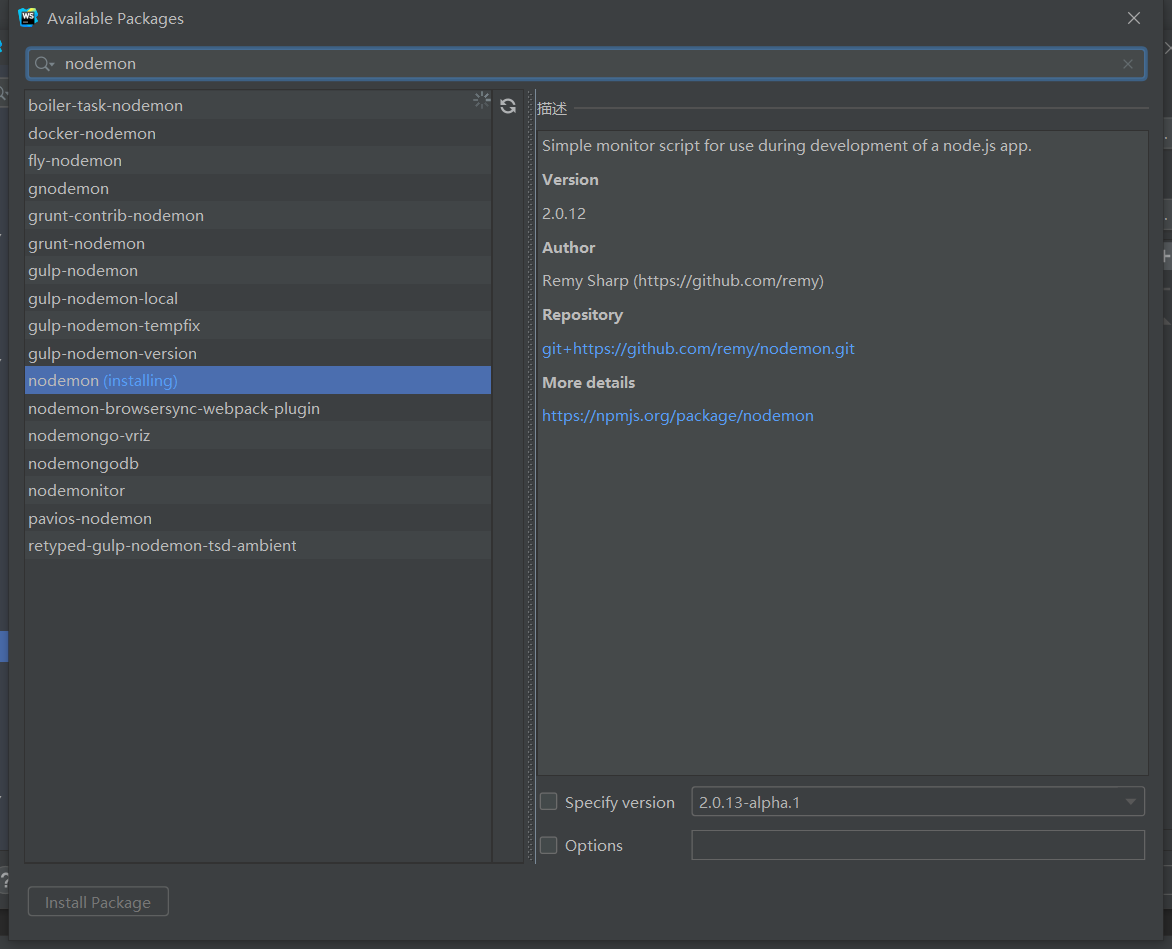
//将自动转换的数据显示到div中  
***result***.innerHTML = "name = " + xhr.response.name + "</br>" + "age = " + xhr.response.age;

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>服务端响应JSON数据</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
<div id="result"></div>  
<script type="text/javascript">  
 //获取响应结果div  
 const ***result*** = ***document***.getElementById("result");  
 //绑定键盘按下事件  
 ***window***.onkeydown =function () {  
 //创建ajax对象  
 const xhr = new ***XMLHttpRequest***();  
 //设置响应体数据的类型，将获取到的JSON字符串手动转换为js对象  
 xhr.responseType = "json";  
 //初始化 设置请求方法和url  
 xhr.open("POST", "http://localhost:8000/json-server");  
 //设置请求体并发送  
 xhr.send("username=ccc&password=123456");  
 //事件绑定 处理服务端返回的结果  
 //on这个单词有when的意思，当...时  
 //readystate（准备状态）是xhr对象中的属性，表示状态 0 （被初始化）1（open方法调用完毕） 2（send方法调用完毕） 3（服务端返回了部分结果） 4（服务端返回了所有结果）  
 //change（改变），当改变的时候触发  
 //会触发四次  
 xhr.onreadystatechange = function () {  
 //xhr.readyState获取返回状态  
 if (xhr.readyState === 4){//当服务端返回了所有结果时  
 //判断响应状态码 200 404 403 401 500  
 //响应状态码中2开头的表示成功，并非只有200  
 if (xhr.status >=200 && xhr.status < 300){//xhr.status获取状态码，获取结果成功时  
 //处理结果 行 头 空行 体  
 // console.log(xhr.status);//响应行的状态码  
 // console.log(xhr.statusText);//响应行的状态字符串  
 // console.log(xhr.getAllResponseHeaders())//所有响应头  
 // console.log(xhr.response);//响应体  
 // //将获取到的JSON字符串手动转换为js对象  
 // let data = JSON.parse(xhr.response);  
 // //将手动转换的数据显示到div中  
 // result.innerHTML = "name = " + data.name + "</br>" + "age = " + data.age;  
 //将自动转换的数据显示到div中  
 ***result***.innerHTML = "name = " + xhr.response.name + "</br>" + "age = " + xhr.response.age;  
 }else {  
  
 }  
 }  
 }  
 }  
</script>  
</body>  
</html>

# nodemon自动重启工具安装

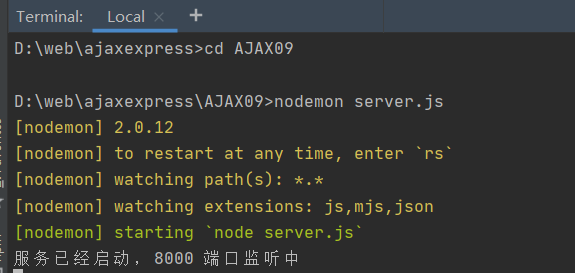
·自动重启node.js相关工具的应用





·重新启动server服务

- nodemon server.js



# IE缓存问题解决

·在一些浏览器中(IE),由于缓存机制的存在，ajax 只会发送的第一次请求，剩 余多次请求不会在发送给浏览器而是直接加载缓存中的数据。

·浏览器的缓存是根据 url 地址来记录的，所以我们只需要修改 url 地址 即可避免缓存问题

·get和post的解决方法相同

·传入t=当前时间的参数

//初始化 设置请求方法和url，并设置需要传递的参数  
xhr.open("GET","http://localhost:8000/ie-server?t=" + ***Date***.now());

//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//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  
//如果url路径是/server则执行以下代码  
//get请求  
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 GET");  
});  
  
//post请求  
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 POST");  
});  
  
//任意类型请求  
app.all("/json-server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //所有响应头信息都可以添加  
 response.setHeader("Access-Control-Allow-Headers","\*");  
 //响应一个数据的对象  
 const data = {  
 name: "atguigu",  
 age:18  
 }  
 //将数据转换成json字符串  
 let str = ***JSON***.stringify(data);  
 //设置响应体  
 response.send(str);  
});  
  
//针对IE缓存  
app.get("/ie-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 IE - 2");  
});  
  
//针对IE缓存  
app.post("/ie-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 IE - 3");  
});  
  
//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");  
})

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>IE缓存问题解决</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
<button>点击发送请求</button>  
<div id="result"></div>  
<script type="text/javascript">  
 //获取button元素  
 const ***btn*** = ***document***.getElementsByTagName("button")[0];  
 //获取响应结果div  
 const ***result*** = ***document***.getElementById("result");  
 //绑定事件  
 ***btn***.onclick = function () {  
 //创建ajax对象  
 const xhr = new ***XMLHttpRequest***();  
 //初始化 设置请求方法和url，并设置需要传递的参数  
 xhr.open("GET","http://localhost:8000/ie-server?t=" + ***Date***.now());  
 // xhr.open("POST","http://localhost:8000/ie-server?t=" + Date.now());  
 //发送  
 xhr.send();  
 //事件绑定 处理服务端返回的结果  
 //on这个单词有when的意思，当...时  
 //readystate（准备状态）是xhr对象中的属性，表示状态 0 （被初始化）1（open方法调用完毕） 2（send方法调用完毕） 3（服务端返回了部分结果） 4（服务端返回了所有结果）  
 //change（改变），当改变的时候触发  
 //会触发四次  
 xhr.onreadystatechange = function () {  
 //xhr.readyState获取返回状态  
 if (xhr.readyState === 4){//当服务端返回了所有结果时  
 //判断响应状态码 200 404 403 401 500  
 //响应状态码中2开头的表示成功，并非只有200  
 if (xhr.status >=200 && xhr.status < 300){//xhr.status获取状态码，获取结果成功时  
 //处理结果 行 头 空行 体  
 // console.log(xhr.status);//响应行的状态码  
 // console.log(xhr.statusText);//响应行的状态字符串  
 // console.log(xhr.getAllResponseHeaders())//所有响应头  
 // console.log(xhr.response);//响应体  
 //将响应体显示到div中  
 ***result***.innerHTML = xhr.response;  
 }else {  
  
 }  
 }  
 }  
 }  
</script>  
</body>  
</html>

# AJAX请求超时与网络异常处理

//超时设置，如果2秒后还没获取到数据，则提示  
xhr.timeout = 2000;  
//超时提示  
xhr.ontimeout = function(){  
 ***result***.innerHTML = "请求超时，请稍后重试";  
}

//网络异常回调  
xhr.onerror = function(){  
 ***result***.innerHTML = "网络异常，请稍后重试";  
}

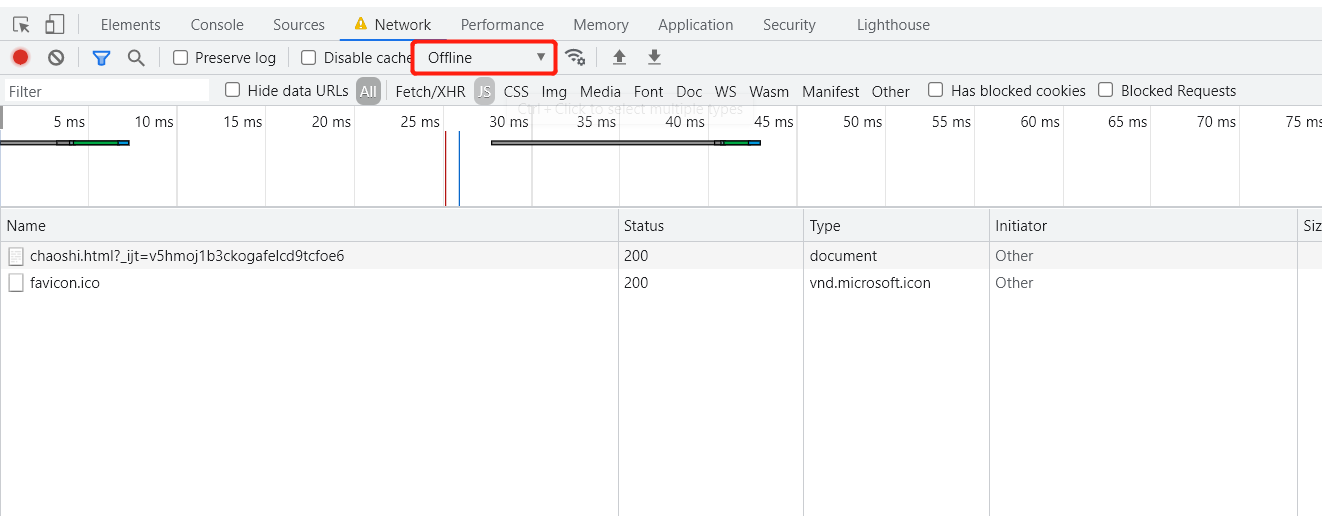
//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//创建路由规则  
//request 是对请求报文的封装  
//response 是对响应报文的封装  
//如果url路径是/server则执行以下代码  
//get请求  
app.get("/server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //设置响应体  
 response.send("HELLO AJAX GET");  
});  
  
//post请求  
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 POST");  
});  
  
//任意类型请求  
app.all("/json-server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //所有响应头信息都可以添加  
 response.setHeader("Access-Control-Allow-Headers","\*");  
 //响应一个数据的对象  
 const data = {  
 name: "atguigu",  
 age:18  
 }  
 //将数据转换成json字符串  
 let str = ***JSON***.stringify(data);  
 //设置响应体  
 response.send(str);  
});  
  
//针对IE缓存  
app.get("/ie-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 IE - 2");  
});  
  
//针对IE缓存  
app.post("/ie-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 IE - 3");  
});  
  
//针对请求超市与网络异常  
app.get("/delay-server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //添加定时器，模拟网络延时，为了测试  
 setTimeout(()=>{  
 //设置响应体  
 response.send("3s后发送");  
 },3000);  
});  
  
  
//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");  
})

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>请求超时与网络异常处理</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
<button>点击发送请求</button>  
<div id="result"></div>  
<script type="text/javascript">  
 //获取button元素  
 const ***btn*** = ***document***.getElementsByTagName("button")[0];  
 //获取响应结果div  
 const ***result*** = ***document***.getElementById("result");  
 //绑定事件  
 ***btn***.onclick = function () {  
 //创建ajax对象  
 const xhr = new ***XMLHttpRequest***();  
 //超时设置，如果2秒后还没获取到数据，则提示  
 xhr.timeout = 2000;  
 //超时提示  
 xhr.ontimeout = function(){  
 ***result***.innerHTML = "请求超时，请稍后重试";  
 }  
 //网络异常回调  
 xhr.onerror = function(){  
 ***result***.innerHTML = "网络异常，请稍后重试";  
 }  
 //初始化 设置请求方法和url，并设置需要传递的参数  
 xhr.open("GET","http://localhost:8000/delay-server");  
 //发送  
 xhr.send();  
 //事件绑定 处理服务端返回的结果  
 //on这个单词有when的意思，当...时  
 //readystate（准备状态）是xhr对象中的属性，表示状态 0 （被初始化）1（open方法调用完毕） 2（send方法调用完毕） 3（服务端返回了部分结果） 4（服务端返回了所有结果）  
 //change（改变），当改变的时候触发  
 //会触发四次  
 xhr.onreadystatechange = function () {  
 //xhr.readyState获取返回状态  
 if (xhr.readyState === 4){//当服务端返回了所有结果时  
 //判断响应状态码 200 404 403 401 500  
 //响应状态码中2开头的表示成功，并非只有200  
 if (xhr.status >=200 && xhr.status < 300){//xhr.status获取状态码，获取结果成功时  
 //处理结果 行 头 空行 体  
 // console.log(xhr.status);//响应行的状态码  
 // console.log(xhr.statusText);//响应行的状态字符串  
 // console.log(xhr.getAllResponseHeaders())//所有响应头  
 // console.log(xhr.response);//响应体  
 //将响应体显示到div中  
 ***result***.innerHTML = xhr.response;  
 }else {  
  
 }  
 }  
 }  
 }  
</script>  
</body>  
</html>

·模拟网络延时的方法

//添加定时器，模拟网络延时，为了测试  
setTimeout(()=>{  
 //设置响应体  
 response.send("3s后发送");  
},3000);

·模拟网络异常的方法



# AJAX取消请求

//取消请求  
***xhr***.abort();

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>AJAX取消请求</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
<button>点击发送请求</button>  
<button>点击取消请求</button>  
<div id="result"></div>  
<script type="text/javascript">  
 //获取发送请求按钮  
 const ***btn*** = ***document***.getElementsByTagName("button")[0];  
 //获取取消请求按钮  
 const ***btn2*** = ***document***.getElementsByTagName("button")[1];  
 //获取响应结果div  
 const ***result*** = ***document***.getElementById("result");  
 //保存xhr  
 let ***xhr*** = null;  
 //绑定发送请求响应事件  
 ***btn***.onclick = function () {  
 //创建ajax对象  
 ***xhr*** = new ***XMLHttpRequest***();  
 //初始化 设置请求方法和url，并设置需要传递的参数  
 ***xhr***.open("GET","http://localhost:8000/delay-server");  
 //发送  
 ***xhr***.send();  
 //事件绑定 处理服务端返回的结果  
 //on这个单词有when的意思，当...时  
 //readystate（准备状态）是xhr对象中的属性，表示状态 0 （被初始化）1（open方法调用完毕） 2（send方法调用完毕） 3（服务端返回了部分结果） 4（服务端返回了所有结果）  
 //change（改变），当改变的时候触发  
 //会触发四次  
 ***xhr***.onreadystatechange = function () {  
 //xhr.readyState获取返回状态  
 if (***xhr***.readyState === 4){//当服务端返回了所有结果时  
 //判断响应状态码 200 404 403 401 500  
 //响应状态码中2开头的表示成功，并非只有200  
 if (***xhr***.status >=200 && ***xhr***.status < 300){//xhr.status获取状态码，获取结果成功时  
 //处理结果 行 头 空行 体  
 // console.log(xhr.status);//响应行的状态码  
 // console.log(xhr.statusText);//响应行的状态字符串  
 // console.log(xhr.getAllResponseHeaders())//所有响应头  
 // console.log(xhr.response);//响应体  
 //将响应体显示到div中  
 ***result***.innerHTML = ***xhr***.response;  
 }else {  
  
 }  
 }  
 }  
 }  
  
 //绑定取消请求响应事件  
 ***btn2***.onclick = function () {  
 //取消请求  
 ***xhr***.abort();  
 }  
</script>  
</body>  
</html>

# AJAX请求重复发送问题

·如果服务器响应慢，用户又疯狂发送请求，服务器就会接收很多相同的请求

·如果之前有相同请求，则把之前发送的请求取消

//标识请求是否正在发送AJAX请求的变量，初始为未发送  
let ***isSending*** = false;

//判断标识变量，如果正在发送请求  
if (***isSending***){  
 //取消之前的发送请求  
 ***xhr***.abort();  
 ***console***.log("正在发送请求中");  
}

//将发送状态设为true，表示正在发送请求  
***isSending*** = true;

//请求发送完成时，将发送状态设为false  
***isSending*** = false;

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>重复请求问题处理</title>  
 <style type="text/css">  
 #result{  
 width: 200px;  
 height: 100px;  
 border: solid 1px #00B7FF;  
 }  
 </style>  
</head>  
<body>  
<button>点击发送请求</button>  
<div id="result"></div>  
<script type="text/javascript">  
 //获取发送请求按钮  
 const ***btn*** = ***document***.getElementsByTagName("button")[0];  
 //获取响应结果div  
 const ***result*** = ***document***.getElementById("result");  
 //标识请求是否正在发送AJAX请求的变量，初始为未发送  
 let ***isSending*** = false;  
 //保存xhr  
 let ***xhr*** = null;  
 //绑定发送请求响应事件  
 ***btn***.onclick = function () {  
 //判断标识变量，如果正在发送请求  
 if (***isSending***){  
 //取消之前的发送请求  
 ***xhr***.abort();  
 ***console***.log("正在发送请求中");  
 }  
 //创建ajax对象  
 ***xhr*** = new ***XMLHttpRequest***();  
 //将发送状态设为true，表示正在发送请求  
 ***isSending*** = true;  
 //初始化 设置请求方法和url，并设置需要传递的参数  
 ***xhr***.open("GET","http://localhost:8000/delay-server");  
 //发送  
 ***xhr***.send();  
 //事件绑定 处理服务端返回的结果  
 //on这个单词有when的意思，当...时  
 //readystate（准备状态）是xhr对象中的属性，表示状态 0 （被初始化）1（open方法调用完毕） 2（send方法调用完毕） 3（服务端返回了部分结果） 4（服务端返回了所有结果）  
 //change（改变），当改变的时候触发  
 //会触发四次  
 ***xhr***.onreadystatechange = function () {  
 //xhr.readyState获取返回状态  
 if (***xhr***.readyState === 4){//当服务端返回了所有结果时  
 //请求发送完成时，将发送状态设为false  
 ***isSending*** = false;  
 //判断响应状态码 200 404 403 401 500  
 //响应状态码中2开头的表示成功，并非只有200  
 if (***xhr***.status >=200 && ***xhr***.status < 300){//xhr.status获取状态码，获取结果成功时  
 //处理结果 行 头 空行 体  
 // console.log(xhr.status);//响应行的状态码  
 // console.log(xhr.statusText);//响应行的状态字符串  
 // console.log(xhr.getAllResponseHeaders())//所有响应头  
 // console.log(xhr.response);//响应体  
 //将响应体显示到div中  
 ***result***.innerHTML = ***xhr***.response;  
 }else {  
  
 }  
 }  
 }  
 }  
  
</script>  
</body>  
</html>

# jQuery发送AJAX请求

//jQuery发送AJAX请求  
app.all("/jQuery-server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 const data = {name:"张三",age:18};  
 //设置响应体  
 response.send(***JSON***.stringify(data));  
  
});

//jQuery发送GET请求  
//第一个参数是请求服务的url  
//第二个参数是要发送的参数对象  
//第三个参数是一个回调函数,data是响应体  
//第四个参数是响应体类型，可传可不传  
$.get("http://localhost:8000/jQuery-server",{a:100,b:200,c:300},function (data) {  
 //打印响应体  
 ***console***.log(data);  
},"json");

//jQuery发送通用型请求  
$.ajax({  
 //url  
 url: "http://localhost:8000/jQuery-server",  
 //参数  
 data:{  
 a:100,  
 b:200,  
 c:300  
 },  
 //请求类型  
 type:"GET",  
 //成功的回调  
 success:function (data) {  
 ***console***.log(data);  
 }  
});

//jQuery发送AJAX请求  
app.all("/jQuery-server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //所有响应头信息都可以添加  
 response.setHeader("Access-Control-Allow-Headers","\*");  
 //添加定时器，模拟网络延时，为了测试  
 setTimeout(()=>{  
 const data = {name:"张三",age:18};  
 //设置响应体  
 response.send(***JSON***.stringify(data));  
 },200);  
});

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>jQuery 发送 AJAX 请求</title>  
 <!--crossorigin="anonymous"可以防止警告-->  
 <link crossorigin="anonymous" href="https://cdn.bootcss.com/twitter-bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">  
 <script crossorigin="anonymous" src="https://cdn.bootcdn.net/ajax/libs/jquery/3.5.1/jquery.min.js"></script>  
</head>  
<body>  
<div class="container">  
 <h2 class="page-header">jQuery发送AJAX请求 </h2>  
 <button class="btn btn-primary">GET</button>  
 <button class="btn btn-danger">POST</button>  
 <button class="btn btn-info">通用型方法ajax</button>  
</div>  
<script>  
 var ***$button*** = $("button");  
 ***$button***.eq(0).click(function () {  
 //jQuery发送GET请求  
 //第一个参数是请求服务的url  
 //第二个参数是要发送的参数对象  
 //第三个参数是一个回调函数,data是响应体  
 //第四个参数是响应体类型，可传可不传  
 $.get("http://localhost:8000/jQuery-server",{a:100,b:200,c:300},function (data) {  
 //打印响应体  
 ***console***.log(data);  
 },"json");  
 })  
  
 ***$button***.eq(1).click(function () {  
 //jQuery发送post请求  
 //第一个参数是请求服务的url  
 //第二个参数是要发送的参数对象  
 //第三个参数是一个回调函数,data是响应体  
 $.post("http://localhost:8000/jQuery-server",{a:100,b:200,c:300},function (data) {  
 //打印响应体  
 ***console***.log(data);  
 })  
 })  
  
 ***$button***.eq(2).click(function () {  
 //jQuery发送通用型请求  
 $.ajax({  
 //url  
 url: "http://localhost:8000/jQuery-server",  
 //参数  
 data:{  
 a:100,  
 b:200,  
 c:300  
 },  
 //响应体结果，可穿可不传  
 dataType:"json",  
 //请求类型  
 type:"GET",  
 //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！！");  
 },  
 headers:{  
 c:300,  
 d:400  
 }  
 });  
 })  
</script>  
</body>  
</html>

# Axios发送AJAX请求

·文档地址

- <https://github.com/axios/axios>

<!--引入axios-->  
<script crossorigin="anonymous" src="https://cdn.bootcdn.net/ajax/libs/axios/0.19.2/axios.js"></script>

//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//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  
//如果url路径是/server则执行以下代码  
//get请求  
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 GET");  
});  
  
//post请求  
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 POST");  
});  
  
//任意类型请求  
app.all("/json-server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //所有响应头信息都可以添加  
 response.setHeader("Access-Control-Allow-Headers","\*");  
 //响应一个数据的对象  
 const data = {  
 name: "atguigu",  
 age:18  
 }  
 //将数据转换成json字符串  
 let str = ***JSON***.stringify(data);  
 //设置响应体  
 response.send(str);  
});  
  
//针对IE缓存  
app.get("/ie-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 IE - 2");  
});  
  
//针对IE缓存  
app.post("/ie-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 IE - 3");  
});  
  
//针对请求超市与网络异常  
app.get("/delay-server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //添加定时器，模拟网络延时，为了测试  
 setTimeout(()=>{  
 //设置响应体  
 response.send("3s后发送");  
 },3000);  
});  
  
//jQuery发送AJAX请求  
app.all("/jQuery-server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //所有响应头信息都可以添加  
 response.setHeader("Access-Control-Allow-Headers","\*");  
 //添加定时器，模拟网络延时，为了测试  
 setTimeout(()=>{  
 const data = {name:"张三",age:18};  
 //设置响应体  
 response.send(***JSON***.stringify(data));  
 },200);  
});  
  
//axios发送AJAX请求  
app.all("/axios-server",(request,response)=>{  
 //设置响应头，设置允许跨域  
 response.setHeader("Access-Control-Allow-Origin","\*");  
 //所有响应头信息都可以添加  
 response.setHeader("Access-Control-Allow-Headers","\*");  
 const data = {name:"张三",age:18};  
 //设置响应体  
 response.send(***JSON***.stringify(data));  
});  
  
//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");  
})

<!DOCTYPE html>  
<html lang="en">  
  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>axios 发送 AJAX请求</title>  
 <!--引入axios-->  
 <script crossorigin="anonymous" src="https://cdn.bootcdn.net/ajax/libs/axios/0.19.2/axios.js"></script>  
</head>  
  
<body>  
<button>GET</button>  
<button>POST</button>  
<button>AJAX</button>  
  
<script>  
 const ***btns*** = ***document***.getElementsByTagName("button");  
 // btns[0].onclick = function () {  
 // //axios发送GET请求  
 // //第一个参数是请求服务的url  
 // //第二个参数是要发送的参数对象  
 // axios.get("http://localhost:8000/axios-server",{  
 // //url参数  
 // params:{  
 // id:100,  
 // vip:7  
 // },  
 // //请求头信息  
 // headers:{  
 // name:"atguigu",  
 // age:"18"  
 // }  
 // });  
 // }  
  
 //配置 baseURL  
 axios.defaults.baseURL = 'http://127.0.0.1:8000';  
  
 ***btns***[0].onclick = function () {  
 //GET 请求  
 axios.get('/axios-server', {  
 //url 参数  
 params: {  
 id: 100,  
 vip: 7  
 },  
 //请求头信息  
 headers: {  
 name: 'atguigu',  
 age: 20  
 }  
 //下面的then是数据返回结果处理,value是返回结果的对象，包含返回的所有信息  
 }).then(value => {  
 ***console***.log(value);  
 });  
 }  
  
  
 ***btns***[1].onclick = function () {  
 //POST请求  
 //第一个参数是url  
 //第二个参数是请求体  
 //第三个参数是配置  
 axios.post('/axios-server', {  
 username: 'admin',  
 password: 'admin'  
 }, {  
 //url 参数  
 params: {  
 id: 200,  
 vip: 9  
 },  
 //请求头参数  
 headers: {  
 height: 180,  
 weight: 180,  
 }  
 });  
 }  
  
 ***btns***[2].onclick = function () {  
 axios({  
 //请求方法  
 method:"POST",  
 //url  
 url:"/axios-server",  
 //url参数  
 params:{  
 vip:10,  
 level:30  
 },  
 //头信息  
 headers:{  
 a:100,  
 b:200  
 },  
 //请求体参数  
 data:{  
 username: 'admin',  
 password: 'admin'  
 }  
 }).then(response=>{  
 ***console***.log(response);  
 //响应状态码  
 ***console***.log(response.status);  
 })  
 }  
</script>  
</body>  
  
</html>

# 使用fetch函数发送AJAX请求

·文档地址

<https://developer.mozilla.org/zh-CN/docs/Web/API/WindowOrWorkerGlobalScope/fetch>

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>fetch 发送 AJAX请求</title>  
</head>  
<body>  
<button>AJAX请求</button>  
<script>  
  
 const ***btn*** = ***document***.querySelector('button');  
  
 ***btn***.onclick = function(){  
 //第一个参数是url和url参数  
 //第二个参数是配置请求方法、请求头和请求体的对象  
 fetch("http://localhost:8000/fetch-server?vip=10",{  
 //请求方法  
 method:"POST",  
 //请求头  
 headers:{  
 name:"atguigu",  
 age:18  
 },  
 //请求体  
 body:"username=admin&password=admin"  
 }).then(data =>{  
 // console.log(data);  
 //获取文本结果  
 // return data.text();  
 //获取json结果  
 return data.json();  
 }).then(data =>{  
 ***console***.log(data);  
 })  
 }  
</script>  
</body>  
</html>

# 跨域

## 同源策略

·同源策略(Same-Origin Policy)最早由 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。 违背同源策略就是跨域。

·AJAX默认遵循同源策略

·满足同源策略的AJAX请求

const express = require('express');  
  
const app = express();  
  
app.get('/home', (request, response)=>{  
 //打开一个页面  
 //\_\_dirname是当前js文件的绝对路径  
 //./是执行node命令的相对路径  
 response.sendFile(\_\_dirname + '/index.html');  
});  
  
app.get('/data', (request, response)=>{  
 //打开一个页面  
 //\_\_dirname是当前js文件的绝对路径  
 //./是执行node命令的相对路径  
 response.sendFile(\_\_dirname + '/htmls/ceshi.html');  
});  
  
app.listen(9000, ()=>{  
 ***console***.log("服务已经启动...");  
});

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>首页</title>  
</head>  
<body>  
 <h1>尚硅谷</h1>  
 <button>点击获取用户数据</button>  
 <div id="result">  
  
 </div>  
 <script>  
 const ***btn*** = ***document***.querySelector('button');  
 const ***result*** = ***document***.getElementById("result");  
 ***btn***.onclick = function(){  
 const x = new ***XMLHttpRequest***();  
 //这里因为是满足同源策略的, 所以 url 可以简写  
 x.open("GET",'/data');  
 //发送  
 x.send();  
 //  
 x.onreadystatechange = function(){  
 if(x.readyState === 4){  
 if(x.status >= 200 && x.status < 300){  
 // console.log(x.response);  
 ***result***.innerHTML = x.response;  
 }  
 }  
 }  
 }  
 </script>  
</body>  
</html>

# 如何解决跨域

·违背同源策略就是跨域

·跨域经常出现

## JSONP解决跨域

·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

·在网页有一些标签天生具有跨域能力，比如：img link iframe script。 JSONP 就是利用 script 标签的跨域能力来发送请求的

### JSONP的实现原理

//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//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  
//如果url路径是/server则执行以下代码  
//get请求  
  
app.all("/jsonp-server",function (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("console.log('hello jsonp')");  
 const data = {  
 name: '尚硅谷atguigu'  
 };  
 //将js对象转换为json字符串  
 let str = ***JSON***.stringify(data);  
 //返回结果  
 //end不会加响应头，将str传，不是分号，是``  
 // response.end(`handle(${str})`);  
 response.end("handle("+str+")");  
});  
  
  
//监听端口启动服务  
app.listen(8000,function () {  
 ***console***.log("服务已经启动，8000 端口监听中");  
})

<!DOCTYPE html>  
<html lang="en">  
  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>原理演示</title>  
 <style>  
 #result {  
 width: 300px;  
 height: 100px;  
 border: solid 1px #78a;  
 }  
 </style>  
</head>  
  
<body>  
 <div id="result"></div>  
 <script>  
 //处理数据  
 function handle(data) {  
 //获取 result 元素  
 const result = ***document***.getElementById('result');  
 result.innerHTML = data.name;  
 }  
 </script>  
 <!-- <script src="http://127.0.0.1:5500/%E8%AF%BE%E5%A0%82/%E4%BB%A3%E7%A0%81/7-%E8%B7%A8%E5%9F%9F/2-JSONP/js/app.js"></script> -->  
 <script src="http://127.0.0.1:8000/jsonp-server"></script>  
</body>  
  
</html>

### 原生JSONP实践

//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//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  
//如果url路径是/server则执行以下代码  
//get请求  
  
app.all("/jsonp-server",function (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("console.log('hello jsonp')");  
 const data = {  
 name: '尚硅谷atguigu'  
 };  
 //将js对象转换为json字符串  
 let str = ***JSON***.stringify(data);  
 //返回结果  
 //end不会加响应头，将str传，不是分号，是``  
 // response.end(`handle(${str})`);  
 response.end("handle("+str+")");  
});  
  
//用户名检测是否存在  
app.all("/check-username",function (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("console.log('hello jsonp')");  
 const data = {  
 exist:1,  
 msg:'用户名已经存在'  
 };  
 //将js对象转换为json字符串  
 let str = ***JSON***.stringify(data);  
 //返回结果  
 //end不会加响应头，将str传，不是分号，是``  
 // response.end(`handle(${str})`);  
 response.end("handle("+str+")");  
});  
  
//监听端口启动服务  
app.listen(8000,function () {  
 ***console***.log("服务已经启动，8000 端口监听中");  
})

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>案例</title>  
</head>  
<body>  
 用户名: <input type="text" id="username">  
 <p></p>  
 <script>  
 //获取 input 元素  
 const ***input*** = ***document***.querySelector('input');  
 const ***p*** = ***document***.querySelector('p');  
   
 //声明 handle 函数  
 function handle(data){  
 //将边框变红  
 ***input***.style.border = "solid 1px #f00";  
 //修改 p 标签的提示文本  
 ***p***.innerHTML = data.msg;  
 }  
  
 //绑定事件  
 ***input***.onblur = function(){  
 //获取用户的输入值  
 let username = this.value;  
 //向服务器端发送请求 检测用户名是否存在  
 //1. 创建 script 标签  
 const 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 插入到文档中  
 ***document***.body.appendChild(script);  
 }  
 </script>  
</body>  
</html>

### jQuery发送jsonp请求

//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//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  
//如果url路径是/server则执行以下代码  
//get请求  
  
app.all("/jsonp-server",function (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("console.log('hello jsonp')");  
 const data = {  
 name: '尚硅谷atguigu'  
 };  
 //将js对象转换为json字符串  
 let str = ***JSON***.stringify(data);  
 //返回结果  
 //end不会加响应头，将str传，不是分号，是``  
 // response.end(`handle(${str})`);  
 response.end("handle("+str+")");  
});  
  
//用户名检测是否存在  
app.all("/check-username",function (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("console.log('hello jsonp')");  
 const data = {  
 exist:1,  
 msg:'用户名已经存在'  
 };  
 //将js对象转换为json字符串  
 let str = ***JSON***.stringify(data);  
 //返回结果  
 //end不会加响应头，将str传，不是分号，是``  
 // response.end(`handle(${str})`);  
 response.end("handle("+str+")");  
});  
  
  
app.all("/jquery-jsonp-server",function (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("console.log('hello jsonp')");  
 const data = {  
 name: "尚硅谷",  
 city:["北京","上海","深圳"]  
 };  
 //将js对象转换为json字符串  
 let str = ***JSON***.stringify(data);  
 //接收 callback参数  
 let cb = request.query.callback;  
 //返回结果  
 //end不会加响应头，将str传，不是分号，是``  
 // response.end(`handle(${str})`);  
 response.end(cb + "("+str+")");  
});  
//监听端口启动服务  
app.listen(8000,function () {  
 ***console***.log("服务已经启动，8000 端口监听中");  
})

# 设置CORS响应头实现跨域

·https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access\_control\_CORS

·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 请求。跨域资源共享标准新增了一组 HTTP 首部字段，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

·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

//引入express  
const express = require("express");  
//创建应用对象  
const app = express();  
//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  
//如果url路径是/server则执行以下代码  
//get请求  
  
app.all("/jsonp-server",function (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("console.log('hello jsonp')");  
 const data = {  
 name: '尚硅谷atguigu'  
 };  
 //将js对象转换为json字符串  
 let str = ***JSON***.stringify(data);  
 //返回结果  
 //end不会加响应头，将str传，不是分号，是``  
 // response.end(`handle(${str})`);  
 response.end("handle("+str+")");  
});  
  
//用户名检测是否存在  
app.all("/check-username",function (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("console.log('hello jsonp')");  
 const data = {  
 exist:1,  
 msg:'用户名已经存在'  
 };  
 //将js对象转换为json字符串  
 let str = ***JSON***.stringify(data);  
 //返回结果  
 //end不会加响应头，将str传，不是分号，是``  
 // response.end(`handle(${str})`);  
 response.end("handle("+str+")");  
});  
  
  
app.all("/jquery-jsonp-server",function (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("console.log('hello jsonp')");  
 const data = {  
 name: "尚硅谷",  
 city:["北京","上海","深圳"]  
 };  
 //将js对象转换为json字符串  
 let str = ***JSON***.stringify(data);  
 //接收 callback参数  
 let cb = request.query.callback;  
 //返回结果  
 //end不会加响应头，将str传，不是分号，是``  
 // response.end(`handle(${str})`);  
 response.end(cb + "("+str+")");  
});  
  
//cors发送AJAX请求  
app.all("/cors-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 CORS");  
});  
//监听端口启动服务  
app.listen(8000,function () {  
 ***console***.log("服务已经启动，8000 端口监听中");  
})

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>CORS</title>  
 <style>  
 #result{  
 width:200px;  
 height:100px;  
 border:solid 1px #90b;  
 }  
 </style>  
</head>  
<body>  
 <button>发送请求</button>  
 <div id="result"></div>  
 <script>  
 const ***btn*** = ***document***.querySelector('button');  
  
 ***btn***.onclick = function(){  
 //1. 创建对象  
 const x = new ***XMLHttpRequest***();  
 //2. 初始化设置  
 x.open("GET", "http://127.0.0.1:8000/cors-server");  
 //3. 发送  
 x.send();  
 //4. 绑定事件  
 x.onreadystatechange = function(){  
 if(x.readyState === 4){  
 if(x.status >= 200 && x.status < 300){  
 //输出响应体  
 ***console***.log(x.response);  
 }  
 }  
 }  
 }  
 </script>  
</body>  
</html>