1、ajax基本介绍：

ajasx：异步的js和xml

可以在不刷新页面的情况下，向服务器发送http请求，和接收服务器的http相应。

XML：可扩展标记语言，由标签组成，与HTML的区别：

\*XML用来传输和存储数据，HTML用来呈现数据

\*XML没有预定义的标签，标签都是自定义的

早期的ajax使用的是xml格式，服务器返回的也是xml格式的字符串，不过现在的ajax使用的都是json格式，json格式更简洁，且json有自带的数据转换api，数据转换更方便。

JSON:

数据格式基本与js数据类型一样，JSON.xxx()可以将js的数据类型转换成JSON格式，如JSON.stringify()可以将js数据转换成JSON字符串

ajax特点：

优点：

\*可以在不刷新页面情况下发送网络请求

\*允许用户的事件更新页面的内容，即可以在事件函数中发送网络请求

缺点：

\*没有浏览历史，不能回退

\*不能跨域（跨域：一个域名向另一个域名发送请求，JSONP可解决）

\*SEO(搜索引擎优化)不友好，因为页面展示的的数据，大部分是网络请求过来的数据保存在变量中的，这样爬虫就无法通过源代码获得数据

2、GET请求，POST请求

GET和POST是最常用的两种请求类型。

http协议层面：

\*都是http的请求方式，底层都是TCP/IP协议

\*通常GET产生一个TCP数据包，POST产生两个TCP数据包。在网络好的情况下，发一个两个数据包速度没有太大差异，GET/POST都可使用；但是在网络差的情况下，GET可定比POST速度快。

\*GET会把Header和data一起发送，服务器返回200则成功；POST先发送Header，等服务器返回100后再发送data，服务器返回200则成功。

GET请求通常通过url来获取数据，POST请求通常通过表单来发送数据

GET请求和POST请求都可以传递参数

区别：

\*最本质区别：GET的参数包含在url，POST的参数在请求体中

\*GET请求参数有大小限制且只能接受Ascll编码的数据且格式只能url编码，POST请求参数没有大小限制且没有数据的类型限制且有多种格式(也支持url编码)

\*GET参数显示在url上，安全性隐私性差，不适合传递敏感数据，POST在请求体中更安全

\*GET请求会保留在浏览器历史记录中且可bookmark(收藏/书签)，而POST不会

\*GET请求在刷新/回退网页时，无影响；而POST会再次发送请求，因此GET效率更高

\*GET请求会被缓存，POST请求不会被缓存，所以GET效率更高

3、http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，主要是规定了请求(request)和响应(response)两块内容。

浏览器向服务器发送请求，发送内容是请求报文，服务器响应返回的内容是响应报文。

（1）请求报文格式：

\*请求行

请求类型(get,post最常见)--url(query参数)--http协议版本(1.1版本最常用)

\*请求头

host--cookie--content-type(请求体类型)--。。。。。 格式是 名字+:+空格+值

\*请求空行

固定的必须有

\*请求体

GET请求则内容为空，post请求则内容可以不为空

（2）响应报文格式：

\*响应行

http协议版本—状态码—响应状态字符层

\*响应头

对响应体的描述信息 格式是 名字+:+空格+值

\*响应空行

固定的必须有

\*响应体

响应结果

状态码：

\*1xx 服务器收到，需要请求者继续执行 100:服务器收到POST的Header，需要请求者再发送data

\*2xx 表示成功，一般是200

\*3xx 重定向，需要进一步操作来完成请求 301:资源/网页被永久转移到其他url

\*4xx 客户端错误，请求有语法错误或无法完成请求 404:请求的资源/网页不存在

\*5xx 服务器错误，服务器在处理请求过程中发生错误 500:内部服务器错误

F12—network—All--Name中是项目的所有html/js等，点开能看到请求/响应报文

其中，QueryString 能看到GET请求的query参数和POST请求的参数

4、express

ajax需要向服务端请求数据，而express能简便的搭建一个服务器

基于nodeJS的服务端框架

先建立一个存放服务器的文件夹，在该目录下：

npm install express --save

建立xxx.js文件，编写服务端代码：

const express=require('express')

//创建express对象

const app=express()

//创建路由规则

//GET请求时回调

app.get('/aaa',(request,response)=>{ //‘/aaa’是url

//设置允许跨域

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

//响应体

response.send('123')

})

//POST请求时会带哦

app.post('/aaa',(request,response)=>{

//设置允许跨域

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

//响应体

response.send('666')

})

//监听端口，一个端口只能有一个服务监听

app.listen(8000,()=>[

console.log('服务已启动') //启动服务器后的提示信息

])

启动服务器：

node xxx.js

自动重启服务器：

可以在修改，保存服务器js代码后自动重启

npm install nodemon --szve

启动服务器：npx nodemon xxx.js

127.0.0.1:8000/url 可以调试

5、原生ajax发送请求

如在事件函数中：

//1.创建对象

let xhr=new XMLHttpRequest()

/\* 2.初始化，设置请求类型和url，GET请求可以加上query参数 ?k=v&&k=v \*/

xhr.open('GET','http://127.0.0.1:8000/aaa?a=100&&b=200')

//设置请求头，可不写，参数为请求体信息和send()的参数查询字符串的类型

//通常用来做身份校验信息；也可以自定义信息，但是需要服务端特殊处理才能设置

xhr.setRequestHeader('Content-Type','application/x-www-form-urlencoded')

//3.GET请求参数为空，POST请求需要传入参数，参数格式需要服务端能够处理解析的格式

//一般有两种格式 k=v&&k=v 或 k:v&&k:v

xhr.send()

//4.事件绑定，处理服务端返回的结果

/\* xhr.readystate的值 0:未初始化 1:open()调用完毕 2:send()调用完毕 3:服务端返回了部分结果 4:服务端返回了全部结果

该事件是每当xhr.readystate发生改变就触发，共触发4次 0变1变2变3变4 \*/

xhr.onreadystatechange=function(){

//当服务端返回了全部结果时，才处理

if(xhr.readyState==4){

//当状态码为成功时，才处理

if(xhr.status>=200 && xhr.status<=300){

//处理响应结果

console.log(xhr.status) //状态码

console.log(xhr.statusText) //状态字符串

console.log(xhr.getAllResponseHeaders) //相应头

console.log(xhr.response) //响应体

}

}

}

传输JSON数据：

服务端中：

若响应体只是字符串，直接传就可以

若想返回给客服端对象等数据类型，则需要将其转成JSON字符串：

JSON.stringify()

客户端接收：

若服务端返回的只是字符串，直接用就可以

若服务器返回的是转换后的JSON字符串，则组要转化，有两种方式：

\*主动转换：JSON.parse()

\*自动转换：onreadysttatechange事件函数外：xhr.responseType=’jspn’，数据就可以直接用了。

IE浏览器ajax缓存问题：

ajax发送请求得到相应结果后，IE浏览器会缓存相应结果，ajax再次发送请求时，只会得到缓存的相应结果，得不到新的相应结果：

解决：通过给url增加时间戳的query参数，使得每次的请求的url都不同，就不会使用缓存的相应结果。

open() 的url形参：

原：’…./xxx’ 解决：’…./xxx?t=’+Data.now()

请求超时处理：

onreadystatechange事件函数外：

xhr.timeout=数值毫秒 //超时后自动取消请求

xhr.ontimeout=function(){} //超时后回调

网络异常处理：

onreadystatechange事件函数外：

xhr.onerror=function(){} //网络异常时回调

主动取消请求：

在需要的地方，如点击事件内：xhr.abort()

但是发送请求和取消请求在不同的函数作用域，所以xhr要定义在全局：

let xhr=null //必须是let，因为xhr涉及改变

请求时再 xhr=new XMLHttpRequest()

取消请求时 xhr.abort()

解决重复发送相同请求：

当重复发送相同请求时，是没有必要的，服务器也会增大压力。相同的请求发一次就够了。

\*全局定义一个布尔变量let flag=false标识是否发送了请求

\*当flag为true时，取消请求

\*发送请求后，设为true

\*onreadystatechange事件函数内，当readystate==4时就设为false，不需要进入判断状态码。

5、jQuery的ajax使用：

<script

crossorigin="anonymous" src=<https://cdn.bootcdn.net/ajax/libs/jquery/3.5.1/jquery.min.js>

></script>

$.get(‘url’,{k=v,k=v},function(responseBody){},’json’) $.post()参数一样

$.ajax({

url:’...’,

data:{k=v,k=v},

dataType:’json’,

type:’GET’,

success:function(responseBody){},

error:function(){},

timeout: 2000,

header:{}

})

6、跨域：

同源策略：浏览器的一种安全策略，当前网页的url和ajax请求目标资源的url两者之间必须协议，域名，端口保持一致。满足同源策略时，发送请求的url可以省略协议/域名/端口

ajax默认遵循同源策略，无法跨域。

跨域就是违背同源策略，即协议，域名，端口任意一个不同就是跨域。（http和https是不同的协议）

跨域的解决方案：

分为前端解决和后端解决

（1）JSONP

非官方的跨域解决方案，前端解决，只支持GET请求，实现原理是借助script标签实现跨域（img，a，link，script标签本身就具备跨域特性）

原生实现：

服务端：

let data=….

data=JSON.stringify(data)

response.send(`func(${data})`)

客户端：

func(data){//处理}

需要发送请求的内部，如点击事件；

let s=document.createElement(‘script’)

s.src=’url’

document.body.append(s)

通过script标签发送请求，服务端返回一个函数的调用，从而执行func()

jQuery实现；

服务端：

let data=….

data=JSON.stringify(data)

let cb=request.query.callback

response.send(`cb(${data})`)

客户端：

$.getJSON(‘url?callback=?’,function(data){})

callback的值?其实是该回调函数，服务端获取该参数后，将其作为函数调用返回，然后在这个回调函数中对数据进行处理。

（2）CORS

中文为跨域资源共享，后端解决，官方的跨域解决方案，客户端不需要额外的操作，只需要在服务端处理就能完成跨域。支持GET，POST等请求。

服务端只需设置相应头：

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

第一个参数是固定写法，第二个参数是允许发送请求的url，\*表示所有url都允许

axios

原生ajax配置和使用麻烦，而jqeury-ajax有需要导入jquery，都不适合，而vue有专门的axios

安装

npm install axios --save npm install axios –g

测试网站：

httpbin.org

http://123.207.32.32:8000/home/multidata

使用

Import axios from ‘axios’

url参数格式：?key=’val’&key=’val’…

axios({ //配置选项除了url都可以省略

baseURL:’’, //最终url=baseURL+url，默认为空

url:’...’ ,

method: ‘…’ , //省略后默认为get请求

params: {…}, //get专用，可以将url的?后面的参数写在里面，简化url

data:{}, //poet专用，请求体

…..

}).then( res => {…}) //axios会return一个promise对象，then，catch在axios()后面

axios.请求类型(‘url’,{config})

fetch(‘url’,{config})

并发请求：

一个请求需要多个子请求都完成后再开始

除了Promise.all(),还有更简便的方法：

axios.all([ //传入数组

axios(),

axios()

]).then(res=>{}) //所有子请求的结果都在res数组里

若不想通过下标访问res数组，可以

then( axios.spread( (.. .. ..) => {} ) ) 就可以直接用小括号里的…变量直接访问

axios全局配置：

可以给配置选项设置默认值，删除重复代码，如

axios.default.baseURL=”…” //设置公共的url部分，简化url

然而，在真实项目中通常会有多个服务器，ip地址都不同，此时全局配置就不太合适了。

解决方法：

创建axios实例：

const instance1=axios.create(baseConfig) //创建axios实例并配置默认值

instance1(config) //使用与axios()一样

这样就可以创建多个实例来适配多个服务器

封装axios：

像axios这种几乎每个组件都要依赖的框架，如果在每个组件的script里都 import axios ，一旦不再使用该框架，就需要一个个去组件里修改代码，十分麻烦，所以需要对axios封装。

封装成一个js文件，以后更换框架时只需修改自己那个js文件就行。

src下新疆network文件夹，建立request.js，使用时倒入这个文件就行

这里拿到的请求结果，是要返回给调用这个函数的组件的，一般是传入一个回调函数。但是这里axios()本身会返回一个promise对象，在外部使用then就能拿到结果，就不要回调函数了。

import axios from 'axios'

export function request1(config){

//对该实例进行全局配置

const instance1=axios.create({

baseURL:'http://123.207.32.32:8000'

})

//拦截器

instance1.interceptors.request.use()

instance1.interceptors.response.use()

//instance是axios，会返回一个promise，通过return再将此promise返回到外部

return instance1(config)

}

export function request2(cpnfig){}

…

使用时：

Import {request1} from ‘…’

rsequest1({

config

}).then().catch() //请求结果在这里处理

拦截器：

分为请求拦截（拦截配置信息）和相应拦截（拦截请求结果），拦截成功后注意需要返回config和res，主要作用是对配置信息和请求结果进行一些处理，或其它特殊用途。

//拦截请求的配置信息

instance1.interceptors.request.use(config => {

//用处1 增删改查配置信息

//用处2 请求状态时，给页面增加爱一些动画效果

//用处3 请求需要token时(如登录)，提交登录信息

return config

},err => {

console.log(err)

})

//拦截请求结果

instance1.interceptors.response.use(res => {

//对请求结果进行过滤，去掉浏览器给的无用信息

return res.data

},err => {

console.log(err)

})

Vue跨域：

在config文件夹的index.js中的proxyTable中设置