AJAX+Node.js学习记录

Virgil233 已于 2023-11-13 16:01:03 修改

一、AJAX

（1）定义：AJAX是异步的JS和XML，使用XMLHttpRequest对象与服务器通信。可以使用JSON、XML、HTML和text文本等格式发送和接收数据。异步是指它可以在不重新刷新页面的情况下与服务器进行通信、交换数据或更新页面

（2）axios使用：

引入axios.js：https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js

使用axios函数：传入配置对象，再用.then回调函数接收结果并做后续处理

查询参数：params

```
axios({
    url:'目标资源地址',
    params:{
        参数名：值
    }
}).then((result) => {
    //对服务器返回的数据做后续处理
})
```

（3）常用请求方法：get获取数据；post提交数据；put修改数据（全部）；delete删除数据；patch修改数据（部分）


```
axios({
    url:'目标资源地址',
    method:'请求方法',
    data:{
        参数名：值
    }
}).then((result) => {
    //对服务器返回的数据做后续处理
```

```
})
```

(4) 错误处理：

```
axios({
    //请求选项
}).then((result) => {
    //处理数据
}).catch((error) => {
    //处理错误
})
```

(5) 接口文档：https://apifox.com/

(6) form-serialize插件：快速收集表单元素的值

语法：const data = serialize(form,{hash:true,empty:true})，hash设置获取数据结构，true为JS对象，false为查询字符串；empty设置是否获取空值

(7) 使用XMLHttpRequest：

```
/*创建XMLHttpRequest对象*/
const xhr = new XMLHttpRequest()
/*配置请求方法和请求url地址*/
xhr.open('请求方法', '请求url网址?参数名1=值1&参数名2=值2')
/*监听loadend事件，接收响应结果*/
xhr.addEventListener('loadend', () => {
    //响应结果
    console.log(xhr.response)
})
/*发起请求*/
xhr.send()
```


(8) 查询参数对象快捷转换为查询参数字符串

```
cosnt obj = {
    pname,
    cname
}
```

```
const paramsObj = new URLSearchParams(obj)
const queryString = paramsObj.toString()
```

(9) XMLHttpRequest数据提交：

注：xhr判断响应是否成功：看响应状态码xhr.status，如果是200-299之间则成功，否则失败

```
/*创建XMLHttpRequest对象*/
const xhr = new XMLHttpRequest()
/*配置请求方法和请求url地址*/
xhr.open('请求方法','请求url网址')
/*监听loadend事件，接收响应结果*/
xhr.addEventListener('loadend', () => {
    //响应结果
    console.log(xhr.response)
})
/*告诉服务器，传递的内容类型是json字符串*/
xhr.setRequestHeader('Content-Type', 'application/json')
/*准备数据并转成json字符串*/
const user = {username:'aaa', password:'123'}
const userStr = JSON.stringify(user)
/*发起请求*/
xhr.send(userStr)
```

(10) Promise：用于表示一个异步操作的最终完成（或失败）及其结果值

```
//创建Promise对象
const p = new Promise((resolve,rejet) => {
    //执行异步任务并传递结果
    //成功调用：resolve(值)，触发then()执行
    //失败调用：reject(值)，触发catch()执行
})
//接收结果
p.then(result => {
    //成功
```

```
}).catch(error => {11        //失败
})
```

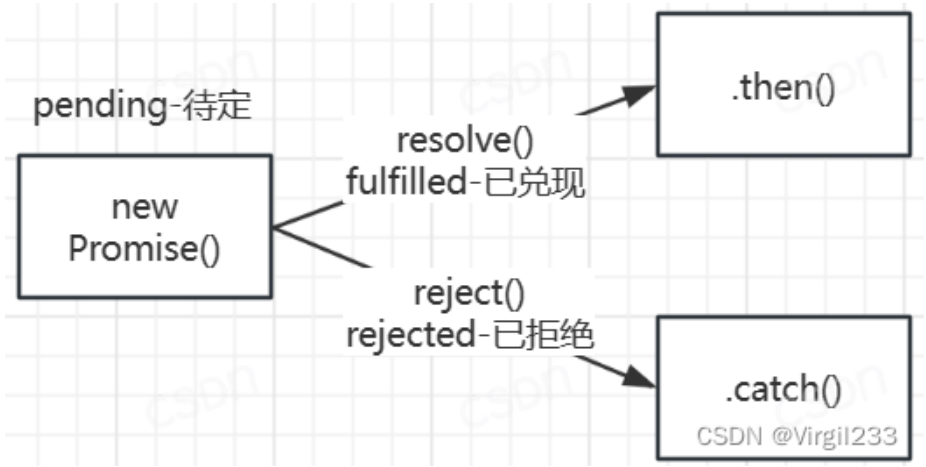
（11）Promise三种状态：

待定（pending）：初始状态，既没有被兑现，也没有被拒绝

已兑现（fulfilled）：操作成功完成

已拒绝（rejected）：操作失败

注：Promise对象一旦被兑现或拒绝后状态无法再改变



（12）基于Promise+XHR封装myAxios函数

```
//定于myAxios函数，接收配置对象，返回Promise对象
function myAxios(config){
    return new Promise((resolve, reject) => {
        //发起XHR请求，默认请求方法为get
        const xhr = new XMLHttpRequest()

        //判断有params选项，携带查询参数
        if(config.params){
            //使用URLSearchParams转换并添加到url上
            const obj = new URLSearchParams(config.params)
            const queryStr = obj.toString()
            //把查询参数字符串拼接到url后面
            config.url += `?${queryStr}`
```

```
        }
        xhr.open(config.method || 'GET', config.url)

        xhr.addEventListener('loadend', () => {
            //调用成功或失败的处理程序
            if(xhe.status >= 200 && xhr.status < 300){
                resolve(JSON.parse(xhr.response))
            }else{
                reject(new Error(xhr.response))
            }
        })

        //判断有data选项，携带请求体
        if(config.data){
            //转换数据类型，在send中发送
            const jsonStr = JSON.stringify(config.data)
            xhr.sendRequestHeader('Content-Type', 'application/json')
            xhr.send(jsonStr)
        }else{
            //没有请求体数据，正常发起请求
            xhr.send()
        }
    })
}
```

（13）同步代码：按照书写代码的顺序一行一行地执行程序，上一行完成后才会执行下一行

异步代码：执行一个可能长期运行任务的同时继续对其他事件作出反应而不用等待任务完成，同时，程序会在任务完成后显示结果。如：setTimeout、setInterval、事件、AJAX

注：Promise本身是同步的，但then和catch回调函数是异步的

（14）回调函数地狱：回调函数中嵌套回调函数，一直嵌套下去，可读性差、异常捕获困难、耦合性高

（15）Promise链式调用：依靠then()方法会返回一个新生成的Promise对象的特性，继续串联下一环任务直到结束，通过链式调用解决回调函数地狱问题

```
/*需求：获取默认第一个省、第一个市、第一个地区并展示在下拉菜单中*/
let pname = ''
//获取省份Promise对象
axios({url:''}).then(result => {
```

```
    pname = result.data.list[0]
    document.querySelector('.province').innerHTML = pname
    //获取城市Promise对象
    return axios({url:'', params:{pname}})
}).then(result => {
    const cname = result.data.list[0]
    document.querySelector('.city').innerHTML = cname
    //获取地区Promise对象
    return axios({url:'', params:{pname, cname}})
}).then(result => {
    const areaName = result.data.list[0]
    document.querySelector('.area').innerHTML = areaName
})
```

（16）async函数和await：可以用一种更简洁的方式写出基于Promise的异步行为，而无需刻意地链式调用Promise。用try-catch捕获错误

```
async function getData(){
    try{
        cosnt pObj = await axios({url:''})
        const pname = pObj.data.list[0]
        cosnt cObj = await axios({url:'', params:{pname}})
        const cname = cObj.data.list[0]
        cosnt aObj = await axios({url:'', params:{pname, cname}})
        const areaName = aObj.data.list[0]

        document.auerySelector('.province').innerHTML = pname
        document.auerySelector('.city').innerHTML = cname
        document.auerySelector('.area').innerHTML = areaName
    }catch(error){
        console.dir(error)
    }
}

getData()
```

（17）事件循环：负责执行代码、收集和处理事件以及执行队列中的子任务。因为JS是单线程，为了让耗时代码不阻塞其他代码运行，所以设计了事件循环模型。如setTimeout即使设置等待时间为0也会在同步代码后执行

JS代码执行顺序：先执行同步代码，遇到异步代码交给宿主浏览器环境执行；异步有了结果后把回调函数放入任务队列排队；当调用栈空闲后，反复调用任务队列里的回调函数

（18）宏任务：由浏览器环境执行的异步代码，如：JS脚本执行事件（script）、setTimeout/setInterval、AJAX请求完成事件、用户交互事件等

微任务：由JS引擎环境执行的异步代码，如Promise.then()

注：宏任务队列和微任务队列中同时有任务时，优先处理微任务

（19）Promise.all静态方法：合并多个Promise对象，等待所有同时成功完成或某一个失败，做后续逻辑

```
const p = Proise.all([Promise对象, Promise对象, ...])
p.then(result => {
    //result结果：[Promise对象成功结果, Promise对象成功结果, ...]
}).catch(error => {
    //第一个失败的Promise对象抛出的异常
})
```

（20）token：访问权限的令牌，本质上是一串字符串，用来判断是否有登录状态等，控制访问权限（前端只能判断是否有token，后端才能判断token的有效性）

```
const token = localStorage.getItem('token')
//没有token则强制跳转登录页
if(!token){
    location.href = '../login/index.html'
}
```

（23）axios请求拦截器：发起请求前触发的配置函数，对请求参数进行额外配置

```
axios.interceptors.request.use(function(config){
    const token = location.getItem('token')
    token && (config.headers.Authorization = `Bearer ${token}`)
    //发送请求前的操作
    return config
}, function(error){
    //对请求错误的操作
    return Promise.reject(error)
```

```
})
```

（24）axios响应拦截器：响应回到then/catch之前触发的拦截函数，对响应结果统一处理

```
axios.interceptors.response.use(function(response){
    //2xx范围内的状态码触发
    //对响应数据的操作，如提取服务器返回的数据对象，返回到发起请求的位置
    const result = response.data
    return result
}, function(error){
    //超出2xx范围的状态码触发
    //对响应错误的操作，如响应状态为401代表身份验证失败
    if(error?.response?.status === 401){
        alert('登录状态过期，请重新登录')
        localStorage.clear()
        location.href = '../login/index.html'
    }
    return Promise.reject(error)
})
```

二、Node.js

（1）Node.js： 跨平台的JS运行环境，用来搭建服务器端的JS应用程序。基于Chrome的V8引擎封装，区别于JS的是没有BOM和DOM。可以独立执行JS代码，利于前端工程化

（2）fs模块——读写文件

```
//加载fs模块对象
const fs = require('fs')
//写入
fs.writeFile('文件路径', '写入内容', err => {
    //写入后回调函数
})
//读取
fs.readFile('文件路径', (err, data) => {
    //读取后回调函数
    //data是文件内容的Buffer数据流
```

```
})
```

（3）path模块——路径处理

问题：Node.js代码中，相对路径是根据终端所在路径查找的，而不是代码文件所在路径，所以可能无法找到想要的文件

解决：使用绝对路径，内置变量__dirname获取当前模块绝对路径，path.join()会使用特定于平台的分隔符作为定界符，将给定路径片段连接在一起

```
//加载path模块
const path = require('path')
//拼接路径
path.join(__dirname, '路径1', '路径2', ...)
```

（4）http模块——创建web服务

```
//加载http模块，创建web服务对象
const http = require('http')
const server = http.createServer()

//监听request请求事件，设置响应头和响应体
server.on('reuest', (req, res) => {
    //设置响应头：内容类型，普通文本；编码格式为utf-8
    res.setHeader('Content-Type', 'text/plain;charset=utf-8)
    res.end('欢迎使用')
})

//配置端口号并启动web服务
server.listen(3000, () => {
    console.log('web服务已启动')
})
```


（5）Node.js模块化：每个文件都被视为一个单独的模块。CommonJS模块是为Node.js打包JS代码的原始方式，但Node.js还支持浏览器和其他JS运行时使用的ECMAScript模块标准

（6）CommonJS标准导入导出模块：

注：导入时，如果是内置模块，直接写名字，如fs、path、http等；如果是自定义模块，写模块文件路径

```
//导出
module.exports = {
    对外属性名1：baseURL,
    对外属性名2：自定义模块中的函数名
}

//导入
const obj = require('模块名或路径')
//obj等同于module.exports导出的对象
```

(7) ECMAScript标准默认导入导出模块：

注：Node.js默认支持CommonJS标准，要使用ECMAScript标准，需要在运行模块所在文件夹新建package.json文件，并设置{"type": "module"}

```
//导出
export default = {
    对外属性名1：baseURL,
    对外属性名2：自定义模块中的函数名
}

//导入
import obj from '模块名或路径'
//obj等同于module.exports导出的对象
```

(8) ECMAScript标准命名导入导出模块：

注：按需加载（只用个别属性）使用命名；全部加载使用默认

```
//导出
export const baseURL = ......
export const getArraySum = ......

//导入，变量名需与导出时同名
import {baseURL, getArraySum} from '模块名或路径'
```


(9) 包：分为项目包（项目代码）和软件包（封装工具和方法）

注：根目录中必须有package.json文件，用来记录包的清单信息。导入软件包时，默认引入index.js模块文件或package.json中main属性指定的模块文件

(10) npm：软件包管理器，用于下载和管理Node.js包

使用：初始化清单文件npm init -y(得到package.json文件，有则跳过此命令)→下载软件包npm i 软件包名称→使用软件包

注：npm i 不加软件包名则根据package.json文件安装所有依赖

（11）本地软件包：当前项目使用，封装属性和方法，存在于node_modules

全局软件包：本机所有项目使用，封装命令和工具，存在于系统设置的位置

全局软件包nodemon：替代node命令，检测代码更改，自动重启程序

使用：安装npm i nodemon -g(-g代表安装到全局环境中)→运行nodemon 目标js文件

文章知识点与官方知识档案匹配，可进一步学习相关知识

Vue入门技能树 > Node.js和npm > Node安装与配置　38374 人正在系统学习中
