AJAX入门与axios使用

什么是AJAX

定义:AJAX是异步的JavaScript和XML.简单来说,就是使用XMLHttpRequest对象与服务器通信. 它可以使用JSON,HTML和text文本等格式发送和接受数据.AJAX具有异步特性,可以在不刷新页面的情况下与服务器通信,交换数据或更新页面.

概念:AJAX是浏览器与服务器进行数据通信的技术.

axios使用

语法:

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

2.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

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

认识URL

定义:***统一的资源定位符***(Uniform Resource Locator),俗称***网页地址***.

eg.

- https://www.baidu.com/index.html --->网页资源
- https://www.itheima.com/images/logo.png --->图片资源
- https://hmajax.itheima.net/api/province --->数据资源

协议

http协议:超文本传输协议,规定浏览器和服务器之间传输数据的**格式**.

https😳/hmajax.itheima.net/api/province

域名

域名:标记服务器在互联网中的**方位**.

https://**hmajax.itheima.net**/api/province

资源路径

资源路径:标记资源在服务器下的**具体位置**.

https://hmajax.itheima.net/**api/province**

查询参数

URL查询参数

定义:浏览器提供给服务器的**额外信息**,让服务器返回浏览器想要的数据

语法:http://xxx.com/xxx/xxx**?参数名1=值1&参数名2=值2**

eg.

http://hmajax.itheima.net/api/city**?pname=河北省**

axios - 查询参数

语法:使用axios的**params**选项

注意:axios在运行时把参数名和值,会拼接到URL**?参数名=值**

城市列表:http://hmajax.itheima.net/api/city?pname=河北省

```
axios({
    url:'http://hmajax.itheima.net/api/city',
    params:{
        pname:'河北省'
    }
}).then(result=>{
    //对服务器返回的数据做后续处理
})
```

常用请求方法和数据提交

常用请求方法

请求方法:对服务器**资源**,要执行**操作**

请求方法	操作
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)

数据提交

场景:当数据需要在服务器上**保存**

axios请求配置

url:请求的URL网址.

methods:请求的方法,**GET**可以省略(不区分大小写)

data:提交数据

```
axios({
    url:'目标资源地址',
    methods:'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
})
```

axios错误处理

语法:在**then**方法的后面,通过点语法调用**catch**方法,传入**回调函数**并定义**形参**

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

```
7 | })
```

处理:注册案例,**重复注册**时通过**弹框**提示用户错误原因

HTTP协议---请求报文

HTTP协议:规定了浏览器发送及服务器返回内容的**格式**

请求报文:浏览器按照HTTP协议要求的**格式**,发送给服务器的**内容**

请求报文的格式

1. **请求行:请求方法,URL**,协议
2. **请求头**:以键值对的格式携带的附加信息,比如:**Content-Type**
3. 空行:分割请求头,空行之后的是发送给服务器的资源
4. **请求体:发送的资源**

请求报文-错误排查

HTTP协议-相应报文

HTTP协议:规定了浏览器发送及服务器返回内容的**格式**

响应报文:服务器按照HTTP协议要求的**格式**,发送给浏览器的**内容**

1. **响应行(状态行)**:协议,**HTTP响应状态码**,状态信息
2. **响应头**:以键值对的格式携带的附加信息,比如:**Content-Type**
3. 空行:分隔响应头,空行之后的是服务器返回的资源
4. **响应体:返回的资源**

HTTP响应状态码

HTTP响应状态码:用来表面请求**是否成功**完成

比如:**404(服务器找不到资源)**

状态码	说明
1xx	信息
2xx	成功
3xx	重定向消息
4xx	客户端错误
5xx	服务端错误

接口文档

接口文档:由**后端**提供的**描述接口**的文章

XMLHttpRequest

XMLHttpRequest_基本使用

AJAX原理 - XMLHttpRequest

定义:***XMLHttpRequest(XHR)对象用于与服务器交互***.通过XMLHttpRequest可以在不刷新页面的情况下请求特点URL,获取数据.这允许网页在不影响用户操作的情况下,更新页面的局部内容.XMLHttpRequest在**AJAX**编程中被大量使用.

关系:axios内部采用XMLHttpRequest与服务器交互

好处:掌握使用XHR与服务器进行数据交互,了解axios内部原理

使用XMLHttpRequest

步骤:

1. 创建XMLHttpRequest对象
2. 配置**请求方法**和请求**URL**地址
3. 监听loadend事件,接收**响应结果**
4. 发起请求

```
const xhr = new XMLHttpRequest()
xhr.open('请求方法','请求url网址')
xhr.addEventListener('loadend',()=>{
    //响应结果
    console.log(xhr.response)
})
xhr.send()
```

XMLHttpRequest_查询参数

定义:浏览器给服务器的额外信息,让服务器返回浏览器想要的数据

语法:http://xxx.com/xxx/xxx?参数名1=值1&参数名2=值2

XMLHttpRequest_数据提交

需求:通过XHR提交用户名和密码,完成注册功能

核心:

请求头设置Content-Type:application/json

请求体携带JSON字符串

```
const xhr = new XMLHttpRequest()
xhr.open('请求方法','请求url网址')
xhr.addEventListener('loadend',()=>{
    //响应结果
    console.log(xhr.response)
})
//告诉服务器,我传递的内容类型,是JSON字符串
xhr.setRequestHeader('Content-Type','application/json')
//准备数据并且转换成JSON字符串
const user = { username:'itheima007',password:'7654321'}
const userStr = JSON.stringify(user)
xhr.send(userStr)
```

Promise

认识_Promise

定义:Promise对象用于表示一个异步操作最终完成(或失败)及其结果值

好处:

1. 逻辑更清晰
2. 了解axios函数内部运作机制
3. 能解决回调函数地狱问题

语法:

```
//1.创建Promise对象
const p = new Promise((resolve,reject)=>{
    //2.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
})
//3.接收结果

```

认识_Promise的三种状态

作用:了解Promise对象如何**关联**的**处理函数**,以及代码的执行顺序

概念:一个Promise对象,必然处于以下几种状态之一

- 待定(pending) :初始状态,既没有被兑现,也没有被拒绝
- 已兑现(fulfilled) :意味着,操作成功完成
- 已拒绝(rejected) :意味着,操作失败

注意:Promise对象一旦被**兑现/拒绝**,就是**已敲定**了,状态无法再被改变

同步代码和异步代码

同步代码:逐行执行,原地**等待结果**后,才继续向下执行

异步代码:调用后**耗时**,不阻塞代码执行,将来完成后触发**回调函数**

JS中有哪些异步代码:

- setTimeout / setInterval
- 事件
- AJAX

异步代码如何接收结果:

依靠回调函数来接收

回调函数地狱

需求:展示默认第一个省,第一个城市,第一个地区在下拉菜单中

概念:在回调函数中**嵌套回调函数**,一直嵌套下去形成了回调函数地狱

缺点:可读性差,异常无法捕获,耦合性很差,牵一发动全身

eg.

```
axios({url:'http://hmajax.itheima.net/api/province'}).then(result=>({
    const pname = result.data.list[0]
    document.querySelector('.province').innerHTML=pname
    axios({url:'http://hmajax.itheima.net/api/city'},params:{pname}).th
en(result=>{
        const cname = result.data.list[0]
        document.querySelector('.city').innerHTML = cname
        axios({url:'http://hmajax.itheima.net/api/area'},params:{pname,
cname}).then(result=>{
                document.querySelector('.area').innerHTML = result.dat
a.list[0]
    })
  })
})
```

Promise_链式调用

概念:依靠then()方法返回**会新生成一个Promise对象**特效,继续串联下一环任务,直到结束

细节:then()回调函数中的**返回值**,会影响新生成的Promise对象**最终状态和结果**

好处:通过链式调用,解决回调函数嵌套问题

```
//1.创建Promise对象-模拟请求省份名字
const p = new Promise((resolve,reject)=>{
    setTimeout(()=>{
        resolve('北京市')
    },2000)
})
//获取省份名字
const p2 = p.then(result => {
    console.log(result)
    //3.创建Promise对象-模拟请求城市名字
    //return Promise对象最终状态和结果,影响到新的Promise对象
    return new Promise((resolve,reject)=>{
        setTimeout(()=>{
            resolve(result+'---北京')
        },2000)
    })
})
p2.then(result => {
    console.log(result)
})
//then()原地的结果是一个新的Promise对象
console.log(p2 === p)
```

async函数和await

定义:async函数施使用async关键字声明的函数,async函数是AsyncFunction构造函数的实例,并且其中允许使用await关键字.async和await关键字让我们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

eg.

```
//获取省市区
async function getDefultArea(){
    const pobj = await axios({url:'http://hmajax.itheima.net/api/provin
ce'})
    const pname = pobj.data.list[0]
    const cobj = await axios({url:'http://hmajax.itheima.net/api/cit
```

```
y'},params:{pname})
    const cname = cobj.data.list[0]
    const aobj = await axios({url:'http://hmajax.itheima.net/api/are
a'},params:{pname,cname})
    const aname = aobj.data.list[0]
    //赋予到页面
    document.querySelector('.province').innerHTML = pname
    document.querySelector('.city').innerHTML = cname
    document.querySelector('.area').innerHTML = aname
}
getDefultArea()
```

async函数和await-捕获错误

使用: try...catch,**try...catch**语句标记要尝试的预计块,并指定一个出现异常时抛出的响应

语法:

```
try{
    //要执行的代码
} catch(error){
    //error接收的是错误信息
    //try里代码,如果有错误,直接进入这里执行
}
```

eg.

```
async function getDefultArea(){
    try{
    const pobj = await axios({url:'http://hmajax.itheima.net/api/provin
ce'})
    const pname = pobj.data.list[0]
    const cobj = await axios({url:'http://hmajax.itheima.net/api/cit
y'},params:{pname})
    const cname = cobj.data.list[0]
    const aobj = await axios({url:'http://hmajax.itheima.net/api/are
a'},params:{pname,cname})
    const aname = aobj.data.list[0]
    //赋予到页面
    document.querySelector('.province').innerHTML = pname
    document.querySelector('.city').innerHTML = cname
    document.querySelector('.area').innerHTML = aname
    } catch(error){
        //2.接着调用catch块,接受错误信息
```

```
            //如果try里某行代码报错后,try中剩余的代码不会执行了
        }
    }
    getDefultArea()
```

事件循环

概念:JavaScript有一个基于**事件循环**的并发模型,**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**.这个模型与其他语言中的模型截然不同,比如C和Java.

原因:JavaScript单线程(某一刻只能执行一行代码),为了让耗时代码不阻塞其他代码执行,设计***事件循环模型***

调用栈

宿主环境

任务队列

什么是事件循环:

执行代码和收集异步任务,在调用栈空闲时,反复调用任务队列里回调函数执行机制

JavaScript内代码如何执行:

- 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**
- 当调用栈**空闲**后,反复调用任务队列里的回调函数

宏任务与微任务

ES6之后引入了Promise对象,让JS引擎也可以发起异步任务

异步任务分为:

- 宏任务:由浏览器环境执行的异步代码
- 微任务:由JS引擎环境执行的异步代码

任务(代码)	执行所在环境
JS脚本执行任务(script)	浏览器
setTimeout/setInterval	浏览器
AJAX请求完成事件	浏览器
用户交互事件等	浏览器

任务(代码)	执行所在环境
Promise对象.then()	JS引擎

JavaScript内代码如何执行:

- 执行第一个script脚本事件宏任务,里面**同步**代码
- 遇到**宏任务/微任务**交给宿主环境,有结果回调函数进入对应队列
- 当执行栈空闲时,**清空微任务**队列,再执行**下一个宏任务**,cong1再来

Promise.all静态方法

概念: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

语法:

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