1、原生ajax

1.1、Ajax简介

Ajax全程Asynchronous JavaScript And XML，即异步的js和XML

通过Ajax可以在浏览器中向服务器发送异步请求，最大优势：不需要刷星就可以获得数据

Ajax不是编程语言，而是一种将现有的标准组合在一起使用的新方式。

1.2、XMl简介

XML　可扩展标记语言。

　　被设计用来传输和储存数据

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，全是自定义标签，用来表示一些数据

现在用json

1.3、Ajax优缺点

- 优点：
  - 可以不用刷新页面就能与服务器端通信
  - 允许根据与用户事件来更新部分页面内容。
- 缺点：
  - 无浏览记录，不能回退
  - 存在跨域问题（**默认不可跨域**）
  - SEO不友好（搜索引擎搜索不到）

1.4、Ajax的使用

1.4.1、Ajax请求的基本操作

- GET请求

```
<body>
  <button>点击发送请求</button>
  <div id="result"></div>

  <script>
    //获取button元素
    const but = document.getElementsByTagName('button')[0];

    //绑定事件
    but.onclick = function(){
      //1.创建对象
      const xhr = new XMLHttpRequest();
      const result = document.getElementById('result')
      //2.初始化 设置请求方法和url
      xhr.open('GET','http://localhost:8000/server');
      //3.发送
```

```
      xhr.send();
      //4.事件绑定 处理服务端返回的结果
      //on 当什么时
      //readystate 是xhr对象中的属性，表示状态0(初始值) 1(open方法调用完毕) 2(send方法调用完毕) 3(服务端返回部分的结果) 4(服务端返回的所有结果)
      xhr.onreadystatechange = function(){
        if(xhr.readyState === 4){
          //判断响应状态码 200 404 403 ...
          //2xx 表示成功
          if(xhr.status >= 200 && xhr.status < 300){
            //处理响应结果 行 头 空行 体
            console.log(xhr.status);//状态码
            console.log(xhr.statusText);//状态字符串
            console.log(xhr.getAllResponseHeaders);//所有响应头
            console.log(xhr.response);//响应体

            result.innerHTML = xhr.response
          }
        }
      }
    }
  </script>
</body>
</html>
```

```
//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
//response 是对响应报文的封装
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('这是用express服务器框架创建的node服务器')
});

//4.监听端口启动服务
app.listen(8000,()=>{
  console.log("服务器创建成功");
})
```

设置url参数（url后面接 " ? " 参数名和值，多个参数用 " & " 连接）

```
 xhr.open('GET','http://localhost:8000/server?a=123&b=321&c=231');
```

- POST请求

```
result.addEventListener("mouseover",function(){
    //1.创建对象
    const xhr = new XMLHttpRequest();
```

```
    //2.初始化 设置类型与 URl
    xhr.open('POST','http://localhost:8000/server');
    //3.发送
    xhr.send();
    //4.事件绑定
    xhr.onreadystatechange = function(){
        //判断
        if(xhr.readyState === 4){
            if(xhr.status >= 200 && xhr.status < 300){
                result.innerHTML = xhr.response;
            }
        }
    }
})
```

```
//设置post响应
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('这是用express服务器框架创建的node服务器,返回post响应')
});
```

设置参数:

```
xhr.send('a=123&b=321&c=231');//无格式要求，但是需要在服务器端中有对应的处理方式
```

设置响应头

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

1.4.2 服务端响应json数据

```
//.html
<script>
    //绑定dom元素
    const result = document.getElementById('result')
    //绑定键盘按下事件
    window.onkeydown = function(){
    //发送请求
    const xhr = new XMLHttpRequest();
    //设置响应体的数据类型
    xhr.responseType = 'json';
    //初始化
    xhr.open('GET','http://localhost:8000/json-server');
    //发送
    xhr.send();
    //事件绑定
    xhr.onreadystatechange = function(){
        if(xhr.readyState === 4){
            if(xhr.status >= 200 && xhr.status < 300){
                // console.log(xhr.response);
                //方式一:手动转换
```

```
                // let data = JSON.parse(xhr.response);
                // console.log(data);
                //方式二：提前设置响应体的数据类型
                result.innerHTML = xhr.response.name;
                console.log(xhr.response);
            }
        }
    }
}
</script>
```

```
//.js
//all响应所有请求
app.all('/json-server',(request,response)=>{
  //设置响应头 设置允许跨域
  response.setHeader('Access-Control-Allow-Origin','*');
  //设置响应体
  // response.send('这是用express服务器框架创建的node服务器,返回json数据');
  const data = {
    name:"周以"
  };
  //对json数据进行转换
  let str = JSON.stringify(data);
  response.send(str);
});
```

1.4.3 ie缓存问题

ie存在服务器端修改数据后 客户端不能同步响应问题

解决方法：设置每条请求有不同的参数

```
xhr.open('GET','http://localhost:8000/server?t='+Date.new())
```

1.4.4 超时与延时响应

```
<script>
    //获取button元素
    const but = document.getElementsByTagName('button')[0];
    //绑定事件
    but.addEventListener('click',function(){
        //1.创建对象
        const xhr = new XMLHttpRequest();
        //2.初始化 设置请求方法和url
        //设置超时 2s
        xhr.timeout = 2000;
        //超时回调
        xhr.ontimeout = function(){
            alert("网络超时！");
        }
        //网络异常回调
        xhr.onerror = function(){
            alert("网络异常")
        }
        xhr.open('GET','http://localhost:8000/delay');
```

```
        //3.发送
        xhr.send();
    })
</script>
```

```
//延时响应
app.get('/delay',(request,response)=>{
  //设置响应头 设置允许跨域
  response.setHeader('Access-Control-Allow-Origin','*');
  //设置服务器延时响应 3s后
  setTimeout(() => {
    response.send('这是用express服务器框架创建的node服务器')
  }, 3000);
  //设置响应体
});
```

1.4.5 取消请求

```
<body>
  <button>发送</button>
  <button>取消</button>
  <script>
    //获取dom元素对象
    const buts = document.querySelectorAll('button');
    let x = null;
    //发送请求
    buts[0].onclick = function(){
      x = new XMLHttpRequest();
      x.open('GET','http://localhost:8000/delay');
      x.send();
    }
    //取消请求，使用abort
    buts[1].onclick = function(){
      x.abort();
    }
  </script>
</body>
```

1.addEventListener可以对同一个元素绑定多个事件，执行顺序从上到下依次执行。而onclick同一个元素只能绑定一个事件，如有多个，后面的事件会覆盖前面的事件。

```
document.getElementById("myDIV").addEventListener("click", function() {
  alert(1)
});
document.getElementById("myDIV").addEventListener("click", function() {
  alert(2)
});
//以上代码会先弹出1，在弹出2
document.getElementById("myDIV").onclick = function () {
  alert(1)
};
document.getElementById("myDIV").onclick = function () {
  alert(2)
```

```
};
//以上代码只会弹出1。
```

2.addEventListener的第三个参数为布尔类型，默认为false，也就是执行的冒泡机制，如为true，则执行捕获机制。

3.addEventListener它对任何 DOM 元素都是有效的，而不仅仅只对 HTML 元素有效。点击查看dom分类

4.注册addEventListener事件时不需要写on，而onclick方式则必须加on。

```
document.getElementById("myDIV").addEventListener("click", myFunction);
document.getElementById("myDIV").onclick = myFunction;
```

5.在移除事件上，onclick使用的是指针指向null，例如`document.onclick = null`，而addEventListener则使用的是独有的移除方法removeListener（要使用此方法，addEventListener必须执行的是外部函数或存在函数名，不然则不能使用）

例如：

```
// 向 <div> 元素添加事件句柄
document.getElementById("myDIV").addEventListener("mousemove", myFunction);

// 移除 <div> 元素的事件句柄
document.getElementById("myDIV").removeEventListener("mousemove", myFunction);

//如是以下类型的代码，则不能使用removeEventListener
document.getElementById("myDIV").addEventListener("mousemove", function() {});
```

6.addEventListener为DOM2级事件绑定，onclick为DOM0级事件绑定。

7.IE678只能使用attachEvent，无addEventListener。

2、jQuery、Axios、fetch中的ajax

2.1、jQuery中的Ajax

```
$('button').eq(0).click(function(){
    $.get('http://localhost:8000/jquery-server',{a:120,b:321},function(data){
        console.log(data);
    })
});
$('button').eq(1).click(function(){
    $.post('http://localhost:8000/jquery-server',{a:120,b:321},function(data){
        console.log(data);
    },'json')
})
//通用方法
$('button').eq(0).click(function(){
    $.get('http://localhost:8000/jquery-server',{a:120,b:321},function(data){
        console.log(data);
    })
});
$('button').eq(1).click(function(){
    $.post('http://localhost:8000/jquery-server',{a:120,b:321},function(data){
```

```
        console.log(data);
    },'json')
})
$('button').eq(2).click(function(){
    $.ajax({
        //url
        url:'http://localhost:8000/delay',
        //参数
        data:{a:100,b:200},
        //请求类型
        type:'GET',
        //响应体结果
        dataType:'json',
        //成功的回调
        success: function(data){
            console.log(data);
        },
        //超时的回调
        timeout: 2000,
        //失败的回调
        error: function(){
            console.log("出错惹");
        },
        header:{
            c:123,
            d:321,
        }
    })
})
```

```
app.all('/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('这是用express服务器框架创建的node服务器')
  const data = {
    name:"周以一号"
  };
  //对json数据进行转换
  let str = JSON.stringify(data);
  response.send(str);
});
```

2.2、Axios中的ajax

```
const but = document.querySelectorAll('button');
//配置baseURL
axios.defaults.baseURL = 'http://localhost:8000';
//get
but[0].onclick = function(){
    axios.get('/axios-server',{
        params:{
            id:12,
            asd:21
        }
    }).then(value =>{
```

```
        console.log(value);
    })
};
//post
but[1].onclick = function(){
    axios.post('/axios-server',{
        userid: 'admin',
        password: 'admin'
    },{
        params: {
            id:12
        }
    })
};
//通用
but[2].onclick = function(){
    axios({
        method: 'post',
        url: '/axios-server',
        params: {
            id:123,
            we:234
        },
        headers: {
            q:123,
            w:132
        },
        data: {
            userid: 'admin',
            pw: 'admin'
        }
    }).then(response => {
        console.log(response);
    })
}
```

```
app.all('/axios-server',(request,response)=>{
  //设置响应头 设置允许跨域
  response.setHeader('Access-Control-Allow-Origin','*');
  // 设置允许自定义请求头
  response.setHeader('Access-Control-Allow-Headers', '*');
  //设置响应体
  // response.send('这是用express服务器框架创建的node服务器')
  const data = {
    name:"周以一号"
  };
  //对json数据进行转换
  response.send(JSON.stringify(data));
});
```

2.3、fetch中的ajax

```
but.onclick = function(){
    fetch('http://localhost:8000/axios-server',{
        method: 'POST',
        headers: {
            name:'zhouyi'
        },
        body:'id=zhouyi&pw=woaini'
    }).then(response =>{
        return response.json();//返回数据
    }).then(response => {
        console.log(response);//数据处理
    })
}
```

3、跨域问题

3.1同源策略

同源策略（Same-Orgin Policy）最早由Netscape公司提出，是一种浏览器的安全策略

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默认同源策略，无法实现跨域请求

同源示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>同源</title>
</head>
<body>
  <h1>zhouyi</h1>
  <button>anniu</button>
  <script>
    const but = document.querySelector('button');
    but.onclick = function(){
      const x = new XMLHttpRequest();
      x.open('GET','/data');
      x.send();
      x.onreadystatechange = function(){
        if(x.readyState === 4){
          if(x.status >= 200 && x.status < 300){
            console.log(x.response);
          }
        }
      }
    }
  </script>
</body>
</html>
```

```
const express = require('express');

const app = express();

app.get('/home',(request,response)=>{
  response.sendFile(__dirname + '/index.html');
});

app.get('/data',(request,response)=>{
  response.send('shuju')
});

app.listen(9000,()=>{
  console.log("服务启动");
})
```

3.2、如何解决跨域问题

3.2.1 JSONP

1、JSONP是什么

JSONP（JSON with Padding）是一个非官方的跨域解决方案，只支持 get 请求

2、JSONP怎么工作的

在网页中有一些标签本身就具有跨域能力，比如：img、link、iframe、script

JSONP就是利用script标签的苦于能力来发送请求的

示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JSONp案例</title>
</head>
<body>
  用户名: <input type="text">
  <p></p>
  <script>
    const inp = document.querySelector('input');
    const p = document.querySelector('p');

    function handle(data){
      inp.style.border = "solid 5px #f00";
      p.innerHTML = data.msg
    }

    inp.onblur = function(){
      let username = this.value;
      const script = document.createElement('script');
      script.src = 'http://localhost:8000/JSONP-server';
      document.body.appendChild(script)
    }
```

```
  </script>
</body>
</html>
```

```
//JSONP实践
app.all('/JSONp-server',(request,response)=>{
  //设置响应头 设置允许跨域
  // response.setHeader('Access-Control-Allow-Origin','*');
  response.setHeader('Access-Control-Allow-Headers', '*');

  const data = {
    name:"周以一号",
    msg: "用户名已经存在"
  };
  //对json数据进行转换
  let str = JSON.stringify(data)
  response.end(`handle(${str})`);
});
```

3.2.3、CORS

CORS（Cross-Origin Resource Sharing），跨域资源共享。CORS是官方的跨域解决方法，它的特点是不需要再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，允许服务器声明那些源站通过浏览器有权限访问哪些资源。

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