day23 JSONP及练习讲解

浏览器的同源策略

浏览器的一个安全策略、对对应的服务器请求接口地址有对应防护 （请求不通 异步请求）导致跨域。

同源策略包含的内容

- 协议 一致
- 端口号一致
- ip地址一致

跨域问题产生的原因

由于同源策略会产生跨域问题

- 协议不一致
- 端口号不一致
- ip地址不一致

文件资源访问也会产生跨域

- 对应的资源位置不一致 （ftp和http的影响）

跨域问题报错

```
Access to XMLHttpRequest at 'https://sp0.baidu.com/5 %E5%90%8C%E6%BA%90%E…%96%E7%95%A5.html:1
a1Fazu8AA54nxGko9WTAnF6hhy/su?wd=miqi&cb=fn' from origin 'http://localhost:8080' has been
blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested
resource.
```

跨域问题解决

服务器端处理

在响应的时候添加对应的响应头 （Access-Control-Allow-Origin）

```
response.setResponseHeader('Access-Control-Allow-Origin','*')
response.setResponseHeader('Access-Control-Allow-Origin-Method','*')
```

使用客户端和服务端的配合 JSONP

- 服务端提供的接口为jsonp接口
- 客户端通过对应的script标签不受跨域影响的特性来接收对应的数据 （通过回调函数接收）

使用代理

- devserver代理 (vue提供的 测试)
- 服务器代理 nginx

跨域问题是因为对应的tcp/udp来导致 所以如果要解决你可以不使用tcp/udp 使用 webScroket（即时通信）

JSONP讲解

概述

JSONP是一种解决跨域的一种的方案，它实际上也是一个get请求，它是利用对应的script标签不受跨域影响的限制来解决对应的跨域问题。（带对应的href src等的标签都不受跨域的影响 script link farmes等）

JSONP解决跨域必须具备的点

- 后端的接口必须为jsonp接口
- 前台使用script scr链入的方式 通过传入对应的回调函数来接收对应的结果

简单jsonp

准备JSONP接口

```
https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/su?wd=miqi&cb=fn
```

准备script标签通过src链入对应的接口地址

```
<script>
    //全局变量属于window
    function callback(result) {
        console.log(result)
    }
</script>
<!-- 准备script标签链入对应的jsonp地址
wd 表示对应的关键词
cb 表示回调函数 （服务器执行的 结果完它会将结果传入回调函数）
-->
<script src="https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/su?wd=吃饭
&cb=callback"></script>
```

jsonp封装

回调函数版本

```
//传递 url
//传递 参数
//传递回调函数对应的参数名
//传递 回调函数
export function jsonp(url, params, paramName, callback) {
  //生成对应的回调函数的名字
  let callbackName = `fn${parseInt(Math.random()*100)+Date.now()}`
  // let callbackName = Symbol() //Symbol不能转为字符串
  //将函数名和对应的回调函数联系到一起 加给window
  window[callbackName] = callback
  //进行url的拼接
  //拆分对应的参数对象拼接
  for (var key in params) {
    //url是否包含? 如果不包含使用?拼接 包含使用&连接
    let s = url.includes('?') ? '&' : '?'
    url += `${s+key}=${params[key]}`
  }
  //拼接对应的回调函数名及回调函数
  let s = url.includes('?') ? '&' : '?'
  url += `${s+paramName}=${callbackName}`
  //创建对应的script标签
  let script = document.createElement('script')
```

```
  //等待script标签加载完成再指定对应的src
  //指定对应的scr地址
  script.src = url
  //添加到对应的body中
  document.body.append(script)
  script.onload = function () {
    //再删除
    document.body.removeChild(script)
    //删除变量
    delete window[callbackName]
  }
}
```

promise版本

```
//传递 url
//传递 参数
//传递回调函数对应的参数名
//传递 回调函数
export function jsonp(url, params, paramName) {
  return new Promise((resolve, reject) => {
    //生成对应的回调函数的名字
    let callbackName = `fn${parseInt(Math.random()*100)+Date.now()}`
    // let callbackName = Symbol() //Symbol不能转为字符串
    //数据传递给then  调用resolve
    window[callbackName] = resolve //相当于 resolve(result)
    //进行url的拼接
    //拆分对应的参数对象拼接
    for (var key in params) {
      //url是否包含? 如果不包含使用?拼接 包含使用&连接
      let s = url.includes('?') ? '&' : '?'
      url += `${s+key}=${params[key]}`
    }
    //拼接对应的回调函数名及回调函数
    let s = url.includes('?') ? '&' : '?'
    url += `${s+paramName}=${callbackName}`
    //创建对应的script标签
    let script = document.createElement('script')
    //等待script标签加载完成再指定对应的src
    //指定对应的scr地址
    script.src = url
    //添加到对应的body中
    document.body.append(script)
    script.onload = function () {
      //再删除
      document.body.removeChild(script)
      //删除变量
      delete window[callbackName]
    }
  })
}
```