跨域

什么是跨域

跨域是指一个域下的文档或脚本试图去请求另一个域下的资源，这里跨域是广义的。

广义的跨域:

```
1.) 资源跳转：  A链接、重定向、表单提交

2.) 资源嵌入：  <link>、<script>、<img>、<frame>等dom标签，还有样式中background:url()、@font-face()等文件外链
3.) 脚本请求：  js发起的ajax请求、dom和js对象的跨域操作等
```

其实我们通常所说的跨域是狭义的，是由浏览器同源策略限制的一类请求场景。

什么是同源策略?

同源策略/SOP（Same origin policy）是一种约定，由Netscape公司1995年引入浏览器，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，浏览器很容易受到XSS、CSFR等攻击。所谓同源是指"协议+域名+端口"三者相同，即便两个不同的域名指向同一个ip地址，也非同源。

同源策略限制以下几种行为：

```
1.) Cookie、LocalStorage 和 IndexDB 无法读取
2.) DOM 和 Js对象无法获得
3.) AJAX 请求不能发送
```

常见跨域场景

```
URL                                  说明                    是否允许通信
http://www.domain.com/a.js
http://www.domain.com/b.js           同一域名，不同文件或路径  允许
http://www.domain.com/lab/c.js

http://www.domain.com:8000/a.js
```

```
http://www.domain.com/b.js          同一域名，不同端口
        不允许

http://www.domain.com/a.js
https://www.domain.com/b.js         同一域名，不同协议
        不允许

http://www.domain.com/a.js
http://192.168.4.12/b.js            域名和域名对应相同ip
        不允许

http://www.domain.com/a.js
http://x.domain.com/b.js            主域相同，子域不同
        不允许
http://domain.com/c.js

http://www.domain1.com/a.js
http://www.domain2.com/b.js         不同域名
        不允许
```

> 只要是端口号之前的有一个不一样属于跨域

演示跨域

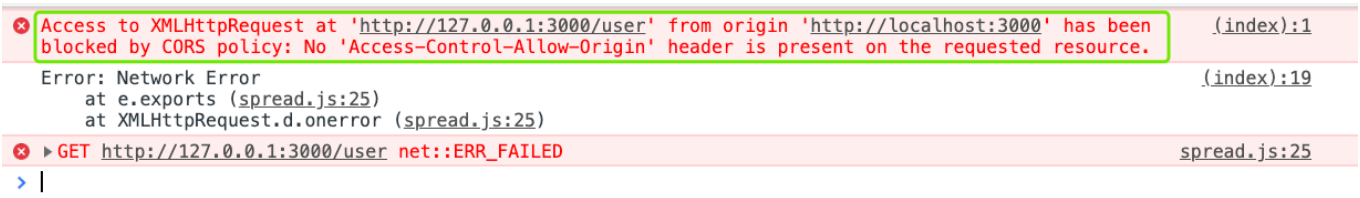
server.js

```
const http = require('http');
const fs = require('fs');
http.createServer((req, res) => {
    const { url, method } = req;
    if (url === '/' && method === 'GET') {
        // 读取首页
```

```
        fs.readFile('./index.html', (err, data) => {
            if (err) {
                res.statusCode = 500;//服务器内部错误
                res.end('500- Interval Serval Error!');
            }
            res.statusCode = 200;//设置状态码
            res.setHeader('Content-Type', 'text/html');
            res.end(data);
        })
    } else if (url === '/user' && method === 'GET') {
        res.statusCode = 200;//设置状态码
        res.setHeader('Content-Type', 'application/json');
        res.end(JSON.stringify([{ name: "小马哥" }]));
    }

}).listen(3000);
```

axios发起请求

```
axios.get('http://127.0.0.1:3000/user').then(res => {
    console.log(res.data);

}).catch(err => {
    console.log(err);

})
```



跨域常用解决方案

1. 通过jsonp跨域
2. **跨域资源共享（CORS 最常用）**
3. nginx代理跨域
4. nodejs中间件代理跨域

通过JSONP跨域

通常为了减轻web服务器的负载，我们把js、css，img等静态资源分离到另一台独立域名的服务器上，在html页面中再通过相应的标签从不同域名下加载静态资源，而被浏览器允许，基于此原理，我们可以通过动态创建script，再请求一个带参网址实现跨域通信

axios最新版本已经不支持jsonp方法了,不想因为一个jsonp请求就又去引一个依赖，所以决定自己封装一下

```
axios.jsonp = (url) => {
    if (!url) {
        console.error('Axios.JSONP 至少需要一个url参数!')
        return;
    }
    return new Promise((resolve, reject) => {
        window.jsonCallBack = (result) => {
            resolve(result)
        }
        var JSONP = document.createElement("script");
        JSONP.type = "text/javascript";
        JSONP.src = `${url}callback=jsonCallBack`;
        document.getElementsByTagName("head")
[0].appendChild(JSONP);
        setTimeout(() => {
            document.getElementsByTagName("head")
[0].removeChild(JSONP)
```

```
        }, 500)
    })
}

// 第一种 通过jsonp
axios.jsonp('http://127.0.0.1:3000/user?')
    .then(res=>{
    console.log(res);
}).catch(err=>{
    console.log(err);

})
```

server.js

```
const express = require('express');
const fs = require('fs');
const app = express();
// 中间件方法
// 设置node_modules为静态资源目录
// 将来在模板中如果使用了src属性
http://localhost:3000/node_modules
app.use(express.static('node_modules'))
app.get('/',(req,res)=>{
  fs.readFile('./index.html',(err,data)=>{
    if(err){
      res.statusCode = 500;
      res.end('500 Interval Serval Error! ');
    }
    res.statusCode = 200;
    res.setHeader('Content-Type','text/html');
    res.end(data);
  })
```

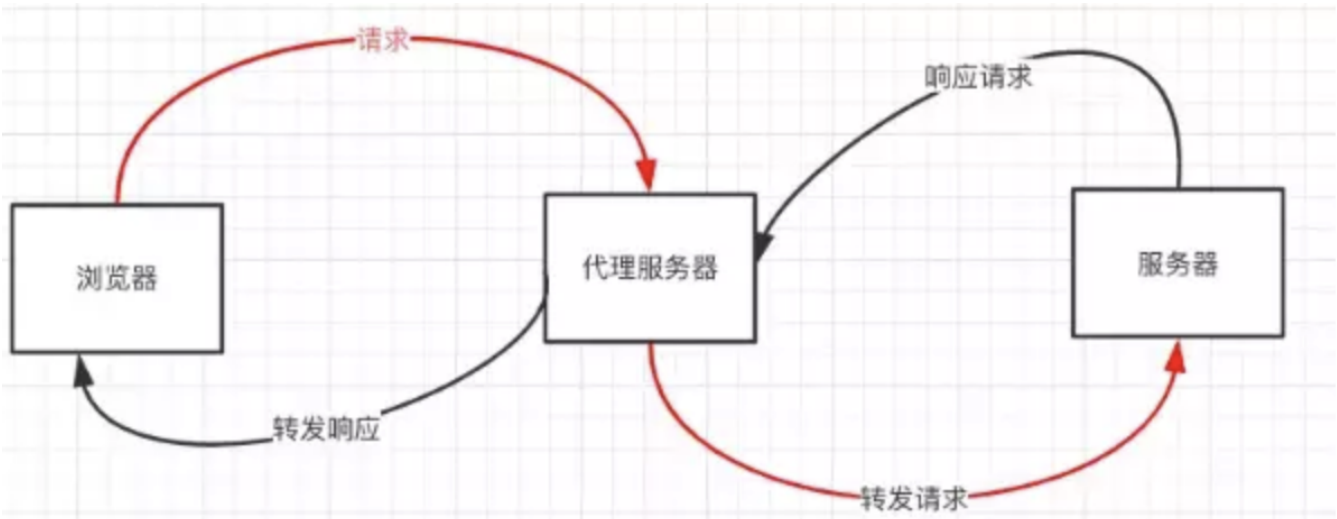
```
})
// app.set('jsonp callback name', 'cb')
app.get('/api/user',(req,res)=>{
  console.log(req);
  // http://127.0.0.1:3000/user?cb=jsonCallBack
  // const cb = req.query.cb;
  // cb({})
  // res.end(`${cb}(${JSON.stringify({name:"小马哥"})})`)
  res.jsonp({name:'小马哥'})
})
app.listen(3000);
```

jsonp缺点：只能实现get一种请求。

Nodejs中间件代理跨域

实现原理：**同源策略是浏览器需要遵循的标准，而如果是服务器向服务器请求就无需遵循同源策略。** 代理服务器，需要做以下几个步骤：

- 接受客户端请求 。
- 将请求 转发给服务器。
- 拿到服务器 响应 数据。
- 将 响应 转发给客户端。



index.html文件,通过代理服务器http://127.0.0.1:8080向目标服务器http://localhost:3000/user请求数据

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>

<body>
  <script src='/axios/dist/axios.js'></script>
  <h2>中间件代理跨域</h2>
  <script>
    axios.defaults.baseURL = 'http://localhost:8080';
    axios.get('/user')
      .then(res => {
        console.log(res);

      })
      .catch(err => {
        console.log(err);

      })
  </script>

</body>

</html>
```

proxyServer.js代理服务器

```
const express = require('express');
const { createProxyMiddleware } = require('http-proxy-middleware');
const app = express();

// 代理服务器操作
//设置允许跨域访问该服务.
app.all('*', function (req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Headers', 'Content-Type');
  res.header('Access-Control-Allow-Methods', '*');
  res.header('Content-Type', 'application/json;charset=utf-8');
  next();
});

// http-proxy-middleware
// 中间件 每个请求来之后 都会转发到 http://localhost:3001 后端服务器
app.use('/', createProxyMiddleware({ target: 'http://localhost:3001', changeOrigin: true }));

app.listen(8080);
```

业务服务器server.js

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

```
// 中间件方法
// 设置node_modules为静态资源目录
// 将来在模板中如果使用了src属性
http://localhost:3000/node_modules
app.use(express.static('node_modules'))
app.get('/',(req,res)=>{
  fs.readFile('./index.html',(err,data)=>{
    if(err){
      res.statusCode = 500;
      res.end('500 Interval Serval Error! ');
    }
    res.statusCode = 200;
    res.setHeader('Content-Type','text/html');
    res.end(data);
  })
})
// app.set('jsonp callback name', 'cb')

app.get('/user',(req,res)=>{
  res.json({name:'小马哥'})
})
app.listen(3001);
```

通过CORS跨域

```
//设置允许跨域访问该服务.
app.all('*', function (req, res, next) {
    /// 允许跨域访问的域名：若有端口需写全（协议+域名+端口），若没有端口末尾不用加'/'
    res.header('Access-Control-Allow-Origin', '*');
    res.header('Access-Control-Allow-Headers', 'Content-Type');
    res.header('Access-Control-Allow-Methods', '*');
    res.header('Content-Type', 'application/json;charset=utf-8');
    next();
});
```

具体实现：

- 响应简单请求:动词为get/post/head,如果没有自定义请求头,`Content-Type`是`application/x-www-form-urlencoded`,`multipar/form-data`或`text/plagin`之一,通过添加以下解决

```
res.header('Access-Control-Allow-Origin', '*');
```

- 响应prefight请求,需要响应浏览器发出的options请求(预检请求),并根据情况设置响应头

```
//设置允许跨域访问该服务.
app.all('*', function (req, res, next) {
    res.header('Access-Control-Allow-Origin', 'http://localhost:3002');
    //允许令牌通过
    res.header('Access-Control-Allow-Headers', 'Content-Type,X-Token');
    res.header('Access-Control-Allow-Methods', 'GET,POST,PUT');
    //允许携带cookie
    res.header('Access-Control-Allow-Credentials', 'true');
    res.header('Content-Type', 'application/json;charset=utf-8');
    next();
});
```

前端

在这里给大家补充了axios相关用法，具体的看视频

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>

<body>
```

```
  <script src='/axios/dist/axios.js'></script>
  <script src="https://cdn.bootcss.com/qs/6.9.1/qs.js"></script>
  <h2>CORS跨域</h2>
  <script>
    axios.defaults.baseURL = 'http://127.0.0.1:3002';
    axios.defaults.headers.common['Authorization'] = 'xaxadadadadad';
    axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';

    axios.interceptors.request.use(function (config) {
      let data = config.data;
      data = Qs.stringify(data);
      config.data = data;
      // 在发送请求之前做些什么
      return config;
    }, function (error) {
      // 对请求错误做些什么
      return Promise.reject(error);
    });
    axios.post('/login', {
        username: 'xiaomage',
        password: 123
      }, {
        // headers: {
        //   'Authorization': 'adjahdj1313131'
        // },
        // 表示跨域请求时需要使用凭证 允许携带cookies
        withCredentials: true
      })
      .then(res => {
        console.log(res);
```

```
        }).catch(err => {
          console.log(err);

        })
    </script>

</body>

</html>
```

使用第三方插件cors

```
npm i cors -S
```

server.js

```
const cors = require('cors');
//简单使用
app.use(cors())
```

配置选项参考链接

ex:

```
app.use(cors({
  origin:'http://localhost:3002', //设置原始地址
  credentials:true, //允许携带cookie
  methods:['GET','POST'], //跨域允许的请求方式
  allowedHeaders:'Content-Type,Authorization' //允许请求头的携带信息
}))
```

nginx反向代理

实现原理类似于Node中间件代理，需要你搭建一个中转nginx服务器，用于转发请求。

使用nginx反向代理实现跨域，是最简单的跨域方式。只需要修改nginx的配置即可解决跨域问题，支持所有浏览器，支持session，不需要修改任何代码，并且不会影响服务器性能。

实现思路：通过nginx配置一个代理服务器（域名与domain1相同，端口不同）做跳板机，反向代理访问domain2接口，并且可以顺便修改cookie中domain信息，方便当前域cookie写入，实现跨域登录。

```
// proxy服务器
server {
    listen       81;
    server_name  www.domain1.com;
    location / {
        proxy_pass   http://www.domain2.com:8080;  #反向代理
        proxy_cookie_domain www.domain2.com www.domain1.com; #修改cookie里域名
        index  index.html index.htm;
        # 当用webpack-dev-server等中间件代理接口访问nignx时，此时无浏览器参与，故没有同源限制，下面的跨域配置可不启用
        add_header Access-Control-Allow-Origin http://www.domain1.com;  #当前端只跨域不带cookie时，可为*
        add_header Access-Control-Allow-Credentials true;
    }
}
```

爬虫

什么是爬虫

> **网络爬虫**（又被称为网页蜘蛛，网络机器人，在 FOAF 社区中间，更经常的称为网页追逐者），是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本。另外一些不常使用的名字还有蚂蚁、自动索引、模拟程序或者蠕虫。

爬虫的分类

- 通用网络爬虫(全网爬虫)

  > 爬行对象从一些 种子URL 扩充到整个 Web，主要为门户站点搜索引擎和大型 Web 服务提供商采集数据。

- 聚焦网络爬虫(主题网络爬虫)

  > 是 指选择性 地爬行那些与预先定义好的主题相关页面的网络爬虫。

- 增量式网络爬虫

  > 指对已下载网页采取增量式更新和 只爬行新产生的或者已经发生变化网页 的爬虫，它能够在一定程度上保证所爬行的页面是尽可能新的页面。

- Deep Web爬虫

> 爬行对象是一些在用户填入关键字搜索或登录后才能访问到的 `深层网页信息` 的爬虫。

爬虫的爬行策略

- 通用网络爬虫（全网爬虫）

> 深度优先策略、广度优先策略

- 聚焦网络爬虫（主题网络爬虫）

> 基于内容评价的爬行策略（内容相关性），基于链接结构评价的爬行策略、基于增强学习的爬行策略（链接重要性），基于语境图的爬行策略（距离，图论中两节点间边的权重）

- 增量式网络爬虫

> 统一更新法、个体更新法、基于分类的更新法、自适应调频更新法

- `Deep Web` 爬虫

> `Deep Web` 爬虫爬行过程中最重要部分就是表单填写，包含两种类型：基于领域知识的表单填写、基于网页结构分析的表单填写

现代的网页爬虫的行为通常是四种策略组合的结果：

> 选择策略：决定所要下载的页面； 重新访问策略：决定什么时候检查页面的更新变化； 平衡礼貌策略：指出怎样避免站点超载； 并行策略：指出怎么协同达到分布式抓取的效果；

简单的网页爬虫流程

1. 确定爬取对象（网站/页面）
2. 分析页面内容（目标数据/DOM结构）
3. 确定开发语言、框架、工具等
4. 编码 测试，爬取数据
5. 优化

实战:百度新闻爬虫

确定爬取对象(网站/页面)

> 百度新闻 （news.baidu.com/）

分析页面内容（目标数据/DOM结构）

确定开发语言、框架、工具等

> node.js(express) + vscode

编码

- 安装依赖包

```
npm i express superagent cheerio
```

> `express` （使用express来搭建一个简单的Http服务器。当然，你也可以使用node中自带的 `http` 模块） `superagent` （superagent是node里一个非常方便的、轻量的、渐进式的第三方客户端请求代理模块，用他来请求目标页面） `cheerio` （cheerio相当于node版的jQuery，用过jQuery的同学会非常容易上手。它主要是用来获取抓取到的页面元素和其中的数据信息）

1. **使用express启动简易的本地Http服务器**

```
//index.js
const express = require('express');
const app = express();

app.get('/',(req,res)=>{
    res.send('hello world');
})

let server = app.listen(3000, function () {
  let host = server.address().address;
  let port = server.address().port;
  console.log('Your App is running at http://%s:%s', host, port);
});

```

> 访问：http://localhost:3000/ 看到 `hello world` 页面

2. **分析百度新闻首页的新闻信息**

> 百度新闻首页大体上分为“热点新闻”、“本地新闻”、“国内新闻”、“国际新闻”.....等。这次我们先来尝试抓取左侧 `“热点新闻”` 和下方的 `“本地新闻”` 两处的新闻数据。
>
> `F12` 打开 `Chrome` 的控制台，审查页面元素，经过查看左侧“热点新闻”信息所在 `DOM` 的结构，我们发现所有的“热点新闻”信息（包括新闻标题和新闻页面链接）都在 `id` 为 `#pane-news` 的 `<div>` 下面 `<ul>` 下 `<li>` 下的 `<a>` 标签中。用 `jQuery` 的选择器表示为： `#pane-news ul li a` 。

3. 为了爬取新闻数据，首先我们要用superagent请求目标页面，获取整个新闻首页信息

```
1  // 引入所需要的第三方包
2  const superagent= require('superagent');
3
4  let hotNews = [];                                    // 热点新
   闻
5  let localNews = [];                                  // 本地新
   闻
6
7  /**
8   * index.js
9   * [description] - 使用superagent.get()方法来访问百度新闻首页
10  */
11 superagent.get('http://news.baidu.com/').end((err, res) =>
   {
12   if (err) {
```

```
    // 如果访问失败或者出错，会这行这里
    console.log(`热点新闻抓取失败 - ${err}`)
  } else {
   // 访问成功，请求http://news.baidu.com/页面所返回的数据会包含在res
    // 抓取热点新闻数据
    hotNews = getHotNews(res)
  }
});
```

1. 获取页面信息后，我们来定义一个函数 `getHotNews()` 来抓取页面内的“热点新闻”数据。

```
/**
 * index.js
 * [description] - 抓取热点新闻页面
 */
// 引入所需要的第三方包
const cheerio = require('cheerio');

let getHotNews = (res) => {
  let hotNews = [];
  // 访问成功，请求http://news.baidu.com/页面所返回的数据会包含在res.text中。

  /* 使用cheerio模块的cherrio.load()方法，将HTMLdocument作为参数传入函数
     以后就可以使用类似jQuery的$(selectior)的方式来获取页面元素
   */
  let $ = cheerio.load(res.text);

  // 找到目标数据所在的页面元素，获取数据
  $('div#pane-news ul li a').each((idx, ele) => {
```

```
    // cherrio中$('selector').each()用来遍历所有匹配到的DOM元素
    // 参数idx是当前遍历的元素的索引，ele就是当前便利的DOM元素
    let news = {
      title: $(ele).text(),        // 获取新闻标题
      href: $(ele).attr('href')    // 获取新闻网页链接
    };
    hotNews.push(news)              // 存入最终结果数组
  });
  return hotNews
};
```

这里要多说几点：

1. `async/await` 据说是异步编程的终级解决方案,它可以让我们以同步的思维方式来进行异步编程。`Promise` 解决了异步编程的“回调地狱”，async/await同时使异步流程控制变得友好而有清晰，有兴趣的同学可以去了解学习一下，真的很好用。
2. `superagent` 模块提供了很多比如 `get` 、 `post` 、 `delte` 等方法，可以很方便地进行Ajax请求操作。在请求结束后执行 `.end()` 回调函数。 `.end()` 接受一个函数作为参数，该函数又有两个参数 `error和res` 。当请求失败， `error` 会包含返回的错误信息，请求成功， `error` 值为 `null` ，返回的数据会包含在 `res` 参数中。
3. `cheerio` 模块的 `.load()` 方法，将 `HTML document` 作为参数传入函数，以后就可以使用类似jQuery的$(selectior)的方式来获取页面元素。同时可以使用类似于 `jQuery` 中的 `.each()` 来遍历元素。此外，还有很多方法，大家可以自行Google/Baidu

1. 将抓取的数据返回给前端浏览器

```
/**
 * [description] - 跟路由
 */
// 当一个get请求 http://localhost:3000时，就会后面的async函数
app.get('/', async (req, res, next) => {
  res.send(hotNews);
});
```

OK！！这样，一个简单的百度“热点新闻”的爬虫就大功告成啦！！

> 简单总结一下，其实步骤很简单：
>
> 1. `express` 启动一个简单的 `Http` 服务
> 2. 分析目标页面 `DOM` 结构，找到所要抓取的信息的相关 `DOM` 元素
> 3. 使用 `superagent` 请求目标页面
> 4. 使用 `cheerio` 获取页面元素，获取目标数据
> 5. 返回数据到前端浏览器

现在，继续我们的目标，抓取“本地新闻”数据（编码过程中，我们会遇到一些有意思的问题） 有了前面的基础，我们自然而然的会想到利用和上面相同的方法“本地新闻”数据。

1. 分析页面中“本地新闻”部分的 `DOM` 结构

> `F12` 打开控制台，审查“本地新闻” `DOM` 元素，我们发现，“本地新闻”分为两个主要部分，“左侧新闻”和右侧的“新闻资讯”。这所有目标数据都在 `id` 为 `#local_news` 的 `div` 中。“左侧新闻”数据又在 `id` 为 `#localnews-focus` 的 `ul` 标签下的 `li` 标签下的 `a` 标签中，包括新闻标题和页面链接。“本地资讯”数据又在 `id` 为 `#localnews-zixun` 的 `div` 下的 `ul` 标签下的 `li` 标签下的 `a` 标签中，包括新闻标题和页面链接。

1. OK！分析了 `DOM` 结构，确定了数据的位置，接下来和爬取 `“热点新闻”` 一样，按部就班，定义一个 `getLocalNews()` 函数，爬取这些数据。

```
**
 * [description] - 抓取本地新闻页面
 */
let getLocalNews = (res) => {
  let localNews = [];
  let $ = cheerio.load(res);

  // 本地新闻
```

```
  $('ul#localnews-focus li a').each((idx, ele) => {
    let news = {
      title: $(ele).text(),
      href: $(ele).attr('href'),
    };
    localNews.push(news)
  });

  // 本地资讯
  $('div#localnews-zixun ul li a').each((index, item) => {
    let news = {
      title: $(item).text(),
      href: $(item).attr('href')
    };
    localNews.push(news);
  });

  return localNews
};
```

对应的，在`superagent.get()`中请求页面后，我们需要调用`getLocalNews()`函数，来爬去本地新闻数据。 `superagent.get()`函数修改为：

```
superagent.get('http://news.baidu.com/').end((err, res) =>
{
  if (err) {
    // 如果访问失败或者出错，会这行这里
    console.log(`热点新闻抓取失败 - ${err}`)
  } else {
    // 访问成功，请求http://news.baidu.com/页面所返回的数据会包含在res
    // 抓取热点新闻数据
    hotNews = getHotNews(res)
    localNews = getLocalNews(res)
  }
});
```

同时，我们要在 `app.get()` 路由中也要将数据返回给前端浏览器。 `app.get()` 路由代码修改为：

```
/**
 * [description] - 跟路由
 */
// 当一个get请求 http://localhost:3000时，就会后面的async函数
app.get('/', async (req, res, next) => {
  res.send({
    hotNews: hotNews,
    localNews: localNews
  });
});
```

> 编码完成，激动不已！！ `DOS` 中让项目跑起来，用浏览器访问 `http://localhost:3000`

尴尬的事情发生了！！返回的数据只有热点新闻，而本地新闻返回一个空数组[]。检查代码，发现也没有问题，但为什么一直返回的空数组呢?

一个有意思的问题

> 为了找到原因，首先，我们看看用`superagent.get('http://news.baidu.com/').end((err, res) => {})`请求百度新闻首页在回调函数`.end()`中的第二个参数res中到底拿到了什么内容?

```
// 新定义一个全局变量 pageRes
let pageRes = {};         // supergaent页面返回值

// superagent.get()中将res存入pageRes
superagent.get('http://news.baidu.com/').end((err, res) =>
{
  if (err) {
    // 如果访问失败或者出错，会这行这里
    console.log(`热点新闻抓取失败 - ${err}`)
  } else {
   // 访问成功，请求http://news.baidu.com/页面所返回的数据会包
含在res
   // 抓取热点新闻数据
   // hotNews = getHotNews(res)
   // localNews = getLocalNews(res)
   pageRes = res
  }
});

// 将pageRes返回给前端浏览器，便于查看
app.get('/', async (req, res, next) => {
  res.send({
    // {}hotNews: hotNews,
```

```
22      // localNews: localNews,
23      pageRes: pageRes
24    });
25  });
```

> 可以看到，返回值中的 `text` 字段应该就是整个页面的 `HTML` 代码的字符串格式。为了方便我们观察，可以直接把这个 `text` 字段值返回给前端浏览器，这样我们就能够清晰地看到经过浏览器渲染后的页面。

修改给前端浏览器的返回值

```
app.get('/', async (req, res, next) => {
  res.send(pageRes.text)
}
```

> 审查元素才发现，原来我们抓取的目标数据所在的 `DOM` 元素中是空的，里面没有数据！ 到这里，一切水落石出！在我们使用 `superagent.get()` 访问百度新闻首页时， `res` 中包含的获取的页面内容中，我们想要的“本地新闻”数据还没有生成， `DOM` 节点元素是空的，所以出现前面的情况！抓取后返回的数据一直是空数组 `[ ]`

> 在控制台的 `Network` 中我们发现页面请求了一次这样的接口： `http://localhost:3000/widget?id=LocalNews&ajax=json&t=1526295667917` ， 接口状态 `404` 。这应该就是百度新闻获取 `“本地新闻”` 的接口，到这里一切都明白了！“本地新闻”是在页面加载后动态请求上面这个接口获取的，所以我们用 `superagent.get()` 请求的页面再去请求这个接口时，接口 `URL` 中 `hostname` 部分变成了本地 `IP` 地址，而本机上没有这个接口，所以 `404` ， 请求不到数据。

找到原因，我们来想办法解决这个问题！！

使用第三方 `npm` 包，模拟浏览器访问百度新闻首页，在这个模拟浏览器中当 `“本地新闻”` 加载成功后，抓取数据，返回给前端浏览器。

使用Nightmare自动化测试工具

> `Electron` 可以让你使用纯 `JavaScript` 调用 `Chrome` 丰富的原生的接口来创造桌面应用。你可以把它看作一个专注于桌面应用的 `Node.js` 的变体，而不是 `Web` 服务器。其基于浏览器的应用方式可以极方便的做各种响应式的交互
>
> `Nightmare` 是一个基于 `Electron` 的框架，针对 `Web` 自动化测试和爬虫，因为其具有跟 `PlantomJS` 一样的自动化测试的功能可以在页面上模拟用户的行为触发一些异步数据加载，也可以跟 `Request` 库一样直接访问 `URL` 来抓取数据，并且可以设置页面的延迟时间，所以无论是手动触发脚本还是行为触发脚本都是轻而易举的。

安装依赖

```
npm i nightware -S
```

给 `index.js` 中新增如下代码：

```
const Nightmare = require('nightmare');          // 自动化测试包，处理动态页面
const nightmare = Nightmare({ show: true });     // show:true  显示内置模拟浏览器

/**
 * [description] - 抓取本地新闻页面
 * [nremark] - 百度本地新闻在访问页面后加载js定位IP位置后获取对应新闻，
 * 所以抓取本地新闻需要使用 nightmare 一类的自动化测试工具，
```

```
 * 模拟浏览器环境访问页面，使js运行，生成动态页面再抓取
 */
// 抓取本地新闻页面
nightmare
.goto('http://news.baidu.com/')
.wait("div#local_news")
.evaluate(() =>
document.querySelector("div#local_news").innerHTML)
.then(htmlStr => {
  // 获取本地新闻数据
  localNews = getLocalNews(htmlStr)
})
.catch(error => {
  console.log(`本地新闻抓取失败 - ${error}`);
})
```

修改 `getLocalNews()` 函数为：

```
/**
 * [description]- 获取本地新闻数据
 */
let getLocalNews = (htmlStr) => {
  let localNews = [];
  let $ = cheerio.load(htmlStr);

  // 本地新闻
  $('ul#localnews-focus li a').each((idx, ele) => {
    let news = {
      title: $(ele).text(),
      href: $(ele).attr('href'),
    };
    localNews.push(news)
  });
```

```
16
17    // 本地资讯
18    $('div#localnews-zixun ul li a').each((index, item) => {
19      let news = {
20        title: $(item).text(),
21        href: $(item).attr('href')
22      };
23      localNews.push(news);
24    });
25
26    return localNews
27  }
```

修改`app.get('/')`路由为：

```
1  /**
2   * [description] - 跟路由
3   */
4  // 当一个get请求 http://localhost:3000时，就会后面的async函数
5  app.get('/', async (req, res, next) => {
6    res.send({
7      hotNews: hotNews,
8      localNews: localNews
9    })
10 });
```

好了，大功告成~~~

总结

1. `express`启动一个简单的`Http`服务
2. 分析目标页面`DOM`结构，找到所要抓取的信息的相关`DOM元`素
3. 使用`superagent`请求目标页面

> 4. **动态页面（需要加载页面后运行JS或请求接口的页面）可以使用Nightmare模拟浏览器访问**
> 5. 使用 `cheerio` 获取页面元素，获取目标数据

socket实现聊天室

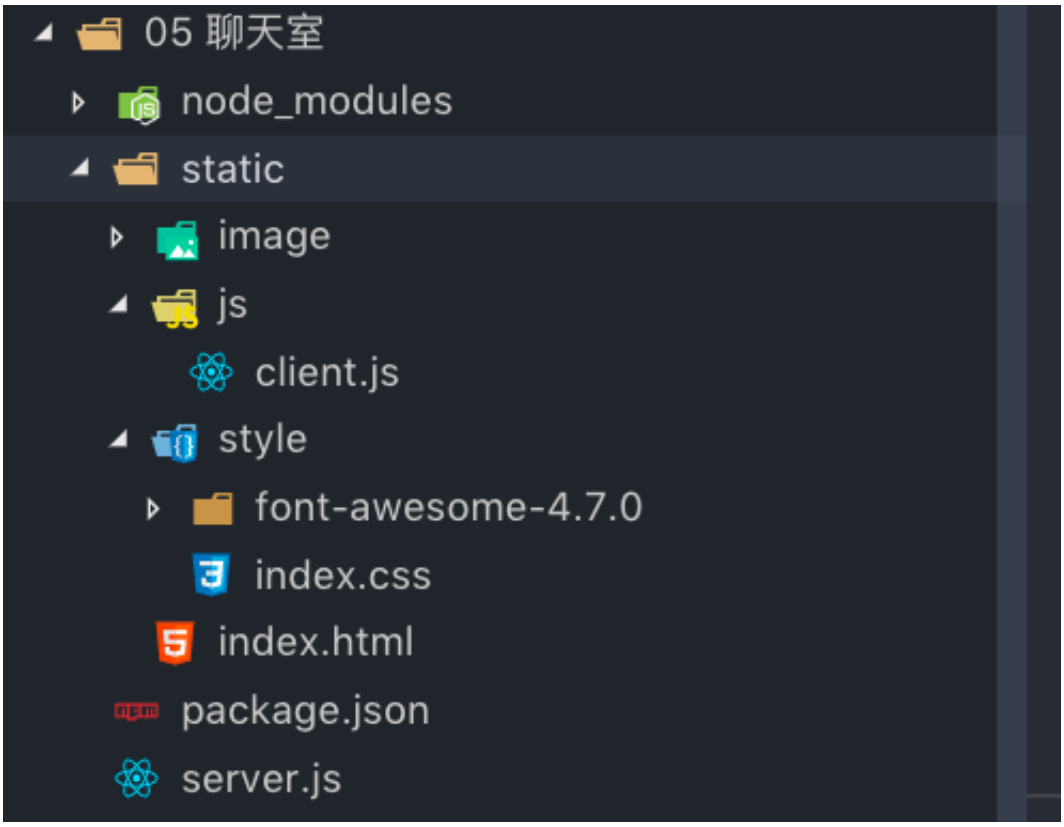
案例:实现聊天室

基于node+express+socket.io+vue+flex来实现简易的聊天室

实现功能:

- 登录检测
- 系统提示在线人员状态(进入/离开)
- 接收和发送消息
- 自定义消息字体颜色
- 支持发送表情
- 支持发送图片
- 支持发送窗口震动

项目结构:



index.html

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <link rel="stylesheet" href="style/index.css">
    <link rel="stylesheet" href="style/font-awesome-4.7.0/css/font-awesome.min.css">
  </head>

  <body>
    <div id="app">
      <div class="name" v-if='isShow'>
```

```
16          <!-- <h2>请输入你的昵称</h2> -->
17          <input @keyup.enter='handleClick' type="text"
    id="name" placeholder="请输入昵称..." autocomplete="off"
18                 v-model='username'>
19          <button id="nameBtn" @click='handleClick'>确 定
    </button>
20        </div>
21        <div class="main" :class='{shaking:isShake}'>
22          <div class="header">
23            <img src="image/logo.jpg">
24            ❤️聊天室
25          </div>
26          <div id="container">
27            <div class="conversation">
28              <ul id="messages">
29                <li v-for='(user,index) in userSystem'
    :class='user.side'>
30                  <div v-if='user.isUser'>
31                    <img :src="user.img" alt="">
32                    <div>
33                      <span>{{user.name}}</span>
34                      <p :style="{color: user.color}" v-
    html='user.msg'>
35                      </p>
36                    </div>
37                  </div>
38                  <p class='system' v-else>
39                    <span>{{nowDate}}</span><br />
40                    <span v-if='user.status'>{{user.name}}
    {{user.status}}了聊天室</span>
41                    <span v-else>{{user.name}}发送了一个窗口抖
    动</span>
42                  </p>
```

```
43
44                </li>
45              </ul>
46              <form action="">
47                <div class="edit">
48                  <input type="color" id="color" v-
    model='color'>
49                  <i title="自定义字体颜色" id="font"
    class="fa fa-font">
50                  </i><i @click='handleSelectEmoji'
    @dblclick='handleDoubleSelectEmoji' title="双击取消选择"
    class="fa fa-smile-o" id="smile">
51                  </i><i @click='handleShake' title="单击页面
    震动" id="shake" class="fa fa-bolt">
52                  </i>
53                  <input type="file" id="file">
54                  <i class="fa fa-picture-o" id="img"></i>
55                  <div class="selectBox" v-
    show='isEmojiShow'>
56                    <div class="smile" id="smileDiv">
57                      <p>经典表情</p>
58                      <ul class="emoji">
59                        <li v-for='(emojiSrc,i) in emojis'
    :id='i' :key='i'>
60                          <img :src="emojiSrc" :alt="i+1"
    @click='handleEmojiImg(i+1)'>
61                        </li>
62                      </ul>
63                    </div>
64                  </div>
65                </div>
66                <!-- autocomplete禁用自动完成功能 -->
```

```
67              <textarea id="m" v-model='msgVal' autofocus
    @keyup.enter='handleSendMsg'></textarea>
68              <button class="btn rBtn" id="sub"
    @click='handleSendMsg'>发送</button>
69              <button class="btn" id="clear"
    @click='handleLogout'>关闭</button>
70            </form>
71          </div>
72          <div class="contacts">
73            <h1>在线人员(<span id="num">{{userInfo.length}}
    </span>)</h1>
74            <ul id="users">
75              <li v-for='(user,index) in userInfo'>
76                <img :src="user.img" alt="">
77                <span>{{user.username}}</span>
78              </li>
79            </ul>
80            <p v-if='userInfo.length==0'>当前无人在线哟~</p>
81          </div>
82        </div>
83      </div>
84    </div>
85    <script
    src="https://cdn.bootcss.com/socket.io/2.2.0/socket.io.js">
    </script>
86    <script
    src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js">
    </script>
87    <script src='js/client.js'></script>
88  </body>
89
90  </html>
```

server.js

```
const express = require('express');
const app = express();1
const http = require('http').Server(app);
const io = require('socket.io')(http);
const port = process.env.PORT || 5000;

let users = [];//存储登录的用户
let userInfo = [];//存储用户姓名和头像

app.use('/',express.static(__dirname+'/static'));
app.get('/', function (req, res) {
  res.sendFile(__dirname + '/index.html');
});

io.on('connection', function (socket) {
  console.log('连接成功');
  socket.on('login', function (user) {
    const {username} = user;
    console.log(users.indexOf(username))

    if (users.indexOf(username)>-1){
      socket.emit('loginError');

    }else{
      // 存储用户名
      users.push(username);
      userInfo.push(user);
      io.emit('loginSuc');
      socket.nickName = username;
```

```
30        // 系统通知
31        io.emit('system', {
32          name: username,
33          status: '进入'
34        })
35
36        // 显示在线人员
37        io.emit('disUser',userInfo);
38        console.log('一个用户登录');
39
40      }
41
42    });
43    // 发送窗口事件
44    socket.on('shake',()=>{
45      socket.emit('shake',{
46        name:'您'
47      })
48      // 广播消息
49      socket.broadcast.emit('shake',{
50        name:socket.nickName
51      })
52    });
53    // 发送消息事件
54    socket.on('sendMsg',(data)=>{
55
56      let img = '';
57      for(let i = 0; i < userInfo.length;i++){
58        if(userInfo[i].username === socket.nickName){
59          img = userInfo[i].img;
60        }
61      }
62      // 广播
```

```
63      socket.broadcast.emit('receiveMsg', {
64        name: socket.nickName,
65        img: img,
66        msg: data.msgVal,
67        color: data.color,
68        type: data.type,
69        side: 'left',
70        isUser:true
71      });
72      socket.emit('receiveMsg', {
73        name: socket.nickName,
74        img: img,
75        msg: data.msgVal,
76        color: data.color,
77        type: data.type,
78        side: 'right',
79        isUser: true
80      });
81
82    })
83
84    // 断开连接时
85    socket.on('disconnect',()=>{
86      console.log('断开连接');
87
88      let index = users.indexOf(socket.nickName);
89      if(index > -1){
90        users.splice(index,1);//删除用户信息
91        userInfo.splice(index,1);//删除用户信息
92
93        io.emit('system',{
94          name:socket.nickName,
95          status:'离开'
```

```
        })
        io.emit('disUser,userInfo'); //重新渲染
        console.log('一个用户离开');

    }
  })

});


http.listen(3000, () => {
  console.log('listen on 3000端口');
})
```

client.js

```

const vm = new Vue({
  el: '#app',
  data() {
    return {
      username: '',
      msgVal: '',
      isShow: true,
      nowDate: new Date().toTimeString().substr(0, 8),
      userHtml: '',
      userInfo: [],
      isShake: false,
      timer: null,
      userSystem: [],
      color: '#000000',
```

```
      emojis: [],
      isEmojiShow: false
    }
  },
  methods: {
    handleClick() {
      var imgN = Math.floor(Math.random() * 4) + 1; // 随机分配头像
      if (this.username) {
        this.socket.emit('login', {
          username: this.username,
          img: 'image/user' + imgN + '.jpg'
        });
      }

    },
    shake() {
      this.isShake = true;
      clearTimeout(this.timer);
      this.timer = setTimeout(() => {
        this.isShake = false;
      }, 500);
    },
    handleShake(e) {
      this.socket.emit('shake');
    },
    // 发送消息
    handleSendMsg(e) {
      e.preventDefault();
      if (this.msgVal) {
        this.socket.emit('sendMsg', {
          msgVal: this.msgVal,
          color: this.color,
```

```
48            type: 'text'
49          })
50          this.msgVal = '';
51        }
52      },
53      scrollBottom() {
54        this.$nextTick(() => {
55          const div = document.getElementById('messages');
56          div.scrollTop = div.scrollHeight;
57        })
58      },
59      initEmoji() {
60        for (let i = 0; i < 141; i++) {
61          this.emojis.push(`image/emoji/emoji (${i +
    1}).png`);
62        }
63      },
64      // 点击微笑 弹出表情
65      handleSelectEmoji() {
66        this.isEmojiShow = true;
67      },
68      handleDoubleSelectEmoji() {
69        this.isEmojiShow = false;
70      },
71      // 用户点击发送表情
72      handleEmojiImg(index) {
73        this.isEmojiShow = false;
74        this.msgVal = this.msgVal + `[emoji${index}]`;
75      },
76      handleLogout(e) {
77        e.preventDefault();
78        this.socket.emit('disconnect')
79      }
```

```
  },
  created() {
    const socket = io();
    this.socket = socket;
    this.socket.on('loginSuc', () => {
      this.isShow = false;
    });
    this.socket.on('loginError', () => {
      alert('用户名已存在,请重新输入');
      this.username = '';
    })
    // 系统提示消息
    this.socket.on('system', (user) => {
      console.log(user);
      this.userSystem.push(user);
      this.scrollBottom()
    })
    // 显示在线人员
    this.socket.on('disUser', (userInfo) => {
      this.userInfo = userInfo;
    })
    //监听抖动事件
    this.socket.on('shake', (user) => {
      this.userSystem.push(user);
      this.shake();
      this.scrollBottom();
    })
    this.socket.on('receiveMsg', (obj) => {
      let msg = obj.msg;
      let content = ''
      if (obj.type === 'img') {
      }
      // 提取文字中的表情加以渲染
```

```
113         while (msg.indexOf('[') > -1) {  // 其实更建议用正则将[]中的内容提取出来
114             var start = msg.indexOf('[');
115             var end = msg.indexOf(']');
116             content += '<span>' + msg.substr(0, start) + '</span>';
117             content += '<img src="image/emoji/emoji%20(' + msg.substr(start + 6, end - start - 6) + ').png">';
118             msg = msg.substr(end + 1, msg.length);
119         }
120         content += '<span>' + msg + '</span>';
121         obj.msg = content;
122         this.userSystem.push(obj);
123         // 滚动条总是在最底部
124         this.scrollBottom();
125       })
126       // 渲染表情
127       this.initEmoji();
128     }
129
130 })
```