

axios: 基于promise的HTTP第三方库, 可用于客户端和服务端node.js

axios特性

- 1.支持promiseAPI
- 2.拦截请求和响应（可以在请求前，响应前做一些操作，如在请求头中加入信息）
- 3.转换请求数据和响应数据（在请求时，对敏感信息加密；响应时解密）
- 4.取消请求
- 5.自动转换JSON数据（一般在http请求时，传输的为字符串）
- 6.客户端支持防御XSRF

axios浏览器支持

火狐，Opera，谷歌，IE(8以上)

axios请求方式

axios请求方法：get,post,put,patch,delete

get:获取数据

post:提交数据（表单提交+文件上传）

put:更新数据——将所有数据推送到后端（服务端）——一般用于新建

patch:更新数据——只将修改的数据送到后端（服务端）——一般用于更新

delete:删除数据

参数：

1. post,put,patch-三个参数：url路径，请求数据，config配置

请求数据data格式两种：

1.form-data：表单提交（图片文件上传）

2.application/json

- 2.get,delete-两个参数：url路径，config配置

axios使用

1.get (Get.vue)

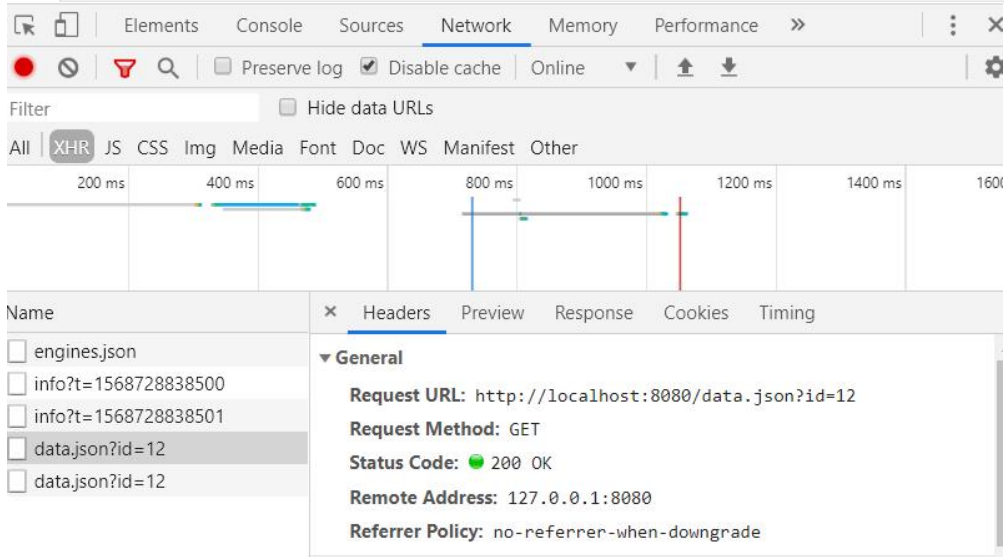
```
1 //在created生命周期
2 created(){
3   // 执行请求
4   // 第一种写法
5   // 传参请求—http://localhost:8080/data.json?id
6   axios.get('/data.json',{
7     params:{id:12}}).then((res)=>{console.log(res)})
8   // 另一种写法
9   // 传参
10  axios({
```

```

11     method: 'get',
12     url: '/data.json',
13     params: {id: 12}
14   }).then((res) => { console.log(res) })
15 }

```

1 访问: <http://localhost:8080/get>



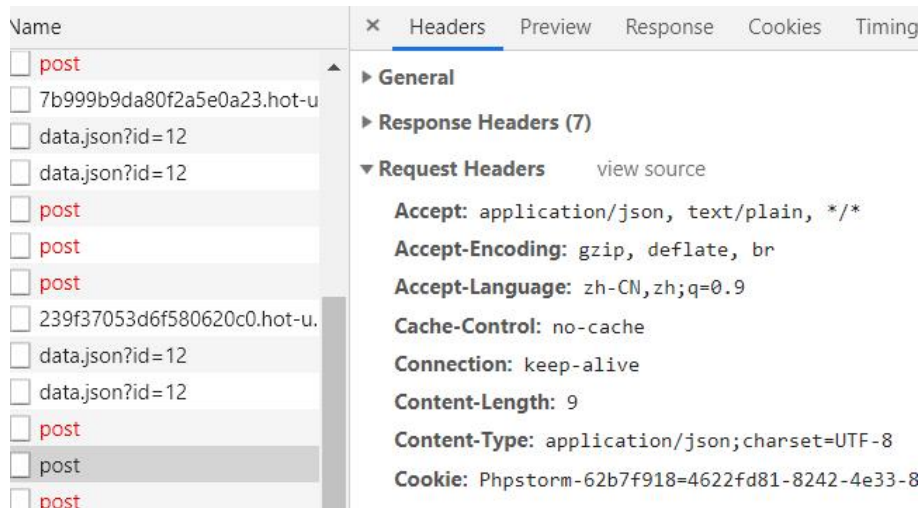
2.post请求

- 1 post-三个参数: url路径, {}请求数据, config配置
- 2 请求数据data格式两种:
- 3 1.form-data: 表单提交 (图片文件上传)
- 4 2.application/json

```

1 //application/json
2 let data = {id:12}
3 axios.post('/post', data).then(res => {
4   console.log(res)
5 })
6 // 另一种写法
7 // 传参
8 axios({
9   method: 'post',
10  url: '/post',
11  data: data
12 }).then((res) => {
13   console.log(res)
14 })

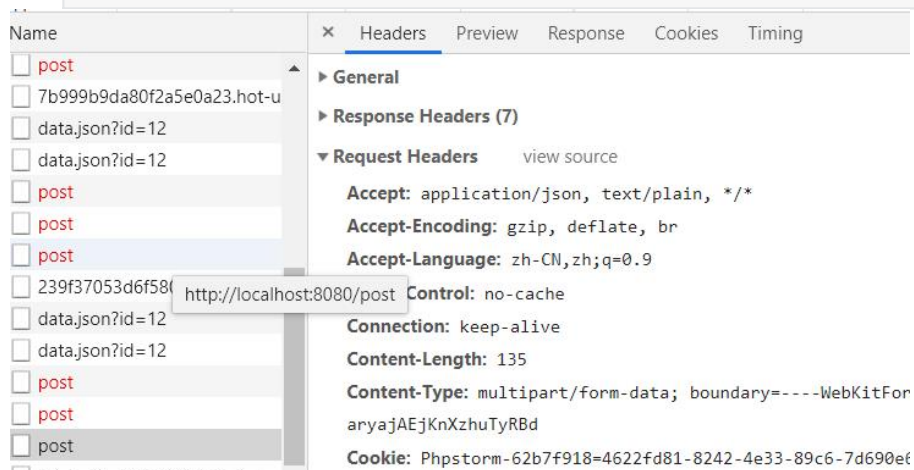
```



```

1 // form-data请求
2 // 创建一个FormData格式的对象
3 let formData = new FormData()
4 let data = {id:12}
5 // 对data遍历（key对应id）
6 for(let key in data){
7     // 将键值添加到formData中
8     formData.append(key,data[key])
9 }
10 axios.post('/post',formData).then(res=>{
11     console.log(res)
12 })

```



3.put请求

```

1 // put请求
2 axios.put('/put',data).then(res=>{
3     console.log(res)
4 })

```

4.patch请求

```
1 // patch请求
2 axios.patch('/patch',data).then(res=>{
3   console.log(res)
4 })
```

5.delete请求

```
1 // delete请求-两个参数
2
3 // 若接口需要在url后拼接参数
4 axios.delete('/delete',{
5   params:{
6     id:12
7   }
8 }).then(res=>{
9   console.log(res)
10 })
11
12 // 不在url传输
13 axios.delete('/delete',{
14   data:{
15     id:12
16   }
17 }).then(res=>{
18   console.log(res)
19 })
20
21 // 不用别名
22 axios({
23   method:'delete',
24   url:'/delete',
25   params:{},
26   data:{}
27 }).then(res=>{
28   console.log(res)
29 })
30
```

并发请求：同时进行多个请求，并统一处理返回值

- 1 两个方法：
- 2 `axios.all()`-参数：数组，数组内的值是一个一个`axios`请求
- 3 `axios.spread()`-作用是：在`axios.all()`多个请求完成的时候，将返回数据进行分割处理
- 4 然后对每一个返回值统一进行处理

```
1 //在created生命周期
2 created(){
3   axios.all(
4     [
5       axios.get('/data.json'),
6       axios.get('/city.json'),
7     ]
8     // then()内的参数为axios.spread()
9   ).then(
10     // axios.spread()内的参数为回调函数
11     // axios.all()有几个方法，就需要有几个对应的返回值
12     axios.spread((dataRes,cityRes)=>{
13       console.log(dataRes,cityRes)
14     })
15   )
16 }
```

创建axios实例

```
1 //在created生命周期
2 created(){
3   // 当项目中只有一种后端语言时,可直接:
4   axios.get('/data.json')
5
6   // 当后端接口地址有多个，且超时时长不一样
7   // 创建实例
8   // axios.create(axios配置信息)方法
9   let axios1 = axios.create({
10     // 域名
11     baseURL: 'http://localhost:8080',
12     // 超时时长（默认是1000毫秒），超时报401
13     timeout:1000
14   })
15   let axios2 = axios.create({
```

```

16     // 域名
17     baseURL: 'http://localhost:9090',
18     // 超时时长（默认是1000毫秒），超时报401
19     timeout: 5000
20 })
21
22 // 使用实例
23 axios1.get('/data.json').then(res=>{
24     console.log(res)
25 })
26 axios2.get('/city.json').then(res=>{
27     console.log(res)
28 })
29 }

```

axios配置参数

```

1 //在created生命周期
2 created(){
3     axios.create({
4         // 请求域名，基本地址
5         baseURL: 'http://localhost:8080',
6         // 请求超时时长（毫秒）
7         timeout: 1000,
8         // 请求路径url
9         url: '/data.json',
10        // 请求方法
11        method: 'get,post,patch,put,delete',
12        // 设置请求头
13        headers: {},
14        // 比如登录后一些接口需要限权
15        // headers:{
16        //     // 识别当前登陆人信息
17        //     token: ''
18        // }
19        //请求参数拼接在url上
20        params: {},
21        // 请求参数放在请求体中
22        data: {}
23    })

```

```
24 // get()后为相对路径,请求时会和baseUrl拼接—http://localhost:8080/data.  
    json  
25 axios.get('/data.json',{config:上述的配置参数})  
26 }
```

axios配置方法

```
1 // 1.axios全局配置(常修改的全局配置timeout和baseUrl)  
2     axios.defaults.timeout = 1000  
3     axios.defaults.baseUrl = 'http://localhost:8080'  
4 // 2.axios实例配置  
5     // 若不添加参数,则默认全局配置  
6     let instance = axios.create()  
7     // 修改配置  
8     instance.defaults.timeout = 3000  
9 // 3.axios请求配置  
10    instance.get('/data.json',{  
11        timeout:3000  
12    })  
13 // 4.优先级-高--->低  
14 // 请求配置>实例配置>全局配置
```

axios配置具体使用

```
1 //在created生命周期内  
2 // 实际开发中,全局配置很少用到  
3     // 如:有两种接口  
4     // http://localhost:8080  
5     // http://localhost:9090  
6  
7     // 创建实例  
8     let instance1 = axios.create({  
9         baseUrl:'http://localhost:8080',  
10        timeout:1000  
11    })  
12    let instance2 = axios.create({  
13        baseUrl:'http://localhost:9090',  
14        timeout:3000  
15    })  
16    // 使用实例请求  
17    // instance1用到的参数: baseUrl, timeout,url,method,params
```

```

18     instance1.get('/data.json',{
19         params:{}
20     }).then(res=>{
21         console.log(res)
22     })
23     // instance2用到的参数: baseUrl, timeout: 5000,url,method
24     instance2.get('/city.json',{
25         //当实例的配置不符合时
26         timeout:5000,
27     }).then(res=>{
28         console.log(res)
29     })
30

```

axios拦截器 (请求拦截器+响应拦截器)

```

1     //在created生命周期
2     created(){
3         // 拦截器: 在请求或响应被处理前拦截它们
4         // (在发起请求前做些处理, 在响应回来后做些处理)
5         // 请求拦截器+响应拦截器
6         //(use()方法有两个参数: 1, 请求前的回调; 2.请求错误的回调)
7
8         // 请求拦截器
9         axios.interceptors.request.use
10         (config=>{
11             // 在发送请求前做些什莫, 处理后需要return出去
12             return config
13         },err=>{
14             // 在请求错误时做些什么
15             return Promise.reject(err)
16         })
17
18         // 响应拦截器
19         axios.interceptors.response.use
20         (res=>{
21             // 请求成功, 对响应数据做些处理, 处理后需要return出去
22             return res
23
24         },err=>{
25             // 响应错误做些什莫

```



```

26     return Promise.reject(err)
27   })
28   // 请求成功的数据，会到res;请求失败的数据，会到err
29   axios.get('').then(res=>{
30     //成功
31   }).catch(err=>{
32     //失败
33   })
34 }

```

- 1 请求错误和响应错误的区别：
- 2 请求错误：发送请求，但没有到达后端，报错**404**
- 3 响应错误：发送请求，有到达后端，返回的错误

拦截器使用（举例）：登录状态

```

1  // 举例：登录状态（登录，后端会返回一个参数（一个编码的字符串）token:''）
2    // token内包含了加密的个人信息，通过加密信息可以识别身份
3    // 一般token，放在请求头headers内
4
5    // 使用请求拦截器
6
7    // 访问-需要登录的接口
8    //声明实例(在开发中，给实例添加拦截器和属性，不会给axios设置内容。若设置则
   可能会造成全局污染)
9    let instance = axios.create({})
10   instance.interceptors.request.use
11   (config=>{
12     // 给token赋值
13     config.headers.token = ''
14
15     // return 回去
16     return config
17   })
18   // 若访问-不需要登录的接口
19   let instance2 = axios.create({})

```

拦截器使用—移动端开发-弹窗（使用请求拦截器+响应拦截器）

```

1    // 移动端开发-使用请求拦截器
2    let instance3 = axios.create({})
3    instance3.interceptors.request.use

```

```

4      (config=>{
5          // 有一弹窗，在请求前将弹窗弹出
6          $('#model').show()
7          // return
8          return config
9      })
10     // 请求后，将弹窗隐藏
11     // 使用响应拦截器
12     instance3.interceptors.response.use
13     (res=>{
14         // 弹窗隐藏
15         $('#model').hide()
16         return res
17     })

```

axios错误处理

```

1  //在created生命周期
2  created(){
3
4      // 错误处理:请求错误时进行的处理
5      // 请求错误和响应错误一般都会汇聚到catch中
6
7      // 执行请求
8      axios.get('/data.json').then((res)=>{
9          // 成功
10         console.log(res)
11     }).catch(err=>{
12         // 失败
13     })
14
15     // 举例：实际开发中，一般添加统一的错误处理
16     // 创建实例
17     let instance = axios.create({})
18     // 请求拦截器
19     instance.interceptors.request.use
20     (config=>{
21         console.log(config)
22     },err=>{
23         // 请求错误-一般http状态码以4开头，常见：401超时；404not found
24         // 添加2s弹窗提示

```

```

25     $('#model').show()
26     setTimeout(()=>{
27         $('#model').hide()
28     },2000)
29     return Promise.reject(err)
30 })
31 // 响应拦截器
32 instance.interceptors.response.use
33 (res=>{
34     console.log(res)
35 },err=>{
36     // 添加2s弹窗提示
37     $('#model').show()
38     setTimeout(()=>{
39         $('#model').hide()
40     },2000)
41     // 响应错误处理-一般http状态码以5开头，500-系统错误；502-系统重启
42     return Promise.reject(err)
43 })
44
45 // 请求
46 instance.get('data.json').then(res=>{
47     //成功
48     console.log(res)
49 }).catch(err=>{
50     // 错误处理
51     console.log(err)
52 })
53 }

```

两个JSON+contactList:

第一个json:称为“不确定通信”，判断接口是否能请求到后端服务；通，则继而请求第二个json

Name	× Headers Preview Response Timing
<input type="checkbox"/> engines.json	Request URL: http://localhost:9000/api/contact/new/json
<input type="checkbox"/> info?t=1568799147873	Request Method: OPTIONS
<input type="checkbox"/> info?t=1568799147875	Status Code: 204 No Content
<input type="checkbox"/> contactList	http://192.168.16.105:8081/sockjs-node/info?t=1568799147875
<input type="checkbox"/> json	Referrer Policy: no-referrer-when-downgrade
<input type="checkbox"/> json	▼ Response Headers
<input type="checkbox"/> contactList	Access-Control-Allow-Credentials: true
	Access-Control-Allow-Headers: Content-Type, Authorization, Accept
	Access-Control-Allow-Methods: GET, POST, DELETE, PUT, PATCH
	Access-Control-Allow-Origin: *
	Access-Control-Max-Age: 5
	Connection: keep-alive
	Date: Wed, 18 Sep 2019 09:33:01 GMT
	Varv: Origin

第二个json：真正的请求

Name	× Headers Preview Response Timing
<input type="checkbox"/> engines.json	▼ General
<input type="checkbox"/> info?t=1568799147873	Request URL: http://localhost:9000/api/contact/new/json
<input type="checkbox"/> info?t=1568799147875	Request Method: POST
<input type="checkbox"/> contactList	Status Code: 200 OK
<input type="checkbox"/> json	Remote Address: [::1]:9000
<input type="checkbox"/> json	Referrer Policy: no-referrer-when-downgrade
<input type="checkbox"/> contactList	▼ Response Headers
	Access-Control-Allow-Origin: *
	Access-Control-Expose-Headers: WWW-Authenticate, Server-Authorization
	Connection: keep-alive
	Content-Length: 68
	Content-Type: application/json; charset=utf-8
	Origin: http://localhost:8081
	Referer: http://localhost:8081/contactList
	Sec-Fetch-Mode: cors
	User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X) AppleWebKit/604.1.38 (KHTML, like Gecko) Version/11.0 Mobile/15A372 Safari/604.1
	▼ Request Payload view source
	▼ {tel: "11012011912", name: "猪八戒"}
	name: "猪八戒"
	tel: "11012011912"

返回值：

Name	× Headers Preview Response Timing
<input type="checkbox"/> engines.json	▼ {code: 200, data: {tel: "11012011912", name: "猪八戒", id: 37}} code: 200 ▶ data: {tel: "11012011912", name: "猪八戒", id: 37}
<input type="checkbox"/> info?t=1568799147873	
<input type="checkbox"/> info?t=1568799147875	
<input type="checkbox"/> contactList	
<input type="checkbox"/> json	
<input type="checkbox"/> json	
<input type="checkbox"/> contactList	

第三个contactList: 联系人列表

Name	× Headers Preview Response Timing
<input type="checkbox"/> engines.json	▼ General Request URL: http://localhost:9000/api/contactList Request Method: GET Status Code: 🟢 200 OK Remote Address: [::1]:9000 Referrer Policy: no-referrer-when-downgrade
<input type="checkbox"/> info?t=1568799147873	
<input type="checkbox"/> info?t=1568799147875	
<input type="checkbox"/> contactList	
<input type="checkbox"/> json	
<input type="checkbox"/> json	
<input type="checkbox"/> contactList	▼ Response Headers Access-Control-Allow-Origin: * Access-Control-Expose-Headers: WWW-Authenticate, Server-Authorization Connection: keep-alive Content-Length: 1753 Content-Type: application/json; charset=utf-8

Name	× Headers Preview Response Timing
<input type="checkbox"/> engines.json	▼ {code: 200, data: [{name: "张三", tel: "13000000000", id: 1}, {name: "李四", tel: "13000000000", id: 2}]} code: 200 ▶ data: [{name: "张三", tel: "13000000000", id: 1}, {name: "李四", tel: "13000000000", id: 2}]
<input type="checkbox"/> info?t=1568799147873	
<input type="checkbox"/> info?t=1568799147875	
<input type="checkbox"/> contactList	
<input type="checkbox"/> json	
<input type="checkbox"/> json	
<input type="checkbox"/> contactList	http://localhost:9000/api/contact/new/json