课程依赖:

* promise
* ajax HTTP

1. axios 是目前前端最热门的请求工具, 用来浏览器向服务器发送 AJAX 请求进行数据交换，和在nodejs端发送http请求。
    a. axios是基于promise的http客户端可在浏览器和nodejs中运行
    b. axios可在响应之前做准备工作，响应回来之后对结果做一些预处理
2. 因为axios需要用到服务器，所以需要使用json server来搭建服务，可以得到一个假的rest api 用三十秒时间
3. 如何发送ajax请求：
    a. 调用axios函数即axios(),
    b. axios()函数接受参数是一个对象，对象中包括一些属性请求类型methods、请求url、请求体信息data(post和put用的着)、url参数、请求头信息等，即

```
axios({
  methos:'GET',
  url:'http;//localhost:3000/posts/2',
}),
axios({
  methos:'POST',
  url:'http;//localhost:3000/posts',
  data:{
    title:"尚硅谷",
    name:"张三",
  }
}),
```

    c. axios函数返回的结果是一个promise对象所以可以用then方法指定成功的回调，获取成功的结果，即

```
axios({
  methos:'GET',
  url:'http;//localhost:3000/posts/2',
}).then(response=>{
  console.log(response);
})
```

    4.axios其他方式请求发送：
    d. axios的一些方法来发送请求
    axios.request(config): 等同于 axios(config)请求方式

    axios.get(url[, config]): 发 get 请求
    axios.delete(url[, config]): 发 delete 请求

axios.post(url[, data, config]): 发 post 请求
axios.put(url[, data, config]): 发 put 请求

```
//发送 GET 请求
btns[0].onclick = function(){
  // axios()
  axios.request({
    method:'GET',
    url: 'http://localhost:3000/comments'
  }).then(response => {
    console.log(response);
  })
}
//发送 POST 请求
btns[1].onclick = function(){
  // axios()
  axios.post(
    'http://localhost:3000/comments',
    {
      "body": "喜大普奔",
      "postId": 2
    }).then(response => {
      console.log(response);
    })
}
```

```
axios.defaults.xxx: 请求的默认全局配置
axios.interceptors.request.use(): 添加请求拦截器
axios.interceptors.response.use(): 添加响应拦截器

axios.create([config]): 创建一个新的 axios(它没有下面的功能)

axios.Cancel(): 用于创建取消请求的错误对象
axios.CancelToken(): 用于创建取消请求的 token 对象
axios.isCancel(): 是否是一个取消请求的错误
axios.all(promises): 用于批量执行多个异步请求
axios.spread(): 用来指定接收所有成功数据的回调函数的方法
```

5.axios请求响应结果结构：

consfig:配置对象，里面包含很多内容，里面包括请求类型，url,请求体等数据

data{}:响应体的结果，就是服务器返回的结果，里面内容是对象，因为axios自动将服务器返回结果进行json解析转成对象，方便做处理

header:响应头信息，响应报文包括四个部分：响应头，响应行，响应空行，响应体，这

里是响应头信息。

request:原生的Ajax请求对象，axios发送Ajax请求，而发送Ajax请求就要用到XHLHttpRequest实例对象，而request属性所保存的当前axios发送请求时所创建的ajax请求对象也就是XHLHttpRequest实例对象，

status:响应状态

statusText:响应状态字符串

6.axios请求对象：

1.axios请求对象:指axios在调用时，他所接收的参数对象，看他里面有哪些配置内容config,不仅是axios，还有request,get,post等的配置

(常用)url：指明给谁发送请求，设置url参数

(常用)method：请求类型：get等

(常用)baseURL：设定url的基础结构，axios内部会自动把url(后面的路径)和baseUrl(共有的部分)进行结合成最终的url结果。

transformRequest：对请求的数据进行处理，在将处理后的结果向服务器进行发送，对请求参数进行一个预处理

transformResponse:对响应的结果进行一些改变，之后再用自定义回调去处理结果，对响应结果进行一个预处理

(常用)headers:头信息，对请求头信息进行配置，如在进行身份校验的时候要求在头信息中加入一个特殊的标识来检验你的请求是否满足条件，这时就可以借助headers对请求头信息进行控制

(常用)params(常用): {},设定url参数的，向服务端发送请求，在发送请求时传递url参数

```
* axios({
*     url: '/post',
*     // /post?a=100&b=200
*     // /post/a/100/b/200
*     // /post/a.100/b.200
*     params: {
*        a:100,
*        b:200
*     }
* })
```

paramsSerializer（用的少）:参数序列化的配置项，用的少，对请求参数做一个序列化，转化成一个字符串，

```
 * axios({
*     url: '/post',
*     // /post?a=100&b=200
```

先转换成后面两种,就用到paramsSerializer对这个对象做一个个处理，转换成最终形式上的字符串，与服务器做一个统一

```
*     // /post/a/100/b/200
*     // /post/a.100/b.200
*
* })
```

(常用)data(用的多):一种是对象形式{}，一种是字符串形式''，对象形式axios会将其转成json格式字符串传递，如果是字符串形式，axios就直接传递了。

(常用)timeout:超时时间，发送请求时超时时间，超过时间就取消

withCredentials:跨域请求时，对cookic的携带做一个设置，false是不携带，true是随跨域请求时可以对cookic做一个携带

adapter:对请求的适配器做一个设置，有两种：一种是ajax的，另一个是在nodejs中发送http请求的，两个运行环境

auth（用少）:请求基础就是验证，设置用户名和密码的

responseType: 对响应体结果的格式做一个设置，默认是json格式，默认服务器返回结果是json格式的，结果回来后会自动将其做一个转换

responseEncoding:响应结果编码utf-8

xsrfCookieName:跨域，跨站请求的一个标识，对cookic的名字做一个设置，对头信息做一个设置，这是一个安全设置，保证请求是来自我们自己得客户端而不是来自于未知的其他的网站的页面，起保护作用，保护作用要结合服务器做一个说明,服务器返回结果会返回唯一的标识，下次发送请求的时候需要再传递过去，服务器认了之后检测没有问题才会给你做响应，这就是这个流程，为什么他能实现保护呢,因为有些网站页面里会加入一些链接向我们的服务器发送请求，如果不做唯一的标识去检验的话，可能这个页面里面发送的请求就直接对结果产生影响，加上唯一标识后，我们的客户端就是可以发送的，其他的网页再发送请求时是不能携带标识参数的，他没有，所以就可以避免跨站攻击了

xsrfHeaderName:对名字做标识的

onUploadProgress:上传时的回调

onDownloadProgress:下载时的回调

maxContentLength: 设置http响应体的最大尺寸

maxBodyLength:请求体的最大尺寸

validateStatus:对响应结果的成功做一个设置，有成功与失败的规则或者自己设置

maxRedirects:最大跳转次数5次，向一个服务发送请求做了跳转，做了跳转之后要不要继续往前做请求，只用在nodejs中,在前端ajax中用不到

beforeRedirect:

socketPath:设定socket文件的位置，向docker守护进程发送请求的，做数据转发,与proxy也有关系,如果这俩都用了，优先使用socket文件配置

httpAgent: 对客户端信息做一些设置，keepalive等是否保持连接

httpsAgent: 同上

proxy: 代理，用在服务端的nodejs中，一般做爬虫，如果你用一个ip去向服务器发送请求去抓取数据，很容易被别人禁你的ip，这个时候可以借助中间代理，做很多的代理，疯狂切换去发送请求，这样就可以很好的获取服务器的数据，还有一些投票也是用来代理

cancelToken:是对ajax请求做一个取消的设置

signal:

decompress:对响应结果是不是做一个解压，在nodejs中设置，在ajax中没法做解压

7.默认配置：是对重复设置的默认编写

```
//默认配置
axios.defaults.method = 'GET';//设置默认的请求类型为 GET
axios.defaults.baseURL = 'http://localhost:3000';//设置基础 URL
```

axios.defaults.params = {id:100};//默认url后面会加id=100参数
axios.defaults.timeout = 3000;//三秒之后取消请求

```
btns[0].onclick = function(){
        axios({
            url: '/posts'
        }).then(response => {
            console.log(response);
        })
    }
```

8.axios创建实例对象发送ajax请求

```
//创建实例对象 /getJoke
const duanzi = axios.create({
  baseURL: 'https://api.apiopen.top',
  timeout: 2000
});
```

```
const onather = axios.create({
    baseURL: 'https://b.com',
    timeout: 2000
});
//这里  duanzi 与 axios 对象的功能几近是一样的
// duanzi({
//     url: '/getJoke',
// }).then(response => {
//     console.log(response);
// });

duanzi.get('/getJoke').then(response => {
    console.log(response.data)
})
```

9.axios拦截器

1.axios拦截器

```
// Promise
// 设置请求拦截器  config 配置对象
axios.interceptors.request.use(function (config) {
  console.log('请求拦截器 成功 - 1号');
  //修改 config 中的参数
  config.params = {a:100};
```

```
    return config;
}, function (error) {
    console.log('请求拦截器 失败 - 1号');
    return Promise.reject(error);
});

axios.interceptors.request.use(function (config) {
    console.log('请求拦截器 成功 - 2号');
    //修改 config 中的参数
    config.timeout = 2000;
    return config;
}, function (error) {
    console.log('请求拦截器 失败 - 2号');
    return Promise.reject(error);
});

// 设置响应拦截器
axios.interceptors.response.use(function (response) {
    console.log('响应拦截器 成功 1号');
    return response.data;
    // return response;
}, function (error) {
    console.log('响应拦截器 失败 1号')
    return Promise.reject(error);
});

axios.interceptors.response.use(function (response) {
    console.log('响应拦截器 成功 2号')
    return response;
}, function (error) {
    console.log('响应拦截器 失败 2号')
    return Promise.reject(error);
});

//发送请求
axios({
    method: 'GET',
    url: 'http://localhost:3000/posts'
}).then(response => {
    console.log('自定义回调处理成功的结果');
    console.log(response);
});
```

2.axios拦截器工作原理
3.模拟实现axios拦截器功能

10.axios取消请求
1.axios取消请求
2.axios取消请求工作原理
3.模拟实现axios取消请求功能
11.axios文件结构说明
12.axios创建过程和模拟实现
1.axios创建过程
2.axios对象创建过程模拟实现
13.axios发送请求过程详解和模拟实现axios发送请求
1.axios发送请求过程详解
2.模拟实现axios发送请求
14.axios源码分析总结