1. 文档

```
https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest
```

2. 理解

```
使用XMLHttpRequest (XHR)对象可以与服务器交互，也就是发送ajax请求
前端可以获取到数据，而无需让整个的页面刷新。
这使得Web页面可以只更新页面的局部，而不影响用户的操作。
```

3. 区别一般的HTTP请求与ajax请求

```
ajax请求是一种特别的http请求
对服务器端来说，没有任何区别，区别在浏览器端
浏览器端发请求：只有XHR或fetch发出的才是ajax请求，其它所有的都是非ajax请求
浏览器端接收到响应
    一般请求：浏览器一般会直接显示响应体数据，也就是我们常说的刷新/跳转页面
    ajax请求：浏览器不会对界面进行任何更新操作，只是调用监视的回调函数并传入响应相关数据
```

4. 使用语法

```
XMLHttpRequest(): 创建XHR对象的构造函数
status: 响应状态码值，比如200，404
statusText: 响应状态文本
readyState: 标识请求状态的只读属性
    0: 初始
    1: open()之后
    2: send()之后
    3: 请求中
    4: 请求完成
onreadystatechange: 绑定readyState改变的监听
responseType: 指定响应数据类型，如果是'json'，得到响应后自动解析响应体数据
response: 响应体数据，类型取决于responseType的指定
timeout: 指定请求超时时间，默认为0代表没有限制
ontimeout: 绑定超时的监听
onerror: 绑定请求网络错误的监听
open(): 初始化一个请求，参数为: (method, url[, async])
send(data): 发送请求
abort(): 中断请求
getResponseHeader(name): 获取指定名称的响应头值
getAllResponseHeaders(): 获取所有响应头组成的字符串
setRequestHeader(name, value): 设置请求头
```

5. XHR的简单封装

1). 特点

```
函数的返回值为promise，成功的结果为response，异常的结果为error
能处理多种类型的请求：GET/POST/PUT/DELETE
函数的参数为一个配置对象
响应json数据自动解析为js
```

2) 编码实现

```
/*
使用XHR封装发送ajax请求的通用函数
  返回值：promise
  参数为配置对象
    url：请求地址
    params：包含所有query请求参数的对象
    data：包含所有请求体参数数据的对象
    method：为请求方式
*/
function axios({url, params={}, data={}, method='GET'}) {
  // 返回一个promise对象
  return new Promise((resolve, reject) => {
    // 创建一个XHR对象
    const request = new XMLHttpRequest()

    // 根据params拼接query参数
    let queryStr = Object.keys(params).reduce((pre, key) => {
      pre += `&${key}=${params[key]}`
      return pre
    }, '')
    if (queryStr.length>0) {
      queryStr = queryStr.substring(1)
      url += '?' + queryStr
    }
    // 请求方式转换为大写
    method = method.toUpperCase()

    // 初始化一个异步请求(还没发请求)
    request.open(method, url, true)
    // 绑定请求状态改变的监听
    request.onreadystatechange = function () {
      // 如果状态值不为4，直接结束(请求还没有结束)
      if (request.readyState !== 4) {
        return
      }
      // 如果响应码在200~~299之间，说明请求都是成功的
      if (request.status>=200 && request.status<300) {
        // 准备响应数据对象
        const responseData = {
          data: request.response,
          status: request.status,
          statusText: request.statusText
        }
```

```
        // 指定promise成功及结果值
        resolve(responseData)
      } else { // 请求失败了
        // 指定promise失败及结果值
        const error = new Error('request error staus '+ request.status)
        reject(error)
      }
    }

    // 指定响应数据格式为json ==> 内部就是自动解析好
    request.responseType = 'json'

    // 如果是post/put请求
    if (method==='POST' || method==='PUT') {
      // 设置请求头: 使请求体参数以json形式传递
      request.setRequestHeader('Content-Type', 'application/json;charset=utf-8')
      // 包含所有请求参数的对象转换为json格式
      const dataJson = JSON.stringify(data)
      // 发送请求, 指定请求体数据
      request.send(dataJson)
    } else {// GET/DELETE请求
      // 发送请求
      request.send(null)
    }
  })
}
```

3). 测试

```
function testGet() {
  axios({
    url: 'http://localhost:3000/comments',
    // url: 'http://localhost:3000/comments2',
    params: {id: 5, body: 'aaaa'},
  }).then(response => {
    console.log('get success', response.data, response)
  }).catch(error => {
    alert(error.message)
  })
}

function testPost() {
  axios({
    url: 'http://localhost:3000/comments',
    // url: 'http://localhost:3000/comments2',
    method: 'POST',
    data: { body: 'aaaa', postId: 2 }
  }).then(response => {
    console.log('post success', response.data, response)
  }).catch(error => {
    alert(error.message)
  })
```

```
}

function testPut() {
  axios({
    // url: 'http://localhost:3000/comments/6',
    url: 'http://localhost:3000/comments/3',
    method: 'put',
    data: {body: 'abcdefg', "postId": 2}
  }).then(response => {
    console.log('put success', response.data, response)
  }).catch(error => {
    alert(error.message)
  })
}

function testDelete() {
  axios({
    url: 'http://localhost:3000/comments/6',
    method: 'delete',
  }).then(response => {
    console.log('delete success', response.data, response)
  }).catch(error => {
    alert(error.message)
  })
}
```