11. AJAX

什么是AJAX?

AJAX(**Async JavaScript and XML): 实现数据请求 + 客户端渲染**

- AJAX解决了网页异步刷新的问题

AJAX核心：

基于XMLHttpRequest创建HTTP请求

创建xhr实例

打开一个URL地址「发送请求前的一些配置信息」

1. method 请求方式: **GET**(get/delete/head/options...) / **POST**(post/put/patch...)

GET和POST在官方定义中是没有明确的区别的，但是浏览器或者开发的时候，都有一套约定俗成的规范：

- GET请求传递给服务器的信息，除了请求头传递以外，要求基于URL问号传参传递给服务器

  xhr.open('GET', './1.json?lx=1&name=xxx')

- POST请求要求传递给服务器的信息，是基于请求主体传递

  xhr.send('lx=1&name=xxx')

GET与POST的区别:

1. **GET传递的信息不如POST多**，因为URL有长度限制「IE→2KB」，超过这个长度的信息会被自动截掉，这样导致传递内容过多，最后服务器收到的信息是不完整的, **POST理论上是没有限制的**，但是传递的东西越多，速度越慢，可能导致浏览器报传输超时的错误，所以实际上我们会自己手动做限制.
2. **GET会产生缓存「浏览器默认产生的，不可控的缓存」**：两次及以上，请求相同的API接口，并且传递的参数也一样，浏览器可能会把第一次请求的信息直接返回，而不是从服务器获取最新的信息！！

   xhr.open('GET', './1.json?lx=1&name=xxx&_'+Math.random())

   在请求URL的**末尾设置随机数**，以此来清除GET缓存的副作用

3. POST相对于GET来讲更安全一些：**GET传递的信息是基于URL末尾拼接**，这个随便做一些劫持或者修改，都可以直接改了，而**POST请求主体信息的劫持**，没那么好做！！但是“互联网面前，人人都在裸奔”！！所以不管什么方式，只要涉及安全的信息，都需要手动加密「因为默认所有的信息传输都是明文的」！！

监听请求的过程，在不同的阶段做不同的处理「包含获取服务器的响应信息」

1. 监听请求的过程，在不同的阶段做不同的处理「包含获取服务器的响应信息」

ajax状态 xhr.readyState:

- 0 **UNSENT** 未发送
- 1 **OPENED** 请求发送
- 2 **HEADERS_RECEIVED** 响应头信息已经返回
- 3 **LOADING** 响应主体信息正在处理
- 4 **DONE** 响应主体信息已经返回

1. HTTP状态码 xhr.status/xhr.statusText
   - **200** OK
   - **202** Accepted ：服务器已接受请求，但尚未处理（异步）
   - **204** No Content：服务器成功处理了请求，但不需要返回任何实体内容
   - **206** Partial Content：服务器已经成功处理了部分 GET 请求（断点续传 Range/If-Range/Content-Range/Content-Type:”multipart/byteranges”/Content-Length....）

- **301** Moved Permanently 永久转移 「域名迁移」
- **302** Move Temporarily 临时转移 「负载均衡」
- **304** Not Modified
- **305** Use Proxy

- **400** Bad Request : 请求参数有误
- **401** Unauthorized：权限（Authorization）
- **403** Forbidden 服务器拒绝执行「为啥可能会已响应主体返回」
- **404** Not Found 地址错误
- **405** Method Not Allowed 请求方式不被允许
- **408** Request Timeout 请求超时

- **500** Internal Server Error 未知服务器错误
- **503** Service Unavailable 超负荷
- **505** HTTP Version Not Supported
- ……

获取响应主体信息 xhr.response/responseText/responseXML...

服务器返回的响应主体信息的格式

+ 字符串「一般是JSON字符串」 「最常用」
+ XML格式数据
+ 文件流格式数据「buffer/二进制...」
+ ...

+ 获取响应头信息 xhr.getResponseHeader/getAllResponseHeaders

发送请求「send中传递的信息，就是设置的请求主体信息」

基于请求主体传递给服务器的数据格式是有要求的「Postman接口测试工具」

- form-data 主要应用于文件的上传或者表单数据提交

```
xhr.setRequestHeader('Content-Type', 'multipart/form-data');
let fd = new FormData;
fd.append('lx', 0);
fd.append('name', 'xxx');
xhr.send(fd);
```

- x-www-form-urlencoded格式的字符串

  格式：“lx=1&name=xxx” 「常用」

  Qs库：$npm i qs

  Qs.stringify/parse:实现对象和urlencoded格式字符串之间的转换

```
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
xhr.send(Qs.stringify({
lx: 0,
name: 'xxx'
}));
```

- raw字符串格式

  普通字符串 → text/plain

  JSON字符串 → application/json ⇒ JSON.stringify/parse 「常用」

  XML格式字符串 → application/xml

  ……

- binary进制数据文件「buffer/二进制...」

  一般也应用于文件上传

  图片 → image/jpeg

  EXCEL → application/vnd.openxmlformats-officedocument.spreadsheetml.sheet

  ...

- GraphQL

axios

http://www.axios-js.com/zh-cn/docs/

基于Promise封装的ajax库，核心XMLHttpRequest

- axios 函数(对象)
  1. CancelToken 用于取消ajax请求
  2. all 基于promise.all实现ajax的并行，当所有的ajax请求都成功，整体才会返回一个成功
  3. promise实例
  4. spread 解析出基于all返回的结果

5. create 创建一个新的实例，来做单独的全局配置
6. defaults 全局默认配置
7. get/delete/head/options 发送对应方式的请求
8. post/put/patch 发送对应方式的请求
9. request 发送请求
10. interceptors
11. request 请求拦截器
12. response 响应拦截器

```
// 基于axios发送请求，最后返回的都是promise实例
let formData = new FormData()
formData.append('file', 'xxx')
formData.append('size', '1024')

axios({
    // baseURL+url：最终请求的地址
    baseURL: 'http://127.0.0.1:8888',
    url: '/user/list',
    method: 'post',
    // params:基于URL末尾拼接参数的方式，把params对象一项项传递给服务器
    params: {
        lx: 0,
        from: 'wx'
    },
    // 内部有的方法，params对象最后可以拼接到URL的末尾，内部就是基于这个方法处理的
    paramsSerializer: function (params) {
        return Qs.stringify(params, {
            arrayFormat: 'brackets'
        });
    },
    // data:只针对POST系列请求，设置请求主体传递的信息，默认会把对象变为 application/json 字符串传递给服务器
    data: {
        file: 'xxx',
        size: 1024
    },
    // 在POST请求下，把请求主体信息发送给服务器之前，对请求主体信息进行处理
    transformRequest: function (data) {
        return Qs.stringify(data);
    }

    // 值：FormData\binary\raw...
    data: formData,
    transformRequest: function (data) {
        if (_.isPlainObject(data)) {
            // application/json && x-www-form-urlencoded
            return Qs.stringify(data);
        }
        return data;
    },
    // 设置请求头信息
    headers: {
        // 所有请求通用
        'Content-Type': 'multipart/form-data',
        common: {
            'X-Token': 'xxx'
        },
        // 可以只针对某种请求设置
        post: {
            'lx': 1
```

```
        },
        get: {
            'lx': 0
        }
    },
    // 零散配置信息
    timeout: 0,
    withCredentials: true,
    // 预设服务器返回的数据格式：不论服务器返回啥格式，内部会转换为我们预设的格式 json/arraybuffer/blob/document/text...
    responseType: 'json',
    // 监听上传/下载进度
    onUploadProgress: function (progressEvent) {},
    onDownloadProgress: function () {},
    // 内部规定，HTTP状态码为多少，算是请求成功，返回成功Promise，否则返回失败的!!
    validateStatus: function (status) {
        return status >= 200 && status < 300;
    }
}); */

// axios([config])
// axios.request([config])
// axios.get/head/delete/options([url],[config])
// axios.post/put/patch([url],[data],[config])

axios
  .get('http://127.0.0.1:8888/user/list2', {
    params: {
      lx: 1,
      from: 'wx'
    }
  })
  .then(response => {
    // 服务器返回的状态码和validateStatus指定的匹配条件一致（READY-STATE===4）
    // config 设定的配置项
    // headers 响应头信息「对象」
    // request 原生的XHR对象
    // status/statusText 状态码和状态码的描述
    // data 响应主体信息
    console.log('成功', response)
    return response.data
  })
  .then(data => {
    // 获取响应主体信息，完成对应的业务逻辑
    // ...
  })
  .catch(reason => {
    // 服务器返回的状态码不与validateStatus条件一致「最起码服务器有返回」
    // 压根服务器啥都没返回「例如：断网」
    // 当前请求超时或者被取消
    //   + config
    //   + request
    //   + toJSON
    //   + message 错误信息
    //   + response 如果是网络层失败，是没有response，如果只是axios层失败，是存在response
    //   + isAxiosError 是否为axios层面失败
    console.dir(reason)
  })
```

请求成功和失败

1. 网络层失败 请求没有发送成功，或者没有任何的响应「没有完成一个HTTP事物」
2. AXIOS层失败

   + 服务器一定有返回

    + 只不过状态码和validateStatus不一致
    + 超时或者取消请求
3. 业务层失败
    + 一般都是服务器根据业务需求，基于类似于code等标志，来区分不同的业务形态和结果