AJAX

- `ajax` 全名 `async javascript and XML`
- 是前后台交互的能力
- 也就是我们客户端给服务端发送消息的工具，以及接受响应的工具
- 是一个 **默认异步** 执行机制的功能

AJAX 的优势

1. 不需要插件的支持，原生 js 就可以使用
2. 用户体验好（不需要刷新页面就可以更新数据）
3. 减轻服务端和带宽的负担
4. 缺点： 搜索引擎的支持度不够，因为数据都不在页面上，搜索引擎搜索不到

AJAX 的使用

- 在 js 中有内置的构造函数来创建 ajax 对象
- 创建 ajax 对象以后，我们就使用 ajax 对象的方法去发送请求和接受响应

创建一个 ajax 对象

```
// IE9及以上
const xhr = new XMLHttpRequest()

// IE9以下
const xhr = new ActiveXObject('Mricosoft.XMLHTTP')
```

- 上面就是有了一个 ajax 对象
- 我们就可以使用这个 `xhr` 对象来发送 ajax 请求了

配置链接信息

```
const xhr = new XMLHttpRequest()

// xhr 对象中的 open 方法是来配置请求信息的
// 第一个参数是本次请求的请求方式 get / post / put / ...
// 第二个参数是本次请求的 url
// 第三个参数是本次请求是否异步，默认 true 表示异步，false 表示同步
// xhr.open('请求方式', '请求地址', 是否异步)
```

```
xhr.open('get', './data.php')
```

- 上面的代码执行完毕以后，本次请求的基本配置信息就写完了

发送请求

```
const xhr = new XMLHttpRequest()
xhr.open('get', './data.php')

// 使用 xhr 对象中的 send 方法来发送请求
xhr.send()
```

- 上面代码是把配置好信息的 ajax 对象发送到服务端

一个基本的 ajax 请求

- 一个最基本的 ajax 请求就是上面三步
- 但是光有上面的三个步骤，我们确实能把请求发送的到服务端
- 如果服务端正常的话，响应也能回到客户端
- 但是我们拿不到响应
- 如果想拿到响应，我们有两个前提条件
  i. 本次 HTTP 请求是成功的，也就是我们之前说的 http 状态码为 200 ~ 299
  ii. ajax 对象也有自己的状态码，用来表示本次 ajax 请求中各个阶段

ajax 状态码

- ajax 状态码 - `xhr.readyState`
- 是用来表示一个 ajax 请求的全部过程中的某一个状态
  - `readyState === 0`： 表示未初始化完成，也就是 `open` 方法还没有执行
  - `readyState === 1`： 表示配置信息已经完成，也就是执行完 `open` 之后
  - `readyState === 2`： 表示 `send` 方法已经执行完成
  - `readyState === 3`： 表示正在解析响应内容
  - `readyState === 4`： 表示响应内容已经解析完毕，可以在客户端使用了
- 这个时候我们就会发现，当一个 ajax 请求的全部过程中，只有当 `readyState === 4` 的时候，我们才可以正常使用服务端给我们的数据
- 所以，配合 http 状态码为 200 ~ 299
  - 一个 ajax 对象中有一个成员叫做 `xhr.status`
  - 这个成员就是记录本次请求的 http 状态码的
- 两个条件都满足的时候，才是本次请求正常完成

readyStateChange

- 在 ajax 对象中有一个事件，叫做 `readyStateChange` 事件
- 这个事件是专门用来监听 ajax 对象的 `readyState` 值改变的的行为
- 也就是说只要 `readyState` 的值发生变化了，那么就会触发该事件
- 所以我们就在这个事件中来监听 ajax 的 `readyState` 是不是到 4 了

```
const xhr = new XMLHttpRequest()
xhr.open('get', './data.php')

xhr.send()

xhr.onreadyStateChange = function () {
 // 每次 readyState 改变的时候都会触发该事件
 // 我们就在这里判断 readyState 的值是不是到 4
 // 并且 http 的状态码是不是 200 ~ 299
 if (xhr.readyState === 4 && /^2\d{2|$/.test(xhr.status)) {
 // 这里表示验证通过
 // 我们就可以获取服务端给我们响应的内容了
  }
}
```

responseText

- ajax 对象中的 `responseText` 成员
- 就是用来记录服务端给我们的响应体内容的
- 所以我们就用这个成员来获取响应体内容就可以

```
const xhr = new XMLHttpRequest()
xhr.open('get', './data.php')

xhr.send()

xhr.onreadyStateChange = function () {
 if (xhr.readyState === 4 && /^2\d{2|$/.test(xhr.status)) {
 // 我们在这里直接打印 xhr.responseText 来查看服务端给我们返回的内容
 console.log(xhr.responseText)
 }
}
```

使用 ajax 发送请求时携带参数

- 我们使用 ajax 发送请求也是可以携带参数的
- 参数就是和后台交互的时候给他的一些信息
- 但是携带参数 get 和 post 两个方式还是有区别的

发送一个带有参数的 get 请求

- get 请求的参数就直接在 url 后面进行拼接就可以

```
const xhr = new XMLHttpRequest()
// 直接在地址后面加一个 ?，然后以 key=value 的形式传递
// 两个数据之间以 & 分割
xhr.open('get', './data.php?a=100&b=200')

xhr.send()
```

  - 这样服务端就能接受到两个参数
  - 一个是 a，值是 100
  - 一个是 b，值是 200

发送一个带有参数的 post 请求

- post 请求的参数是携带在请求体中的，所以不需要再 url 后面拼接

```
const xhr = new XMLHttpRequest()
xhr.open('get', './data.php')

// 如果是用 ajax 对象发送 post 请求，必须要先设置一下请求头中的 content-type
// 告诉一下服务端我给你的是一个什么样子的数据格式
xhr.setRequestHeader('content-type', 'application/x-www-form-urlencoded')

// 请求体直接再 send 的时候写在 () 里面就行
// 不需要问号，直接就是 'key=value&key=value' 的形式
xhr.send('a=100&b=200')
```

  - `application/x-www-form-urlencoded` 表示的数据格式就是 `key=value&key=value`

同源策略

- **同源策略是由浏览器给的**
  浏览器不允许我们向别人发送请求，只能向自己的服务器发送请求

- 浏览器不允许我们向别人发送请求，只能向自己的服务器发送请求
- 当我们想向别人的服务器发送请求的时候，就会被浏览器阻止了
- 什么是 “别人的服务器” 呢？
  - **当 请求协议/域名/端口号 有任意一个不同的时候，那么就算是别人的服务器**
  - 这个时候就会触发同源策略
- 我们管触发了 **同源策略** 的请求叫做跨域请求

实现一个跨域请求

- 有的时候我们是需要实现跨域请求的
- 我们需要多个服务器给一个页面提供数据
- 那么这个时候我们就要想办法解决跨域问题

JSONP

- `jsonp` 是我们实现跨域请求的手段，是把我们之前的东西组合在一起使用的技术手段而已
- 利用的是 script 标签来实现

script 标签的本质

- 浏览器给我们提供了一个 `script` 标签
- 它的本质就是请求一个外部资源，是不受到同源策略的影响的
- 同时 `script` 标签的 `src` 属性，也是一种请求，也能被服务器接收到
- 并且：
  - **script标签的src属性请求回来的东西是一个字符串，浏览器会把这个字符串当作 js 代码来执行**
- 所以我们就可以利用这个 `script` 标签的 `src` 属性来进行跨域请求了

配置代理（了解）

- 代理，分成两种，正向代理和反向代理

正向代理

- 有一个客户端需要向一个非同源的服务器B发送请求
- 我们搭建一个和客户端同源的服务器A
- 当客户端发送请求的时候，由服务器A来接受
- 再由服务器A向服务器B发送请求，因为 **同源策略是由浏览器给的**，服务器之间没有
- 服务器B接受到请求以后，会处理请求，并把响应返回给服务器A
- 再由服务器A把响应给到客户端就可以了

- 再由服务器把响应给到客户端就可以了
- 我们就可以用这个方式来进行跨域请求了

反向代理

- 反向代理一般是用来做负载均衡的
- 当我请求一个服务器的时候，其实请求的是服务器端设置的代理服务器
- 由代理服务器把若干大量的请求分发给不同的服务器进行处理
- 再由服务器把响应给到代理服务器
- 代理服务器返回给客户端