什么是 AJAX？

AJAX = 异步 JavaScript 和 XML。
AJAX = Asynchronous JavaScript and XML（异步的 JavaScript 和 XML）
AJAX 是一种用于创建快速动态网页的技术。
通过在后台与服务器进行少量数据交换，AJAX 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

传统的网页（不使用 AJAX）如果需要更新内容，必需重载整个网页面。

什么是 XMLHttpRequest？

Javascript 本身并未具备向服务器发送请求的能力，要么使用 XMLHttpRequest 对象发送请求。XMLHttpRequest 是一个浏览器接口，使得 Javascript 可以进行 HTTP(S)通信，这就是我们熟悉的 AJAX。

在 Ajax 应用程序中，XmlHttpRequest 对象负责将用户信息以异步通信地发送到服务器端，并接收服务器返回的响应信息和数据。

XMLHttpRequest 提供了一系列的属性和方法，来向服务器发送异步的 http 请求；在服务器处理用户请求的过程中，XMLHttpRequest 通过属性的状态值来实时反映 http 请求所处的状态，并根据这些状态指示 Javascript 做相应的处理；当服务器顺利完成响应用户行为的动作、并将响应数据返回时，XMLHttpRequest 提供的 response 系列方法，可以将这些响应数据以文本、XML Document 对象、Ado Stream 对象或者 unsigned byte 数组的方式组装起来，提供给 Javascript 处理。

使用 XMLHttpRequest (XHR)对象可以与服务器交互，可以从服务器获取数据，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。

早期，各个浏览器的实现都不同，HTML5 之后，W3C 进行了统一。目前所有现代浏览器都支持。

var xhr = new XMLHttpRequest();

* xhr.readyState：XMLHttpRequest 对象的状态，等于 4 表示数据已经接收完毕。
* xhr.status：服务器返回的状态码，等于 200 表示一切正常。
* xhr.responseText：服务器返回的文本数据
* xhr.responseXML：服务器返回的 XML 格式的数据
* xhr.statusText：服务器返回的状态文本。

新版本的 XMLHttpRequest 对象，针对老版本的缺点，做出了大幅改进。

* 可以设置 HTTP 请求的时限,timeout。
* 可以使用 FormData 对象管理表单数据。
* 可以上传文件。
* 可以请求不同域名下的数据（跨域请求）。
* 可以获取服务器端的二进制数据。
* 可以获得数据传输的进度信息
  *

```
var xhr = new XMLHttpRequest();
var form = document.getElementById('myform');
      var formData = new FormData(form);
      formData.append('secret', ‘123456');
     formData.append('file', ‘……’);
xhr.send(formData);
```

除了 IE 8 和 IE 9，主流浏览器都支持 CORS，IE 10 也将支持这个功能。服务器端的设置

实现 AJAX 的基本步骤

1. 创建 XMLHttpRequest 对象
2. 创建 HTTP 请求，链接服务器
3. 设置请求头（setRequestHeader）
4. 发送请求
5. 设置响应 HTTP 请求状态变化的函数

1. 创建 XMLHttpRequest 对象

2. 创建 http 请求，链接服务器

创建了 XMLHttpRequest 对象之后,必须为 XMLHttpRequest 对象创建 HTTP 请求,用于说明 XMLHttpRequest 对象要从哪里获取数据

创建 HTTP 请求可以使用 XMLHttpRequest 对象的 open()方法,其语法代码如下所示:
xhr.open(method,URL,flag)
method：该参数用于指定 HTTP 的请求方法，一共有 get、post、head、put、delete 五种方法，常用的方法为 get 和 post。大小写不敏感
URL：该参数用于指定 HTTP 请求的 URL 地址，可以是绝对 URL，也可以是相对 URL。
flag：该参数为可选参数，参数值为布尔型。该参数用于指定是否使用异步方式。true 表示异步方式、false 表示同步方式，默认为 true。

xml.open('get','http://localhost:3000/info',true);

3. 设置请求头

XMLHttpRequest.setRequestHeader() 是设置 HTTP 请求头部的方法。此方法必须在 open() 方法和 send() 之间调用。如果多次对同一个请求头赋值，只会生成一个合并了多个值的请求头。
语法
myReq.setRequestHeader(header, value);
header
属性的名称。
value
属性的值

以及 setRequestHeader 设置请求头，比如
// xml.setRequestHeader('Authorization','5ddd459c-62be-45c8-91e1-988d2dea3e4e');
xml.setRequestHeader('Content-Type','application/json;charset=UTF-8');

4. 发送请求

如果是 post 请求，需要有参数，如果是 get,请求的参数要拼接在 url 上
XMLHttpRequest.send(data)

5. 设置响应 HTTP 请求状态变化的函数

创建完 HTTP 请求之后，应该就可以将 HTTP 请求发送给 Web 服务器了。然而，发送 HTTP 请求的目的是为了接收从服务器中返回的数据。从创建 XMLHttpRequest 对象开始，到发送数据、接收数据、XMLHttpRequest 对象一共会经历以下 5 种状态。

readyState 说明 XMLHttpRequest 对象请求的状态;(有 5 个状态分别是
0 表示没有初始化；
1 表示读取中
2 表示已读取
3 交互中(接受中)

4 完成

创建 XMLHttpRequest 对象 -> 指定发送地址及发送方法 -> 指定状态变化处理方法 -> 发送请求，请求发送后状态变化了就会自动调用指定的处理方法。

```
xhr.onreadystatechange = getDataFromServer;

function getDataFromServer(){
        if(xhr.readyState==4 && xhr.status==200){
         //设置获取数据的语句
          var response = JSON.parse(xml.responseText);
           if(response.success){
              var data = response.data;
           }
      }
}
```

xhr.status ： 服务器返回的 http 状态码

```
switch (xhr.status) {
      case 400:
        console.log('错误请求')
       break;
      case 401:
        console.log('未授权，请重新登录')
       break;
      case 403:
       console.log('拒绝访问')
       break;
      case 404:
       console.log('请求错误,未找到该资源')
       break;
      case 405:
       console.log('请求方法未允许')
       break;
      case 408:
       console.log('请求超时')
       break;
      case 500:
       console.log('服务器内部错误')
       break;
      case 501:
       console.log('网络未实现')
```

```
        break;
      case 502:
        console.log('网络错误')
        break;
      case 503:
        console.log('服务不可用')
        break;
      case 504:
        console.log('网络超时')
        break;
      case 505:
        console.log('http 版本不支持该请求')
        break;
      default:
        console.log(`连接错误`)
    }
```

JQUERY AJAX

```
$.ajax({
    type: httpMethod,
    cache:false,
    async:false,
    contentType: "application/json; charset=utf-8",
    dataType: "json",//返回值类型
    url: path+url,
    data:jsonData,
    success: function(data){
    },
    error : function(xhr, ts, et) {
    }
});
```

contentType: 发送信息至服务器时内容编码类型，简单说告诉服务器请求类型的数据，我要发什么类型的数据

dataType：告诉服务器，我想得到什么类型的数据，除了常见的 json、XML，还可以指定 html、jsonp、script 或者 text

跨域处理

1. 什么是跨域？

跨域是指从一个域名的网页去请求另一个域名的资源。比如从 www.baidu.com 页面去请求 www.google.com 的资源。但是一般情况下不能这么做，它是由浏览器的同源策略造成的，是浏览器对 JavaScript 施加的安全限制。跨域的严格一点的定义是：只要协议，域名，端口有任何一个的不同，就被当作是跨域

概念：只要协议、域名、端口有任何一个不同，都被当作是不同的域。
通过 XMLHttpRequest 对象实现（IE10 以下不支持）

2. 为什么浏览器要限制跨域访问呢？

原因就是安全问题：如果一个网页可以随意地访问另外一个网站的资源，那么就有可能在客户完全不知情的情况下出现安全问题

3. 为什么要跨域？

既然有安全问题，那为什么又要跨域呢？ 有时公司内部有多个不同的子域，比如一个是 location.company.com ,而应用是放在 app.company.com , 这时想从 app.company.com 去访问 location.company.com 的资源就属于跨域。

4. 解决跨域问题的方法

- cors
- jsonp
- postMessage

- document.domain
- window.name
- http-proxy
- nginx
- websocket
- Webpack 里如何解决跨域

1）跨域资源共享（CORS）

CORS（Cross-Origin Resource Sharing）跨域资源共享，定义了必须在访问跨域资源时，浏览器与服务器应该如何沟通。CORS 背后的基本思想就是**使用自定义的 HTTP 头部让浏览器与服务器进行沟通，从而决定请求或响应是应该成功还是失败**。
服务器端对于 CORS 的支持，主要就是通过设置 Access-Control-Allow-Origin 来进行的。如果浏览器检测到相应的设置，就可以允许 Ajax 进行跨域的访问。

只需要在后台中加上响应头来允许域请求！在被请求的 Response header 中加入以下设置，就可以实现跨域访问了！

```
// 允许跨域访问的域名：若有端口需写全（协议+域名+端口），若没有端口末尾不用加'/'
response.setHeader("Access-Control-Allow-Origin", "http://www.domain1.com");

// 允许前端带认证 cookie：启用此项后，上面的域名不能为'*'，必须指定具体的域名，否则浏览器会提示
response.setHeader("Access-Control-Allow-Credentials", "true");

// 提示 OPTIONS 预检时，后端需要设置的两个常用自定义头
response.setHeader("Access-Control-Allow-Headers", "Content-Type,X-Requested-With");
```

2）通过 jsonp 跨域

JSONP 的原理：通过 script 标签引入一个 js 文件，这个 js 文件载入成功后会执行我们在 url 参数中指定的函数，并且会把我们需要的 json 数据作为参数传入。

为了便于客户端使用数据，逐渐形成了一种非正式传输协议，人们把它称作 JSONP，该协议的一个要点就是允许用户传递一个 callback 参数给服务端，然后服务端返回数据时会将

这个 callback 参数作为函数名来包裹住 JSON 数据，这样客户端就可以随意定制自己的函数来自动处理返回数据了

所以 jsonp 是需要服务器端的页面进行相应的配合的。（即用 JavaScript 动态加载一个 script 文件，同时定义一个 callback 函数给 script 执行而已。）

启动服务，用 D:\ruanmou\NodeJS\myExpress

服务器端：

```
var express = require('express');
var app = express();

app.get('/callback=:cbk',function(req,res){
    var bk = req.params.cbk
    var vt = {name:'Tim',age:28,id:bk};
    res.send(bk+'('+JSON.stringify(vt)+')');
})
app.listen(3300);
```

客户端：

原生方式

```
    <script type="text/javascript">
    function test(jsondata){
        //处理获得的 json 数据
    }
</script>
<script src="http://localhost:3300/callback=test"> </script>
```

jquery 方式：

```
    $.getJSON( ‘http://localhost:3300/callback=?,function(jsondata)'){
        //处理获得的 json 数据
    });
$.ajax({
    url: 'http://localhost:3300',
    type: 'get',
    dataType: 'jsonp',  // 请求方式为 jsonp
    jsonpCallback: "callback",    // 自定义回调函数名
    data: {}
});
```

JSONP 的优缺点

JSONP 的优点是：它不像 XMLHttpRequest 对象实现的 Ajax 请求那样受到同源策略的限制；它的兼容性更好，在更加古老的浏览器中都可以运行，不需要 XMLHttpRequest 或 ActiveX 的支持；并且在请求完毕后可以通过调用 callback 的方式回传结果。

JSONP 的缺点则是：它只支持 GET 请求而不支持 POST 等其它类型的 HTTP 请求；
它只支持跨域 HTTP 请求这种情况，不能解决不同域的两个页面之间如何进行 JavaScript 调用的问题。

3）通过 ngnix 代理跨域

安装 ngnix
1.下载地址 http://nginx.org/en/download.html，解压下载的 nginx 文件。
2. 配置部署，编辑 nginx/conf 下的 nginx.conf，
命令为：vim /usr/local/etc/nginx/nginx.conf
3. 启动 nginx。命令窗口 cd 进入 nginx 安装目录，输入 start nginx 启动 nginx

然后通过浏览器访问 http://127.0.0.1:8088/访问 或者 http://域名:8088/访问

4. nginx 停止命令：nginx -s quit
   nginx 重启命令：nginx -s reload 或者 直接 nginx 启动

mac 电脑系统配置 ngnix
https://www.cnblogs.com/tandaxia/p/8810648.html

1、 nginx 配置解决 iconfont 跨域

浏览器跨域访问 js、css、img 等常规静态资源被同源策略许可，但 iconfont 字体文件(eot|otf|ttf|woff|svg)例外，此时可在 nginx 的静态资源服务器中加入以下配置。

```
location / {
  add_header Access-Control-Allow-Origin *;
}
```

http://localhost:9099/api/info

```
server {
      listen       8001;
      server_name  localhost;
      location / {
         # 静态资源目录
          root   D:/ruanmou/ES5/AJAX/lesson/;
          index  index.html index.htm;
      }
 #凡是 localhost:8001/api 这个样子的，都转发到真正的服务端地址 http://localhost:3000
       location /api {
          # 设置代理服务器的协议和地址
          proxy_pass  http://localhost:3000;
          index  index.html yanshi.html;
       }
    }
```

前端就不用干什么事情了，除了写接口，也没后端什么事情了

4）通过 vue 框架，以及 webpack 的跨域

利用 webpack-dev-server 代理接口跨域。在开发环境下，由于 vue 渲染服务和接口代理服务都是 webpack-dev-server 同一个，所以页面与代理接口之间不再跨域，无须设置 headers 跨域信息了。

```
 webpackdevServer={
       contentBase: buildPath,
       historyApiFallback: true,   //任意的 404 响应都可能需要被替代为 index.html
       host: '0.0.0.0',
       port: 3009,
       proxy: {
           // 页面中调接口的形式：http://localhost:3009/api/info
          "/api": "http://localhost:3000"
          // 代理到后端的服务地址，会拦截所有以 api 开头的请求地址
       }
    };
```

5）WebSocket 协议跨域

WebSocket protocol 是 HTML5 一种新的协议。它实现了浏览器与服务器全双工通信，同时允许跨域通讯，是 server push 技术的一种很好的实现。

可以在支持 HTML5 的浏览器版本中使用 WebSocket 进行数据通信，常见的案例是使用 WebSocket 进行实时数据刷新。

原生 WebSocket API 使用起来不太方便，我们使用 Socket.io，它很好地封装了 webSocket 接口，提供了更简单、灵活的接口，也对不支持 webSocket 的浏览器提供了向下兼容。
<script src="https://cdn.bootcss.com/socket.io/2.2.0/socket.io.js"></script>
WebSocket 是高级 api，不兼容，但是可以使用 socket.io 这个库，这个库做了兼容处理

webSocket 本身不存在跨域问题，所以我们可以利用 webSocket 来进行非同源之间的通信。

websocket 如何实现跨域通信?
原理：
在客户端 利用 webSocket 的 API，可以直接 new 一个 socket 实例，然后通过 open 方法内 send 要传输到后台的值，也可以利用 message 方法接收后台传来的数据。
在服务端通过 new WebSocket.Server({port:3003})实例，利用 message 接收数据，利用 send 向客户端发送数据。

服务端页面 socket.js：

```
/*要使用 ws 协议，那么就要装一个 ws 的包 */
 let WebSocket = require("ws");
 let wss = new WebSocket.Server({port:3003});
 wss.on("connection",function(ws){
  //先连接
    ws.on("message",function(data){
    //用 message 来监听客户端发来的消息
    console.log(data);
    ws.send("你好 1,"+data+"！ ");
    })
 })
```

客户端页面 websocket.html：
http://192.168.1.114:8080/websocket.html

```
<!--
 高级 api  不兼容  但是有一个 socket.io 这个库，是兼容的(一般用这个)
 -->
 <script type="text/javascript">
 //ws 协议是 webSocket 自己创造的
  let socket = new WebSocket("ws://localhost:3003");
  socket.onopen = function(){
     socket.send("我叫 laney");
  }
  socket.onmessage = function(e){
      console.log(e.data); //你好,我叫俞华！
  }
 </script>
```

6.node.js 中使用 http-proxy 创建代理服务器

代理，也称网络代理，是一种特殊网络服务，允许一个终端通过代理服务与另一个终端进行非直接的连接，这样利于安全和防止被攻击。
代理服务器，就是代理网络用户去获取网络信息，就是信息的中转，负责转发。
代理又分 正向代理 和 反向代理：
正向代理：帮助局域网内的用户访问外面的服务。
反向代理：帮助外面的用户访问局域网内部的服务。

安装 http-proxy

```
npm install http-proxy --save
```

代理本地服务

```
const http = require('http');
const httpProxy = require('http-proxy');

//创建一个代理服务
const proxy = httpProxy.createProxyServer();

//创建 http 服务器并监听 8888 端口
let server = http.createServer(function (req, res) {
   //将用户的请求转发到本地 9999 端口上
```

```
    proxy.web(req, res, {
        target: 'http://localhost:9999'
    });
    //监听代理服务错误
    proxy.on('error', function (err) {
        console.log(err);
    });
});
server.listen(8888, '0.0.0.0');
```

9999 端口服务代码：

```
const http = require('http');
http.createServer(function (req, res) {
    res.end('port : 9999');
}).listen(9999, '0.0.0.0');
```

当们在本地访问 8888 端口时，proxy 会帮我们把请求代理到 9999 端口服务，然后返回数据。