## 什么是浏览器的同源策略?

所谓同源是指：域名、协议、端口必须都一致。

DOM同源策略：禁止对不同源页面DOM进行操作。这里主要场景是iframe跨域的情况，不同域名的iframe是限制互相访问的。

XmlHttpRequest同源策略：禁止使用XHR对象向不同源的服务器地址发起HTTP请求。 只要协议、域名、端口有任何一个不同，都被当作是不同的域，之间的请求就是跨域操作。

跨域限制主要是为了安全考虑。

**跨域的解决方案?**

由于现在大部分都是采用前后分离的方式开发。经常会和跨域打交道。

Vue解决跨域:

proxyTable:

在webpack的配置文件config的index文件中添加proxyTable配置代理跨域.

proxyTable: {

'/apis': {

target: "https://cnodejs.org", // 需要进行代理跨域主机

secure: true, // 如果是https接口，需要配置这个参数

changeOrigin: true, //是否跨域

pathRewrite: {

'^/apis': '' //请求结束 将url里面 /apis重写成"" }

}

}

}

React解决跨域:

使用中间件http-proxy-middleware:

安装中间件之后,在src下建立setupProxy.js文件,写入配置,好处是可以配置多个代理:

const proxy = require('http-proxy-middleware');

module.exports = function (app) {

app.use(proxy('/apis', {

target: "https://api.juooo.com",

secure: false,

changeOrigin: true,

pathRewrite: {

"^/apis": "/"

}

}));

app.use(proxy('/apm', {

target: "https://m.juooo.com",

secure: false,

changeOrigin: true,

pathRewrite: {

"^/apm": "/"

}

}));

}

JSONP跨域: 动态创建 script 标签，然后利用 src 属性进行跨域.原理是src请求网络资源不受同源策略的影响:

大概的实现就是封装jsonp函数,传入url和回调函数,动态创建script标签请求跨域,跨域成功执行回调函数.

function jsonp({ url, params, callback }) {

return new Promise((resolve, reject) => {

let script = document.createElement('script')

window[callback] = function(data) {

resolve(data)

document.body.removeChild(script)

}

params = { ...params, callback }

let arrs = []

for (let key in params) {

arrs.push(`${key}=${params[key]}`)

}

script.src = `${url}?${arrs.join('&')}`

document.body.appendChild(script)

})

}

jsonp({

url: 'http://localhost:3000/say',

params: { wd: 'Iloveyou' },

callback: 'show'

}).then(data => {

console.log(data)

})

CORS跨域: 后台开启允许跨域.

此外还有location.hash ,postMessage , document.domain等实现跨域.

## TCP连接为什么需要三次握手，两次不可以吗？或者四次、五次可以吗？

TCP握手详解

建立连接协议（三次握手）：

第一次握手：客户端发送syn包（syn=x）的数据包到服务器，并进入SYN\_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=x+1），同时自己也发送一个SYN包（syn=y），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=y+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

3次握手完成两个重要的功能，既要双方都做好发送数据的准备工作(双方都知道彼此已准备好)，也要允许双方就初始序列号进行协商，这个序列号在握手过程中被发送和确认。

如果把三次握手改成仅需要两次握手，死锁是可能发生的。客户端认为连接还未建立成功，将忽略服务端发来的任何数据分组，等待连接确认应答。而服务端在发出的数据超时后，重复发送同样的数据。这样就形成了死锁。

**什么是Ajax? jQuery.ajax,和axios和fetch之间有什么区别?各有什么优缺点?**

Ajax的全称Asynchronous JavaScript + XML(异步JavaScript和XML).Ajax是一种技术方案，最核心的依赖是浏览器提供的XMLHttpRequest对象，是这个对象使得浏览器可以发出HTTP请求与接收HTTP响应。实现了在页面不刷新个情况下和服务器进行数据交互。

手写ajax:

var xhr = new XMLHttpRequest()

xhr.open('GET', 'http://hello.php', true)

xhr.onreadystatechange = function(){

if(xhr.readyState === 4) {

if((xhr.status >= 200 && xhr.status < 300) || xhr.status == 304){

//成功了

console.log(xhr.responseText)

} else {

console.log('服务器异常')

}

}

}

Jquery.ajax基于原生的XHR开发，XHR本身的架构不清晰.

针对MVC的编程,不符合现在前端MVVM的浪潮

JQuery整个项目太大，单纯使用ajax却要引入整个JQuery非常的不合理.

axios:

axios从 node.js 创建 http 请求;

支持 Promise API

客户端支持防止CSRF;

提供了一些并发请求的接口;

不支持jsonp，不过可以引入jsonp模块.

fetch:

fetch基于Promise,脱离了XHR，是ES规范里新的实现方式.

更加底层，提供的API丰富（request, response）

但是fetch只对网络请求报错，对400，500都当做成功的请求，需要封装去处理.

fetch默认不会带cookie，需要添加配置项

fetch没有办法原生监测请求的进度，而XHR可以.

fetch('https://api.myjson.com/bins/of6pw')

.then(function(response) {

return response.json();

})

.then(function(res) {

console.log(res);

});

**前端需要掌握的加密技术?**

使用 https;

使用md5加密;

将密码+时间戳加密后传输，然后服务端再解密.

**Get请求和post请求的区别?**

GET 和 POST 只是 HTTP 协议中两种请求方式（异曲同工），而 HTTP 协议是基于 TCP/IP 的应用层协议，无论 GET 还是 POST，用的都是同一个传输层协议，所以在传输上，没有区别。

1.url可见性：

get，参数url可见

post，url参数不可见

2.传输数据的大小：

get一般传输数据大小不超过2k-4k.

post请求传输数据的大小没有限制.

3.数据传输上：

get，通过拼接url进行传递参数

post，通过body体传输参数

4.后退页面的反应：

get请求页面后退时，不产生影响.

post请求页面后退时，会重新提交请求.

5.安全性：

原则上post肯定要比get安全，毕竟post传输参数时url不可见.

**什么是Web Workers**

H5规范提供了js分线程的实现, 取名为: Web Workers

相关API:

\* Worker: 构造函数, 加载分线程执行的js文件

\* Worker.prototype.onmessage: 用于接收另一个线程的回调函数

\* Worker.prototype.postMessage: 向另一个线程发送消息

不足

\* worker内代码不能操作DOM(更新UI)

\* 不能跨域加载JS

\* 不是每个浏览器都支持这个新特性

**对http协议有了解吗?**

1.HTTP也叫作超文本传输协议，全称：Hyper Transfer Protocol。可以传输文字、图像、视频和各种流文件。

2. HTTP是无状态的：同一个客户端，连续发起多次请求，对HTTP服务器来说，都是新的请求，HTTP没办法知道来自一个客户端。

3. HTTP消息结构:

客户端发送一个HTTP请求到服务器的请求消息包括以下格式：请求行（request line）、请求头部（header）、空行和请求数据四个部分组成

HTTP响应也由四个部分组成，分别是：状态行、消息报头、空行和响应正文.

4.基本请求方法:

GET：向特定的资源发出请求。

POST：向指定资源提交数据进行处理请求,数据被包含在请求体中。

PUT：向指定资源位置上传其最新内容.

HEAD：向服务器索要与GET请求相一致的响应，只不过响应体将不会被返回。

5.基本状态码:

以2开头 （请求成功）表示成功处理了请求的状态代码.

以3开头 （请求被重定向）表示要完成请求，需要进一步操作。

以4开头 （请求错误）这些状态代码表示请求可能出错，妨碍了服务器的处理。

以5开头（服务器错误）这些状态代码表示服务器在尝试处理请求时发生内部错误。

**http和https的区别?**

HTTP特点：

1.https协议需要申请证书，一般免费证书较少，因而需要一定费用.

2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

5. 苹果现在强制所有iOS App使用HTTPS加密。安卓9.0以上也要使用https协议.

**一个url的是由哪些部分组成的?**

1.协议部分：URL的协议部分为http：或https:

2.服务器hostname(主机名）：是指存放资源的服务器的域名系统 (DNS) 主机名或 IP 地址。

3. 端口部分：跟在域名后面的是端口,默认是80

4.path部分：从域名后的第一个“/”开始到最后一个“/”为止，是path部分。

5. ?query(查询)：可选，用于给动态网页传递参数.

6. hash:从“#”开始到最后，都是hash部分。

**url中#(hash)的含义?**

1.hash 属性是一个可读可写的字符串，该字符串是 URL 的锚部分.

2.“#”代表网页中的一个位置。其右面的字符，就是该位置的标识符浏览器读取这个URL后，会自动将print位置滚动至可视区域。（单页应用）

3.HTTP请求不包括'#',所以单单改变#后的部分，浏览器只会滚动到相应位置，不会重新加载网页。

4.使用window.location.hash读取#值.

5.onhashchange事件,这是一个HTML 5新增的事件，当#值发生变化时，就会触发这个事件。

**用户输入一个url地址,按下回车之后发生了什么?**

互联网个网络xu设备间的通信都必须遵守TCP/IP协议标准。利用TCP/IP进行通信时，会通过分层顺序与对端进行通信，发送数据会依次经过应用层，传输层，网络层，数据链路层，物理层。而接受数据的顺序刚好相反。

1.用户在应用层输入url

2. 应用层DNS解析域名，获得对端IP地址.

3. 应用层客户端发送HTTP请求:

HTTP请求包括请求报头和请求主体。其中请求报头中包含了请求方法，请求资源，请求所使用协议，以及返回的资源是否需要缓存，客户端是否需要发送cookie等。

4. 传输层TCP传输报文:

传输层的TCP协议提供可靠的字节流服务，他为了方便传输，将大块的数据分割成以报文段为基本单位的数据包进行管理。为了保证可靠性传输，TCP采用三次握手来保证可靠性传输。

5. 网络层IP协议查询MAC地址.

IP协议把TCP分割的数据包传送给接收方。而要保证却是能够传送给对方主机还需要MAC地址，也就是物理地址。IP地址和MAC地址是一一对应关系。一个设备有且只有一个MAC地址。IP地址可以更换，MAC不会变。

6. 数据到达链路层

7. 服务器接受请求

服务端主机的网卡接收到数据后，驱动操作系统拿到数据，自下而上进行解包,从原路返回响应.

响应报文中包括协议名称/协议版本，状态码，状态码描述等信息。其中常见状态码：200 表示请求资源成功;301：永久重定向，表示资源已经永久性重定向到指定位置;304:缓存;500,服务器错误;404页面未找到.

答案2:

1. 域名解析,解析主机名得到对应的IP地址.

2. 在Internet中所有的传输都是通过TCP/IP进行的,发起TCP3次握手.

3. 建立TCP连接后发起HTTP请求.

4. 服务器端响应http请求，浏览器得到html代码

5. 浏览器解析html代码，并请求html代码中的资源.

6. 浏览器对页面进行渲染呈现给用户.

**请求报文包含了哪些部分?**

1. 请求行:

包括请求方法字段、URL字段和HTTP协议版本

例如：GET /index.html HTTP/1.1和GET/sample.jspHTTP/1.1

get方法将数据拼接在url后面，传递参数受限

请求方法有：GET、POST、HEAD、PUT、OPTIONS、TRACE、CONNECT

2. 请求头(key value形式), 包含浏览器，主机，接受的编码方式和压缩方式,例如:

Accept:image/gif.image/jpeg,\*/\* //请求格式

Accept-Language:zh-cn //语言

Connection:Keep-Alive //持续TCP连接

Host:localhost //主机

User-Agent:Mozila/4.0(compatible;MSIE5.01;Window NT5.0) //浏览器信息

Accept-Encoding:gzip,deflate // 压缩方式

3.空行, 是一个回车符和换行符,表示请求头结束, 通知服务器以下不再有请求头.

4. 请求体(数据) : post方法中，会把数据以key value形式发送请求,包含在请求体(body)里.

**说说你所知道的web安全及防护措施.**

SQL注入、XSS、

CSRF攻击:

跨站点请求伪造（Cross-Site Request Forgeries），冒充用户发起请求.

预防:

验证码；强制用户必须与应用进行交互，才能完成最终请求.

尽量使用 post ，限制 get 使用；get 太容易被拿来做 csrf 攻击；

token 验证 CSRF 防御机制是公认最合适的方案。

XSS攻击:

攻击者想尽一切办法将可以执行的代码注入到网页中. 比如攻击的代码被服务端写入进数据库中，这种攻击危害性很大，因为如果网站访问量很大的话，就会导致大量正常访问页面的用户都受到攻击。

预防:

转义字符:

首先，对于用户的输入应该是永远不信任的。最普遍的做法就是转义输入输出的内容，对于引号、尖括号、斜杠进行转义.

使用函数过滤script 标签.

CSP

CSP 本质上就是建立白名单，开发者明确告诉浏览器哪些外部资源可以加载和执行。我们只需要配置规则，如何拦截是由浏览器自己实现的。

设置 HTTP Header 中的 Content-Security-Policy

设置 meta 标签的方式 <meta http-equiv="Content-Security-Policy">开启白名单.

**了解token加密吗? token的原理：**

第一步：后端随机产生一个 token，把这个token 保存到 session 状态中；同时后端把这个token 交给前端页面；

第二步：前端页面提交请求时，把 token 加入到请求数据或者头信息中，一起传给后端；

后端验证前端传来的 token 与 session 是否一致，一致则合法，否则是非法请求。

Cookie,session, LocalStorage和sessionStorage的区别?

高并发请求优化方案(秒杀活动)?

1.限流，点击提交按钮后按钮置灰，显示为正在排队中，等处理结束或若干秒后，才允许用户点击.

2.限制用户在n秒之内只能提交一次请求.

3. 限频率，同一个UID，限制访问频率，做页面缓存，n秒内到达站点层的请求，均返回同一页面

实现轮询,什么是长轮询和短轮询?

什么是webSocket?

怎么在获取当前页面的url地址?