



# 一面 2：JS-Web-API 知识点与高频考题解析

除 ES 基础之外，Web 前端经常会用到一些跟浏览器相关的 API，接下来我们一起梳理一下。

## 知识点梳理

- BOM 操作
- DOM 操作
- 事件绑定
- Ajax
- 存储

## BOM

BOM（浏览器对象模型）是浏览器本身的一些信息的设置和获取，例如获取浏览器的宽度、高度，设置让浏览器跳转到哪个地址。

- `navigator`
- `screen`
- `location`
- `history`

这些对象就是一堆非常简单粗暴的 API，没有任何技术含量，讲起来一点意思都没有，大家去 MDN 或者 w3school 这种网站一查就都明白了。面试的时候，面试官基本不会出太多这方面的题目，因为只要基础知识过关了，这些 API 即便你记不住，上网一查也都知道。下面列举一下常用功能的代码示例

获取浏览器特性（即俗称的 UA）然后识别客户端，例如判断是不是 Chrome 浏览器



```
console.log(isChrome)
```

获取屏幕的宽度和高度

```
console.log(screen.width)
console.log(screen.height)
```

js

获取网址、协议、path、参数、hash 等

```
// 例如当前网址是 https://juejin.im/timeline/frontend?a=10&b=10#some
console.log(location.href) // https://juejin.im/timeline/frontend?a=10&b=10#some
console.log(location.protocol) // https:
console.log(location.pathname) // /timeline/frontend
console.log(location.search) // ?a=10&b=10
console.log(location.hash) // #some
```

js

另外，还有调用浏览器的前进、后退功能等

```
history.back()
history.forward()
```

js

## DOM

题目：DOM 和 HTML 区别和联系

### 什么是 DOM

讲 DOM 先从 HTML 讲起，讲 HTML 先从 XML 讲起。XML 是一种可扩展的标记语言，所谓可扩展就是它可以描述任何结构化的数据，它是一棵树！

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
```



```
<other>
  <a></a>
  <b></b>
</other>
</note>
```

HTML 是一个有既定标签标准的 XML 格式，标签的名字、层级关系和属性，都被标准化（否则浏览器无法解析）。同样，它也是一棵树。

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div>
    <p>this is p</p>
  </div>
</body>
</html>
```

html

我们开发完的 HTML 代码会保存到一个文档中（一般以 `.html` 或者 `.htm` 结尾），文档放在服务器上，浏览器请求服务器，这个文档被返回。因此，最终浏览器拿到的是一个文档而已，文档的内容就是 HTML 格式的代码。

但是浏览器要把这个文档中的 HTML 按照标准渲染成一个页面，此时浏览器就需要将这堆代码处理成自己能理解的东西，也得处理成 JS 能理解的东西，因为还得允许 JS 修改页面内容呢。

基于以上需求，浏览器就需要把 HTML 转变成 DOM，HTML 是一棵树，DOM 也是一棵树。对 DOM 的理解，可以暂时先抛开浏览器的内部因素，先从 JS 着手，即可以认为 DOM 就是 JS 能识别的 HTML 结构，一个普通的 JS 对象或者数组。

## 获取 DOM 节点

最常用的 DOM API 就是获取节点，其中常用的获取方法如下面代码示例：

```
// 通过 id 获取
var div1 = document.getElementById('div1') // 元素

// 通过 tagname 获取
var divList = document.getElementsByTagName('div') // 集合
console.log(divList.length)
console.log(divList[0])

// 通过 class 获取
var containerList = document.getElementsByClassName('container') // 集合

// 通过 CSS 选择器获取
var pList = document.querySelectorAll('p') // 集合
```

js

题目：property 和 attribute 的区别是什么？



property



`className` `nodeName` `nodeType` 属性。注意，这些都是 JS 范畴的属性，符合 JS 语法的。

js

```
var pList = document.querySelectorAll('p')
var p = pList[0]
console.log(p.style.width) // 获取样式
p.style.width = '100px' // 修改样式
console.log(p.className) // 获取 class
p.className = 'p1' // 修改 class

// 获取 nodeName 和 nodeType
console.log(p.nodeName)
console.log(p.nodeType)
```

### attribute

property 的获取和修改，是直接改变 JS 对象，而 attribute 是直接改变 HTML 的属性，两种有很大的区别。attribute 就是对 HTML 属性的 get 和 set，和 DOM 节点的 JS 范畴的 property 没有关系。

js

```
var pList = document.querySelectorAll('p')
var p = pList[0]
p.getAttribute('data-name')
p.setAttribute('data-name', 'juejin')
p.getAttribute('style')
p.setAttribute('style', 'font-size:30px;')
```

而且，get 和 set attribute 时，还会触发 DOM 的查询或者重绘、重排，频繁操作会影响页面性能。

题目：DOM 操作的基本 API 有哪些？

### DOM 树操作

新增节点



```
// 添加新节点
var p1 = document.createElement('p')
p1.innerHTML = 'this is p1'
div1.appendChild(p1) // 添加新创建的元素

// 移动已有节点。注意，这里是“移动”，并不是拷贝
var p2 = document.getElementById('p2')
div1.appendChild(p2)
```

### 获取父元素

```
var div1 = document.getElementById('div1')
var parent = div1.parentElement
```

js

### 获取子元素

```
var div1 = document.getElementById('div1')
var child = div1.childNodes
```

js

### 删除节点

```
var div1 = document.getElementById('div1')
var child = div1.childNodes
div1.removeChild(child[0])
```

js

还有其他操作的API，例如获取前一个节点、获取后一个节点等，但是面试过程中经常考到的就是上面几个。

---

## 事件

### 事件绑定

普通的事件绑定写法如下：



```
// event.preventDefault() // 阻止默认行为
// event.stopPropagation() // 阻止冒泡
console.log('clicked')
})
```

为了编写简单的事件绑定，可以编写通用的事件绑定函数。这里虽然比较简单，但是会随着后文的讲解，来继续完善和丰富这个函数。

```
// 通用的事件绑定函数
function bindEvent(elem, type, fn) {
  elem.addEventListener(type, fn)
}
var a = document.getElementById('link1')
// 写起来更加简单了
bindEvent(a, 'click', function(e) {
  e.preventDefault() // 阻止默认行为
  alert('clicked')
})
```

js

最后，如果面试被问到 IE 低版本兼容性问题，我劝你果断放弃这份工作机会。现在互联网流量都在 App 上，IE 占比越来越少，再去为 IE 浪费青春不值得，要尽量去做 App 相关的工作。

题目：什么是事件冒泡？

## 事件冒泡

```
<body>
  <div id="div1">
    <p id="p1">激活</p>
    <p id="p2">取消</p>
    <p id="p3">取消</p>
    <p id="p4">取消</p>
  </div>
  <div id="div2">
    <p id="p5">取消</p>
    <p id="p6">取消</p>
  </div>
</body>
```

html

js

```
var body = document.body
bindEvent(body, 'click', function (e) {
    // 所有 p 的点击都会冒泡到 body 上，因为 DOM 结构中 body 是 p 的上级节点，事件会沿着 DOM 树向上冒泡
    alert('取消')
})

var p1 = document.getElementById('p1')
bindEvent(p1, 'click', function (e) {
    e.stopPropagation() // 阻止冒泡
    alert('激活')
})
```

如果我们在 `p1` `div1` `body` 中都绑定了事件，它是会根据 DOM 的结构来冒泡，从下到上挨个执行的。但是我们使用 `e.stopPropagation()` 就可以阻止冒泡

题目：如何使用事件代理？有何好处？

## 事件代理

我们设定一种场景，如下代码，一个 `<div>` 中包含了若干个 `<a>`，而且还能继续增加。那如何快捷方便地为所有 `<a>` 绑定事件呢？

html

```
<div id="div1">
  <a href="#">a1</a>
  <a href="#">a2</a>
  <a href="#">a3</a>
  <a href="#">a4</a>
</div>
<button>点击增加一个 a 标签</button>
```

这里就会用到事件代理。我们要监听 `<a>` 的事件，但要把具体的事件绑定到 `<div>` 上，然后看事件的触发点是不是 `<a>`。

js

```
var div1 = document.getElementById('div1')
div1.addEventListener('click', function (e) {
    // e.target 可以监听到触发点击事件的元素是哪一个
})
```





```
        alert(target.innerHTML)
    }
})
```

我们现在完善一下之前写的通用事件绑定函数，加上事件代理。

```
js

function bindEvent(elem, type, selector, fn) {
    // 这样处理，可接收两种调用方式 bindEvent(div1, 'click', 'a', function () {...}) 和 bin
    if (fn == null) {
        fn = selector
        selector = null
    }

    // 绑定事件
    elem.addEventListener(type, function (e) {
        var target
        if (selector) {
            // 有 selector 说明需要做事件代理
            // 获取触发时间的元素，即 e.target
            target = e.target
            // 看是否符合 selector 这个条件
            if (target.matches(selector)) {
                fn.call(target, e)
            }
        } else {
            // 无 selector，说明不需要事件代理
            fn(e)
        }
    })
}
```

然后这样使用，简单很多。

```
js

// 使用代理，bindEvent 多一个 'a' 参数
var div1 = document.getElementById('div1')
bindEvent(div1, 'click', 'a', function (e) {
    console.log(this.innerHTML)
})

// 不使用代理
var a = document.getElementById('a1')
bindEvent(div1, 'click', function (e) {
```

最后，使用代理的优点如下：

- 使代码简洁
- 减少浏览器的内存占用

## Ajax

### XMLHttpRequest

题目：手写 XMLHttpRequest 不借助任何库

这是很多奇葩的、个性的面试官经常用的手段。这种考查方式存在很多争议，但是你不能完全说它是错误的，毕竟也是考查对最基础知识的掌握情况。

```
js
var xhr = new XMLHttpRequest()
xhr.onreadystatechange = function () {
  // 这里的函数异步执行，可参考之前 JS 基础中的异步模块
  if (xhr.readyState == 4) {
    if (xhr.status == 200) {
      alert(xhr.responseText)
    }
  }
}
xhr.open("GET", "/api", false)
xhr.send(null)
```

当然，使用 jQuery、Zepto 或 Fetch 等库来写就更加简单了，这里不再赘述。

### 状态码说明

上述代码中，有两处状态码需要说明。xhr.readyState 是浏览器判断请求过程中各个阶段的，xhr.status 是 HTTP 协议中规定的不同结果的返回状态说明。

xhr.readyState 的状态码说明：

- 0 -代理被创建，但尚未调用 open() 方法。



- 3 - 下载中， `responseText` 属性已经包含部分数据。
- 4 - 下载操作已完成

题目：HTTP 协议中，response 的状态码，常见的有哪些？

`xhr.status` 即 HTTP 状态码，有 `2xx` `3xx` `4xx` `5xx` 这几种，比较常用的有以下几种：

- `200` 正常
- `3xx`
  - `301` 永久重定向。如 `http://xxx.com` 这个 GET 请求（最后没有 `/`），就会被 `301` 到 `http://xxx.com/`（最后是 `/`）
  - `302` 临时重定向。临时的，不是永久的
  - `304` 资源找到但是不符合请求条件，不会返回任何主体。如发送 GET 请求时，head 中有 `If-Modified-Since: xxx`（要求返回更新时间是 `xxx` 时间之后的资源），如果此时服务器端资源未更新，则会返回 `304`，即不符合要求
- `404` 找不到资源
- `5xx` 服务器端出错了

看完要明白，为何上述代码中要同时满足 `xhr.readyState == 4` 和 `xhr.status == 200`。

## Fetch API

目前已经有一个获取 HTTP 请求更加方便的 API：`Fetch`，通过 `Fetch` 提供的 `fetch()` 这个全局函数方法可以很简单地发起异步请求，并且支持 `Promise` 的回调。但是 `Fetch` API 是比较新的 API，具体使用的时候还需要查查 [caniuse](https://caniuse.com/fetch)，看下其浏览器兼容情况。

看一个简单的例子：

```
js
fetch('some/api/data.json', {
  method: 'POST', // 请求类型 GET、POST
  headers: {}, // 请求的头信息，形式为 Headers 对象或 ByteString
  body: {}, // 请求发送的数据 blob、BufferSource、FormData、URLSearchParams (get 或 head 方法)
  mode: '', // 请求的模式，是否跨域等，如 cors、no-cors 或 same-origin
  credentials: '', // cookie 的跨域策略，如 omit、same-origin 或 include
  cache: '', // 请求的 cache 模式: default、no-store、reload、no-cache、force-cache 或 only
}).then(function(response) { ... });
```



回数据) 多种类型, 比如支持二进制文件、字符串和 `formData` 等。

### 跨域

题目: 如何实现跨域?

浏览器中有 **同源策略**, 即一个域下的页面中, 无法通过 Ajax 获取到其他域的接口。例如有一个接口 `http://m.juejin.com/course/ajaxcourse/recom?cid=459`, 你自己的一个页面 `http://www.yourname.com/page1.html` 中的 Ajax 无法获取这个接口。这正是命中了“同源策略”。如果浏览器哪些地方忽略了同源策略, 那就是浏览器的安全漏洞, 需要紧急修复。

url 哪些地方不同算作跨域?

- 协议
- 域名
- 端口

但是 HTML 中几个标签能逃避过同源策略——`<script src="xxx">`、``、`<link href="xxxx">`, 这三个标签的 `src/href` 可以加载其他域的资源, 不受同源策略限制。

因此, 这使得这三个标签可以做一些特殊的事情。

- `<img>` 可以做打点统计, 因为统计方并不一定是同域的, 在讲解 JS 基础知识异步的时候有过代码示例。除了能跨域之外, `<img>` 几乎没有浏览器兼容问题, 它是一个非常古老的标签。
- `<script>` 和 `<link>` 可以使用 CDN, CDN 基本都是其他域的连接。
- 另外 `<script>` 还可以实现 JSONP, 能获取其他域接口的信息, 接下来马上讲解。

但是请注意, 所有的跨域请求方式, 最终都需要信息提供方来做出相应的支持和改动, 也就是要经过信息提供方的同意才行, 否则接收方是无法得到它们的信息的, 浏览器是不允许的。

### 解决跨域 - JSONP

首先, 有一个概念你要明白, 例如访问 `http://coding.m.juejin.com/classindex.html` 的时候, 服务器端就一定有一个 `classindex.html` 文件吗? —— 不一定, 服务器可以拿到这个



器也可以动态生成文件并返回。OK，接下来正式开始。

例如我们的网站和掘金网，肯定不是一个域。我们需要掘金网提供一个接口，供我们来获取。首先，我们在自己的页面这样定义

```
html
<script>
window.callback = function (data) {
  // 这是我们跨域得到信息
  console.log(data)
}
</script>
```

然后掘金网给我提供了一个 <http://coding.m.juejin.com/api.js>，内容如下（之前说过，服务器可动态生成内容）

```
js
callback({x:100, y:200})
```

最后我们在页面中加入 `<script src="http://coding.m.juejin.com/api.js"></script>`，那么这个js加载之后，就会执行内容，我们就得到内容了。

### 解决跨域 - 服务器端设置 http header

这是需要在服务器端设置的，作为前端工程师我们不用详细掌握，但是要知道有这么个解决方案。而且，现在推崇的跨域解决方案是这一种，比 JSONP 简单许多。

```
js
response.setHeader("Access-Control-Allow-Origin", "http://m.juejin.com/"); // 第二个参数
response.setHeader("Access-Control-Allow-Headers", "X-Requested-With");
response.setHeader("Access-Control-Allow-Methods", "PUT,POST,GET,DELETE,OPTIONS");

// 接收跨域的cookie
response.setHeader("Access-Control-Allow-Credentials", "true");
```



### cookie

cookie 本身不是用来做服务器端存储的（计算机领域有很多这种“狗拿耗子”的例子，例如 CSS 中的 float），它是设计用来在服务器和客户端进行信息传递的，因此我们的每个 HTTP 请求都带着 cookie。但是 cookie 也具备浏览器端存储的能力（例如记住用户名和密码），因此就被开发者用上了。

使用起来也非常简单，`document.cookie = ....` 即可。

但是 cookie 有它致命的缺点：

- 存储量太小，只有 4KB
- 所有 HTTP 请求都带着，会影响获取资源的效率
- API 简单，需要封装才能用

### localStorage 和 sessionStorage

后来，HTML5 标准就带来了 `sessionStorage` 和 `localStorage`，先拿 `localStorage` 来说，它是专门为了浏览器端缓存而设计的。其优点有：

- 存储量增大到 5MB
- 不会带到 HTTP 请求中
- API 适用于数据存储 `localStorage.setItem(key, value)` `localStorage.getItem(key)`

`sessionStorage` 的区别就在于它是根据 session 过去时间而实现，而 `localStorage` 会永久有效，应用场景不同。例如，一些需要及时失效的重要信息放在 `sessionStorage` 中，一些不重要但是不经常设置的信息，放在 `localStorage` 中。

另外告诉大家一个小技巧，针对 `localStorage.setItem`，使用时尽量加入到 `try-catch` 中，某些浏览器是禁用这个 API 的，要注意。

---

### 小结

---

本小节总结了 W3C 标准中 Web-API 部分，面试中常考的知识点，这些也是日常开发中最常用的 API 和知识。



写下你的留言

**Johnson君**

比较粗糙基础，进一般小公司足够，凭这些想进bat还是有距离。

▲ 0 评论 2天前

**janeluck** 前端工程师

文章末尾提示的localStorage.setItem, 使用时尽量加入到try-catch。 开发时遇到过safari的无痕浏览模式下就会禁用localStorage, setItem会导致程序运行抛错。

▲ 5 评论 1月前

**aloha66**

事件代理的e.nodeName是不是应该写成e.target.nodeName呢？

▲ 2 评论 1月前

**霍霖笙箫** 前端工程师

就当又复习一遍了，基础看多少遍都不嫌多啊，希望秋招顺利。其实screen几个常用的可以多例几个，顺便补充下 src和link的区别，我也是读了之后顺便科普了下这个。跨域2种少了点吧，CORS可以多科普些 web socket, postMessage, proxyServer可以提点。谢谢作者~~

▲ 0 评论 1月前

**张三丰是也**

不能只用 navigator.userAgent 中是否含有 "Chrome" 来判断是否是 Chrome 浏览器，因为 Edge 浏览器中也包含 "Chrome"

▲ 0 评论 1月前

**fuiyu**

能容有点简单

▲ 0 1条评论 2月前



标题内双题名咋办，这么混乱的内容

▲ 0 1条评论 2月前

**frank26** 前端开发

xhr.open("GET", "/api", false)

《红宝书》为保证浏览器兼容性，这句话应该放到给readyState加监听事件之后。。。

▲ 2 1条评论 2月前

**稻谷的谷** 学生/前端

xhr.readyState的状态码说明与MDN似乎不太一样

▲ 0 2条评论 2月前

**dduke** 前端 @ 上海闯奇

跨域相关的 现在不是比较常用 使用 nginx 代理 或者 node gateway

▲ 1 1条评论 3月前

**聆道人** 歪脖frontend @ 浪迹天涯

早晨打卡

▲ 1 2条评论 3月前

**题歌** Web Developer @ Cloudpick

打卡 复习基础知识

▲ 0 1条评论 3月前

**csz.Seven** 前端开发

打卡

▲ 0 1条评论 3月前

**Raoul1996** 前端实习生 @ 不洗碗工作室

赞

▲ 0 评论 3月前





## Web 前端面试指南与高频考题解析

共

▲ 0

评论 3月前