HTML中的JavaScript

<script>元素

将JS插入HTML的主要方法
网景创造出来，最早在Netscape Navigator 2 中实现的
以下8个属性

- async：可选。表示立即下载脚本，但是不阻止其他页面动作比如下载资源或等待其他脚本加载。只对外部脚本文教有效
- crossorigin：可选。配置相关请求的CROS设置。默认不使用CROS。anonymous配置文件请求不必设置凭据标志；use-credentials设置凭据标志，出战请求会包含凭据
- defer：可选，表示脚本可以延迟到文档完全被解析和显示之后再执行。只对外部脚本文件有效。IE7以及更早版本可以对内指定。
- integrity：可选。允许比对接收到的资源和指定的加密签名以验证子资源完整性（SRI,Subresource Integrity）。如果接收到的资源的签名与这个属性指定的签名不匹配，则页面会报错，脚本不会执行。一般给CDN用，防止提供恶意内容
- src：可选。表示包含要执行的代码的外部文件。
- charset：可选。使用src属性指定的代码字符集。很少使用，因为大部分浏览器不在乎这个值
- language：弃用。Javascript，Javascript 1.2，VBScript。大部分浏览器忽略
- type：可选，代替language，表示代码块中脚本语言的内容类型（也称MIME类型：多媒体邮件拓展）。按照惯例，这个值始终都是"text/javascript"，尽管"text/javascript"和"text/ecmascript"都已经废弃了。JavaScript 文件的 MIME 类型通常是"application/x-javascript"，不过给type 属性这个值有可能导致脚本被忽略。
  在非 IE 的浏览器中有效的其他值还有"application/javascript"和"application/ecmascript"。如果这个值是 module，则代码会被当成 ES6 模块，而且只有这时候代码中才能出现 import 和 export 关键字。主要是"module"来区分ES6

使用方法：

1. 直接嵌入代码:出现字符串</script>就认为结束，如果必须有用到</script>，请转义<\/script>
2. 在网页中包含外部 JavaScript 文件，src导入:<script src="example.js"/> ,XHTML看到，可以忽略结束标签，但是这个语法HTML不能使用，无效HTML,有些浏览器不能正常处理，如IE

浏览器不会检查JavaScript文件的拓展名是否为.js。这就为使用服务器端脚本语言动态生成 JavaScript 代码，或者在浏览器中将JavaScript扩展语言（如TypeScript，或React的 JSX）转译为JavaScript提供了可能性。不过要注意，服务器经常会根据文件扩展来确定响应的正确 MIME 类型。如果不打算使用.js扩展名，一定要确保服务器能返回正确的 MIME 类型。

直接嵌入与src，优先src。如果在<script>中既用了src又嵌入了JS代码，浏览器只会下载src，忽略行内代码。
src可以是一个完整的URL，而且指向的资源可以不在同一个域中。
过程：浏览器解析资源，向src的路径发送一个GET请求，假定是个JavaScript文件。这个请求不受SameSite策

略限制，返回并执行的JS还是受限制（返回的JS发起的请求是受SameSite限制的）。因为是GET请求，依然受HTTP/HTTPS协议限制

这个功能就支持了CDN，但是CDN有可能被黑，会提供恶意代码，所以需要integrity验证

浏览器按照<script>出现顺序依次解释他们，前提是没有defer和async。第二个必须等第一个解释完成才能继续...

标签位置

过去所有<script>元素都放在<head>标签内，为了把外部的CSS和JS文件都集中放在一起。

问题:
所有的JS都放在head就意味着必须把所有的JS代码都下载、解析、解释后，才能渲染页面（head之后才开始渲染）。
解决方案
现在通常都丢到body标签中的页面内容后面如：

```
<!DOCTYPE html>
<html>
 <head>
 <title>Example HTML Page</title>
 </head>
 <body>
 <!-- 这里是页面内容 -->
 <script src="example1.js"></script>
 <script src="example2.js"></script>
 </body>
</html>
```

这样页面就能先渲染后处理JS文件，浏览器空白时间变少，用户体验增加

推迟执行脚本 defer

defer--->先下载，等整个页面解析完后再运行。
最好只有一个defer文件，推迟执行的脚本不一定总会按顺序执行或者在 DOMContentLoaded事件之前执行
把defer放页面底部是因为浏览器的支持程度不一样

动态加载脚本

js可以使用DOM API，通过向DOM中动态加入script元素同样可以加载指定脚本。

```
let script = document.createElement('script');
script.src = 'gibberish.js';
document.head.appendChild(script);
```

在把 HTMLElement 元素添加到 DOM且执行到这段代码之前不会发送请求。默认异步方式加载=加了async。但是并不是所有浏览器支持async所有可以统一为同步加载

```
let script = document.createElement('script');
script.src = 'gibberish.js';
```

```
script.async = false;
document.head.appendChild(script);
```

动态加载脚本获取的资源对浏览器预加载不可见，影响获取资源队列的优先级，可以再头文件中显示声明

```
<link rel="preload" href="gibberish.js">
```

XHTML中的变化

可扩展超文本标记语言（XHTML，Extensible HyperText Markup Language）是HTML作为XML的应用重新包装的结果。
必须指定type为text/javascript，HTML可以没有
已经退出舞台
XHTML规则比HTML严格：<会被认为标签的开始，所以<一般用< 替代
或者用CDATA代码块包裹

```
<script type="text/javascript"><![CDATA[
 function compare(a, b) {
 if (a < b) {
 console.log("A is less than B");
 } else if (a > b) {
 console.log("A is greater than B");
 } else {
 console.log("A is equal to B");
 }
 }
]]></script>
```

但是有的浏览器支持XHTML但是不支持CDADA块，需要//注释

行内代码与外部文件

尽量放在外部文件

- 可维护性
- 缓存：浏览器会根据特定的设置缓存所有外部链接的JavaScript 文件，这意味着如果两个页面都用到同一个文件，则该文件只需下载一次。这最终意味着页面加载更快。
- 适应未来

文档模式

IE5.5发明了文档模式的概念，即可以使用dectype切换文档模式

起初的文档模式：

- 混杂模式：让 IE 像 IE5 一样（支持一些非标准的特性）
- 标准模式：让 IE 具有兼容标准的行为

后来又加入了第三种

- 准标准模式：浏览器支持很多标准的特性，但是没有标准规定得那么严格

IE11已弃用

<noscript>元素

针对早期浏览器不支持JS的问题
符合以下情况之一

- 浏览器不支持脚本
- 浏览器对脚本的支持被关闭
  noscript的内容会被渲染，否额不会被渲染

总结

JavaScript 是通过<script>元素插入到 HTML页面中的。
这个元素可用于把 JavaScript 代码嵌入到HTML 页面中，跟其他标记混合在一起，也可用于引入保存在外部文件中的 JavaScript。
本章的重点可以总结如下。

- 要包含外部 JavaScript 文件，必须将 src 属性设置为要包含文件的 URL。文件可以跟网页在同一台服务器上，也可以位于完全不同的域。
- 所有<script>元素会依照它们在网页中出现的次序被解释。在不使用 defer 和 async 属性的情况下，包含在<script>元素中的代码必须严格按次序解释。
- 对不推迟执行的脚本，浏览器必须解释完位于<script>元素中的代码，然后才能继续渲染页面的剩余部分。为此，通常应该把<script>元素放到页面末尾，介于主内容之后及</body>标签之前。
- 可以使用 defer 属性把脚本推迟到文档渲染完毕后再执行。推迟的脚本原则上按照它们被列出的次序执行。
- 可以使用 async 属性表示脚本不需要等待其他脚本，同时也不阻塞文档渲染，即异步加载。异步脚本不能保证按照它们在页面中出现的次序执行。
- 通过使用<noscript>元素，可以指定在浏览器不支持脚本时显示的内容。如果浏览器支持并启用脚本，则<noscript>元素中的任何内容都不会被渲染。