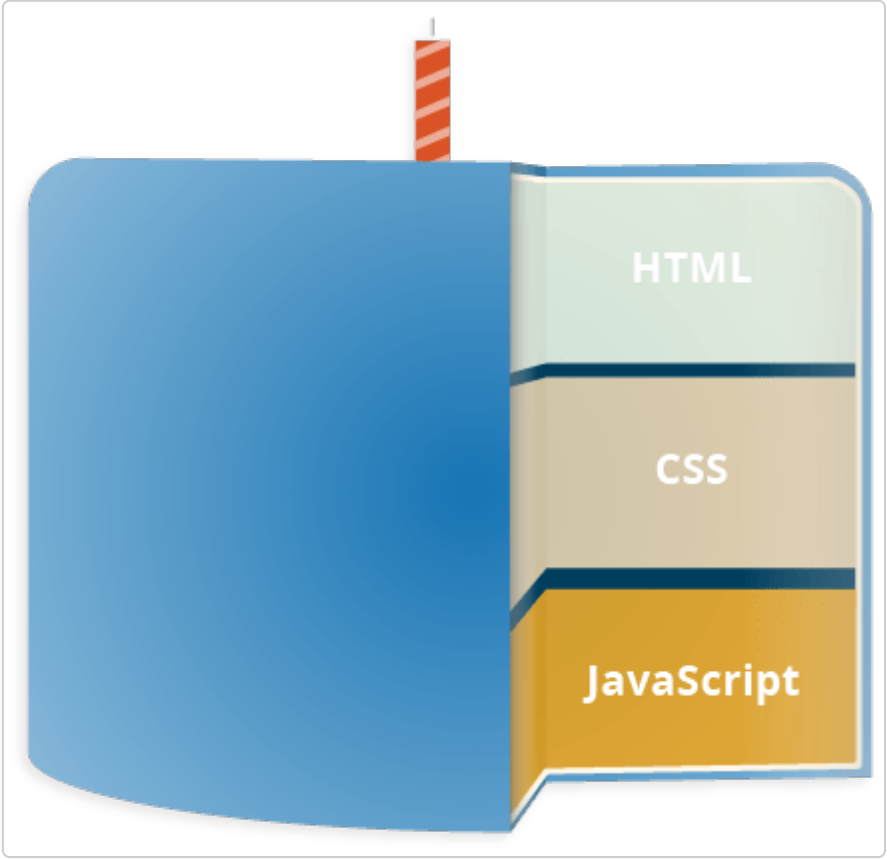
什么是JavaScript?

欢迎来到 MDN 初学者的 JavaScript 课程！在本文中，我们将从高层次审视 JavaScript，回答“它是什么？”等问题。和“你能用它做什么？”，并确保你对 JavaScript 的用途感到满意。

先决条件：	基本的计算机知识，对 HTML 和 CSS 有基本的了解。
客观的：	熟悉什么是 JavaScript，它能做什么，以及它如何融入网站。

高级定义

JavaScript 是一种脚本或编程语言，允许您在网页上实现复杂的功能——每次网页所做的不仅仅是坐在那里并显示静态信息供您查看——显示及时的内容更新、交互式地图、动画 2D/ 3D 图形、滚动视频自动点唱机等——您可以打赌，JavaScript 很可能参与其中。它是标准 Web 技术夹心蛋糕的第三层，其中两层（HTML和CSS）我们已经在学习区的其他部分进行了更详细的介绍。



- HTML是我们用来构建 Web 内容并为其赋予意义的标记语言，例如定义段落、标题和数据表，或在页面中嵌入图像和视频。
- CSS是一种样式规则语言，我们使用它来将样式应用到我们的 HTML 内容，例如设置背景颜色和字体，以及在多列中布置我们的内容。
- JavaScript是一种脚本语言，可让您创建动态更新的内容、控制多媒体、动画图像以及几乎所有其他内容。（好吧，不是所有的东西，但是你可以用几行 JavaScript 代码实现的东西是惊人的。）

这三层很好地建立在另一层之上。我们以一个简单的文本标签为例。我们可以使用 HTML 对其进行标记以赋予其结构和用途：

```
<p>Player 1: Chris</p>
```

Player 1: Chris

然后我们可以在组合中添加一些 CSS 以使其看起来不错：

```
p {
  font-family: "helvetica neue", helvetica, sans-serif;
  letter-spacing: 1px;
  text-transform: uppercase;
  text-align: center;
  border: 2px solid rgb(0 0 200 / 0.6);
  background: rgb(0 0 200 / 0.6);
  color: rgb(255 255 255 / 1);
  box-shadow: 1px 1px 2px rgb(0 0 200 / 0.4);
  border-radius: 10px;
  padding: 3px 10px;
  display: inline-block;
  cursor: pointer;
}
```

最后，我们可以添加一些 JavaScript 来实现动态行为：

```
const para = document.querySelector("p");

para.addEventListener("click", updateName);

function updateName() {
  const name = prompt("Enter a new name");
  para.textContent = `Player 1: ${name}`;
}
```

试着点击这个最后一个版本的文本标签，看看会发生什么（另请注意，您可以在 GitHub 上找到这个演示——查看源代码，或实时运行）！

JavaScript 可以做的远不止这些——让我们更详细地探讨一下。

那么它到底能做什么呢?

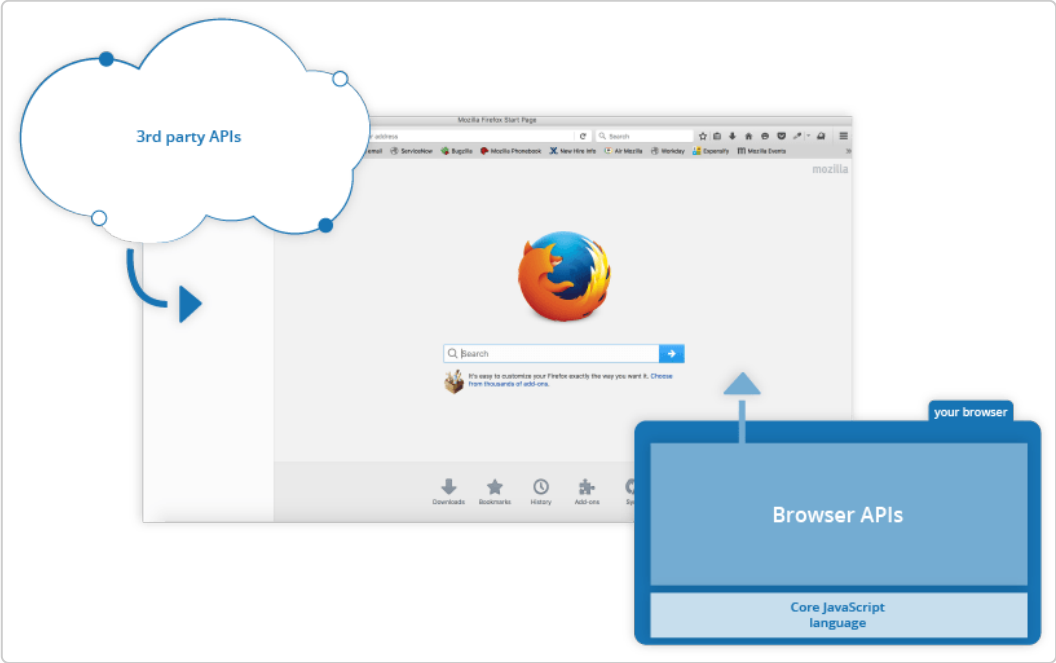
核心客户端 JavaScript 语言包含一些常见的编程功能，允许您执行以下操作：

- 在变量中存储有用的值。例如，在上面的示例中，我们要求输入一个新名称，然后将该名称存储在一个名为 的变量中 `name` 。
- 对文本片段（在编程中称为“字符串”）的操作。在上面的示例中，我们采用字符串“Player 1:”并将其连接到变量 `name` 以创建完整的文本标签，例如“Player 1: Chris”。
- 运行代码以响应网页上发生的某些事件。我们 `click` 在上面的示例中使用了一个事件来检测何时单击标签，然后运行更新文本标签的代码。
- 以及更多！

然而，更令人兴奋的是构建在客户端 JavaScript 语言之上的功能。所谓的**应用程序编程接口**（**API**）为您提供了在 JavaScript 代码中使用的额外超能力。

API 是现成的代码构建块集，允许开发人员实施原本难以或不可能实施的程序。他们为编程做的事情与现成的家具套件为家庭建造做的事情一样——把现成的板材用螺丝拧在一起做成书架比自己设计、去寻找要容易得多正确的木头，将所有的面板切割成合适的尺寸和形状，找到尺寸合适的螺丝，然后将*它们*拼在一起做成一个书架。

它们通常分为两类。



浏览器 API内置于您的 Web 浏览器中，能够从周围的计算机环境中公开数据，或者执行有用的复杂操作。例如：

- 允许 DOM (Document Object Model) API 您操作 HTML 和 CSS，创建、删除和更改 HTML，为您的页面动态应用新样式等。每次您看到页面上出现弹出窗口，或显示一些新内容（如我们在上面看到的简单的演示）例如，这就是 DOM 的实际应用。
- 检索 Geolocation API 地理信息。这就是Google 地图 能够找到您的位置并将其绘制在地图上的方式。
- 和 Canvas API WebGL 允许您创建动画 2D 和 3D 图形。人们正在使用这些网络技术做一些令人惊奇的事情——请参阅Chrome Experiments 和webglsamples 。
- 音频和视频 API喜欢 HTMLMediaElement 并 WebRTC 允许您使用多媒体做一些非常有趣的事情，例如直接在网页中播放音频和视频，或者从您的网络摄像头抓取视频并将其显示在其他人的计算机上（尝试我们的简单快照演示以 获取这个想法）。

> **注意：**上面的许多演示在较旧的浏览器中都无法运行——在进行实验时，最好使用现代浏览器（例如 Firefox、Chrome、Edge

> 或 Opera）来运行您的代码。您需要考虑跨浏览器测试当您接近交付生产代码（即真实客户将使用的真实代码）时会更详细。

默认情况下，第三方 API并未内置于浏览器中，您通常必须从 Web 上的某个地方获取它们的代码和信息。例如：

- Twitter API 允许你做一些事情，比如在你的网站上显示你最新的推文。
- Google Maps API 和OpenStreetMap API 允许您将自定义地图嵌入您的网站，以及其他此类功能。

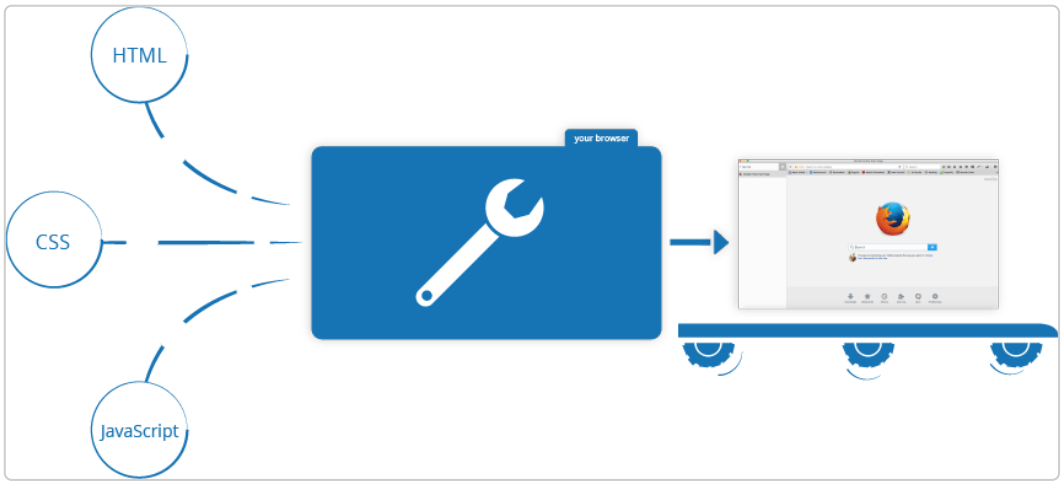
> **注意：**这些 API 是高级 API，我们不会在本模块中介绍其中的任何内容。您可以在我们的客户端 Web API 模块中找到更多关于这些的信息。

还有更多可用的！但是，暂时不要过度兴奋。在学习 JavaScript 24 小时后，您将无法构建下一个 Facebook、Google 地图或 Instagram——首先要了解很多基础知识。这就是你来这里的原因——让我们继续吧！

JavaScript 在您的页面上做什么？

在这里，我们实际上将开始查看一些代码，并在此过程中探索在页面中运行一些 JavaScript 时实际发生的情况。

让我们简要回顾一下当您在浏览器中加载网页时发生的事情（首先在我们的 CSS 工作原理一文中讨论）。当您在浏览器中加载网页时，您是在执行环境（浏览器选项卡）中运行代码（HTML、CSS 和 JavaScript）。这就像一个接收原材料（代码）并输出产品（网页）的工厂。



JavaScript 的一个非常常见的用途是通过文档对象模型 API（如上所述）动态修改 HTML 和 CSS 以更新用户界面。请注意，Web 文档中的代码通常按照其在页面上出现的顺序加载和执行。如果在要修改的 HTML 和 CSS 之前加载和运行 JavaScript，则可能会发生错误。您将在本文后面的"脚本加载策略"部分了解解决此问题的方法。

浏览器安全

每个浏览器选项卡都有自己单独的用于运行代码的桶（这些桶在技术术语中称为"执行环境"）——这意味着在大多数情况下每个选项卡中的代码是完全独立运行的，并且一个选项卡中的代码不能直接影响另一个选项卡中的代码——或另一个网站上的代码。这是一个很好的安全措施——如果不是这样，那么盗版者就可以开始编写代码来窃取其他网站的信息，以及其他类似的坏事。

> **注意：** 有一些方法可以安全地在不同网站/选项卡之间发送代码和数据，但这些是我们不会在本课程中介绍的高级技术。

JavaScript 运行顺序

当浏览器遇到一段 JavaScript 时，它通常会按顺序从上到下运行它。这意味着您需要小心放置事物的顺序。例如，让我们回到我们在第一个示例中看到的 JavaScript 块：

```
const para = document.querySelector("p");
```

```
para.addEventListener("click", updateName);

function updateName() {
  const name = prompt("Enter a new name");
  para.textContent = `Player 1: ${name}`;
}
```

在这里，我们选择一个文本段落（第 1 行），然后为其附加一个事件侦听器（第 3 行），以便在单击该段落时 `updateName()` 运行代码块（第 5-8 行）。代码 `updateName()` 块（这些类型的可重用代码块称为“函数”）要求用户输入新名称，然后将该名称插入段落以更新显示。

如果你调换了前两行代码的顺序，它将不再起作用——相反，你会在浏览器开发者控制台中得到一个错误返回—— `TypeError: para is undefined` 。这意味着该 `para` 对象还不存在，因此我们无法为其添加事件侦听器。

> **注意：** 这是一个非常常见的错误——在尝试对它们进行操作之前，您需要注意代码中引用的对象是否存在。

解释代码与编译代码

您可能会听到在编程上下文中**解释**和**编译的** 术语。在解释型语言中，代码从上到下运行，并立即返回运行代码的结果。您不必在浏览器运行之前将代码转换为不同的形式。代码以对程序员友好的文本形式接收，并直接从中进行处理。

另一方面，编译语言在被计算机运行之前被转换（编译）成另一种形式。例如，C/C++被编译成机器码，然后由计算机运行。该程序是从二进制格式执行的，它是从原始程序源代码生成的。

JavaScript 是一种轻量级的解释型编程语言。Web 浏览器接收原始文本形式的 JavaScript 代码并从中运行脚本。**从技术的角度来看，大多数现代 JavaScript 解释器实际上使用一种称为即时编译的**技术来提高性能；在使用脚本时，JavaScript 源代码被编译成更快的二进制格式，以便尽快运行。然而，JavaScript 仍然被认为是一种解释型语言，因为编译是在运行时处理的，而不是提前处理的。

两种类型的语言各有优势，但我们现在不讨论它们。

服务器端与客户端代码

您可能还会听到**服务器端**和**客户端**代码这两个术语，尤其是在 Web 开发的上下文中。客户端代码是在用户计算机上运行的代码——当查看网页时，页面的客户端代码被下载，然后由浏览器运行和显示。在本模块中，我们明确讨论**客户端 JavaScript**。

另一方面，服务器端代码在服务器上运行，然后将其结果下载并显示在浏览器中。流行的服务器端 Web 语言示例包括 PHP、Python、Ruby、ASP.NET，甚至 JavaScript！JavaScript 也可以用作服务器端语言，例如在流行的 Node.js 环境中——您可以在我们的动态网站 – 服务器端编程主题中找到有关服务器端 JavaScript 的更多信息。

动态代码与静态代码

动态 一词用于描述客户端 JavaScript 和服务器端语言——它指的是更新网页/应用程序的显示以在不同情况下显示不同内容、根据需要生成新内容的能力。服务器端代码在服务器上动态生成新内容，例如从数据库中提取数据，而客户端 JavaScript 在客户端浏览器中动态生成新内容，例如创建一个新的 HTML 表格，用服务器请求的数据填充它，然后在向用户显示的网页中显示表格。两种上下文中的含义略有不同，但相关，并且两种方法（服务器端和客户端）通常一起工作。

没有动态更新内容的网页称为**静态**网页——它始终显示相同的内容。

如何将 JavaScript 添加到您的页面?

JavaScript 以类似于 CSS 的方式应用于您的 HTML 页面。CSS 使用 `<link>` 元素来应用外部样式表和 `<style>` 元素来将内部样式表应用到 HTML，而 JavaScript 在 HTML 的世界中只需要一个朋友——元素 `<script>` 。让我们了解这是如何工作的。

内部JavaScript

1. 首先，制作我们的示例文件apply-javascript.html 的本地副本。将它保存在一个合适的目录中。

2. 在网络浏览器和文本编辑器中打开文件。您会看到 HTML 创建了一个包含可单击按钮的简单网页。
3. 接下来，转到您的文本编辑器并在您的头脑中添加以下内容——就在您的结束 `</head>` 标记之前：

```
<script>
  // JavaScript goes here
</script>
```

4. 现在我们将在我们的元素中添加一些 JavaScript， `<script>` 使页面做一些更有趣的事情——在“// JavaScript goes here”行下方添加以下代码：

```
document.addEventListener("DOMContentLoaded", () => {
  function createParagraph() {
    const para = document.createElement("p");
    para.textContent = "You clicked the button!";
    document.body.appendChild(para);
  }

  const buttons = document.querySelectorAll("button");

  for (const button of buttons) {
    button.addEventListener("click", createParagraph);
  }
});
```

5. 保存你的文件并刷新浏览器——现在你应该看到当你点击按钮时，一个新的段落被生成并放在下面。

> **注意：**如果您的示例似乎不起作用，请再次执行这些步骤并检查您是否正确完成了所有操作。您是否将起始代码的本地副本保存为 `.html` 文件？您是否在标签 `<script>` 之前添加了元素 `</head>`？您是否完全按照所示输入了 JavaScript？**JavaScript 区分大小写，而且非常繁琐，因此您需要完全按照所示输入语法，否则可能无法运行。**

> **注意：** 您可以在 GitHub 上看到此版本为apply-javascript-internal.html （也可以实时查看 ）。

外部脚本

这很好用，但是如果我们想将 JavaScript 放在外部文件中怎么办？现在让我们探讨一下。

1. 首先，在与示例 HTML 文件相同的目录中创建一个新文件。调用它 `script.js` ——确保它有 .js 文件扩展名，因为这是它被识别为 JavaScript 的方式。
2. `<script>` 用以下内容 替换当前元素：

   ```
   <script src="script.js" defer></script>
   ```

3. 在里面 `script.js` ，添加以下脚本：

   ```
   function createParagraph() {
     const para = document.createElement("p");
     para.textContent = "You clicked the button!";
     document.body.appendChild(para);
   }

   const buttons = document.querySelectorAll("button");

   for (const button of buttons) {
     button.addEventListener("click", createParagraph);
   }
   ```

4. 保存并刷新你的浏览器，你应该会看到同样的东西！它的工作原理是一样的，但现在我们已经在外部文件中获得了 JavaScript。就组织代码和使其可跨多个 HTML 文件重用而言，这通常是一件好事。另外，HTML 更易于阅读，而无需在其中转储大量脚本。

> **注意：** 您可以在 GitHub 上看到此版本为apply-javascript-external.html 和script.js （也可以实时查看 ）。

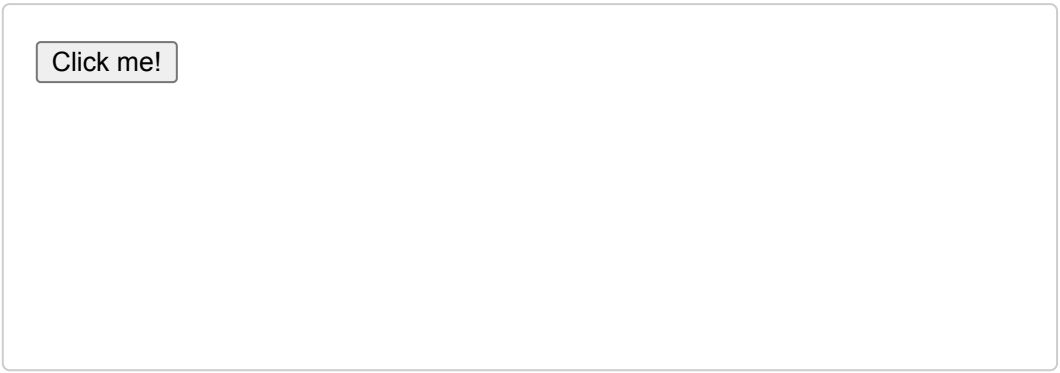
内联 JavaScript 处理程序

请注意，有时您会发现 HTML 中存在一些实际的 JavaScript 代码。它可能看起来像这样：

```
function createParagraph() {
  const para = document.createElement("p");
  para.textContent = "You clicked the button!";
  document.body.appendChild(para);
}
```

```
<button onclick="createParagraph()">Click me!</button>
```

您可以在下面尝试我们演示的这个版本。



该演示具有与前两节完全相同的功能，只是该 `<button>` 元素包含一个内联 `onclick` 处理程序，使函数在按下按钮时运行。

但是，请不要这样做。用 JavaScript 污染 HTML 是一种不好的做法，而且效率低下——您必须 `onclick="createParagraph()"` 在每个要应用 JavaScript 的按钮上包含该属性。

改为使用 addEventListener

不要在 HTML 中包含 JavaScript，而是使用纯 JavaScript 结构。该 `querySelectorAll()` 功能允许您选择页面上的所有按钮。然后，您可以遍

历按钮，为每个 using 分配一个处理程序 `addEventListener()` 。代码如下所示：

```
const buttons = document.querySelectorAll("button");

for (const button of buttons) {
  button.addEventListener("click", createParagraph);
}
```

这可能比属性长一点 `onclick` ，但它适用于所有按钮——无论页面上有多少按钮，也不管添加或删除了多少按钮。不需要更改 JavaScript。

> **注意：** 尝试编辑您的版本 `apply-javascript.html` 并在文件中添加更多按钮。重新加载时，您应该会发现单击所有按钮都会创建一个段落。整洁吧?

脚本加载策略

在正确的时间加载脚本涉及许多问题。没有什么像看起来那么简单！一个常见的问题是页面上的所有 HTML 都是按照它出现的顺序加载的。如果您正在使用 JavaScript 来操作页面上的元素（或者更准确地说，文档对象模型），如果 JavaScript 在您尝试执行某些操作的 HTML 之前加载和解析，您的代码将无法工作。

在上面的代码示例中，在内部和外部示例中，在解析 HTML 正文之前，JavaScript 在文档的头部加载并运行。这可能会导致错误，因此我们使用了一些结构来绕过它。

在内部示例中，您可以在代码周围看到这样的结构：

```
document.addEventListener("DOMContentLoaded", () => {
  // …
});
```

这是一个事件侦听器，它侦听浏览器的 `DOMContentLoaded` 事件，这表示 HTML 正文已完全加载和解析。此块中的 JavaScript 将在触发该事件后才

会运行，因此可以避免错误（您将在本课程的后面部分了解事件）。

在外部示例中，我们使用更现代的 JavaScript 特性来解决这个问题，即 `defer` 属性，它告诉浏览器在 `<script>` 到达标记元素后继续下载 HTML 内容。

```
<script src="script.js" defer></script>
```

在这种情况下，脚本和 HTML 将同时加载并且代码将起作用。

> **注意：** 在外部情况下，我们不需要使用 `DOMContentLoaded` 事件，因为 `defer` 属性为我们解决了问题。我们没有将 `defer` 解决方案用于内部 JavaScript 示例，因为它 `defer` 仅适用于外部脚本。

这个问题的老式解决方案过去是将脚本元素放在主体的底部（例如，就在标记之前 `</body>` ），以便在解析完所有 HTML 后加载它。此解决方案的问题在于，在加载 HTML DOM 之前，脚本的加载/解析被完全阻止。在具有大量 JavaScript 的大型网站上，这可能会导致严重的性能问题，从而降低您的网站速度。

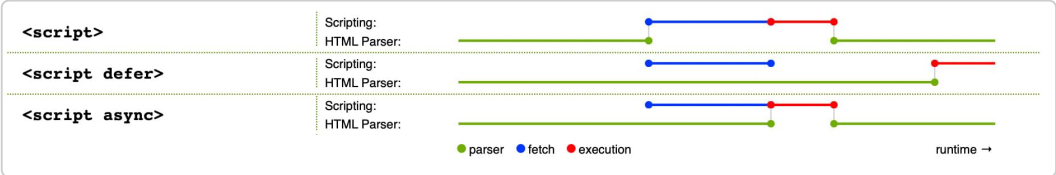
异步和延迟

实际上，我们可以使用两个现代功能来绕过阻塞脚本的问题——`async` 和 `defer` （我们在上面看到的）。让我们看看这两者之间的区别。

使用该 `async` 属性加载的脚本将在获取脚本时下载脚本而不会阻塞页面。但是，一旦下载完成，脚本就会执行，这会阻止页面呈现。您无法保证脚本将以任何特定顺序运行。最好 `async` 在页面中的脚本彼此独立运行并且不依赖于页面上的其他脚本时使用。

使用该属性加载的脚本 `defer` 将按照它们在页面上出现的顺序加载。它们在页面内容全部加载完毕后才会运行，如果您的脚本依赖于现有的 DOM（例如，它们修改页面上的一个或多个元素），这将很有用。

以下是不同脚本加载方法的直观表示，以及这对您的页面意味着什么：



此图像来自HTML 规范，根据 CC BY 4.0 许可条款复制并裁剪为缩小版本。

例如，如果您有以下脚本元素：

```
<script async src="js/vendor/jquery.js"></script>

<script async src="js/script2.js"></script>
```

mdn web docs

GitLab

用于软件创新的 One DevOps 平台。消除点解决方案蔓延。30 天免费试用。

Mozilla 广告

不想看广告?

您不能依赖脚本加载的顺序。 `jquery.js` 可能会在加载之前或之后加载 `script2.js` ， `script3.js` 如果是这种情况，这些脚本中依赖的任何函数 `jquery` 都会产生错误，因为 `jquery` 在脚本运行时不会定义。

`async` 当你有一堆后台脚本要加载时，应该使用它，而你只想尽快将它们安装到位。例如，也许您有一些游戏数据文件要加载，游戏实际开始时会需要这些文件，但现在您只想继续显示游戏介绍、标题和大厅，而不会被脚本加载阻止.

使用 `defer` 属性加载的脚本（见下文）将按照它们在页面中出现的顺序运行，并在脚本和内容下载后立即执行：

```
<script defer src="js/vendor/jquery.js"></script>

<script defer src="js/script2.js"></script>

<script defer src="js/script3.js"></script>
```

在第二个例子中，我们可以确定 `jquery.js` will load before `script2.js` 和 `script3.js` that `script2.js` will load before `script3.js` 。它们在页面

内容全部加载完毕后才会运行，如果您的脚本依赖于现有的 DOM（例如，它们修改页面上的多个元素之一），这将很有用。

总结一下：

- `async` 并且 `defer` 两者都指示浏览器在单独的线程中下载脚本，而页面的其余部分（DOM 等）正在下载，因此页面加载在获取过程中不会被阻止。
- 具有属性的脚本 `async` 将在下载完成后立即执行。这会阻塞页面并且不保证任何特定的执行顺序。
- 具有属性的脚本 `defer` 将按它们所在的顺序加载，并且仅在所有内容加载完成后才执行。
- 如果您的脚本应该立即运行并且它们没有任何依赖关系，那么请使用 `async`.
- 如果您的脚本需要等待解析并依赖于其他脚本和/或 DOM 就位，请使用加载它们 `defer` 并将它们的相应 `<script>` 元素按照您希望浏览器执行它们的顺序放置。

评论

与 HTML 和 CSS 一样，可以在 JavaScript 代码中写入注释，这些注释将被浏览器忽略，并且存在的目的是向您的开发人员同事提供有关代码如何工作的说明（以及您，如果您在之后返回到您的代码六个月，不记得你做了什么）。注释非常有用，您应该经常使用它们，尤其是对于大型应用程序。有两种类型：

- 单行注释写在双正斜杠 (//) 之后，例如

```
// I am a comment
```

- 在字符串 /* 和 */ 之间写入多行注释，例如

```
/*
  I am also
  a comment
*/
```

因此，例如，我们可以使用如下注释来注释上一个演示的 JavaScript：

```
// Function: creates a new paragraph and appends it to the
bottom of the HTML body.

function createParagraph() {
  const para = document.createElement("p");
  para.textContent = "You clicked the button!";
  document.body.appendChild(para);
}

/*
  1. Get references to all the buttons on the page in an array
format.
  2. Loop through all the buttons and add a click event listener
to each one.

  When any button is pressed, the createParagraph() function
will be run.
*/

const buttons = document.querySelectorAll("button");

for (const button of buttons) {
  button.addEventListener("click", createParagraph);
}
```

> **注意：**一般来说，注释多总比少好，但是如果你发现自己添加了很多注释来解释什么是变量（你的变量名可能应该更直观），或者解释非常简单的操作（也许你的代码过于复杂）。

概括

到此为止，这是您进入 JavaScript 世界的第一步。我们从理论开始，开始让您了解为什么要使用 JavaScript 以及您可以用它做什么。在此过程中，您看到了一些代码示例，并了解了 JavaScript 如何与您网站上的其余代码相适应，等等。

JavaScript 现在可能看起来有点令人生畏，但不要担心 - 在本课程中，我们将通过简单的步骤带您完成它，这些步骤将在以后有意义。在下一篇文章中，我们将直接进入实践，让您直接进入并构建您自己的 JavaScript 示例。

此页面最后修改于 2023 年 3 月 5 日由MDN 贡献者提供。