**页面生命周期：DOMContentLoaded，load，beforeunload，unload**

HTML 页面的生命周期包含三个重要事件：

* DOMContentLoaded —— 浏览器已完全加载 HTML，并构建了 DOM 树，但像 <img> 和样式表之类的外部资源可能尚未加载完成。
* load —— 浏览器不仅加载完成了 HTML，还加载完成了所有外部资源：图片，样式等。
* beforeunload/unload —— 当用户正在离开页面时。

每个事件都是有用的：

* DOMContentLoaded 事件 —— DOM 已经就绪，因此处理程序可以查找 DOM 节点，并初始化接口。
* load 事件 —— 外部资源已加载完成，样式已被应用，图片大小也已知了。
* beforeunload 事件 —— 用户正在离开：我们可以检查用户是否保存了更改，并询问他是否真的要离开。
* unload 事件 —— 用户几乎已经离开了，但是我们仍然可以启动一些操作，例如发送统计数据。

我们探索一下这些事件的细节。

**[DOMContentLoaded](https://zh.javascript.info/onload-ondomcontentloaded" \l "domcontentloaded)**

DOMContentLoaded 事件发生在 document 对象上。

我们必须使用 addEventListener 来捕获它：

document.addEventListener("DOMContentLoaded", ready);

// 不是 "document.onDOMContentLoaded = ..."

例如：

<script>

function ready() {

alert('DOM is ready');

// 图片目前尚未加载完成（除非已经被缓存），所以图片的大小为 0x0

alert(`Image size: ${img.offsetWidth}x${img.offsetHeight}`);

}

document.addEventListener("DOMContentLoaded", ready);

</script>

<img id="img" src="<https://en.js.cx/clipart/train.gif?speed=1&cache=0>">

在示例中，DOMContentLoaded 处理程序在文档加载完成后触发，所以它可以查看所有元素，包括它下面的 <img> 元素。

但是，它不会等待图片加载。因此，alert 显示其大小为零。

乍一看，DOMContentLoaded 事件非常简单。DOM 树准备就绪 —— 这是它的触发条件。它并没有什么特别之处。

**[DOMContentLoaded 和脚本](https://zh.javascript.info/onload-ondomcontentloaded" \l "domcontentloaded-he-jiao-ben)**

当浏览器处理一个 HTML 文档，并在文档中遇到 <script> 标签时，就会在继续构建 DOM 之前运行它。这是一种防范措施，因为脚本可能想要修改 DOM，甚至对其执行 document.write 操作，所以 DOMContentLoaded 必须等待脚本执行结束。

因此，DOMContentLoaded 肯定在下面的这些脚本执行结束之后发生：

<script>

document.addEventListener("DOMContentLoaded", () => {

alert("DOM ready!");

});

</script>

<script src="<https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.3.0/lodash.js>"></script>

<script>

alert("Library loaded, inline script executed");

</script>

在上面这个例子中，我们首先会看到 “Library loaded…”，然后才会看到 “DOM ready!”（所有脚本都已经执行结束）。

**不会阻塞 DOMContentLoaded 的脚本**

此规则有两个例外：

1. 具有 async 特性（attribute）的脚本不会阻塞 DOMContentLoaded，[稍后](https://zh.javascript.info/script-async-defer) 我们会讲到。
2. 使用 document.createElement('script') 动态生成并添加到网页的脚本也不会阻塞 DOMContentLoaded。

**[DOMContentLoaded 和样式](https://zh.javascript.info/onload-ondomcontentloaded" \l "domcontentloaded-he-yang-shi)**

外部样式表不会影响 DOM，因此 DOMContentLoaded 不会等待它们。

但这里有一个陷阱。如果在样式后面有一个脚本，那么该脚本必须等待样式表加载完成：

<link type="text/css" rel="stylesheet" href="style.css">

<script>

// 在样式表加载完成之前，脚本都不会执行

alert(getComputedStyle(document.body).marginTop);

</script>

原因是，脚本可能想要获取元素的坐标和其他与样式相关的属性，如上例所示。因此，它必须等待样式加载完成。

当 DOMContentLoaded 等待脚本时，它现在也在等待脚本前面的样式。

**[浏览器内建的自动填充](https://zh.javascript.info/onload-ondomcontentloaded" \l "liu-lan-qi-nei-jian-de-zi-dong-tian-chong)**

Firefox，Chrome 和 Opera 都会在 DOMContentLoaded 中自动填充表单。

例如，如果页面有一个带有登录名和密码的表单，并且浏览器记住了这些值，那么在 DOMContentLoaded 上，浏览器会尝试自动填充它们（如果得到了用户允许）。

因此，如果 DOMContentLoaded 被需要加载很长时间的脚本延迟触发，那么自动填充也会等待。你可能在某些网站上看到过（如果你使用浏览器自动填充）—— 登录名/密码字段不会立即自动填充，而是在页面被完全加载前会延迟填充。这实际上是 DOMContentLoaded 事件之前的延迟。

**[window.onload](https://zh.javascript.info/onload-ondomcontentloaded" \l "window-onload)**

当整个页面，包括样式、图片和其他资源被加载完成时，会触发 window 对象上的 load 事件。可以通过 onload 属性获取此事件。

下面的这个示例正确显示了图片大小，因为 window.onload 会等待所有图片加载完毕：

<script>

window.onload = function() { // 与此相同 window.addEventListener('load', (event) => {

alert('Page loaded');

// 此时图片已经加载完成

alert(`Image size: ${img.offsetWidth}x${img.offsetHeight}`);

};

</script>

<img id="img" src="<https://en.js.cx/clipart/train.gif?speed=1&cache=0>">

**[window.onunload](https://zh.javascript.info/onload-ondomcontentloaded" \l "windowonunload)**

当访问者离开页面时，window 对象上的 unload 事件就会被触发。我们可以在那里做一些不涉及延迟的操作，例如关闭相关的弹出窗口。

有一个值得注意的特殊情况是发送分析数据。

假设我们收集有关页面使用情况的数据：鼠标点击，滚动，被查看的页面区域等。

自然地，当用户要离开的时候，我们希望通过 unload 事件将数据保存到我们的服务器上。

有一个特殊的 navigator.sendBeacon(url, data) 方法可以满足这种需求，详见规范 <https://w3c.github.io/beacon/>。

它在后台发送数据，转换到另外一个页面不会有延迟：浏览器离开页面，但仍然在执行 sendBeacon。

使用方式如下：

let analyticsData = { /\* 带有收集的数据的对象 \*/ };

window.addEventListener("unload", function() {

navigator.sendBeacon("/analytics", JSON.stringify(analyticsData));

});

* 请求以 POST 方式发送。
* 我们不仅能发送字符串，还能发送表单以及其他格式的数据，在 [Fetch](https://zh.javascript.info/fetch) 一章有详细讲解，但通常它是一个字符串化的对象。
* 数据大小限制在 64kb。

当 sendBeacon 请求完成时，浏览器可能已经离开了文档，所以就无法获取服务器响应（对于分析数据来说通常为空）。

还有一个 keep-alive 标志，该标志用于在 [fetch](https://zh.javascript.info/fetch) 方法中为通用的网络请求执行此类“离开页面后”的请求。你可以在 [Fetch API](https://zh.javascript.info/fetch-api) 一章中找到更多相关信息。

如果我们要取消跳转到另一页面的操作，在这里做不到。但是我们可以使用另一个事件 —— onbeforeunload。

**[window.onbeforeunload](https://zh.javascript.info/onload-ondomcontentloaded" \l "window.onbeforeunload)**

如果访问者触发了离开页面的导航（navigation）或试图关闭窗口，beforeunload 处理程序将要求进行更多确认。

如果我们要取消事件，浏览器会询问用户是否确定。

你可以通过运行下面这段代码，然后重新加载页面来进行尝试：

window.onbeforeunload = function() {

return false;

};

由于历史原因，返回非空字符串也被视为取消事件。在以前，浏览器曾经将其显示为消息，但是根据 [现代规范](https://html.spec.whatwg.org/#unloading-documents) 所述，它们不应该这样。

这里有个例子：

window.onbeforeunload = function() {

return "There are unsaved changes. Leave now?";

};

它的行为已经改变了，因为有些站长通过显示误导性和恶意信息滥用了此事件处理程序。所以，目前一些旧的浏览器可能仍将其显示为消息，但除此之外 —— 无法自定义显示给用户的消息。

**[readyState](https://zh.javascript.info/onload-ondomcontentloaded" \l "readystate)**

如果我们将 DOMContentLoaded 事件处理程序设置在文档加载完成之后，会发生什么？

很自然地，它永远不会运行。

在某些情况下，我们不确定文档是否已经准备就绪。我们希望我们的函数在 DOM 加载完成时执行，无论现在还是以后。

document.readyState 属性可以为我们提供当前加载状态的信息。

它有 3 个可能值：

* loading —— 文档正在被加载。
* interactive —— 文档被全部读取。
* complete —— 文档被全部读取，并且所有资源（例如图片等）都已加载完成。

所以，我们可以检查 document.readyState 并设置一个处理程序，或在代码准备就绪时立即执行它。

像这样：

function work() { /\*...\*/ }

if (document.readyState == 'loading') {

// 仍在加载，等待事件

document.addEventListener('DOMContentLoaded', work);

} else {

// DOM 已就绪！

work();

}

还有一个 readystatechange 事件，会在状态发生改变时触发，因此我们可以打印所有这些状态，就像这样：

// 当前状态

console.log(document.readyState);

// 状态改变时打印它

document.addEventListener('readystatechange', () => console.log(document.readyState));

readystatechange 事件是跟踪文档加载状态的另一种机制，它很早就存在了。现在则很少被使用。

但是为了完整起见，让我们看看完整的事件流。

这是一个带有 <iframe>，<img> 和记录事件的处理程序的文档：

<script>

log('initial readyState:' + document.readyState);

document.addEventListener('readystatechange', () => log('readyState:' + document.readyState));

document.addEventListener('DOMContentLoaded', () => log('DOMContentLoaded'));

window.onload = () => log('window onload');

</script>

<iframe src="iframe.html" onload="log('iframe onload')"></iframe>

<img src="<http://en.js.cx/clipart/train.gif>" id="img">

<script>

img.onload = () => log('img onload');

</script>

此示例运行 [在 sandbox 中](https://plnkr.co/edit/ct5SNvrHCA75b2KZ?p=preview)。

典型输出：

1. [1] initial readyState:loading
2. [2] readyState:interactive
3. [2] DOMContentLoaded
4. [3] iframe onload
5. [4] img onload
6. [4] readyState:complete
7. [4] window onload

方括号中的数字表示发生这种情况的大致时间。标有相同数字的事件几乎是同时发生的（± 几毫秒）。

* 在 DOMContentLoaded 之前，document.readyState 会立即变成 interactive。它们俩的意义实际上是相同的。
* 当所有资源（iframe 和 img）都加载完成后，document.readyState 变成 complete。这里我们可以发现，它与 img.onload（img 是最后一个资源）和 window.onload 几乎同时发生。转换到 complete 状态的意义与 window.onload 相同。区别在于 window.onload 始终在所有其他 load 处理程序之后运行。

**[总结](https://zh.javascript.info/onload-ondomcontentloaded" \l "zong-jie)**

页面生命周期事件：

* 当 DOM 准备就绪时，document 上的 DOMContentLoaded 事件就会被触发。在这个阶段，我们可以将 JavaScript 应用于元素。
  + 诸如 <script>...</script> 或 <script src="..."></script> 之类的脚本会阻塞 DOMContentLoaded，浏览器将等待它们执行结束。
  + 图片和其他资源仍然可以继续被加载。
* 当页面和所有资源都加载完成时，window 上的 load 事件就会被触发。我们很少使用它，因为通常无需等待那么长时间。
* 当用户想要离开页面时，window 上的 beforeunload 事件就会被触发。如果我们取消这个事件，浏览器就会询问我们是否真的要离开（例如，我们有未保存的更改）。
* 当用户最终离开时，window 上的 unload 事件就会被触发。在处理程序中，我们只能执行不涉及延迟或询问用户的简单操作。正是由于这个限制，它很少被使用。我们可以使用 navigator.sendBeacon 来发送网络请求。
* document.readyState 是文档的当前状态，可以在 readystatechange 事件中跟踪状态更改：
  + loading —— 文档正在被加载。
  + interactive —— 文档已被解析完成，与 DOMContentLoaded 几乎同时发生，但是在 DOMContentLoaded 之前发生。
  + complete —— 文档和资源均已加载完成，与 window.onload 几乎同时发生，但是在 window.onload 之前发生。