# JavaScript 类型转换

## typeof 操作符

typeof "John"                 // 返回 string   
typeof 3.14                   // 返回 number  
typeof NaN                    // 返回 number  
typeof false                  // 返回 boolean  
typeof [1,2,3,4]              // 返回 object  
typeof {name:'John', age:34}  // 返回 object  
typeof new Date()             // 返回 object  
typeof function () {}         // 返回 function  
typeof myCar                  // 返回 undefined (if myCar is not declared)  
typeof null                   // 返回 object

请注意：

* NaN 的数据类型是 number
* 数组(Array)的数据类型是 object
* 日期(Date)的数据类型为 object
* null 的数据类型是 object
* 未定义变量的数据类型为 undefined

如果对象是 JavaScript Array 或 JavaScript Date ，我们就无法通过 **typeof** 来判断他们的类型，因为都是 返回 Object。

## 将数字转换为字符串

全局方法 **String()** 可以将数字转换为字符串。

该方法可用于任何类型的数字，字母，变量，表达式：

String(x)         // 将变量 x 转换为字符串并返回  
String(123)       // 将数字 123 转换为字符串并返回  
String(100 + 23)  // 将数字表达式转换为字符串并返回

## 将字符串转换为数字

全局方法 **Number()** 可以将字符串转换为数字。

字符串包含数字(如 "3.14") 转换为数字 (如 3.14).

空字符串转换为 0。

其他的字符串会转换为 NaN (不是个数字)。

Number("3.14")    // 返回 3.14  
Number(" ")       // 返回 0   
Number("")        // 返回 0  
Number("99 88")   // 返回 NaN

parseFloat()

定义和用法

parseFloat() 函数可解析一个字符串，并返回一个浮点数。

该函数指定字符串中的首个字符是否是数字。如果是，则对字符串进行解析，直到到达数字的末端为止，然后以数字返回该数字，而不是作为字符串。

<script type="text/javascript">

document.write(parseFloat("10"))

document.write(parseFloat("10.00"))

document.write(parseFloat("10.33"))

document.write(parseFloat("34 45 66"))

document.write(parseFloat(" 60 "))

document.write(parseFloat("40 years"))

document.write(parseFloat("He was 40"))

</script>

输出

10

10

10.33

34

60

40

NaN

## ParseInt

## parseInt()

在判断字符串是否是数字值前，parseInt() 和 parseFloat() 都会仔细分析该字符串。

parseInt() 方法首先查看位置 0 处的字符，判断它是否是个有效数字；如果不是，该方法将返回 NaN，不再继续执行其他操作。但如果该字符是有效数字，该方法将查看位置 1 处的字符，进行同样的测试。这一过程将持续到发现非有效数字的字符为止，此时 parseInt() 将把该字符之前的字符串转换成数字。

例如，如果要把字符串 "12345red" 转换成整数，那么 parseInt() 将返回 12345，因为当它检查到字符 r 时，就会停止检测过程。

字符串中包含的数字字面量会被正确转换为数字，比如 "0xA" 会被正确转换为数字 10。不过，字符串 "22.5" 将被转换成 22，因为对于整数来说，小数点是无效字符。

var iNum1 = parseInt("12345red"); //返回 12345

var iNum1 = parseInt("0xA"); //返回 10

var iNum1 = parseInt("56.9"); //返回 56

var iNum1 = parseInt("red"); //返回 NaN

## isNaN() 函数

## 定义和用法

isNaN() 函数用于检查其参数是否是非数字值。

说明

isNaN() 函数可用于判断其参数是否是 NaN，该值表示一个非法的数字（比如被 0 除后得到的结果）。

如果把 NaN 与任何值（包括其自身）相比得到的结果均是 false，所以要判断某个值是否是 NaN，不能使用 == 或 运算符。正因为如此，isNaN() 函数是必需的。

## 提示和注释

**提示：**isNaN() 函数通常用于检测 parseFloat() 和 parseInt() 的结果，以判断它们表示的是否是合法的数字。当然也可以用 isNaN() 函数来检测算数错误，比如用 0 作除数的情况。

<script>

document.write(isNaN(123));

document.write(isNaN(-1.23));

document.write(isNaN(5-2));

document.write(isNaN(0));

document.write(isNaN("Hello"));

document.write(isNaN("2005/12/12"));

</script>

输出：

false

false

false

false

true

true

## 一元运算符 +

**Operator +**可用于将变量转换为数字：

var y = "5";      // y 是一个字符串  
var x = + y;      // x 是一个数字

如果变量不能转换，它仍然会是一个数字，但值为 NaN (不是一个数字):

var y = "John";   // y 是一个字符串  
var x = + y;      // x 是一个数字 (NaN)

## 将布尔值转换为数字

全局方法 **Number()** 可将布尔值转换为数字。

Number(false)     // 返回 0  
Number(true)      // 返回 1

## 将日期转换为数字

全局方法 **Date()** 可将日期转换为数字。

d = new Date();  
Number(d)          // 返回 1404568027739

日期方法 **getTime()** 也有相同的效果。

d = new Date();  
d.getTime()        // 返回 1404568027739

## 自动转换类型 Type Conversion

当 JavaScript 尝试操作一个 "错误" 的数据类型时，会自动转换为 "正确" 的数据类型。

以下输出结果不是你所期望的：

5 + null    // 返回 5         because null is converted to 0  
"5" + null  // 返回"5null"   because null is converted to "null"  
"5" + 1     // 返回 "51"      because 1 is converted to "1"    
"5" - 1     // 返回 4         because "5" is converted to 5

# JavaScript HTML DOM

通过 HTML DOM，可访问 JavaScript HTML 文档的所有元素。

通过可编程的对象模型，JavaScript 获得了足够的能力来创建动态的 HTML。

* JavaScript 能够改变页面中的所有 HTML 元素
* JavaScript 能够改变页面中的所有 HTML 属性
* JavaScript 能够改变页面中的所有 CSS 样式
* JavaScript 能够对页面中的所有事件做出反应

**查找 HTML 元素**

通常，通过 JavaScript，您需要操作 HTML 元素。

为了做到这件事情，您必须首先找到该元素。有三种方法来做这件事：

* 通过 id 找到 HTML 元素
* 通过标签名找到 HTML 元素
* 通过类名找到 HTML 元素

<p id="intro">你好世界!</p>

<p>该实例展示了 <b>getElementById</b> 方法!</p>

<script>

x=document.getElementById("intro");

document.write("<p>文本来自 id 为 intro 段落: " + x.innerHTML + "</p>");

</script>

<p>你好世界!</p>

<div id="main">

<p> DOM 是非常有用的。</p>

<p>该实例展示了 <b>getElementsByTagName</b> 方法</p>

</div>

<script>

var x=document.getElementById("main");

var y=x.getElementsByTagName("p");

document.write('id="main"元素中的第一个段落为：' + y[1].innerHTML);

</script>

## 通过类名找到 HTML 元素

本例通过 **[getElementsByClassName](http://www.w3cschool.cn/jsref/met-document-getelementsbyclassname.html" \t "_blank)** 函数来查找 class="intro" 的元素：

<p class="intro">你好世界!</p>

<p>该实例展示了 <b>getElementsByClassName</b> 方法!</p>

<script>

x=document.getElementsByClassName("intro");

document.write("<p>文本来自 class 为 intro 段落: " + x[0].innerHTML + "</p>");

</script>

<p><b>注意：</b>Internet Explorer 8 及更早 IE 版本不支持 getElementsByClassName() 方法

**HTML DOM 教程**

在本教程接下来的篇幅中，您将学到：

* 如何改变 HTML 元素的内容 (innerHTML)
* 如何改变 HTML 元素的样式 (CSS)
* 如何对 HTML DOM 事件对出反应
* 如何添加或删除 HTML 元素

## 改变 HTML 输出流

JavaScript 能够创建动态的 HTML 内容：

在 JavaScript 中，document.write() 可用于直接向 HTML 输出流写内容。

document.write(Date());

绝对不要在文档加载完成之后使用 document.write()。这会覆盖该文档。

## 改变 HTML 内容

修改 HTML 内容的最简单的方法时使用 innerHTML 属性。

如需改变 HTML 元素的内容，请使用这个语法：

document.getElementById(id).innerHTML=new HTML

<p id="p1">Hello World!</p>

<script>

document.getElementById("p1").innerHTML="新文本!";

</script>

<p>以上段落通过脚本修改文本。</p>

<h1 id="header">旧标题</h1>

<script>

var element=document.getElementById("header");

element.innerHTML="新标题";

</script>

<p>"旧标题" 被 "新标题" 取代。</p>

实例讲解：

* 上面的 HTML 文档含有 id="header" 的 <h1> 元素
* 我们使用 HTML DOM 来获得 id="header" 的元素
* JavaScript 更改此元素的内容 (innerHTML)

## 改变 HTML 属性

如需改变 HTML 元素的属性，请使用这个语法：

document.getElementById(id).attribute=new value

本例改变了 <img> 元素的 src 属性：

<img id="image" src="/attachments/cover/cover\_docker.png" width="160" height="120">

<script>

document.getElementById("image").src="/attachments/cover/cover\_javascript.png";

</script>

<p>原图片为 cover\_docker.png,脚本将图片修改为 cover\_javascript.png</p>

实例讲解：

* 上面的 HTML 文档含有 id="image" 的 <img> 元素
* 我们使用 HTML DOM 来获得 id="image" 的元素
* JavaScript 更改此元素的属性（把 "smiley.gif" 改为 "landscape.jpg"）

**HTML DOM 改变 CSS**

## 改变 HTML 样式

如需改变 HTML 元素的样式，请使用这个语法：

document.getElementById(id).style.property=new style

<p id="p1">Hello World!</p>

<p id="p2">Hello World!</p>

<script>

document.getElementById("p2").style.color="blue";

document.getElementById("p2").style.fontFamily="Arial";

document.getElementById("p2").style.fontSize="larger";

</script>

<p>以上段落通过脚本修改。</p>

<h1 id="id1">我的标题 1</h1>

<button type="button" onclick="document.getElementById('id1').style.color='red'">

点我!</button>

## JavaScript HTML DOM 事件

HTML DOM 使 JavaScript 有能力对 HTML 事件做出反应。

## 对事件做出反应

我们可以在事件发生时执行 JavaScript，比如当用户在 HTML 元素上点击时。

如需在用户点击某个元素时执行代码，请向一个 HTML 事件属性添加 JavaScript 代码：

onclick=JavaScript

HTML 事件的例子：

* 当用户点击鼠标时
* 当网页已加载时
* 当图像已加载时
* 当鼠标移动到元素上时
* 当输入字段被改变时
* 当提交 HTML 表单时
* 当用户触发按键时

<h1 onclick="this.innerHTML='Ooops!'">点击文本!</h1>

<script>

function changetext(id){

id.innerHTML="Ooops!";

}

</script>

<h1 onclick="changetext(this)">点击文本!</h1>

## 使用 HTML DOM 来分配事件

HTML DOM 允许您使用 JavaScript 来向 HTML 元素分配事件：

<p>点击按钮执行 <em>displayDate()</em> 函数.</p>

<button id="myBtn">点这里</button>

<script>

document.getElementById("myBtn").onclick=function(){displayDate()};

function displayDate(){

document.getElementById("demo").innerHTML=Date();

}

</script>

<p id="demo"></p>

在上面的例子中，名为 displayDate 的函数被分配给 id=myButn" 的 HTML 元素。

按钮点击时Javascript函数将会被执行。

## onload 和 onunload 事件

onload 和 onunload 事件会在用户进入或离开页面时被触发。

onload 事件可用于检测访问者的浏览器类型和浏览器版本，并基于这些信息来加载网页的正确版本。

onload 和 onunload 事件可用于处理 cookie。

<script>

function checkCookies(){

if (navigator.cookieEnabled==true){

alert("Cookies 可用")

}

else{

alert("Cookies 不可用")

}

}

</script>

<p>弹窗-提示浏览器cookie是否可用。</p>

function mymessage(){

alert("消息在 onload 事件触发后弹出。");

}

## onmouseover 和 onmouseout 事件

onmouseover 和 onmouseout 事件可用于在用户的鼠标移至 HTML 元素上方或移出元素时触发函数。

<div

onmouseover="mOver(this)"

onmouseout="mOut(this)" style="background-color:#D94A38;width:120px;height:20px;padding:40px;">

MouseOver Me

</div>

<script>

function mOver(obj){

obj.innerHTML="Thank You"

}

function mOut(obj){

obj.innerHTML="Mouse Over Me"

}

</script>

## onmousedown、onmouseup 以及 onclick 事件

onmousedown, onmouseup 以及 onclick 构成了鼠标点击事件的所有部分。首先当点击鼠标按钮时，会触发 onmousedown 事件，当释放鼠标按钮时，会触发 onmouseup 事件，最后，当完成鼠标点击时，会触发 onclick 事件。

# JavaScript Window

## Window 对象

所有浏览器都支持 window 对象。它表示浏览器窗口。

所有 JavaScript 全局对象、函数以及变量均自动成为 window 对象的成员。

全局变量是 window 对象的属性。

全局函数是 window 对象的方法。

甚至 HTML DOM 的 document 也是 window 对象的属性之一：

window.document.getElementById("header");

与此相同：

document.getElementById("header");

**Window 尺寸**

有三种方法能够确定浏览器窗口的尺寸（浏览器的视口，不包括工具栏和滚动条）。

对于Internet Explorer、Chrome、Firefox、Opera 以及 Safari：

* window.innerHeight - 浏览器窗口的内部高度
* window.innerWidth - 浏览器窗口的内部宽度

对于 Internet Explorer 8、7、6、5：

* document.documentElement.clientHeight
* document.documentElement.clientWidth

或者

* document.body.clientHeight
* document.body.clientWidth

实用的 JavaScript 方案（涵盖所有浏览器）：

<p id="demo"></p>

<script>

var w=window.innerWidth

|| document.documentElement.clientWidth

|| document.body.clientWidth;

var h=window.innerHeight

|| document.documentElement.clientHeight

|| document.body.clientHeight;

x=document.getElementById("demo");

x.innerHTML="浏览器window宽度: " + w + ", 高度: " + h + "。"

</script>

该例显示浏览器窗口的高度和宽度：（不包括工具栏/滚动条）

**其他 Window 方法**

一些其他方法：

* window.open() - 打开新窗口
* window.close() - 关闭当前窗口
* window.moveTo() - 移动当前窗口
* window.resizeTo() - 调整当前窗口的尺寸

## JavaScript 弹窗

可以在 JavaScript 中创建三种消息框：警告框、确认框、提示框。

## 警告框

警告框经常用于确保用户可以得到某些信息。

当警告框出现后，用户需要点击确定按钮才能继续进行操作。

window.alert("sometext");

**window.alert()** 方法可以不带上window对象，直接使用**alert()**方法。

## 确认框

确认框通常用于验证是否接受用户操作。

当确认卡弹出时，用户可以点击 "确认" 或者 "取消" 来确定用户操作。

当你点击 "确认", 确认框返回 true， 如果点击 "取消", 确认框返回 false。

window.confirm("sometext");

**window.confirm()** 方法可以不带上window对象，直接使用**confirm()**方法。

<p>点击按钮，显示确认框。</p>

<button onclick="myFunction()">点我</button>

<p id="demo"></p>

<script>

function myFunction(){

var x;

var r=confirm("按下按钮!");

if (r==true){

x="你按下了\"确定\"按钮!";

}

else{

x="你按下了\"取消\"按钮!";

}

document.getElementById("demo").innerHTML=x;

}

</script>

**JavaScript 计时事件**

javaScript 一个设定的时间间隔之后来执行代码

我们称之为计时事件

通过使用 JavaScript，我们有能力做到在一个设定的时间间隔之后来执行代码，而不是在函数被调用后立即执行。我们称之为计时事件。

在 JavaScritp 中使用计时事件是很容易的，两个关键方法是:

* setInterval() - 间隔指定的毫秒数不停地执行指定的代码。
* setTimeout() - 暂停指定的毫秒数后执行指定的代码

**Note:** setInterval() 和 setTimeout() 是 HTML DOM Window对象的两个方法。

## setInterval() 方法

setInterval() 间隔指定的毫秒数不停地执行指定的代码

window.setInterval("javascript function",milliseconds);

**window.setInterval()** 方法可以不使用window前缀，直接使用函数**setInterval()**。

setInterval() 第一个参数是函数（function）。

第二个参数间隔的毫秒数

**注意:** 1000 毫秒是一秒。

<p>单击按钮每3秒出现一个“Hello”警告框。</p>

<p>再次点击警告框，经过3秒出现新的警告框，这将一直执行 ...</p>

<button onclick="myFunction()">点我</button>

<script>

function myFunction(){

setInterval(function(){alert("Hello")},3000);

}

</script>

## 如何停止执行?

clearInterval() 方法用于停止 setInterval() 方法执行的函数代码。

window.clearInterval(intervalVariable)

**window.clearInterval()** 方法可以不使用window前缀，直接使用函数**clearInterval()**。

要使用 clearInterval() 方法, 在创建计时方法时你必须使用全局变量：

myVar=setInterval("javascript function",milliseconds);

然后你可以使用clearInterval() 方法来停止执行。

<p>页面上显示时钟：</p>

<p id="demo"></p>

<button onclick="myStopFunction()">停止时钟</button>

<script>

var myVar=setInterval(function(){myTimer()},1000);

function myTimer(){

var d=new Date();

var t=d.toLocaleTimeString();

document.getElementById("demo").innerHTML=t;

}

function myStopFunction(){

clearInterval(myVar);

}

</script>

## setTimeout() 方法

window.setTimeout("javascript 函数",毫秒数);

setTimeout() 方法会返回某个值。在上面的语句中，值被储存在名为 t 的变量中。假如你希望取消这个 setTimeout()，你可以使用这个变量名来指定它。

setTimeout() 的第一个参数是含有 JavaScript 语句的字符串。这个语句可能诸如 "alert('5 seconds!')"，或者对函数的调用，诸如 alertMsg()"。

第二个参数指示从当前起多少毫秒后执行第一个参数。

提示：1000 毫秒等于一秒。

<p>点击按钮，在等待 3 秒后弹出 "Hello"。</p>

<button onclick="myFunction()">点我</button>

<script>

function myFunction(){

setTimeout(function(){alert("Hello")},3000);

}

</script>

## 如何停止执行?

clearTimeout() 方法用于停止执行setTimeout()方法的函数代码。

window.clearTimeout(timeoutVariable)

**window.clearTimeout()** 方法可以不使用window 前缀。

要使用clearTimeout() 方法, 你必须在创建超时方法中（setTimeout）使用全局变量:

myVar=setTimeout("javascript function",milliseconds);

如果函数还未被执行，你可以使用 clearTimeout() 方法来停止执行函数代码。

<p>点击第一个按钮等待3秒后出现"Hello"弹框。</p>

<p>点击第二个按钮来阻止第一个函数运行。（你必须在3秒之前点击它）。</p>

<button onclick="myFunction()">点我</button>

<button onclick="myStopFunction()">停止弹框</button>

<script>

var myVar;

function myFunction(){

myVar=setTimeout(function(){alert("Hello")},3000);

}

function myStopFunction(){

clearTimeout(myVar);

}

# JavaScript 对象

JavaScript 中的所有事物都是对象：字符串、数值、数组、函数...

此外，JavaScript 允许自定义对象

**所有事物都是对象**

JavaScript 提供多个内建对象，比如 String、Date、Array 等等。 对象只是带有属性和方法的特殊数据类型。

* 布尔型可以是一个对象。
* 数字型可以是一个对象。
* 字符串也可以是一个对象
* 日期是一个对象
* 数学和正则表达式也是对象
* 数组是一个对象
* 甚至函数也可以是对象

对象只是一种特殊的数据。对象拥有**属性**和**方法**。

## 访问对象的属性

属性是与对象相关的值。

访问对象属性的语法是：

objectName.propertyName

这个例子使用了 String 对象的 length 属性来获得字符串的长度：

var message="Hello World!";  
var x=message.length;

## 访问对象的方法

方法是能够在对象上执行的动作。

您可以通过以下语法来调用方法：

objectName.methodName()

这个例子使用了 String 对象的 toUpperCase() 方法来将文本转换为大写：

var message="Hello world!";  
var x=message.toUpperCase();

**创建 JavaScript 对象**

通过 JavaScript，您能够定义并创建自己的对象。

创建新对象有两种不同的方法：

* 定义并创建对象的实例
* 使用函数来定义对象，然后创建新的对象实例

## 创建直接的实例

这个例子创建了对象的一个新实例，并向其添加了四个属性：

person=new Object();  
person.firstname="John";  
person.lastname="Doe";  
person.age=50;  
person.eyecolor="blue";

替代语法（使用对象 literals）

person={firstname:"John",lastname:"Doe",age:50,eyecolor:"blue"};

## 使用对象构造器

本例使用函数来构造对象：

<script>

function person(firstname,lastname,age,eyecolor){

this.firstname=firstname;

this.lastname=lastname;

this.age=age;

this.eyecolor=eyecolor;

}

myFather=new person("John","Doe",50,"blue");

document.write(myFather.firstname + " is " + myFather.age + " years old.");

</script>

在JavaScript中，this通常指向的是我们正在执行的函数本身，或者是指向该函数所属的对象（运行时）

## 把属性添加到 JavaScript 对象

您可以通过为对象赋值，向已有对象添加新属性：

假设 personObj 已存在 - 您可以为其添加这些新属性：firstname、lastname、age 以及 eyecolor：

person.firstname="John";  
person.lastname="Doe";  
person.age=30;  
person.eyecolor="blue";  
x=person.firstname;

## 把方法添加到 JavaScript 对象

方法只不过是附加在对象上的函数。

在构造器函数内部定义对象的方法：

function person(firstname,lastname,age,eyecolor){

this.firstname=firstname;

this.lastname=lastname;

this.age=age;

this.eyecolor=eyecolor;

this.changeName=changeName;

function changeName(name){

this.lastname=name;

}

}

myMother=new person("Sally","Rally",48,"green");

myMother.changeName("Doe");

document.write(myMother.lastname);

changeName() 函数 name 的值赋给 person 的 lastname 属性。

## 避免低效率的脚本位置

HTML 4 规范指出 <script> 标签可以放在 HTML 文档的<head>或<body>中，并允许出现多次。Web 开发人员一般习惯在 <head> 中加载外链的 JavaScript，接着用 <link> 标签用来加载外链的 CSS 文件或者其他页面信息。

低效率脚本位置示例：

<html>

<head>

<title>Source Example</title>

<script type="text/javascript" src="script1.js"></script>

<script type="text/javascript" src="script2.js"></script>

<script type="text/javascript" src="script3.js"></script>

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

</head>

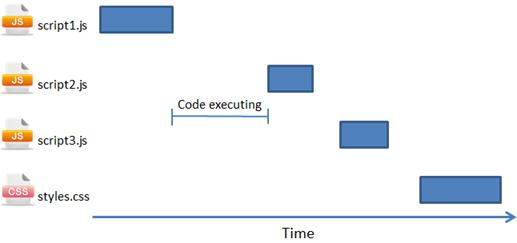
<body>

<p>Hello world!</p>

</body>

</html>

然而这种常规的做法却隐藏着严重的性能问题。在清单 2 的示例中，当浏览器解析到 <script> 标签（第 4 行）时，浏览器会停止解析其后的内容，而优先下载脚本文件，并执行其中的代码，这意味着，其后的 styles.css 样式文件和<body>标签都无法被加载，由于<body>标签无法被加载，那么页面自然就无法渲染了。因此在该 JavaScript 代码完全执行完之前，页面都是一片空白。下图描述了页面加载过程中脚本和样式文件的下载过程。



我们可以发现一个有趣的现象：第一个 JavaScript 文件开始下载，与此同时阻塞了页面其他文件的下载。此外，从 script1.js 下载完成到 script2.js 开始下载前存在一个延时，这段时间正好是 script1.js 文件的执行过程。每个文件必须等到前一个文件下载并执行完成才会开始下载。在这些文件逐个下载过程中，用户看到的是一片空白的页面。  
  
从 IE 8、Firefox 3.5、Safari 4 和 Chrome 2 开始都允许并行下载 JavaScript 文件。这是个好消息，因为<script>标签在下载外部资源时不会阻塞其他<script>标签。遗憾的是，JavaScript 下载过程仍然会阻塞其他资源的下载，比如样式文件和图片。尽管脚本的下载过程不会互相影响，但页面仍然必须等待所有 JavaScript 代码下载并执行完成才能继续。因此，尽管最新的浏览器通过允许并行下载提高了性能，但问题尚未完全解决，脚本阻塞仍然是一个问题。  
  
由于脚本会阻塞页面其他资源的下载，因此推荐将所有<script>标签尽可能放到<body>标签的底部，以尽量减少对整个页面下载的影响。

推荐的代码放置位置示例：

<head>

<title>Source Example</title>

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

</head>

<body>

<p>Hello world!</p>

<!-- Example of efficient script positioning -->

<script type="text/javascript" src="script1.js"></script>

<script type="text/javascript" src="script2.js"></script>

<script type="text/javascript" src="script3.js"></script>

</body>

</html>

这段代码展示了在 HTML 文档中放置<script>标签的推荐位置。尽管脚本下载会阻塞另一个脚本，但是页面的大部分内容都已经下载完成并显示给了用户，因此页面下载不会显得太慢。这是优化 JavaScript 的首要规则：将脚本放在底部。