# 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

## JavaScript 对象

对象只是一种特殊的数据。对象拥有**属性**和**方法**。

属性也可以是对象，字符串，数值等等

## 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：

字面量创建

* 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

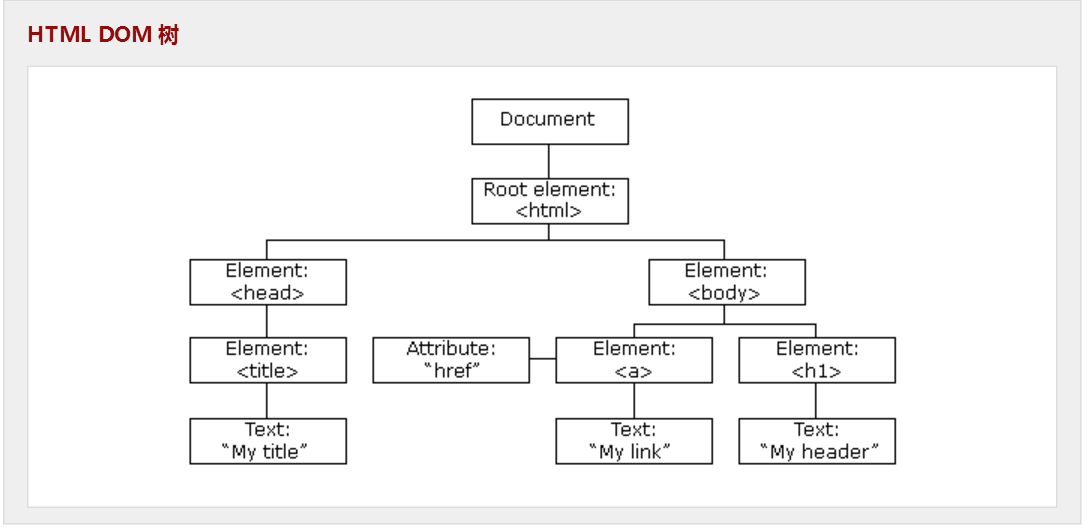
## 创建直接的实例

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"};



对象和基本数据类型赋值问题

值的副本，值的引用

## 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;

}

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。

## 基本类型的对象特殊问题

## 基本数据类型：Undefined、Null、Boolean、Number、String

## 基本类型值是指简单的数据段，5种基本类型是按值访问的，因为可以操作保存在变量中的实际值

## 基本类型的值在内存中占据固定大小的空间，被保存在栈内存中。从一个变量向另一个变量复制基本类型的值，会创建这个值的一个副本

## 不能给基本类型的值添加属性

## 引用数据类型：Object

## 引用类型值是指那些可以由多个值构成的对象。js不允许直接访问内存中的位置，也就是不能直接访问操作对象的内存空间。在操作对象时，实际上是在操作对象的引用而不是实际的对象

## 引用类型的值是对象，保存在堆内存中，包含引用类型值的变量实际上包含的并不是对象本身，而是一个指向该对象的指针。从一个变量向另一个变量复制引用类型的值，复制的其实是指针，因此两个变量最终都指向同一个对象。

## 对于引用类型的值，可以为其添加属性和方法，也可以改变和删除其属性和方法

## 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>

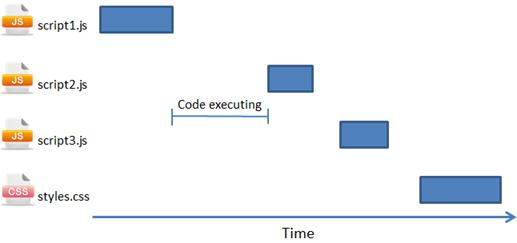
<body>

<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。