对象

对象的创建

对象是一个包含相关数据和方法的集合（通常由一些变量和函数组成，我们称之为对象里面的属性和方法）对象的创建有两种形式：

> let obj=new Object(); let obj1={};

向对象中添加属性

对象有属性和属性值（属性名不能重复），添加属性也有两种方法。

> obj.name=“张三”; obj[“age”]=18; console.log(obj);

向对象中添加方法

> obj.say=function(){console.log(“我是张三”)} console.log(obj.say());

对象的遍历

> let obj={name:“张三”,age:18,height:“180cm”}; for(let k in obj){ console.log(obj[k]); }

练习

请用对象的形式创建一个名字为可可的狗对象，具体信息如下：

名称(name)：可可 类型（type）：阿拉斯加犬 年龄(age)：5岁 颜色(color)：红色（添加的属性） 技能(skill)：输出“汪汪叫、摇尾巴”（添加的方法）> let dog={ name:“可可”，type:“阿拉斯加犬”，age:“5岁” } dog.color=“红色” dog.skill=function(){ console.log(“汪汪叫，摇尾巴”) }

console与控制台

Console 对象用于 JavaScript 调试。JavaScript 原生中默认是没有 Console 对象，这是宿主对象（也就是浏览器）提供的内置对象。 用于访问调试控制台, 在不同的浏览器里效果可能不同。Console 对象常见的两个用途

- 显示网页代码运行时的错误信息。
- 提供了一个命令行接口，用来与网页代码互动。

打开控制台的方式

- 直接按 F12
- 鼠标右键，点击“检查”

console对象方法

console.log 是最常用的

console.log方法用于在控制台输出信息。它可以接受一个或多个参数，将它们连接起来输出。

> console.log(‘a’, ‘b’, ‘c’)

自动在每次输出的结尾，添加换行符

>

> console.log(1); console.log(2); console.log(3);

console上述的集中度支持printf的占位符格式，支持的占位符有：字符（%s）、整数（%d或%i）、浮点数（%f）和对象（%o）：

> console.log(“%s年%d月%d日”，“2023”,1,2);

%c 占位符是最常用的。使用 %c 占位符时，对应的后面的参数必须是 CSS 语句，用来对输出内容进行 CSS 渲染。常见的输出方式有两种： 图片输出 文字样式输出

> console.log(“%c新年快乐！”， “color: red; font-size: 20px”);

console.info是console.log方法的别名，用法完全一样。

console.dir

dir方法用来对一个对象进行检查（inspect），并以易于阅读和打印的格式显示。

> let obj = { name: “张三”, age: 18, height: “180cm” }; console.dir(obj);

上面代码显示dir方法的输出结果，比log方法更易读，信息也更丰富。

console.dirml

dirxml方法主要用于以目录树的形式，显示 DOM 节点。

```
<body>
    <div id="info">
        <p>新年快乐！</p>
    </div>
</body>
</html>
<script>
let info = document.getElementById('info');
 console.dirxml(info);
</script>
```

console.assert

console.assert方法主要用于程序运行过程中，进行条件判断，如果不满足条件，就显示一个错误，但不会中断程序执行。这样就相当于提示用户，内部状态不正确。

它接受两个参数，第一个参数是表达式，第二个参数是字符串。只有当第一个参数为false，才会提示有错误，在控制台输出第二个参数，否则不会有任何结果

> console.assert(false, ‘判断条件不成立’)

console.trace

console.trace方法显示当前执行的代码在堆栈中的调用路径。 >function add(a,b){ > console.trace(); >return a+b; } >let x = add3(1,1); > function add3(a,b){return add2(a,b);} > function add2(a,b){return add1(a,b);} > function add1(a,b){return add(a,b);}

console.group()，console.groupEnd()，console.groupCollapsed()

console.group和console.groupEnd这两个方法用于将显示的信息分组。它只在输出大量信息时有用，分在一组的信息，可以用鼠标折叠/展开

>

```
console.group（‘一级分组’）; console.log（‘一级分组的内容’）; console.group（‘二级分组’）; console.log（‘二级分组的内容’）; console.groupEnd(); // 二级分组结束 console.groupEnd(); // 一级分组结束
```

console.time()，console.timeEnd()

这两个方法用于计时，可以算出一个操作所花费的准确时间。

```
console.time（‘计时’）; let array= new Array(1000000); for (let i = array.length - 1; i >= 0; i–) { array[i] = new Object(); }; console.timeEnd（‘计时’）;
```

console.count()

count方法用于计数，输出它被调用了多少次

```
function greet(user) { console.count(); return ‘hi’ + user; } greet（‘bob’）greet（‘alice’）greet（‘bob’）
```

console.table

对于某些复合类型的数据，console.table方法可以将其转为表格显示。

```
let languages = [ { name: “JavaScript”，age: 19}, { name: “TypeScript”，age: 20 }, { name: “CoffeeScript”，age: 21 }]; console.table(languages);
```

console.warn和console.error

warn方法和error方法也是在控制台输出信息，它们与log方法的不同之处在于，warn方法输出信息时，在最前面加一个黄色三角，表示警告；error方法输出信息时，在最前面加一个红色的叉，表示出错。同时，还会高亮显示输出文字和错误发生的堆栈。其他方面都一样。

```
console.error（‘Error: %s (%i)’，‘Server is not responding’，500) console.warn（‘Warning! Too few nodes (%d)’，document.childNodes.length)
```

可以这样理解，log方法是写入标准输出（stdout），warn方法和error方法是写入标准错误（stderr）。

控制台

浏览器控制台中，除了使用console对象，还可以使用一些控制台自带的命令行方法

$_属性返回上一个表达式的值。

```
2 + 2 $_
```

$0 ~ 4

控制台保存了最近5个在 Elements 面板选中的 DOM 元素，$0代表倒数第一个（最近一个），$1代表倒数第二个，以此类推直到$4。

inspect(object)

inspect(object)方法打开相关面板，并选中相应的元素，显示它的细节。DOM 元素在Elements面板中显示，比如inspect(document)会在 Elements 面板显示document元素。JavaScript 对象在控制台面板Profiles面板中显示，比如inspect(window)。

keys(object)，values(object)

keys(object)方法返回一个数组，包含object的所有键名。

values(object)方法返回一个数组，包含object的所有键值。

```
let o = { ‘p1’： ‘a’， ‘p2’： ‘b’ }; keys(o) values(o)
```

其他方法

命令行 API 还提供以下方法。

- clear()：清除控制台的历史。
- copy(object)：复制特定 DOM 元素到剪贴板。
- dir(object)：显示特定对象的所有属性，是console.dir方法的别名。
- dirxml(object)：显示特定对象的 XML 形式，是console.dirxml方法的别名。

DOM基础

DOM简介

文档对象模型 (DOM) 是 HTML 和 XML 文档的编程接口。它提供了对文档的结构化的表述，并定义了一种方式可以使从程序中对该结构进行访问，从而改变文档的结构，样式和内容。DOM 将文档解析为一个由节点和对象（包含属性和方法的对象）组成的结构集合。简言之，它会将 web 页面和脚本或程序语言连接起来。一个 web 页面是一个文档。这个文档可以在浏览器窗口或作为 HTML 源码显示出来。但上述两个情况中都是同一份文档。文档对象模型（DOM）提供了对同一份文档的另一种表现，存储和操作的方式。DOM 是 web 页面的完全的面向对象表述，它能够使用如 JavaScript 等脚本语言进行修改。 DOM 是 W3C（万维网联盟）的标准。 DOM 定义了访问 HTML 和 XML 文档的标准：

> “W3C 文档对象模型 （DOM） 是中立于平台和语言的接口，它允许程序和脚本动态地访问和更新文档的内容、结构和样式。”

W3C DOM 标准被分为 3 个不同的部分：

- 核心 DOM - 针对任何结构化文档的标准模型
- XML DOM - 针对 XML 文档的标准模型
- HTML DOM - 针对 HTML 文档的标准模型

html DOM

HTML DOM 是：

- HTML 的标准对象模型
- HTML 的标准编程接口
- W3C 标准

HTML DOM 定义了所有 HTML 元素的对象和属性，以及访问它们的方法。 换言之，HTML DOM 是关于如何获取、修改、添加或删除 HTML 元素的标准。

html DOM节点

在 HTML DOM 中，所有事物都是节点。DOM 是被视为节点树的 HTML。 根据 W3C 的 HTML DOM 标准，HTML 文档中的所有内容都是节点：

- 整个文档是一个文档节点
- 每个 HTML 元素是元素节点

- HTML 元素内的文本是文本节点
- 每个 HTML 属性是属性节点
- 注释是注释节点

HTML DOM 将 HTML 文档视作树结构。这种结构被称为节点树：

通过 HTML DOM，树中的所有节点均可通过 JavaScript 进行访问。所有 HTML 元素（节点）均可被修改，也可以创建或删除节点

节点父、子和同胞

节点树中的节点彼此拥有层级关系。

父（parent）、子（child）和同胞（sibling）等术语用于描述这些关系。父节点拥有子节点。同级的子节点被称为同胞（兄弟或姐妹）。

- 在节点树中，顶端节点被称为根（root）
- 每个节点都有父节点、除了根（它没有父节点）
- 一个节点可拥有任意数量的子节点
- 同胞是拥有相同父节点的节点

下面的图片展示了节点树的一部分，以及节点之间的关系：

样例

```
<html>
  <head>
    <title>DOM 教程</title>
  </head>
  <body>
    <h1>DOM 第一课</h1>
    <p>Hello world!</p>
  </body>
</html>
```

从上面的 HTML 中：

```
* <html> 节点没有父节点；它是根节点
* <head> 和 <body> 的父节点是 <html> 节点
* 文本节点 "Hello world!" 的父节点是 <p> 节点
* <html> 节点拥有两个子节点：<head> 和 <body>
* <head> 节点拥有一个子节点：<title> 节点
* <title> 节点也拥有一个子节点：文本节点 "DOM 教程"
* <h1> 和 <p> 节点是同胞节点，同时也是 <body> 的子节点
* <head> 元素是 <html> 元素的首个子节点
* <body> 元素是 <html> 元素的最后一个子节点
* <h1> 元素是 <body> 元素的首个子节点
* <p> 元素是 <body> 元素的最后一个子节点
```

在 DOM 中，所有 HTML 元素都被定义为对象

document对象概述

document对象是文档的根节点，window.document属性就指向这个对象。也就是说，只要浏览器开始载入HTML文档，这个对象就开始存在了，可以直接调用。

document.childNodes属性返回该对象的所有子节点。对于HTML文档来说，document对象一般有两个子节点。

第一个子节点是document.doctype，表示文档类型节点（DocumentType）。对于HTML5文档来说，该节点就代表<!DOCTYPE html>。

第二个子节点是document.documentElement，表示元素节点（Element），代表：

。

document对象的属性和方法

- getElementById(id)方法返回一个对象，该对象对应着文档里一个特定的元素节点。

> document指当前文件，get即得到，Element即元素，ById即通过id；即通过id值获取当前文件中的元素

- document.createElement

```
let btn = document.createElement("BUTTON");
   document.body.appendChild(btn);
```

- getElementsByTagName()方法将返回一个对象数组，他们分别对应着文档里一个特定的元素节点

```
let get=document.getElementsByTagName("div")
console.log(get[1])
```

- 该方法与 getElementById() 方法相似，但是它查询元素的 name 属性，而不是 id 属性。 另外，因为一个文档中的 name 属性可能不唯一（如 HTML 表单中的单选按钮通常具有相同的 name 属性），所有 getElementsByName() 方法返回的是元素的数组，而不是一个元素。

DOM方法

方法是我们可以在节点（HTML 元素）上执行的动作

一些常用的 HTML DOM 方法：

- getElementById(id) - 获取带有指定 id 的节点（元素）
- appendChild(node) - 插入新的子节点（元素）
- removeChild(node) - 删除子节点（元素）

一些 DOM 对象方法 方法 描述

getElementById() 返回带有指定 ID 的元素。

getElementsByTagName() 返回包含带有指定标签名称的所有元素的节点列表（集合/节点数组）。

getElementsByClassName() 返回包含带有指定类名的所有元素的节点列表。

appendChild() 把新的子节点添加到指定节点。

removeChild() 删除子节点。

replaceChild()　　替换子节点。

insertBefore()　　在指定的子节点前面插入新的子节点。

createAttribute()　　创建属性节点。

> 在 HTML DOM 中，一个属性节点就是一个属性对象，代表 HTML 元素的一个属性。一个元素可以拥有多个属性。

createElement()　　创建元素节点。

createTextNode()　　创建文本节点。

getAttribute()　　返回指定的属性值。

setAttribute()　　把指定属性设置或修改为指定的值。

HTML DOM修改

修改 HTML = 改变元素、属性、样式和事件。 修改 HTML DOM 意味着许多不同的方面：

- 改变 HTML 内容 改变元素内容的最简答的方法是使用 innerHTML 属性
- 改变 CSS 样式
- 改变 HTML 属性
- 创建新的 HTML 元素 如需向 HTML DOM 添加新元素，您首先必须创建该元素（元素节点），然后把它追加到已有的元素上
- 删除已有的 HTML 元素 如需删除 HTML 元素，您必须清楚该元素的父元素
- 改变事件（处理程序）创建了一个新的

  元素：

```
let para=document.createElement(“p”);
```

如需向

元素添加文本，首先必须创建文本节点。创建文本节点：

```
let node=document.createTextNode(“This is a new paragraph.”);
```

向

元素追加文本节点：

```
para.appendChild(node);
```

向已有元素追加这个新元素。 查找到一个已有的元素：

```
let element=document.getElementById(“div1”);
```

向这个已存在的元素追加新元素：

```
element.appendChild(para);
```

- 添加元素节点和文本节点

```
<body>
  <div id="a">

</div>
</body>
</html>
<script>
  let diva=document.getElementById("a");
  let p=document.createElement("p");
  let text=document.createTextNode("hello") ;
  p.appendChild(text) ;
  diva.appendChild(p);
</script>
```

- 添加属性节点

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style type="text/css">
        .democlass {
            color: red;
        }
    </style>
</head>
<body>
<h1>Hello World</h1>
</body>
</html>
<script>
  let h1 = document.getElementsByTagName["H1"](0);
    let att = document.createAttribute("class");
    att.value = "democlass";
    h1.setAttributeNode(att);
</script>
```

补充：className可以用来改变标签元素的css类选择器，从而改变元素的样式

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
   <style>
   .a{
    height: 200px;
    width: 200px;
    background-color:blue ;
   }
   .b{
    height: 200px;
    width: 200px;
    background-color:crimson;
   }
```

```
    </style>
</head>
<body>
    <div class="a"></div>
</body>
</html>
<script>
    let a=document.getElementsByTagName('div')[0].className
    console.log(a)
    document.getElementsByTagName('div')[0].className="b"
</script>
```

- 删除节点

```
<div id="red">
    <h1>红盒子</h1>
</div>
<div id="blue">蓝盒子</div>
```

删除“红盒子”

```
let red = document.getElementById ("red");  //获取红色盒子的引用
let h1 = document.getElementsByTagName("h1")[0];  //获取标题元素的引用
red.removeChild(h1);  //移出红盒子包含的标题元素
```

当使用 removeChild() 方法删除节点时，该节点所包含的所有子节点将同时被删除

DOM属性

属性是节点（HTML 元素）的值，能够获取或设置。 一些常用的 HTML DOM 属性：

- innerHTML - 节点（元素）的文本值
- parentNode - 节点（元素）的父节点
- childNodes - 节点（元素）的子节点
- attributes - 节点（元素）的属性节点

childNodes 属性

```
<ul class="list">
  <li>
  <div>app</div>
  </li>
  <li>Sam</li>
  <li>Lily</li>
</ul>
```

```
    let a = document.getElementsByClassName('list')[0]
    console.log(a.childNodes)
```

> [text, li, text, li, text, li, text]

获取输出之后，里面拥有7个成员,类型为NodeList,可以当数组那样使用，但不可遍历. 里面的li很好理解，就是获取了ul标签里的li标签的DOM信息，但是这个text该怎么理解呢？ 上面的四个text分别代表了上面的三个li标签空隙处的文本信息(包括换行符)，有n个标签就会有n+1个文本空隙，也会有n+1个文本节点。如果想验证的话，就往这些空隙里打字，然后获取输出，再打开相应的节点，查看data属性就自然知道了。 等价于：

```
<ul class="list">
    文本内容
  <li>
  <div>app</div>
  </li>
    文本内容
  <li>Sam</li>
    文本内容
  <li>Lily</li>
    文本内容
</ul>
```

innerHTML 属性

获取元素内容的最简单方法是使用 innerHTML 属性。

innerHTML 属性对于获取或替换 HTML 元素的内容很有用。

在上面的例子中，getElementById 是一个方法，而 innerHTML 是属性。

- innerHTML 属性可用于获取或改变任意 HTML 元素
- innerHTML返回该节点内的所有子节点及其值

nodeName 属性

nodeName 属性规定节点的名称。

nodeName 是只读的

- 元素节点的 nodeName 与标签名相同
- 属性节点的 nodeName 与属性名相同
- 文本节点的 nodeName 始终是 #text
- 文档节点的 nodeName 始终是 #document

注释：nodeName 始终包含 HTML 元素的大写字母标签名。

nodeValue 属性

nodeValue方法返回的是该节点的值，在DOM中主要有三种节点，分别是元素节点、属性节点、文本节点。 元素节点的nodeValue是undefined或者是null的，说明的元素节点本身是没有值的。 文本节点的nodeValue是文本的本身，因为文本节点本身是有值的。 属性节点的nodeValue是其的属性值，很明显属性一般来说都是有值的

nodeType 属性

nodeType 属性返回节点的类型。nodeType 是只读的。

比较重要的节点类型有：

元素类型　NodeType

元素　　1

属性　　2

文本　　3

注释　　8

文档　　9

HTML DOM访问

访问 HTML 元素等同于访问节点

您能够以不同的方式来访问 HTML 元素：

- 通过使用 getElementById() 方法：返回带有指定 ID 的元素
- 通过使用 getElementsByTagName() 方法： 返回带有指定标签名的所有元素（是个列表）
- 通过使用 getElementsByClassName() 方法：查找带有相同类名的所有 HTML 元素

HTML DOM事件

HTML DOM 允许 JavaScript 对 HTML 事件作出反应。

当事件发生时，可以执行 JavaScript，比如当用户点击一个 HTML 元素时。

如需在用户点击某个元素时执行代码，把 JavaScript 代码添加到 HTML 事件属性中：

> onclick= “JavaScript”

HTML 事件的例子：

onclick事件

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>教程</title>
<script>
function myFunction(){
  document.getElementById("demo").innerHTML="Hello World";
}
</script>
</head>
<body>

<p>单击按钮触发函数。</p>
<button onclick="myFunction()">点我</button>
<p id="demo"></p>

</body>
</html>
```

onload 和 onunload 事件

当用户进入或离开页面时，会触发 onload 和 onunload 事件。

在 JavaScript 中，onload 事件在页面完全加载完毕的时候触发。该事件包含所有的图形图像、外部文件（如 CSS、JS 文件等）的加载，也就是说，在页面所有内容全部加载之前，任何 DOM 操作都不会发生。为 window 对象绑定 onload 事件的方法有两种。

- 直接为 window 对象注册页面初始化事件处理函数。

> window.onload = f; function f() { alert(“页面加载完毕”); }

- 在页面 标签中定义 onload 事件处理属性。

```
  <body onload="f()">

<script>
   function f() {
       alert("页面加载完毕");
   }
</script>
```

onload 事件可用于检查访客的浏览器类型和版本，以便基于这些信息来加载不同版本的网页。

对于onunload事件

但所有主流浏览器都无法在关闭页面时执行该事件， 后来调试发现，onunload事件已正常执行，但onunload是在销毁页面对象后触发的 ，此时alert方法已经被锁定销毁，所以不能弹出提示框

onchange 事件

什么时候会发生onchange事件？

- 当元素的值发生改变时，会发生 onchange 事件。
- 对于单选框和复选框，在被选择的状态改变时，发生 onchange 事件。

```
  <p>请从列表中选择一辆新车。</p>
  <select id="mySelect" onchange="myFunction()">
      <option value="A车">A车</option>
      <option value="B车">B车</option>
      <option value="C车">C车</option>
      <option value="D车">D车</option>
  </select>
  <p>当您选择一辆新车时，会触发一个函数，输出所选汽车的值。</p>
  <p id="demo"></p>


<script>
      function myFunction() {
          let x = document.getElementById("mySelect").value;
          document.getElementById("demo").innerHTML = "You selected: " + x;
      }
  </script>
```

onmouseover 和 onmouseout 事件

onmouseover 和 onmouseout 事件可用于在鼠标指针移动到或离开元素时触发函数。

```
    <script>
        function bigImg(x) {
            x.style.height = "640px";
            x.style.width = "640px";
        }
        function normalImg(x) {
            x.style.height = "320px";
            x.style.width = "320px";
        }
    </script>

    <img onmouseover="bigImg(this)" onmouseout="normalImg(this)" border="0" src="./src/1.jpg" alt="Smiley" width="3
        height="320">
    <p>函数 bigImg() 在鼠标指针移动到图片是触发。</p>
    <p>函数 normalImg() 在鼠标指针移出图片是触发.</p>
```

onmousedown、onmouseup 以及 onclick 事件

onmousedown、onmouseup 以及 onclick 事件是鼠标点击的全部过程。首先当某个鼠标按钮被点击时，触发 onmousedown 事件，然后，当鼠标按钮被松开时，会触发 onmouseup 事件，最后，当鼠标点击完成时，触发 onclick 事件。

HTML DOM导航

通过 HTML DOM，能够使用节点关系在节点树中导航。

HTML DOM 节点列表长度:

> length 属性定义节点列表中节点的数量。 可以使用 length 属性来循环节点列表

导航节点关系

可以使用三个节点属性：parentNode、firstChild 以及 lastChild ，在文档结构中进行导航。

```
<html>
<body>
<p>Hello World!</p>
<div>
  <p>DOM 很有用!</p>
  <p>本例演示节点关系。</p>
</div>
</body>
</html>

* 首个 <p> 元素是 <body> 元素的首个子元素（firstChild）
* <div> 元素是 <body> 元素的最后一个子元素（lastChild）
* <body> 元素是首个 <p> 元素和 <div> 元素的父节点（parentNode）

有两个特殊的属性，可以访问全部文档：
* document.documentElement - 全部文档
* document.body - 文档的主体
```

练习题：点击按钮时，通过修改属性值，显示和隐藏盒子

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
```

```
    <title></title>
    <style>
        .show {
            display: block;
        }

        .hide {
            display: none;
        }
    </style>
</head>
<body>
    <button id="btn">隐藏</button>
    <div>
        我显示了
    </div>
    <script>
        //需求：点击button,隐藏盒子。改变文字，在点击按钮，显示出来。
        //步骤：
        //1.获取事件源和相关元素
        //2.绑定事件
        //3.书写事件驱动程序

        //1.获取事件源和相关元素
        let btn = document.getElementById("btn");
        let div1 = document.getElementsByTagName("div")[0];
        //2.绑定事件
        btn.onclick = function () {
            //3.书写事件驱动程序
            //判断btn的innerHTML属性值，如果为隐藏就隐藏盒子，并修改按钮内容为显示。
            //反之，则显示，并修改按钮内容为隐藏
            if (this.innerHTML === "隐藏") {
                div1.className = "hide";
                //修改按钮上的文字（innerHTML）
                btn.innerHTML = "显示";
            } else {
                div1.className = "show";
                //修改按钮上的文字（innerHTML）
                btn.innerHTML = "隐藏";
            }
        }

    </script>
</body>
</html>
```