

DOM+BOM

DOM+BOM

DOM基础

获取页面元素

DOM的层次结构

Element常用属性

Element常用方法

Property与Attribute的区别

0级DOM事件

鼠标事件

键盘事件

其他事件

0级事件的绑定方式

常规绑定方式

动态绑定的方式

事件方法里面的this

第一种情况

第二种情况

第三种情况

事件对象

获取事件对象

分析事件对象

鼠标事件对象

键盘事件

事件流

事件冒泡

取消事件冒泡

事件的绑定者与触发者 【重点】

事件委托

判断事件的触发者

事件方法里的this

事件先与响应

DOM的样式操作

一、行内样式的操作

二、内部样式快的操作

三、外部样式表

四、classList与className的操作

五、获取元素的样式

总结

补充：CSS变量操作

案例

模板引擎art-template

存在的问题

模板的使用

标准语法

 定义模板

 使用模板

简洁语法

案例

DOM补充

自定义属性DataSet

元素大小

 偏移量

 客户区大小

 滚动大小

 综合案例

正则表达式

正则表达式的创建

正则表达式对象的基本方法

正则表达式的规则

 一元符

 反义字符

 原子表与原子组

 重复匹配

 贪婪与惰性

 转义字符

 原子组编号

 前瞻与后顾

 前瞻

 负前瞻

 后顾

 负后顾

正则表达式操作方法

test方法

exec提取方法

match()提取

split分割

search搜索

replace方法

练习与作业

2级DOM事件

事件监听

事件取消

事件的传播方向

2级事件的事件链

断开事件链

2级事件中的this

2级事情的兼容性

取消事件的默认行为

0级事件与2级事件的区别

HTML条件注释

HTML5的多媒体技术

audio

video

文件与base64

摄像头

canvas画布

画布的创建

画笔的创建

ctx基本方法与属性

渐变设置

线性渐变

径向渐变

圆锥渐变

绘制图片

绘制静态的图片

绘制动态的图片

画布的globalCompositeOperation属性

画布的globalAlpha属性

画布的变换

画布的配置保存及还原

使用图像数据

BOM基础

location对象

history对象

navigator对象

window对象

跨页面作用技术 【重难点】

localStorage本地存储

localStorage的注意事项及特点

sessionStorage会话存储

sessionStorage的注意事项及特点

cookie

cookie的注意事项及特点

JSON

JSON.stringify()序列化

JSON.parse()转化

URL对象

URL.createObjectURL()

URL编码与解码

Ajax网络通讯

什么是Ajax

Ajax的创建过程

Ajax对象的分析

Ajax的封装

同步与异步的概念

JSDoc注释

文档头部相关的注释

方法类型注释

变量类型的注释

默写总结

第一次默写

第二次默写

第三次默写

第四次默写

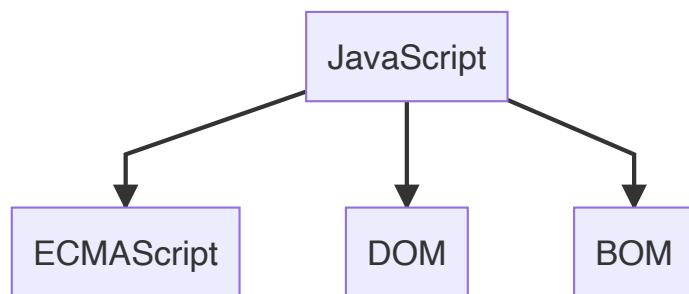
第五次默写

第六次默写

第七次默写

DOM基础

在刚开始接触到JS的时候，我们就说过JS分为3个部分



之前我们所学习的ECMAScript指的就是JavaScript的基础语法，DOM指的就是JS与网页的结合，JS最终是运行在浏览器里面的，它要操作网页，实现网页的特效与功能

DOM就是JS与网页的结合，在结合的时候主要是结合以下两个方法

1. JS 与 HTML 的结合
2. JS 与 CSS 的结合

DOM的全称叫 `document object model` 文档对象模型，它是JS与网页的结合技术，把网页像对象一个用模型的方式去操作

获取页面元素

在DOM里面如果想操作某一个、某些个页面上面的元素前提条件就是先获取这个（些）元素，在DOM里面我们有很多种方式来获取页面的元素

1. 通过ID获取

```
document.getElementById("id名"):HTMLElement  
//如  
document.getElementById("aaa");
```

这一种方式是通过id在页面上面获取一个元素，但有一些特殊情况要注意

- 如果找到了多个元素就只返回第一次找到的元素（这种情况主要是ID重复造成的）
 - 如果没有找到元素就返回 `null`
2. 通过 `class` 名去获取

```
document.getElementsByClassName("class名称"):HTMLCollection  
//如  
document.getElementsByClassName("bbb")
```

这一种方式是通过 `class` 名称去获取一个集合，这个集合里面就是包含了符合要求的集合，它是一个类数组（具备数组的特性，但是不具备数组的方法，我们可以后期通过方法将它转换真正的数组以方便操作）

如果一个都找不到，则返回一个空的集合

3. 通过标签名去获取

```
document.getElementsByTagName("标签名"):HTMLCollection  
//如  
document.getElementsByTagName("li");
```

4. 通过 `name` 属性去获取

```
document.getElementsByName("name名称"):NodeList  
document.getElementsByName("sex")
```

请注意，这一个方法返回的也是一个集合，它是 `NodeList` 类型的集合，与我们之前的 `HTMLCollection` 非常相似，它把符合要求的元素放在这个集合里面，如果找不到就返回一个空集合

关于 `HTMLCollection` 与 `NodeList` 的区别，我在这里先不讲
(`HTMLCollection` 是动态集合，而 `NodeList` 是静态集合)

上面的4个方法就是我们获取DOM元素最基本的就去，比较简单可以直接使用，但是效率非常低（大多数情况之下我们很少使用上面的4个方法）

```
<!-- 选择器 ul>li:last-child -->  
<ul>  
    <li>第1项</li>  
    <li>第2项</li>  
    <li>第3项</li>  
    <li>第4项</li>  
</ul>
```

```
<!-- 选择器 ol>li -->
<ol>
  <li>第1项</li>
  <li>第2项</li>
  <li>第3项</li>
  <li>第4项</li>
</ol>
```

- 如果我想获取 ol 里面的 li 怎么办？
- 如果我想获取 ol 里面的最后一个 li 怎么办？

针对上面的问题，JS借鉴了之前了 CSS 的选择器，衍生出了2个新方法

5. 通过CSS选择器获取单个元素

```
document.querySelector("CSS选择器") :HTMLElement
//如
document.querySelector("#aaa");
document.querySelector(".bbb");
document.querySelector("h2");
```

- 如果找到元素就返回1个元素
- 如果找不到元素就返回 null
- 如果找到多个元素就返回第一个

6. 通过CSS选择器获取元素集合

```
document.querySelectorAll("CSS选择器") :NodeList
//如下
document.querySelectorAll(".bbb");
```

- 这个方法返回的是一个 NodeList 集合
- 它把符合要求的元素全部放在这个集合里面
- 如果没有找到元素就返回一个空集合

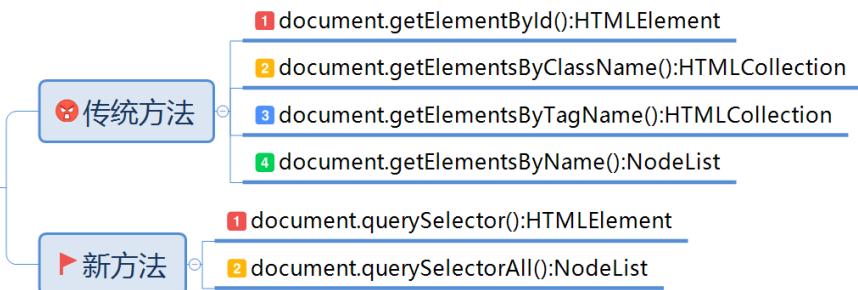
有了上面的 `querySelector()`/`querySelectorAll()` 以后，前面的4个方法只能做为备选方法使用了

👉 小技巧：后期我们会大量使用 `querySelectorAll()` 来获取元素，怎么样简便操作呢？

```
//封装一个方法，快速获取dom元素
function $(selector) {
    return document.querySelectorAll(selector);
}

var list1 = $("ol>li");
var ele = $("ul>li:last-child")[0];
```

获取DOM元素的方法



DOM的层次结构

我们已经可以通过上面的方法来获取页面上面的元素，那么这些元素都是什么数据类型 呢

```
<!DOCTYPE html>
<html lang="zh">
<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>DOM的层次结构</title>
</head>
<body>
    <div id="div1">这是一个div元素</div>
    <p id="p1">这是一个段落标签 </p>
    <input id="txt1" type="text" placeholder="请输入内容">
</body>
<script>
    var div1 = document.querySelector("#div1");
    var p1 = document.querySelector("#p1");
    var txt1 = document.querySelector("#txt1");

    typeof div1;           // "object";
    typeof p1;             // "object";
  
```

```
    typeof txt1;           // "object";
</script>
</html>
```

我们可以看到，我们每次检测的结果都是对象，既然是对象就会有方法与属性。为了更好的弄清楚 DOM到底是什么，也为更好的理解DOM的方法与属性，我们就来看一下DOM的层次结构

元素素材	第一层	第二层	第三层	第四层	第五层
div标签	HTMLDivElement	HTMLElement	Element	Node	EventTarget
p标签	HTMLParagraphElement	HTMLElement	Element	Node	EventTarget
input标签	HTMLInputElement	HTMLElement	Element	Node	EventTarget
document	HTMLDocument	Document		Node	EventTarget

经过对比，我们发现，所以的元素应该都有5层，页面上面的所有标签都是元素，而元素又是对象，所以我们只要掌握公共的父级对象的方法，那么子级对象就可以使用这个方法，这样就极大的提高了我们的学习效率

在这里说明一个，后期DOM的所有知识点都集中在第三层~第五层

第三层讲的是元素，第四层讲的节点，第五层讲的事件

Element常用属性

Element常用的属性其实就是网页元素的对象的属性，英文单词叫 **property**

1. **children** 属性，获取当前元素的所有子级元素，它返回一个 **HTMLCollection** 集合
2. **parentElement** 属性，获取当前元素的父级元素
3. **nextElementSibling** 属性，获取当前元素的下一个兄弟元素
4. **previousElementSibling** 属性，获取当前元素的前一个兄弟元素
5. **firstElementChild** 属性，获取当前元素的第一个子元素，相当于 **children[0]**
6. **lastElementChild** 属性，获取当前元素的最后一个元素，相当于 **children[children.length-1]**
7. **innerHTML** 获取或设置当前元素里面的HTML字符串（文本也包含在内）

```
<body>
  <h1 id="aaa">这是一个标题</h1>
```

```

<div id="div1">
    这是一个盒子
    <a href="#">百度一下</a>
</div>
</body>
<script>
    var aaa = document.querySelector("#aaa");
    var div1 = document.querySelector("#div1");

    aaa.innerHTML;      // '这是一个标题'
    div1.innerHTML;    // '\n        这是一个盒子\n        <a href="#">百度
一下</a>\n        '
//赋值
    aaa.innerHTML = "标哥哥";
    div1.innerHTML = "<button>我是一个按钮</button>";
</script>

```

8. `innerText` 获取或设置当前元素里面的文本字符串（它只能获取文本，获取不到标签）

```

aaa.innerText;      // '这是一个标题'
div1.innerText;    // '这是一个盒子 百度一下'
//只能获取文本，同样也只能设置文本
aaa.innerText = "标哥哥";
div1.innerText = "<button>我是一个按钮</button>";
//这个时候 上面虽然设置的是标签，但是我们是通过innerText设置进去的，它还是以文本的形式显示

```

9. `outerHTML` 属性，获取自己的标签及内部的 `innerHTML`
10. `tagName` 属性，获取当前标签的名称，它是一个大写的字母
11. `value` 属性，获取或设置表单元素的值
12. `checked` 属性，获取或设置表单元素是否被选中，`true` 代表选中，`false` 代表没有选中
13. `childElementCount` 获取子元素的个数，相当于 `children.length` 的属性
14. `className` 属性，用于获取或设置当前元素的 `class` 属性名称

通过设置 `className` 我们可以动态的改变元素的样式

```
<!DOCTYPE html>
```

```

<html lang="zh">
<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>className属性</title>
    <style>
        .a{
            width: 100px;
            height: 100px;
            background-color: pink;
        }
        .b{
            font-weight: bold;
            font-style: italic;
            font-size: 32px;
            border: 5px solid black;
        }
        .c{
            border-radius: 50%;
        }
    </style>
</head>
<body>
    <div id="div1" class="a b">这是一个div元素</div>
</body>
<script>
    var div1 = document.querySelector("#div1");
    div1.className = "a";
    div1.className = "a b c";
</script>
</html>

```

通过这种方式来操作我们的 `className` 以期达到改变元素的样式，看起来很简单，但是还是不怎么方便，后面有方便的办法

15. `classList` 返回当前的 `class` 名称列表，它是一个 `DOMTokenList` 的类型，它是一个类数组，我们可以通过这个对象提供的几个方法来操作样式列表

- `add()` 添加一个 `class` 名称
- `remove()` 删除一个 `class` 名称
- `contains()` 判断包含一个 `class` 的名称，包含就是 `true`，不包含就是 `false`

- `toggle()` 切换，如果原来的样式里面有就删除，没有就添加

Element常用方法

1. `document.createElement(标签名)` 根据一个标签名来创建一个DOM元素

```
var div2 = document.createElement("div");
div2.innerText = "标哥创建的";
```

2. `appendChild()` 向某一个元素的内部的最后追加一个子元素，一个元素只能被一个元素appendChild,不能同时被多个元素appencChild
3. `removeChild()` 删除某一个子元素
4. `remove()` 删除自身
5. `insertAdjacentElement()` 在某一个元素的特定位置插入元素，它的语法格式如下

```
元素.insertAdjacentElement("位置",要插入的元素);
```

这里的位置是最重要的，它有四个固定的值

单词	位置
beforeBegin	开始之前
afterBegin	开始之后
beforeEnd	结束之前
afterEnd	结束之后

```
▼ <body>
  ▼ <ul class="u11"> → ① ◀ 开始
    ▶ <li>...</li>
    ▶ <li>...</li>
    ▶ <li>...</li>
  </ul> → ② ◀ 结束
  <!-- Code injected by live-server -->
```

`insertAdjacentElement` 在插入元素的时候，同一个元素只能插入在一个位置

6. `insertAdjacentHTML()` 在某一个元素的特定位置插入HTML标签字符串

```
元素.insertAdjacentHTML("位置", "要插入的字符串");
//如下
var ul1 = document.querySelector(".ul1");
ul1.firstElementChild.insertAdjacentHTML("afterend", "<li>标哥哥通过字符串插入</li>");
```

这个方法与上面的 `insertAdjacentElement` 是一样的操作，只是这里插入的是一个HTML的字符串，不是元素

7. `cloneNode(deep?:true)` 方法，克隆一个相同的节点，在默认情况下只克隆当前的节点，如果还想克隆子节点，则添加参数 `true`

```
var div4 = div1.cloneNode(true); //深度克隆
```

8. `getAttribute("属性名")` 获取HTML标签上面的属性值【重点】

9. `setAttribute("属性名", "属性值")` 设置HTML标签上面的属性值

10. `removeAttribute("属性名")`，删除某一个HTML标签里面的属性

注意：在HTML标签里面，有很多类似于 `readonly, checked, disabled, required` 等属性的时候，它们都只需要设置属性名就可以了，这些属性，我们把它们叫做单属性

```
<body>
    <!-- <button id="btn1" type="button" disabled="disabled">按钮
    </button> -->
    <button id="btn1" type="button" disabled>按钮1</button>
    <button id="btn2" type="button">按钮2</button>

</body>
<script>
    var btn1 = document.querySelector("#btn1");
    var btn2 = document.querySelector("#btn2");

    // btn1.getAttribute("disabled");
    // btn1.removeAttribute("disabled");
    // btn2.setAttribute("disabled", "disabled");

    //最后一点注意—若
    //所有的单属性Attribute在转换成Property的时候也是有的
    //它的属性值会变成true/false
```

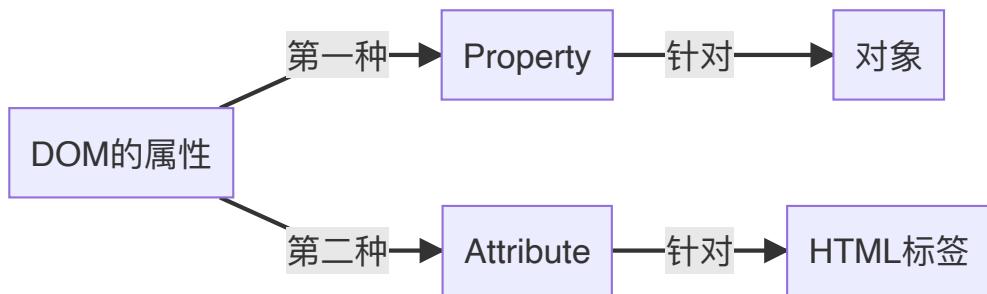
```
btn1.disabled;           //true  
btn1.disabled = false; //取消禁用  
  
btn2.disabled=true//禁用  
</script>
```

Property与Attribute的区别

在之前学习面向对象的时候，我们已经学习过了属性

```
对象.属性名;      //第一种  
对象["属性名"]; //第二种
```

DOM也是一个对象，它是一个非常特殊的对象，它是一个模型对象 **model**，它有2种类型的属性



Property 就是我们之前所学习的属性，如 **children/firstElementChild** 这些都属于DOM对象的属性，这些是依赖于DOM对象存在的

```
<input id="userName" type="text" placeholder="请输入内容">
```

上面的 **id="userName"** 以及 **type="text"** 和 **placeholder="请输入内容"** 这些东西在 **input** 标签里面也叫属性，这种属性叫 **Attribute**

在网页里面，所有HTML标签的Attribute都会默认的转换成DOM对象的Property

```
<body>  
  <input id="userName" type="text" placeholder="请输入内容">  
</body>  
<script>  
  //userName是一个DOM对象  
  //对象就会有属性
```

```
var userName = document.querySelector("#userName");
//如何获取id,type	placeholder的值呢？

//上面的input标签的attribute就转换成了userName这个dom对象的property
userName.id;
userName.type;
userName.placeholder;

//现在以另一种方式来完成
userName.getAttribute("id");
userName.getAttribute("type");
userName.getAttribute("placeholder");
</script>
```

但是HTML标签上面的自定义Attribute是不会转换成DOM对象的Property的

```
<input id="userName" type="text" haha="颜一鸣">
<script>
    //userName是一个DOM对象
    //对象就会有属性
    var userName = document.querySelector("#userName");

    userName.type;
    userName.getAttribute("type");
    // 现在我想知道haha的属性值
    //对于自定义属性，我们只能使用`getAttribute()`
    userName.haha;           //undefined
    userName.getAttribute("haha"); // "颜一鸣"
</script>
```

0级DOM事件

事件就是用户与网页之间的交互行为，当用户触发事件后，会有得到一些反馈结果

事件在DOM里面本身也是一个属性，它是一个特殊的属性，这种属性以on开头

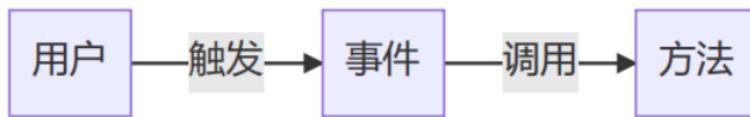
```

<body>
  <button type="button" id="btn1" onclick="console.log('我是一个点击事件')">按钮</button>
  <button type="button" onclick="aaa()">点击事件</button>
</body>
<script>
  var btn = document.querySelector("#btn1")
  function aaa() {
    console.log("我也是一个点击事件")
  }
</script>

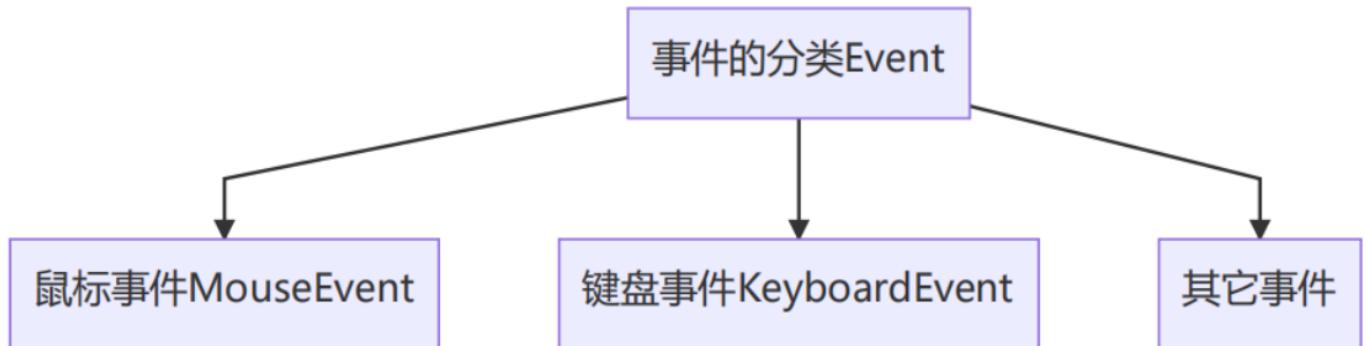
```

在上面的代码里面我们看到 `<button>` 标签里面有一个属性 `onclick`，它就是一个事件，这事件属性接受的属性值是一个方法

上面的 `onclick` 后面跟了一个我们定义的方法 `aaa()`，这个方法它不会自己调用，只有当用户触发这个点击事件后它才回去调用我们传入的方法



当用户在网页上触发了某一个特定条件后，在这个特定的条件下就会调用方法，我们把这个特定的条件就叫做0级DOM事件，我们可以把事件大致分为一下三类



鼠标事件

1. `onclick` 鼠标主键单击事件
2. `ondblclick` 鼠标主键双击事件
3. `onmousedown` 鼠标按键按下的事件

4. **onmouseup** 鼠标按键松开的事件
5. **onmousemove** 鼠标移动的事件
6. **onmouseenter** 鼠标进入以后的事件
7. **onmouseleave** 鼠标离开以后的事件
8. **onmouseover** 鼠标进入以后的事件
9. **onmouseout** 鼠标离开以后的事件

鼠标的进入与离开有两组事件

第一组: **onmouseenter/onmouseleave** 【这一组事件不建议使用, 原因是它们没有事件冒泡, 形成不了事件流】

第二组: **onmouseover/onmouseout**

10. **onmousewheel** 鼠标中键滚动的事件
11. **oncontextmenu** 鼠标上下文菜单事件【同属的说法叫右键菜单】

键盘事件

1. **onkeydown** 键盘按键按下去以后的事件
2. **onkeyup** 键盘按键松开以后的使劲按
3. **onkeypress** 键盘敲击后的事件

onkeydown 所有的按键都能触发, 但是 **onkeypress** 只有字符按键才能触发执行顺序如下

```
keydown--> onkeypress --> onkeyup
```

其他事件

1. **onfocus** 当一个元素获取焦点时候的事件
2. **onblur** 当一个元素失去焦点时候的事件
3. **oninput** 当输入框在输入的时候的事件
4. **onchange** 当表单元素的值发生变化的时候的事件

这个事件与上面的 **oninput** 很像, 但是它需要失去焦点并且前后的值有差异的时候才会触发

5. **onsubmit** 当表单提交的时候会触发

6. `onreset` 当表单被重置的事件
7. `onload` 当加载完成时候的事件
8. `onerror` 当资源加载失败的时候的事件
9. `onscroll` 当一个元素发生滚动的时候的事件
10. `onresize` 当一个元素大小发生改变的事件

0级事件的绑定方式

常规绑定方式

```
<body>
  <button type="button" onclick="console.log('我是一个点击事件')">按钮1</button>
  <button type="button" onclick="aaa()">按钮2</button>
</body>
<script>
  function aaa() {
    console.log("我也是一个点击事件");
  }
</script>
```

上面的代码里面，我们直接在 `onclick` 属性里面传入了一个方法或者直接传入我们想要执行的代码，这其实也就是将一个属性值赋值给一个属性名

动态绑定的方式

上面的例子里面我们说明了，事件本身也是一个属性，既然它是一个属性，我们就可以通过 `对象.属性` 来调用

```

<body>
  <button type="button" onclick="console.log('我也是一个点击事件')">按钮1</button>
  <button type="button" onclick="aaa()">按钮2</button>
  <button type="button">按钮3</button>
</body>
<script>
  var btn = document.querySelector("[type='button']:nth-child(3)")
  btn.onclick = aaa
  function aaa() {
    console.log("我也是一个点击事件");
  }
</script>

```

上面的代码里面就是动态绑定的方式，然后我们学习过匿名函数

```

<body>
  <button type="button">按钮3</button>
</body>
<script>
  var btn = document.querySelector("[type='button']")
  btn.onclick = function () {
    console.log("我也是一个点击事件");
  }
</script>

```

事件方法里面的this

第一种情况

```
<button type="button" onclick="console.log(this)">按钮</button>
```

在这里我们直接在 `onclick` 的地方调用 `this`，这个时候的 `this` 指向它当前的DOM对象

第二种情况

```

<body>
  <button type="button" onclick="aaa()">按钮1</button>
  <!-- 上面的代码可以下面这种写法但是前面我们讲过window调用发放的时候可以省略 -->
  <!-- <button type="button" onclick="window.aaa()">按钮1</button> -->
  <button type="button" onclick="obj.aaa()">按钮2</button>
</body>

```

```

<script>
  function aaa() {
    console.log(this);
    // 这里的this就执行window
  }

  var obj = {
    aaa: function () {
      console.log(this);
      // 这里的this就指向当前调用它的这个对象
    }
  }
</script>

```

在上面的代码里面我们发现这种方式来执行函数里面的 `this` 并没有执行当前的DOM对象吗，而是指向了调用对象

这个时候我们有希望去拿到当前操作DOM对象，我们应该怎么办？

```

10
11 <body>
12   <button type="button" onclick="console.log(this)">按钮1</button>
13   <button type="button" onclick="window.aaa(this)">按钮2</button>
14   <button type="button" onclick="obj.aaa(this)">按钮3</button>
15 </body>
16 <script>
17   function aaa(ele){
18     console.log(ele);
19   }
20
21   var obj = {
22     aaa: function (ele) {
23       console.log(ele);
24     }
25   }
26 </script>
27
28 </html>

```

在这里的this就是当前的dom对象
我们把这个this当初函数的实参传入函数
这样我们就能够拿到当前的这个dom对象了

通过上面这种把 `this` 当作参数传入函数的方式我们就能够拿到当前的DOM对象了

第三种情况

动态绑定的this指向

```
<body>
  <button type="button" class="btn1">按钮1</button>
</body>
<script>
  var btn1 = document.querySelector(".btn1")
  btn1.onclick = function () {
    console.log(this); // btn1
  }
</script>
```

通过这一种动态绑定的 `this`，它是一定会执行当前绑定的这个DOM对象的
通过上面的这三种方式我们就能快速的在事件触发的时候找到当前的DOM对象

事件对象

事件对象是对当前触发的这个事件的详细信息的描述，事件对象叫 `Event`

获取事件对象

之前我们学习了事件的三种绑定方式，现在我们从这三种绑定方式里面来获取事件对象

第一种

```
<button type="button" onclick="console.log(event)">按钮</button>
```

第一种方式直接在行内代码获取，然后系统会自动向这个事件方法内注入一个事件对象 `event`

第二种

```
<button type="button" onclick="aaa(event)">按钮2</button>
<script>
  function aaa(event) {
    // IE兼容性写法
    event = event || window.event
    console.log(event);
  }
</script>
```

第二种写法，这里需要处理IE兼容性

第三种

```
<button type="button" class="btn1">按钮1</button>
<script>
  var btn1 = document.querySelector(".btn1")
  btn1.onclick = function (event) {
    // 这里也需要处理IE兼容性
    event = event || window.event
    console.log(event);
  }
</script>
```

在这一种方式里面，系统会自动的向事件内部注入一个参数 `event`，但是我们仍然需要处理兼容性

分析事件对象

鼠标事件对象

1. `altKey` 触发当前事件的时候是否按下了 `alt` 键
2. `ctrlKey` 触发当前事件的时候是否按下了 `ctrl` 键
3. `shiftKey` 触发当前事件的时候是否按下了 `shift` 键
4. `button` 触发当前事件的按键是哪一个，`0` 表示鼠标主键（左键），`1` 表示鼠标中键，`2` 表示鼠标附键（右键）

下图是 `buttons` 的按键

IE8 及之前版本也提供了 `button` 属性，但这个属性的值与 DOM 的 `button` 属性有很大差异。

- 0：表示没有按下按钮。
 - 1：表示按下了主鼠标按钮。
 - 2：表示按下了次鼠标按钮。
 - 3：表示同时按下了主、次鼠标按钮。
 - 4：表示按下了中间的鼠标按钮。
 - 5：表示同时按下了主鼠标按钮和中间的鼠标按钮。
 - 6：表示同时按下了次鼠标按钮和中间的鼠标按钮。
 - 7：表示同时按下了三个鼠标按钮。
- IE浏览器的button属性与W3C提供的button属性有很大的区别，这里我们以W3C的为主**

5. `clientX/clientY` 代表鼠标事件触发的时候距离浏览器左边或者上面的坐标位置
6. `screenX/screenY` 代表鼠标事件触发的时候距离屏幕左边或者上面的坐标位置
7. `x/y` 这个就是上面的 `clientX/clientY`。出现他的原因也是因为浏览器的兼容性

```
// 如果想要获取当前的触发事件的位置距离浏览器左边或者上面的距离
var x = event.x || event.clientX;
var y = event.y || event.clientY;
```

8. `offsetX/offsetY` 代表鼠标事件触发的位置距离事件触发者的左边或者上面的坐标位置
9. `pageX/pageY` 代表鼠标事件触发的位置距离页面的左边或者上面的坐标位置
10. `type` 代表当前触发的事件类型
11. `path` 代表当前事件的冒泡路径（也叫事件传递路径）
12. `bubbles` 代表当前事件是否冒泡
13. `cancelBubble` 代表当前事件是否取消了冒泡
14. `cancelable` 代表当前的事件是否可以取消冒泡
15. `target` 代表当前事件的触发者
16. `currentTarget` 代表事件的绑定者

键盘事件

1. `keyCode` 代表当前按下的这个按键的编码
2. `key` 代表当前输入的这个字符
3. `repeat` 表示当前事件的触发是否因为之前的按键没有松开而连续触发，`true` 表示没有松开连续的触发

事件流

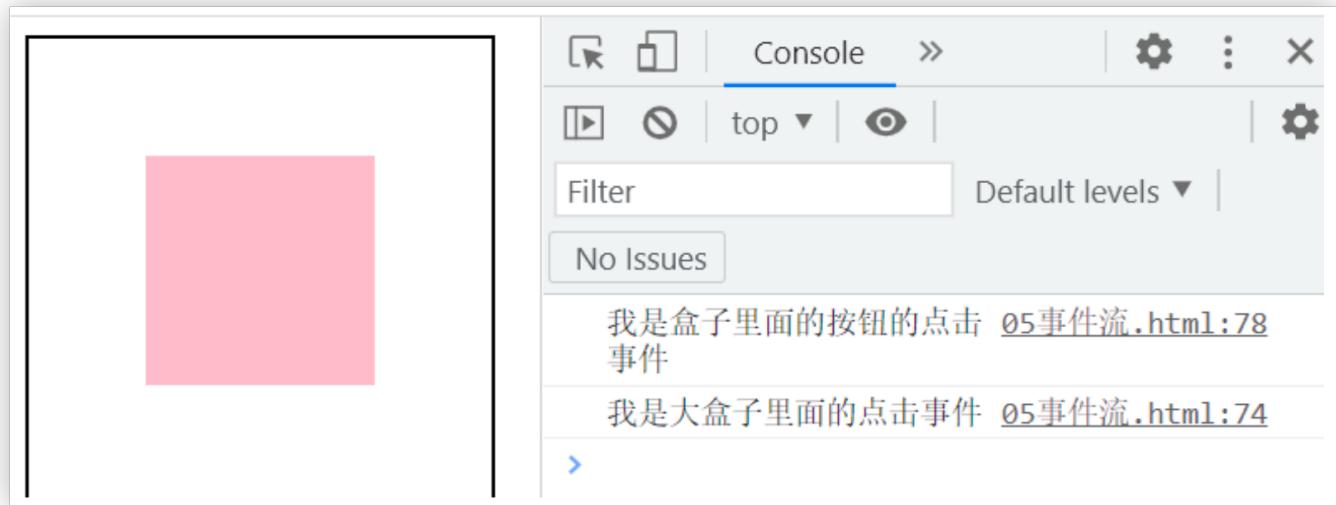
事件流的通俗说法叫做事件传播行为，关于事件的传播请看下面的例子

```
<body>
  <div class="big-box" onclick="aaa(event)">
    <div class="box" onclick="bbb(event)"></div>
  </div>
</body>
<script>
  function aaa(event) {
    console.log("我是大盒子里面的点击事件");
  }

  function bbb(event) {
    console.log("我是盒子里面的按钮的点击事件");
```

```
}
```

```
</script>
```



当我们去点击红色的小盒子的时候，外面的盒子的事件同时也触发了，这个时候小盒子里的 `onclick` 事件就传播到了大盒子里面的 `onclick`，这种现象叫做事件传播，也叫事件冒泡

事件冒泡

事件冒泡是指的是同一类型的事件会在元素的内部由内向外传播，这种现象叫做事件冒泡，上面的例子也是一个事件冒泡的典型例子，同时我们也可以在事件对象里面看到事件冒泡的路径

```
▼ path: Array(6)
  ► 0: div.box
  ► 1: div.big-box
  ► 2: body
  ► 3: html
  ► 4: document
  ► 5: Window {window: Window, self:
    length: 6
  }
  ► [[Prototype]]: Array(0)
```

取消事件冒泡

虽然事件默认是会存在传播行为的，但是我们在一些特定的场景下面我们需要取消事件的冒泡行为

```
<body>
  <div class="big-box" onclick="aaa(event)">
```

```
<div class="box" onclick="bbb(event)"></div>
</div>
</body>
<script>
    function aaa(event) {
        console.log("我是大盒子里面的点击事件");
        console.log(event);
    }

    function bbb(event) {
        event = event || window.event
        event.cancelBubble = true
        event.stopPropagation(); // 阻止事件的传播
        console.log("我是盒子里面的按钮的点击事件");
        console.log(event);
    }
</script>
```

```
<body>
    <div class="big-box" onclick="aaa(event)">
        <div class="box" onclick="bbb(event)"></div>
    </div>
</body>
<script>
    function aaa(event) {
        console.log("我是大盒子里面的点击事件");
        console.log(event);
    }

    function bbb(event) {
        event = event || window.event
        event.cancelBubble = true
        event.stopPropagation(); // 阻止事件的传播
        console.log("我是盒子里面的按钮的点击事件");
        console.log(event);
    }
</script>
```

阻止事件冒泡主要就是设置这两个点

1. `event.cancelBubble = true` 取消事件事件冒泡
2. `event.stopPropagation()` 阻止事件的传播行为

事件的绑定者与触发者【重点】

因为事件是有传播行为的，所以在事件的对象里面就可以看出当前这个事件是被哪一个元素触发的

```
<body>
  <div class="big-box">
    <button type="button">按钮</button>
  </div>
  事件绑定在盒子上面
</body>
<script>
  var box = document.querySelector(".big-box")
  box.onclick = function (event) {
    console.log("currentTarget", event.currentTarget); // 代表事件的绑定者
    console.log("target", event.target); // 表示事件的触发者
  }
</script>
```



当我们去点击里面的按钮的时候，事件是有按钮触发，但是事件有冒泡到了外面的盒子上面
上面的 `target` 表示当前事件的触发者 `currentTarget` 表示当前事件的绑定者

事件委托

因为事件是一个传播行为，所以我们可以认为内部的事件总会冒泡到外部，这样我们就可以处理一个特殊的场景，叫事件委托

请看下面代码

```
<div class="box">
  <button type="button" class="btn1">按钮1</button>
  <button type="button" class="btn1">按钮2</button>
  <button type="button" class="btn1">按钮3</button>
  <button type="button" class="btn1">按钮4</button>
  <h2>这是一个二号标题</h2>
  <p>这是一个段落标题</p>
</div>
<button class="add" onclick="add()">新增</button>
```

要求：将box里面的所有button都绑定一个点击事件，点击后打印当前按钮的文字

第一种方法：使用批量绑定

```
var btnList = document.querySelectorAll(".btn1")
btnList.forEach(function (item) {
  item.onclick = function () {
    console.log("我是" + item.innerText);
  }
})

function add() {
  var btn = document.createElement("button")
  btn.type = "button"
  btn.innerText = "按钮5"
  document.querySelector('.box').appendChild(btn)
}
```

1. 事件的绑定非常麻烦，需要去找到所有 `"class='btn1'"` 的元素然后遍历，然后绑定事件，这么做会非常消耗我们浏览器的性能
2. 当我们调用 `add()` 方法新添加一个按钮后，我们发现新添加的这个按钮没有我们前面绑定的那个事件

第二种方法：利用事件的传播行为来实现

```
<body>
```

```
<div class="box" onclick="aaa(event)">
  <button type="button" class="btn1">按钮1</button>
  <button type="button" class="btn1">按钮2</button>
  <button type="button" class="btn1">按钮3</button>
  <button type="button" class="btn1">按钮4</button>
  <h2>这是一个二号标题</h2>
  <p>这是一个段落标题</p>
</div>
<button class="add" onclick="add()">新增</button>
</body>
<script>
  function aaa(event) {
    if (event.target.className == "btn1") {
      console.log(event.target.innerText);
    }
  }

  function add() {
    var btn = document.createElement("button")
    btn.type = "button"
    btn.className = "btn1"
    btn.innerText = "按钮5"
    document.querySelector('.box').appendChild(btn)
  }
</script>
```

```
<body>
  <div class="box" onclick="aaa(event)">
    <button type="button" class="btn1">按钮1</button>
    <button type="button" class="btn1">按钮2</button>
    <button type="button" class="btn1">按钮3</button>
    <button type="button" class="btn1">按钮4</button>
    <h2>这是一个二号标题</h2>
    <p>这是一个段落标题</p>
  </div>
  <button class="add" onclick="add()">新增</button>
</body>
<script>
```

```
function aaa(event) {
  if (event.target.className == "btn1") {
    console.log(event.target.innerText);
  }
}
```

我们通过判断事件的触发者来选择是否执行里面的代码

```
function add() {
  var btn = document.createElement("button")
  btn.type = "button"
  btn.className = "btn1"
  btn.innerText = "按钮5"
  document.querySelector('.box').appendChild(btn)
}
```

```
</script>
```

通过上面两组方式的对比，我们发下半年在第二种方式里面，本来应该绑定在子集元素 `button` 上面的事件，我们把它绑定在父级的 `box` 上面，这种现象就叫 **事件委托**

判断事件的触发者

在上面的事件委托里面，我们看到事件绑定在父级，然后通过 `event.target` 来决定是否要执行代码，这个时候就有一些问题，如下：

11.1.3 matchesSelector()方法

Selectors API Level 2 规范为 Element 类型新增了一个方法 matchesSelector()。这个方法接收一个参数，即 CSS 选择符，如果调用元素与该选择符匹配，返回 true；否则，返回 false。看例子。

```
if (document.body.matchesSelector("body.page1")) {  
    //true  
}
```

注意在现代化标准浏览器里面
已经将这个方法统一的
调整为matches()方法

在取得某个元素引用的情况下，使用这个方法能够方便地检测它是否会被 querySelector() 或 querySelectorAll() 方法返回。

截至 2011 年年中，还没有浏览器支持 matchesSelector() 方法；不过，也有一些实验性的实现。IE 9+ 通过 msMatchesSelector() 支持该方法，Firefox 3.6+ 通过 mozMatchesSelector() 支持该方法，Safari 5+ 和 Chrome 通过 webkitMatchesSelector() 支持该方法。因此，如果你想使用这个方法，最好是编写一个包装函数。

```
<body>  
    <ul class="box" onclick="aaa(event)">  
        <li>第1项</li>  
        <li>第2项</li>  
        <li>第3项</li>  
        <li>第4项</li>  
        <li>第5项</li>  
        <li>第6项</li>  
    </ul>  
    <button type="button" onclick="add()">新增</button>  
</body>  
<script>  
    function aaa(event) {  
        if (event.target.matches(".box>li:nth-child(odd)")) {  
            console.log(event.target.innerText);  
        }  
    }  
  
    function add() {  
        var li = document.createElement("li")  
        li.innerText = "第7项"  
        document.querySelector(".box").appendChild(li)  
    }  
</script>
```

通过上面的这个 **matches()** 方法，我们就可以快速的判断某一个 Element 的元素是否符合我们想要执行事件代码的要求，这样去判断事件委托的触发者就非常方便了

事件方法里的this

在事件方法的内部 `this` 永远执行事件的绑定者 `currentTarget`

在事件处理程序内部，对象 `this` 始终等于 `currentTarget` 的值，而 `target` 则只包含事件的实际目标。如果直接将事件处理程序指定给了目标元素，则 `this`、`currentTarget` 和 `target` 包含相同的值。来看下面的例子。

```
var btn = document.getElementById("myBtn");
btn.onclick = function(event){
    alert(event.currentTarget === this);      //true
    alert(event.target === this);            //true
};
```

事件方法中的this指向的是事件的绑定者

[DOMEventObjectExample01.htm](#)

这个例子检测了 `currentTarget` 和 `target` 与 `this` 的值。由于 `click` 事件的目标是按钮，因此这三个值是相等的。如果事件处理程序存在于按钮的父节点中（例如 `document.body`），那么这些值是不相同的。再看下面的例子。

事件先与响应

看下面的代码

第一个例子

```
<body>
  <a href="www.baidu.com" target="_self" id="bd">百度一下</a>
</body>
<script>
  var bd = document.querySelector("#bd")
  bd.onclick = function () {
    alert("我是a标签的点击事件")
  }
</script>
```

在上面的代码里面，我们对a标签，同时指定了 `href` 的链接属性吗，以及绑定了 `onclick` 的点击事件，这个时候如果我们点击这个链接，根据事件先于响应的原则，它应该会先触发 `onclick`，再去执行 `href` 的链接跳转

第二个例子

现在我们在以表单标签为例子来看一下

```

<body>
  <form id="form1">
    <input type="text" name="userName">
    <button type="reset">重置</button>
  </form>
</body>
<script>
  var form1 = document.querySelector("#form1")
  form1.onreset = function () {
    alert("我是重置表单事件")
  }
</script>

```

通过 `reset` 的例子我们也看到了，`onreset` 的事件先触发，然后再去执行 `reset` 的重置操作

第三个例子

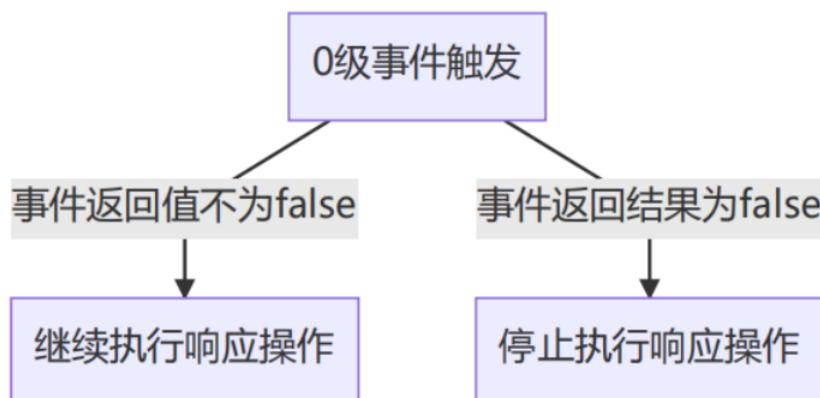
右键菜单事件

```

<body>
  <div class="box"></div>
</body>
<script>
  var box = document.querySelector(".box")
  box.oncontextmenu = function () {
    alert("右键菜单事件")
  }
</script>

```

事件先于响应这是原则，但是也不是所有的事件最终都会触发响应，它是有条件的



如果我们要是在刚刚的事件后面添加一个 `return false`，则默认的响应行为就不会触发了，这种现象在0级事件里面叫事件先与响应，在2级事件里面叫取消事件的默认行为

```
<body>
  <a href="../10事件委托.html" id="aaa">跳转链接</a>

  <a href="../09事件委托.html" onclick="return bbb()">跳转链接2</
    a>
</body>
<script>
  var aaa = document.querySelector("#aaa")
  aaa.onclick = function () {
    alert("点击事件触发")
    // 事件触发后不一定所有的响应都会执行只有当事件的返回值不为false
    // 的时候才会去触发响应
    return false
  }

  function bbb() {
    alert("我是方法bbb")
    return false
  }
</script>
```

我们还需要在标签的onclick上面将这个false返回出去

DOM的样式操作

我们之前学习了如何使用JS来操作页面上面的元素，现在我们来学习一下如何操作页面上面的样式

JS操作样式主要还是使用DOM的属性与方法。在之前我们讲解css的时候，它的存放位置有三个

1. 写在 `style` 属性里面的行内样式
2. 写在 `<style>` 标签里面的内部样式快
3. 通过 `<link>` 导入的外部样式表

一、行内样式的操作

每一个元素上面都有一个 `style` 属性，所以我们在可以通过 `style` 属性来操作元素的样式

```
> box.style
<- CSSStyleDeclaration {0: 'width', accentColor: '', additiveSymbol
  s: '', alignContent: '', alignItems: '', alignSelf: '', ...} ⓘ
  0: "width"
  accentColor: ""
  additiveSymbols: ""
  alignContent: ""
  alignItems: ""
  alignSelf: ""
  alignmentBaseline: ""
  all: ""
  animation: ""
```

如上图所示，`style` 属性里面返回过来的是一个 `CSSStyleDeclaration` 的对象，这个对象里面包含了我们所有的css样式，所以我们只要对这里面的属性赋值就可以了

```
<body>
  <div class="box"></div>
</body>
<script>
  var box = document.querySelector(".box")

  // 下面的代码相当与在html表里面写给一个style属性如何给width赋值为300px
  box.style.width = "300px"
  box.style["font-size"] = "36px"

  // 通过属性名是两个单词通过-拼接的就把-去掉如何紧挨着-的那个字母变成大写转化成驼峰命名法
  box.style.borderRadius = "50px"
</script>
```

在上面的代码里面需要注意，如果css属性里面有 `-` 的，就去掉 `-` 然后后面的首字母转大写

style在操作样式的时候，我们只建议赋值，不建议取值，style在进行取值的时候只能够拿到行内样式里面的属性值。如下所示

```
02行内样式的取值.html > html > head > style > .box
2   <html lang="en">
3
4   <head>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.
0">
8     <title>行内样式的操作</title>
9     <style>
10    .box {
11      width: 200px !important;
12      height: 200px;
13      border: 1px solid black;
14      font-size: 24px;
15      background-color: pink;
16      color: black !important;
17    }

```

我们在通过style获取样式属性值的时候，我们拿到的是设置在行内的color但是css最终展现的结果是需要通过权重值计算后才能得到最终效果这个时候我们获取的样式值就不准确

二、内部样式快的操作

本章节只做了解，如无必要，不要使用

```
<style>
  .box {
    width: 100px;
    height: 100px;
    background-color: lightsalmon;
    margin-top: 10px;
  }
</style>

<body>
  <div class="box">盒子1</div>
  <div class="box">盒子2</div>
  <div class="box">盒子3</div>
</body>
```

为什么需要操作内部样式快，是因为我们在有一个地方需要批量的去改变元素的样式，这个时候我们可以在元素上面给相同的class，在上面的代码中如果我们改变了 `.box` 样式表里面的内容，那么下面的三个使用了 `box` 这个样式的元素都会改变，这一种方式就比我们上面操作 `style` 要快上许多

```
var s = document.querySelector("style")
```

我们在控制台打印这个s，可以看到一个 `sheet` 属性

```
< CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSRuleList, type: 'text/css', href: null, ...} ⓘ
  ▶ cssRules: CSSRuleList {0: CSSStyleRule, 1: CSSStyleRule, length: 2}
    disabled: false
    href: null
  ▶ media: MediaList {length: 0, mediaText: ''}
  ▶ ownerNode: style
    ownerRule: null
    parentStyleSheet: null
  ▶ rules: CSSRuleList {0: CSSStyleRule, 1: CSSStyleRule, length: 2}
    title: null
    type: "text/css"
  ▶ [[Prototype]]: CSSStyleSheet
```

□ **disabled:** 表示样式表是否被禁用的布尔值。这个属性是可读/写的，将这个值设置为 true 可以禁用样式表。

□ **href:** 如果样式表是通过<link>包含的，则是样式表的 URL；否则，是 null。

□ **media:** 当前样式表支持的所有媒体类型的集合。与所有 DOM 集合一样，这个集合也有一个 **length** 属性和一个 **item()** 方法。也可以使用方括号语法取得集合中特定的项。如果集合是空列表，表示样式表适用于所有媒体。在 IE 中，**media** 是一个反映<link>和<style>元素 **media** 特性值的字符串。

□ **ownerNode:** 指向拥有当前样式表的节点的指针，样式表可能是在 HTML 中通过<link>或<style/>引入的（在 XML 中可能是通过处理指令引入的）。如果当前样式表是其他样式表通过 @import 导入的，则这个属性值为 null。IE 不支持这个属性。

□ **parentStyleSheet:** 在当前样式表是通过@import 导入的情况下，这个属性是一个指向导入它的样式表的指针。

□ **title:** **ownerNode** 中 **title** 属性的值。

□ **type:** 表示样式表类型的字符串。对 CSS 样式表而言，这个字符串是"text/css"。

除了 **disabled** 属性之外，其他属性都是只读的。在支持以上所有这些属性的基础上，**CSSStyleSheet** 类型还支持下列属性和方法：

□ **cssRules:** 样式表中包含的样式规则的集合。**IE 不支持这个属性，但有一个类似的 rules 属性。**

□ **ownerRule:** 如果样式表是通过@import 导入的，这个属性就是一个指针，指向表示导入的规则；否则，值为 null。**IE 不支持这个属性。**

□ **deleteRule(index):** 删除 **cssRules** 集合中指定位置的规则。**IE 不支持这个方法，但支持一个类似的 removeRule()方法。**

□ **insertRule(rule, index):** 向 **cssRules** 集合中指定的位置插入 **rule** 字符串。**IE 不支持这个方法，但支持一个类似的 addRule()方法。**

```
s.sheet.cssRules[0].style.width = "200px" // 将第一个规则下面的width设置为200px
s.sheet.cssRules[0].style.borderRadius = "50%"
```

这些都是改变某一个规则里面的某一个样式，除了修改意外，我们还可以新增或者删除某一个规则

```
// 删除某一条规则
s.sheet.deleteRule(1)
s.sheet.removeRule(1) // ie兼容性写法

// 新增一条规则
s.sheet.insertRule(".aaa {font-size: 32px}", 1)
s.sheet.addRule(".aaa {font-size: 32px}", 1) // ie兼容性写法
```

三、外部样式表

它的原理与上面内部样式快的操作一样，如下图所示

```
var link1 = document.querySelector("link")

▼ sheet: CSSStyleSheet
  ► cssRules: CSSRuleList {0: CSSStyleRule, length: 1}
    disabled: false
    href: "http://127.0.0.1:5500/css/01.css"
  ► media: MediaList {length: 0, mediaText: ''}
  ► ownerNode: link
    ownerRule: null
    parentStyleSheet: null
  ► rules: CSSRuleList {0: CSSStyleRule, length: 1}
    title: null
    type: "text/css"
  ► [[Prototype]]: CSSStyleSheet
```

它的操作原同上

四、classList与className的操作

在之前讲DOM的属性的时候，已经接触过这种通过 `classList` 与 `className` 操作页面样式了

```
<style>
  .aaa {
    color: red;
  }

  .bbb {
    color: blue;
  }
```

```
</style>
<body>
  <h2>
    我是标题
  </h2>
</body>
<script>
  var h2 = document.querySelector("h2")
  h2.classList.add("aaa")
  h2.classList.add("bbb")
  h2.classList.remove("aaa")
</script>
```

在上面的四种方式里面，都是对元素样式的赋值操作，如果我们想要获取元素的样式就不要使用上面的办法了

五、获取元素的样式

```
<style>
  .box {
    width: 200px;
    height: 200px;
    border: 2px solid black;
    font-size: 32px;
    color: red !important;
  }
</style>

<body>
  <div class="box" style="color: blue;">
    我是一个盒子
  </div>
</body>
```

在上面的例子中，如果我们直接通过 `box.style.color` 来获取样式得到的是 `blue`，但是，真正展示在页面上面的样式是 `red`，所以通过 `元素.style` 来获取元素的样式值是不对的，它有可能娶不到，也有可取到一个错误的值。

元素样式的取值，最终是需要计算以后得到的，图下图所示：

元素最终的样式是需要通过计算以后得到的，所以我们如果想要获取到元素的值也一定要获取这里的值

margin	-
border	2
padding	-
width	200px
height	200px
border-width	2px
border-color	blue
border-style	solid
border-radius	2px
border-collapse	separate
border-spacing	0px
border-block-start-color	blue
border-block-start-style	solid
border-block-start-width	2px
border-block-end-color	blue
border-block-end-style	solid
border-block-end-width	2px
border-block-color	blue
border-block-style	solid
border-block-width	2px
border-top-color	blue
border-top-style	solid
border-top-width	2px
border-bottom-color	blue
border-bottom-style	solid
border-bottom-width	2px
border-left-color	blue
border-left-style	solid
border-left-width	2px
border-right-color	blue
border-right-style	solid
border-right-width	2px
color	blue element.style
color	red !important .box
display	block
font-size	32px

在DOM操作里面，系统提供了一个方法让我们可以获取元素经过计算以后的样式，这个方法如下：

```
css样式对象 = window.getComputedStyle(元素, 是否为伪元素? );
```

现在我们就去通过这个方法来获取上面元素的最终样式

```
var box = document.querySelector(".box")
var cssObj = window.getComputedStyle(box)
// 这个时候的cssObj就能拿到了所有计算过后的样式值
console.log(cssObj.color);
console.log(cssObj.width);
```

上面就拿到了页面上展示出来的最终值。

这个方法除了能够获取元素，还可以获取伪元素的样式属性

```
<style>
.box {
  width: 200px;
  height: 200px;
  border: 2px solid black;
  font-size: 32px;
  color: red !important;
}

.box::after {
```

```

        content: "我是一个伪元素";
        color: green;
        font-style: italic;
        font-size: 24px;
    }
</style>
<body>
    <div class="box" style="color: blue;">
        我是一个盒子
    </div>
</body>
<script>
    var box = document.querySelector(".box")
    var cssObj = window.getComputedStyle(box, "::after")
    // 这个时候的cssObj就是伪元素.box::after下面的所有样式值
</script>

```

注意：我们在通过 `window.getComputedStyle()` 所得到的样式对象只能取值，不能赋值。

```

> cssObj.color= "rgb(255,0,0)"
✖ ▶ Uncaught DOMException: Failed to set the 'color'      VM262:1
    property on 'CSSStyleDeclaration': These styles are computed,
    and therefore the 'color' property is read-only.
    at <anonymous>:1:13

```

还要注意在IE里面没有这个方法，所以在写代码的时候也可能会遇到下面的情况

IE 不支持 `getComputedStyle()` 方法，但它有一种类似的概念。在 IE 中，每个具有 `style` 属性的元素还有一个 `currentStyle` 属性。这个属性是 `CSSStyleDeclaration` 的实例，包含当前元素全部计算后的样式。取得这些样式的方式也差不多，如下面的例子所示。

```

var myDiv = document.getElementById("myDiv");
var computedStyle = myDiv.currentStyle;

alert(computedStyle.backgroundColor);           //"red"
alert(computedStyle.width);                   //"100px"
alert(computedStyle.height);                  //"200px"
alert(computedStyle.border);                 //undefined

```

```
var box = document.querySelector(".box")
// W3C标准写法
var cssObj = window.getComputedStyle(box)
console.log(cssObj.width);

// 在IE里面的写法
var cssObj1 = box.currentStyle
console.log(cssObj1.width);
```

总结

在获取元素样式的时候我们使用第五点，在给改变样式的值的时候使用前面四种都已，推荐使用第一种与第四种

补充：CSS变量操作

在之前学习CSS的时候我们讲到了CSS的变量，如下

```
<style>
  .box {
    --x: 100px;
    width: var(--x);
    height: var(--x);
    background-color: pink;
  }
</style>
<body>
  <div class="box"></div>
</body>
<script>
  var box = document.querySelector(".box")
  var cssObj = window.getComputedStyle(box)
  // 获取css变量
  var x = cssObj.getPropertyValue("--x")

  // css变量的赋值
  box.style.setProperty("--x", "200px")
</script>
```

在上面的例子里面就恨到的说明了这个CSS变量操作的过程

- 取值我们仍然是找 `getComputedStyle()` 里面得到的这个对象，然后在调

用 `getPropertyValue()` 方法来获取变量的值

- 赋值我们也是找 `style`，然后通过 `setProperty()` 来实现CSS变量的赋值

案例

```
<style>
.out-box {
  --x: 0%;
  width: 200px;
  height: 200px;
  border: 5px solid black;
  border-radius: 50%;
  background-image: conic-gradient(red var(--x), #fff var(--x));
  display: flex;
  justify-content: center;
  align-items: center;
}

.box {
  width: 90%;
  height: 90%;
  border-radius: 50%;
  display: flex;
  justify-content: center;
  align-items: center;
  font-size: 48px;
  background-color: #fff;
}
</style>

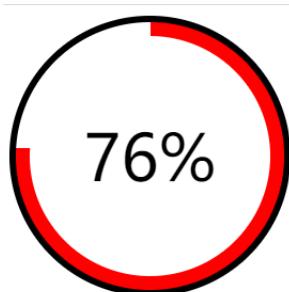
<body>
<div class="out-box">
  <div class="box">
    </div>
  </div>
</body>

<script>
var outBox = document.querySelector(".out-box")
var box = document.querySelector(".box")
var time = 0;
var aaa = setInterval(function () {

```

```
outBox.style.setProperty("--x", time + "%")
box.innerText = time + "%"
time++
if (time == 100) {
    time = 0
}
}, 100)
</script>
```

效果图



模板引擎art-template

模板引擎用于根据定义好的模板来批量生成HTML的一个工具

存在的问题

我们之前希望通过这个一个数据来动态的生成我们的网页结构

```
var stus = [
{
    stuName: "张三",
    sex: "女",
    age: 18,
    hobby: ["看书", "睡觉"]
},
{
    stuName: "李四",
    sex: "女",
    age: 22,
    hobby: ["看书", "睡觉"]
}
// ...这里还有很多
]
```

以前要实现这种动态数据的渲染，我们需要去遍历数据如何通过 `createElement` 来创建元素，如何去追加元素，这么做就很麻烦

```
<body>
  <ul class="list"></ul>
</body>
<script src="./js/stus.js"></script>
<script>
  var list = document.querySelector(".list")
  stus.forEach(function (item, index) {
    var li = document.createElement("li")
    li.innerHTML = "姓名: " + item.stuName + "-----性别: " + item.sex;
    list.appendChild(li)
  })
</script>
```

模板的使用

前面说到了模板引擎就是根据一个定义好的摹本去批量的生成我们的HTML页面，这个固定的模板我们就叫他模板引擎

现在的模板引擎有很多

- `art-template`
- `ejs`
- `jade`
- `pug`
- `hbs`

本堂课我们以 `art-template` 为标准来学习，我们需要从网站上面下周这个程序库，然后引入到网页中

```
<script src="./js/template-web.js"></script>
```

标准语法

`art-template` 有两种语法，一种是标准语法，还有一种是简洁语法，现在我们先从标准语法开始

定义模板

```
<script type="text/html" id="temp1">
<%for(var i = 0;i<arr.length;i++){%>
<%if(arr[i].sex == '男') {%>
    <li>姓名: <%=arr[i].stuName%>-----性别: <%=arr[i].sex%></li>
<%}%>
<%}%>
</script>
```

在上面的模板定义里面，我们使用了 `<%%>` 这个符号，这个符号的作用就是在模板内部可以写 `ECMAScript` 代码，同时我们在这个符号里面写了一个 `for` 循环，同时我们也使用了一个 `if` 的条件判断语句

使用模板

当我们在页面中引入了 `template-web.js` 这个文件后，他就会给我们一个方法，叫 `template`，这个方法是让我们根据数据来批量的生成这个HTML标签

```
<script src="./js/template-web.js"></script>
<script src="./js/stus.js"></script>
<script>
var htmlStr = template("temp1", {
    arr: stus
})
var box = document.querySelector(".box")
box.innerHTML = htmlStr
</script>
```

```

</head>

<body>
  <ul class="box"></ul>
  <script type="type/html" id="temp1">
    <%for(var i = 0;i<arr.length;I++)%>
      <%if(arr[i].sex == '男') %>
        <li>姓名: <%=arr[i].stuName%>-----性别: <%=arr[i].sex%></li>
      <%}%>
    </script>
</body>
<script src='./js/template-web.js'></script>
<script src='./js/stus.js'></script>
<script>
  var htmlStr = template("temp1", {
    arr: stus
  })
  var box = document.querySelector(".box")
  box.innerHTML = htmlStr
</script>

```

```

1  var stus = [
2
3    stuName: "张三",
4    sex: "女",
5    age: 18,
6    hobby: ["看书", "睡觉"]
7  },
8  {
9    stuName: "李四",
10   sex: "女",
11   age: 22,
12   hobby: ["看书", "睡觉"]
13 },
14 {
15   stuName: "王五",
16   sex: "男",
17   age: 18,
18   hobby: ["看书", "睡觉"]
19 },
20 {
21   stuName: "赵六",
22   sex: "男"

```

简洁语法

这个语法使用起来非常简单，只需要记住几个命令即可

```

<script type="type/html" id="temp1">
  {{each arr item index}}
    {{if item.sex == "男"}}
      <li>序号: {{index}}-----姓名: {{item.stuName}}-----性别: {{item.sex}}
    </li>
    {{/if}}
  {{/each}}
</script>

```

在上面的代码里面，我们可以看到，我们使用了 `{{each}}` 来进行遍历，使用了 `{{if}}` 来进行判断，我们可以得出一个结论 `{}</>` 内部就是我们的JS代码

案例

现在有一组，数据需要渲染，性别为男的文字颜色就渲染为蓝色，性别为女的文字颜色就渲染为红色

```

var stus = [
{
  stuName: "张三",
  sex: "女",
  age: 18,
  hobby: ["看书", "睡觉"]
}

```

```
},
{
    stuName: "李四",
    sex: "女",
    age: 22,
    hobby: ["看书", "睡觉"]
}
// ...这里还有很多
]
```

第一种模板写法

```
<script type="type/html" id="temp1">
{{each arr item index}}
{{if item.sex == "男"}}
<li class="blue"> 姓名: {{item.stuName}}-----性别: {{item.sex}}</li>
{{else if item.sex == "女"}}
<li class="red"> 姓名: {{item.stuName}}-----性别: {{item.sex}}</li>
{{/if}}
{{/each}}
</script>
```

第二种模板写法

```
<script type="type/html" id="temp1">
{{each arr item index}}
<li style="color: {{item.sex=='男'?'blue':'red'}}"> 姓名:
{{item.stuName}}-----性别: {{item.sex}}</li>
{{/each}}
</script>
```

第三种模板写法

```
<script type="type/html" id="temp1">
{{each arr item index}}
<li class="{{item.sex=='男'?'blue':'red'}}"> 姓名: {{item.stuName}}-----
性别: {{item.sex}}</li>
{{/each}}
</script>
```

同时我们也可以用上面的数据生成一个 **table**，如下所示

```
<!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>模板引擎</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      list-style-type: none;
    }

    .container {
      width: 80%;
      margin: auto;
    }

    .title {
      text-align: center;
      margin: 20px 0;
    }

    .btn {
      padding: 6px 10px;
      border: none;
      color: #fff;
      border-radius: 3px;
    }

    .btn-del {
      background-color: red;
    }

    .btn-edit {
      background-color: orange;
    }

    .btn-add {
      background-color: lightseagreen;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>模板引擎</h1>
    <h2>功能模块</h2>
    <ul>
      <li>文本替换</li>
      <li>条件判断</li>
      <li>循环语句</li>
      <li>嵌套模板</li>
    </ul>
    <h2>操作按钮</h2>
    <div>
      <button class="btn-del">删除</button>
      <button class="btn-edit">编辑</button>
      <button class="btn-add">添加</button>
    </div>
  </div>
</body>
</html>
```

```
.table1 {
    width: 100%;
    border: 1px solid black;
    border-spacing: 0;
}

.table1 td,
.table1 th {
    text-align: center;
    border: 1px dotted black;
    height: 40px;
}
</style>
</head>

<body>
<div class="container">
    <h2 class="title">学生信息列表</h2>
    <div class="btn-box">
        <button type="button" class="btn btn-add" onclick="showMask()">新增
    </button>
        <button type="button" class="btn btn-del">批量删除</button>
    </div>
    <table class="table1">
    </table>
</div>

<div class="mask">
    <form name="stuForm" class="stuForm">
        <div class="form-item">
            <span class="label">姓名</span>
            <input type="text" id="stuName" class="aaa">
        </div>
        <div class="form-item">
            <span class="label">性别</span>
            <select id="sex" class="aaa">
                <option value="男">男</option>
                <option value="女">女</option>
            </select>
        </div>
        <div class="form-item">
```

```
<span class="label">年龄</span>
<input type="number" id="age" class="aaa">
</div>
<div class="form-item">
    <span class="label">爱好</span>
    <input type="text" id="hobby" class="aaa">
</div>
<div class="form-item">
    <button type="button" class="btn btn-add" onclick="saveData()">保存</button>
    <button type="button" class="btn btn-del" onclick="closeMask()">取消</button>
</div>
</form>
</div>
<script type="type/html" id="stuTemp">
    <tr>
        <th><label><input type="checkbox">全选</label></th>
        <th>姓名</th>
        <th>性别</th>
        <th>年龄</th>
        <th>爱好</th>
        <th>操作</th>
    </tr>
    {{each stuArr item index}}
    <tr>
        <td><input type="checkbox"></td>
        <td>{{item.stuName}}</td>
        <td>{{item.sex}}</td>
        <td>{{item.age}}</td>
        <td>{{item.hobby.toString()}}</td>
        <td>
            <button type="button" class="btn btn-edit">编辑</button>
            <button type="button" class="btn btn-del">删除</button>
        </td>
    </tr>
    {{/each}}
</script>
</body>
<script src=".js/template-web.js"></script>
<script src=".js/stus.js"></script>
<script>
```

```
function renderHtml() {
    var htmlStr = template("stuTemp", {
        stuArr: stus
    })
    document.querySelector(".table1").innerHTML = htmlStr
}
renderHtml()
</script>

</html>
```

DOM补充

自定义属性DataSet

之前在讲DOM的时候提到过2种类型的属性

1. **Property** 它针对的是对象上面的属性
2. **Attribute** 它针对的是HTML标签上面的属性

```
<input type="text" id="aaa" bbb="hello">
```

`type/id/aaa` 都是标签的属性，我们可以把它们称之为 **attribute**，同时我们也讲过，原生的 **attribute** 会转换成 **Property**

```
var aaa = document.querySelector("#aaa");
aaa.type;
aaa.bbb; //不能取值
```

对于自定义属性，DOM并不会帮我们转换成 **property**，这个时候我们就不能通过 **对象.属性** 这一种方式调用，只能通过 **setAttribute/getAttribute** 来完成

现在我们先通过下面的案例来看情况

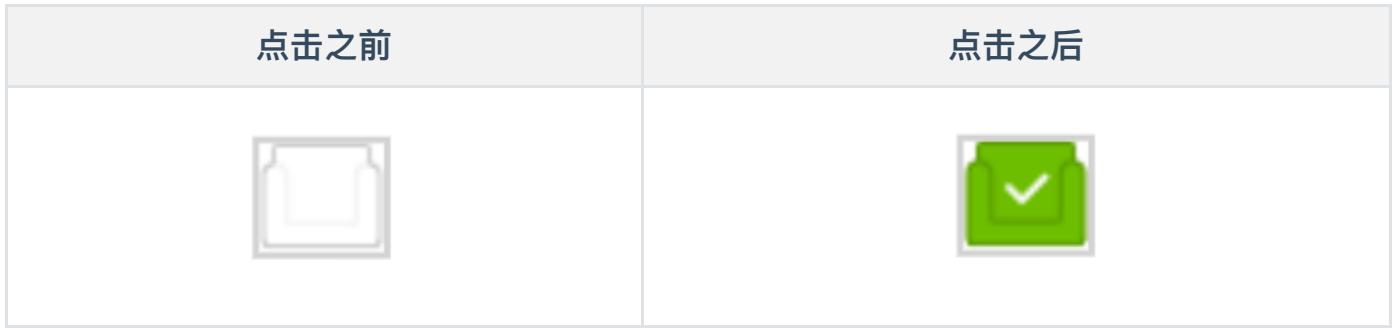
```
<!DOCTYPE html>
<html lang="zh">

<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>最原始的情况</title>
<style>
    .site-box {
        transform: scale(1.5);
        margin: 100px;
        width: 30px;
        height: 26px;
        border: 2px solid lightgray;
        /* background-image: url("img/02.png"); */
        /* background-image: url("img/03.png"); */
    }
    .site-box[bbb="2"]{
        background-image: url("img/02.png");
    }
    .site-box[bbb="3"]{
        background-image: url("img/03.png");
    }
</style>
</head>

<body>
    <div class="site-box" bbb="2" onclick="chooseSite(this)"></div>
</body>
<script>
    function chooseSite(obj){
        if(obj.getAttribute("bbb")=="2"){
            obj.setAttribute("bbb", "3");
        }
        else{
            obj.setAttribute("bbb", "2");
        }
    }
</script>

</html>
```



通过上面的自定义属性，我们可以实现样式的切换效果

自定义属性在DOM操作的时候不会变成 Property，所以我们必须要使用 `setAttribute/getAttribute` 来完成，这样做很麻烦

思路：能否将自定义的属性也转变成 `property` ?

DOM规则，如果要使用自定义的属性，最好在自定义的属性前面加上 `data-`，这样它会把你的自定义属性也转换成 `property`

```
> div1
<   <div id="div1" data-aaa="hello" data-bbb="world"> 这是一个盒子 </div>
> div1.dataset
< ▼ DOMStringMap {aaa: 'hello', bbb: 'world'} ⓘ
  aaa: "hello"
  bbb: "world"
▶ [[Prototype]]: DOMStringMap
```

在DOM里面，所有以 `data-自定义属性` 最终都会转变成 `dataset` 下面的一个属性

```
data-aaa 转变成 dataset.aaa
data-bbb 转变成 dataset.bbb
```

注意事项：所有以 `data-` 开头的自定义属性都不要使用驼峰命名，如果非要用，应该要转义使用

```
<div id="div1" data-userName="标哥" data-aaa="hello" data-bbb="world">
  这是一个盒子
</div>
```

```
<div id="div1" data-username="标哥" data-aaa="hello" data-bbb="world"> 这是一个盒子 </div>
```

这个时候我们可以看到，原来的大写变成了小写，直接使用驼峰是不可以的

如果非要使用，就要转义

```
<div id="div1" data-user-age="18" data-userName="标哥" data-aaa="hello"
data-bbb="world">
    这是一个盒子
</div>
```

在上面 `data-user-age` 就会转换成 `dataset.userAge`

```
<!DOCTYPE html>
<html lang="zh">
<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>自定义属性</title>
    <style>
        .site-box {
            transform: scale(1.5);
            margin: 100px;
            width: 30px;
            height: 26px;
            border: 2px solid lightgray;
        }

        .site-box[data-site-type="2"] {
            background-image: url("img/02.png");
        }

        .site-box[data-site-type="3"] {
            background-image: url("img/03.png");
        }
    </style>
</head>
<body>
    <div class="site-box" data-site-type="2" onclick="chooseSite(this)">
    </div>
</body>
<script>
    function chooseSite(obj) {
        if (obj.dataset.siteType == "2") {
            obj.dataset.siteType = "3";
        }
        else {
```

```
    obj.dataset.siteType = "2";
}
}
</script>
</html>
```

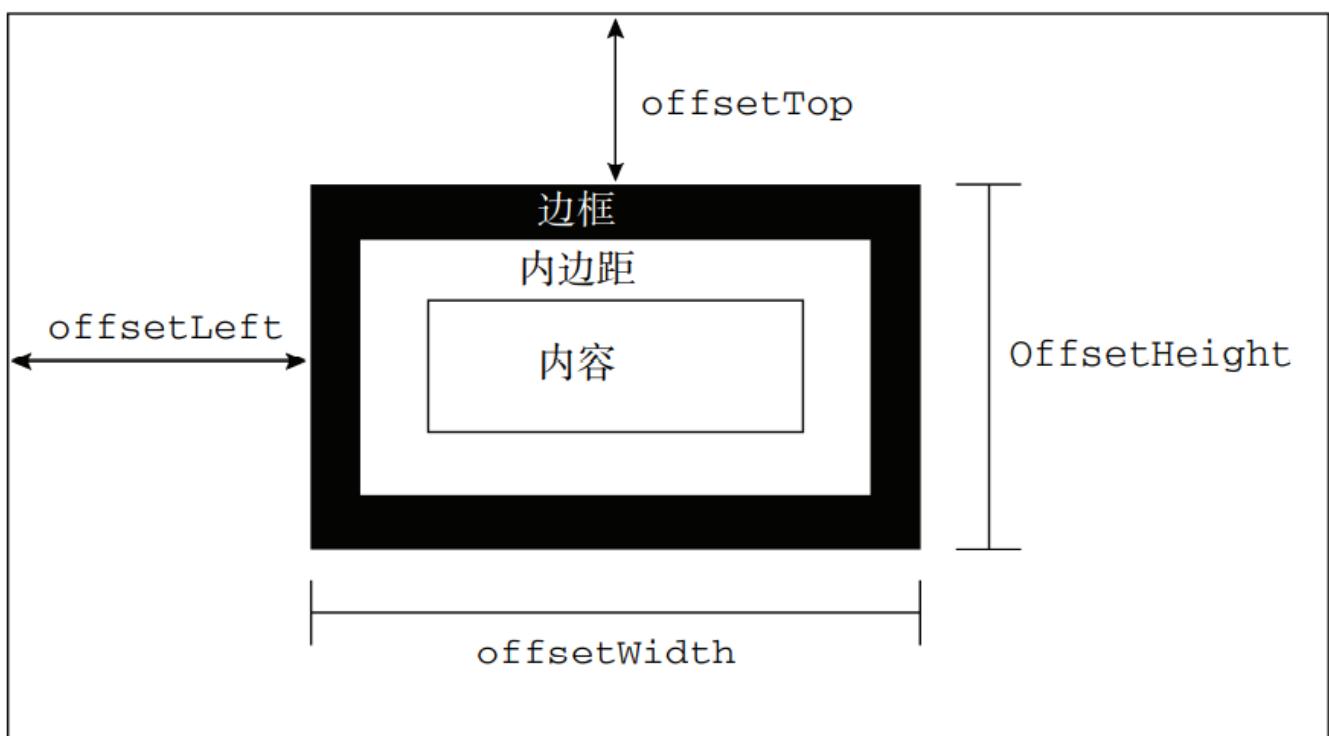
元素大小

元素的大小这点其实与我们这前理解的CSS的盒子模型是很像的，在这个地方它是通过JS的形式表现出来了

偏移量

先要介绍的属性涉及偏移量（offset dimension），包括元素在屏幕上占用的所有可见的空间。元素的可见大小由其高度、宽度决定，包括所有内边距、滚动条和边框大小（注意，不包括外边距）。

offsetParent



1. **offsetWidth**：元素在水平方向上占用的空间大小，以像素计。包括元素的宽度、(可见的) 垂直滚动条的宽度、左边框宽度和右边框宽度 【只读】
2. **offsetHeight**：元素在垂直方向上占用的空间大小，以像素计。包括元素的高度、(可见的) 水平滚动条的高度、上边框高度和下边框高度 【只读】
3. **offsetLeft**：元素距离 **offsetParent** 的左边的距离
4. **offsetTop**：元素距离 **offsetParent** 的上边的距离

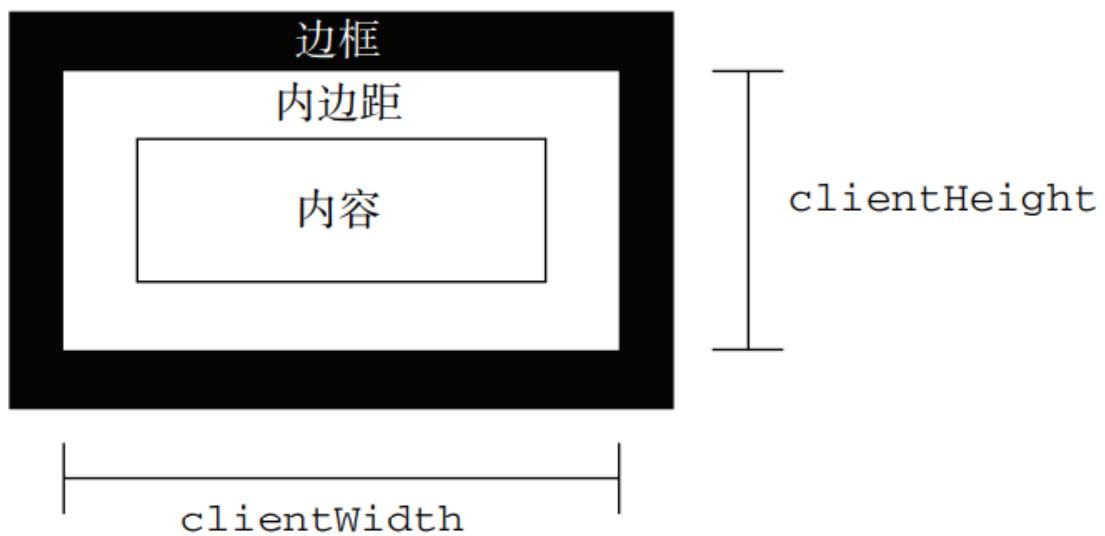
注意事项

1. 所有元素的 `offsetParent` 默认都是 `body`
2. `offsetLeft/offsetTop` 指当前元素距离自己的 `offsetParent` 的距离
3. 如果是"子绝父相"的定位，则子级元素的 `offsetParent` 指的就是外层元素的 `relative` 定位的元素
4. 如果是固定定位，则该元素的 `offsetParent` 就是 `null`

客户区大小

元素的客户区大小 (client dimension) , 指的是元素内容及其内边距所占据的空间大小

`offsetParent`

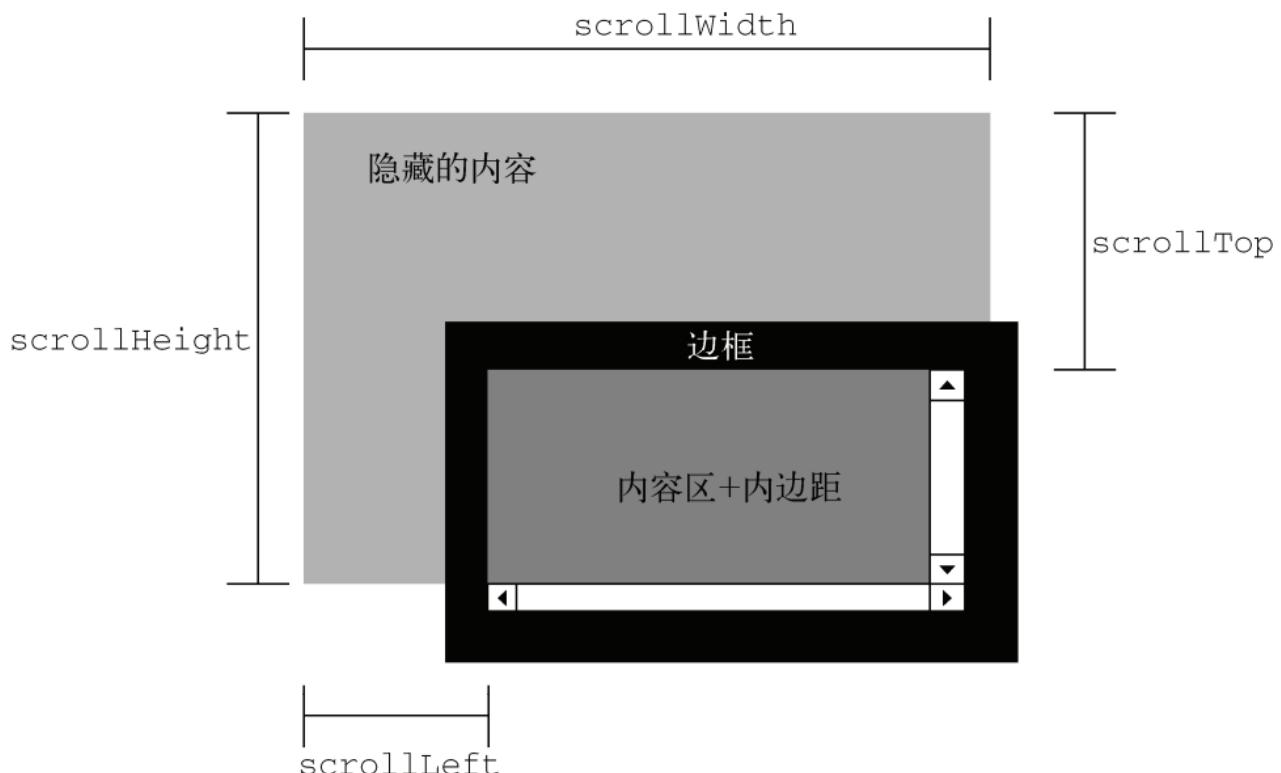


1. `clientWidth` 元素的宽度加上左右的内边距
2. `clientHeight` 元素的高度加上上下内边距

与偏移量相似，客户区大小也是只读的，也是每次访问都要重新计算的。

滚动大小

当一个小盒子里面放在大的内容的时候， 默认情况下盒子是会被溢出的， 我们可以通过添加 `overflow:auto` 来实现滚动条， 用于处理溢出， 这个滚动大小就是元素如果有滚动条的情况下



1. `scrollWidth`：元素在没有滚动条的情况下，它的总宽度【只读】
2. `scrollHeight`：元素在没有滚动条的情况下，它的总高度【只读】
3. `scrollTop`：被隐藏的区域上边的像素（或叫被滚动出去的上边的像素）
4. `scrollLeft`：被隐藏的区域的左边的像素（或叫被滚动出去的左边的像素）

当一个大的盒子里面放一个大盒子， 如果要处理溢出的情况就会使用 `overflow:auto`， 如果使用了这个情况就会出现滚动条， 而有了滚动条就会就 `scrollTop/scrollLeft`。那么， 这个东西到底有什么用呢？

当一个元素有了滚动条，并且在滚动的时候就会触发 `onscroll` 事件

```
<!DOCTYPE html>
<html lang="zh">

<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>onscroll</title>
```

```
<style>
  * {
    margin: 0;
    padding: 0;
    list-style-type: none;
  }

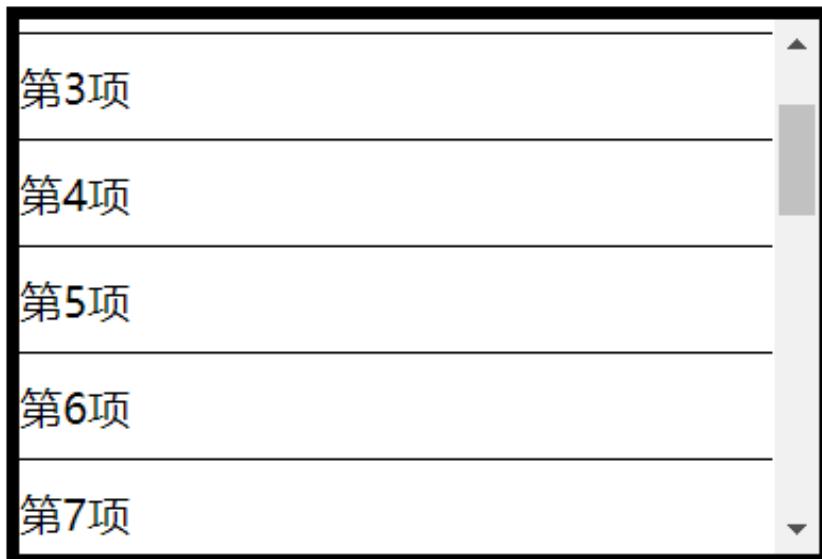
  .box {
    width: 300px;
    height: 200px;
    border: 5px solid black;
    margin: 100px;
    overflow: auto;
  }

  .ul1>li {
    height: 40px;
    display: flex;
    align-items: center;
    border-bottom: 1px solid black;
    box-sizing: border-box;
  }
</style>
</head>

<body>
  <div class="box">
    <ul class="ul1">
      <li>第1项</li>
      <li>第2项</li>
      <li>第3项</li>
      <li>第4项</li>
      <li>第5项</li>
      <li>第6项</li>
      <li>第7项</li>
      <li>第8项</li>
      <li>第9项</li>
      <li>第10项</li>
      <li>第11项</li>
      <li>第12项</li>
      <li>第13项</li>
      <li>第14项</li>
```

```
<li>第15项</li>
<li>第16项</li>
<li>第17项</li>
<li>第18项</li>
<li>第19项</li>
<li>第20项</li>
</ul>
</div>
</body>
<script>
  var box = document.querySelector( ".box" );
  // 当滚动条滚动的时候就会触发onscroll事件
  box.onscroll = function ( event ) {
    console.log( event );
  }
</script>
</html>
```

在上面的代码当中，我们可以看到，当我们去拖动滚动条的时候，它的 `onscroll` 就触发了



综合案例

```
<!DOCTYPE html>
<html lang="zh">

<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>综合案例</title>
<style>
  * {
    margin: 0;
    padding: 0;
    list-style-type: none;
  }

  .box {
    width: 300px;
    height: 200px;
    border: 5px solid black;
    margin: 100px;
    overflow: auto;
    position: relative;
  }

  .ul1 {
    position: absolute;
    width: 100%;
  }

  .ul1>li {
    height: 40px;
    display: flex;
    align-items: center;
    border-bottom: 1px solid black;
    box-sizing: border-box;
  }
</style>
</head>

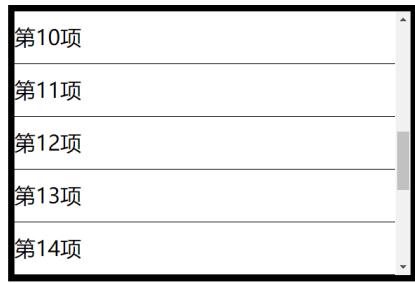
<body>
  <div class="box">
    <ul class="ul1">
      <li>第1项</li>
      <li>第2项</li>
      <li>第3项</li>
      <li>第4项</li>
      <li>第5项</li>
      <li>第6项</li>
      <li>第7项</li>
    </ul>
  </div>
</body>
```

```
<li>第8项</li>
<li>第9项</li>
<li>第10项</li>
<li>第11项</li>
<li>第12项</li>
<li>第13项</li>
<li>第14项</li>
<li>第15项</li>
<li>第16项</li>
<li>第17项</li>
<li>第18项</li>
<li>第19项</li>
<li>第20项</li>
</ul>
</div>
<button type="button" onclick="fn1()">滚动到第10项</button>
<button type="button" onclick="fn2()">回到顶部</button>
</body>
<script>
    var box = document.querySelector(".box");
    var ul1 = document.querySelector(".ul1");
    ul1.offsetParent;          //box

    var li10 = document.querySelector(".ul1>li:nth-child(10)");
    li10.offsetTop;           //ul1

    function fn1() {
        box.scrollTop = li10.offsetTop;
    }
    function fn2(){
        box.scrollTop = 0;
    }
</script>

</html>
```



[滚动到第10项](#) [回到顶部](#)

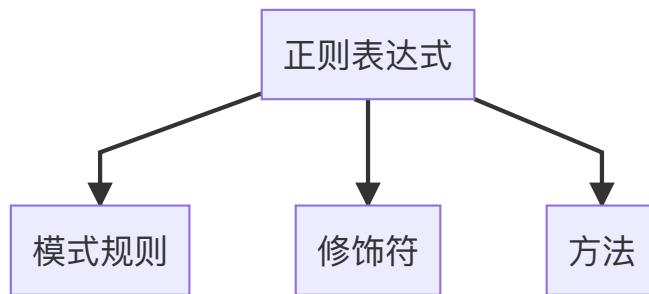
正则表达式

正则表达式并不是JavaScript里面独有的技术，所有的编程语言都有

正则表达式有以下几个特点【它的特点也可以认为是它的使用场景】

1. 只则表达式只对字符串进行操作
2. 正则表达式是根据你所设定的规则对字符串进行“验证”，“提取”，“搜索”，“替换”等操作
3. JavaScript当中的正则表达式它是一个内置对象，它可以直接使用，它的构造函数是 `RegExp` ,或直接使用字面量去创建 /正则规则/

正则表达式一共只有三个部分



正则表达式的创建

正则表达式是一个对象，在使用之前要先创建这个对象，它的构造函数是 `RegExp` ,它的语法格式如下

第一种创建方式

```
var reg = new RegExp(pattern: string | RegExp, flags?: string): RegExp
```

前面的 `pattern` 代表的就是模式规则，后面的 `flags` 代表的就是修饰符

```
var reg = new RegExp(); //创建了一个空的正则  
var reg1 = new RegExp("爱"); //创建了一个普通的正则  
var reg2 = new RegExp("爱", "g"); //创建了正则，后面添加了一个g的修饰符
```

这一种创建方式也是一种很常见的创建方式，但是我们还有更简单的方式它就是字面量创建

第二种创建方式

这一种创建方式使用的是字面量创建的方法，它使用 /规则/ 来完成。语法如下

```
var reg3 = /爱/;  
var reg4 = /爱/g;  
typeof reg3; // "object";
```

正则表达式的创建我们已经学会了，如果要使用正则表达式，就一定要先了解一下正则表达式对象的2个基本方法

正则表达式对象的基本方法

1. `test()` 方法，用于验证某个字符串是否符合正则表达式所定义的规则，验收成功结果就是 `true`，验证失败结果就是 `false`
2. `exec()` 方法，用于提取字符串当中符合正则表达式要求的字符

根据上面的这两个方法，我们才可以开始慢慢接触正则

同时正则表达式在创建的时候还会有一个修饰符可以添加

1. `g` 代表 `global` 全局的意思
2. `i` 代表 `ignore` 忽略的意思，它会忽略英文的大小写
3. `m` 代表 `multiline` 多行的意思，它可以换行操作

```
var reg1 = /杨/;  
reg1.test("杨标");  
reg1.test("标");  
reg1.test("标杨");
```

正则表达式的规则

如果我们想让正则表达式实现我们自己所需要规则，则必须了解正则表达式的规则定义

一元符

元字符	对应说明
.	匹配除换行符之外的任意字符
\w	匹配字母数字下划线，等同于：[a-zA-Z0-9_]
\s	匹配任意空白符
\d	匹配数字，等同于[0-9]
\b	匹配单词边界
	或匹配，如 /x y/ 正则可匹配x或y两个字符
^	匹配字符串的开始
\$	匹配字符串的结束

反义字符

反义字符	对应说明
[^x]	匹配除“x”之外的所有字符，其中“x”可以为任意字符
[^xyz]	同上，匹配除“x、y、z”之外的任意字符
\W	匹配除了字母、数字、下划线之外的所有字符，等同于： [^\w]
\S	匹配除空白符之外的任意字符，等同于： [^\s]
\B	匹配不是单词边界的字符，等同于： [^\b]
\D	匹配不是数字的所有字符，等同于： [^\d]

原子表与原子组

原子表

```
// 我希望有一个正则能够匹配 杨标 , 张标 , 陈标
var reg1 = /^杨|张|陈标$/;
reg1.test("杨标");
reg1.test("张标");
reg1.test("陈标");

//我希望匹配 开头是大小字母, 后面是数字
var reg2 =/^A|B|C|\d$/;

//针对上面的场景, 其实我们是无法更好的实现的, 怎么办呢
```

原子表是可以解决上面的问题的, 原子表以 [] 中括号的形式存在

```
var reg3 = /^[杨张陈]标$/;
reg3.test("杨标");           //true
reg3.test("张标");           //true
reg3.test("陈标");           //true
reg3.test("王标");           //false
```

原子表在做正则操作的时候, 它会把原子表里面的任意一个拿出来做匹配

同时原子表还可以设置区间范围

```
var reg1 = /[0-9]/;        //代表数字0-9
var reg2 = /[a-z]/;        //代表小定字母
var reg3 = /[A-Z]/;        //代表大写字母
```

注意: 原子表里面的范围不能倒着写, 写反了就会报错

原子组

```
//中括号代表原子表
var reg1 = /^[杨张陈]标$/;
//带小括号的就是原子组
var reg2 = /^(杨|张|陈)标$/;
```

原子组在后期的用法里面更多

::TODO

重复匹配

匹配字符	对应说明
*	重复出现零次或多次 {0, }
+	重复出现一次或多次 {1, }
?	重复出现零次或一次 {0, 1}
{n}	重复出现n次
{n,}	至少重复出现n次
{m,n}	重复重现m到n次， 其中， m<n

```
// 我希望有一个正则，它是以杨开头，后面跟2位任意数字
var reg1 = /^杨\d\d$/;
reg1.test("杨11");
reg1.test("杨12");

//我希望有一个正则，它是以杨开头，后面跟10位数字
// var reg2 = /^杨\d\d\d\d\d\d\d\d\d\d\d\d\d\d/;
var reg2 = /^杨\d{10}\$/;

//我希望有一个正则，它是杨开头，后面至少根4个数字
var reg3 = /^杨\d{4, }$/;
reg3.test("杨1234");           //true
reg3.test("杨12345");          //true
reg3.test("杨123");            //false

//正则，它是杨开头，后面跟 4~6个数字
var reg4 = /^杨\d{4,6}\$/;
reg4.test("杨1234");           //true
reg4.test("杨123456");         //true
reg4.test("杨1234567");        //false
reg4.test("杨123");            //false
```

在重复的次数里面，还有一些特殊的重复次数

```
//重复0次或1次
//匹配杨标，后面有可能有0个或1个数字
var reg6 = /^杨标\d{0,1}\$/;
```

```

// 上面的0次或1次的写法，一般会直接使用另一个符号
var reg7 = /^杨标\d?$/;
reg7.test("杨标2");
reg7.test("杨标");

//重复一次或多产欠
//匹配杨标，后面有可能是1个或多个数字
var reg8 = /^杨标\d{1,}$/;
reg8.test("杨标1");
reg8.test("杨标12");
//上面的1次或多次我们可以使用 + 来表示
var reg9 = /^杨标\d+$/;
reg9.test("杨标1");
reg9.test("杨标12");

//匹配任意次数      匹配0次或多次
var reg10 = /^杨标\d*$/;
reg10.test("杨标");
reg10.test("杨标1");
reg10.test("杨标12");

```

贪婪与惰性

相关字符	对应说明
*?	重复任意次，但尽可能少的重复
+?	重复一次或多次，但尽可能少的重复
??	重复零次或一次，但尽可能少的重复
{m,n}?	重复m到n次，但尽可能少的重复
{n,}?	重复n次以上，但尽可能少的重复
{n}?	重复n次，但尽可能少的重复

```

var str = "cbcertydiouycesdfsertd";
//要求：提取以c开始，以d结束，中是任何长度的小写英文字符的内容
var reg1 = /^c[a-z]*d$/;

```

上面的正则表达式是符合要求的，但是它是一个贪婪模式中间的 `[a-z]*` 会贪多个，所以最终匹配的结果就如下

```
cbcertydiouycesdfsertd
```

如果现在我们将上面的正则表达式改变一下，变成惰性模式

```
var reg2 = /c[a-z]*?d/g
```

这个时候我们可以看到它的结果如下

```
cbcertydiouycesdfsertd
```

转义字符

转义字符	对应说明
\xnn	匹配十六进制数
\f	匹配换页符，等同于：\x0c
\n	匹配换行符，等同于：\x0a
\r	匹配回车符，等同于：\x0d
\t	匹配水平制表符，等同于：\x09
\v	匹配垂直制表符，等同于：\x0b
\unnnn	匹配Unicode字符，如：\u00A0

```
//想匹配一个字符串是"[123]"
var reg1 = /^[123]$/;
reg1.test("[123]");

//上面的正则就是错的，因为中括号在正则里面表示 原子表
var reg2 = ^\[123\]$;
reg2.test("[123]");
```

正则表达式里面有一些特殊的东西是需要义的，如 [,] , / . , { , } , * , + , ? , 转义字符使用 \ 表示

原子组编号

在我们学习原子组之前，我们已经知道了在正则里面使用 `()` 可以形成原子组，原子组在之前最大用处可能就是为了让某一个东西形成一个整体。其实原子组还可以进行分组编号

```
var str1 = "<div>Hello</div>";
var str2 = "<H2></H2>";
var str3 = "<p></p>";

var reg1 = /<[a-zA-Z0-9]+><\/[a-zA-Z0-9]+>/;
```

从表现上面看，我们的正则已经符合匹配HTML标签的要求了，但是请看下面的情况

```
var str4 = "<div></p>";
reg1.test(str4);
```

在上面的过程当中，我们发现正则表达式就有问题了，我们希望前面的匹配的东西在后面要继续使用。

在匹配的过程当中，我们要求开始标签与结束标签保持一致，我开始匹配的是 `[a-zA-Z0-9]+` 我最后结束的时候也必须正前面匹配到的保持一致。这个时候我们应该怎么办呢？这个时候我们就要使用原子组的编号

```
var str1 = "<div>Hello</div>";
var str2 = "<H2></H2>";
var str3 = "<p></p>";
var str4 = "<div></p>";
var reg2 = /<([a-zA-Z0-9]+)><\/\1>/;
reg2.test(str1);      //true
reg2.test(str2);      //true
reg2.test(str3);      //true
reg2.test(str4);      //false
```

后面的正则表达式中的 `\1` 代表的就是匹配出来的第1个原子组的内容

我们还可以通过下面的东西来实现原子组编号的应用

```
var str = "fdaffdaaklfjkkklja";
//现在要求找出字符串中连续重复的字符串
var reg1 = /(\w)\1+/g;
```

fdaffdaaklfjkkklja

```
var str = "ababcdefaceced1212rtioio4ybyb";
//在里面找出那些有规律的两个一起重复的字符串
var reg1 = /(\w)(\w)\1\2/g;
```

ababcdefaceced1212rtioio4ybyb

所有的原子组在产生的时候都会有一个编号，这个编号默认是从1开始的，如果要调用某一个编号就使用 \编号 就可以了

但是也有分组以后不产生编号的

()会分组并产生编号

(?:)分组不产生编号

```
var str = "今天又跑到香港去玩了，买了好多东西，购物花了我346.77元";
//我们要把钱提取出来，怎么办？
var reg1 = /\d+(\.\d+)/;           //分组，并产生了编号
reg1.exec(str);

var reg2 = /\d+(?:\.\d+)/;        //分组，不产生编号
reg2.exec(str);
```

```
reg1.exec(str)
▼ (2) [ '346.77', '.77', index: 23, input: '今天又跑到香港去玩了，买了好多东西，购物花了我346.77元', groups: undefined ] ⓘ
  0: "346.77"
  1: ".77"    这里会有一个编号
  groups: undefined
  index: 23
  input: "今天又跑到香港去玩了，买了好多东西，购物花了我346.77元"
  length: 2
  ▶ [[Prototype]]: Array(0)

reg2.exec(str)
▼ [ '346.77', index: 23, input: '今天又跑到香港去玩了，买了好多东西，购物花了我346.77元', groups: undefined ] ⓘ
  0: "346.77"
  groups: undefined
  index: 23
  input: "今天又跑到香港去玩了，买了好多东西，购物花了我346.77元"
  length: 1
  ▶ [[Prototype]]: Array(0)
```

在这里我们就看到了，我们并不需要编号，所以我们的正则表达式就是分组不产生编号

前瞻与后顾

在学习这个知识点之前，一定要弄清楚正则表达式的前后关系【正则表达式右边的方向是前，左边的方向是后】

在学习这个东西之前，还要弄清楚几个点

1. 你要匹配的是什么？
2. 你要限制的是什么？

例如：吕亚宇找一个女朋友，但是它的限制条件是有钱，漂亮，身高170CM以上

1. 匹配条件：女
2. 限制条件：有钱，漂亮，身高170CM

前瞻

前瞻（Look ahead positive）：匹配条件是A，限制条件是A的前面是B

A(?=B)

如果我们想匹配 abc 并且 abc 的前面是 123

```
var str1 = "abc123";
var str2 = "abc456";
var str3 = "123abc";
```

/abc(?=123)/g

文本内容

abc123
abc456
123abc

负前瞻

负前瞻(Look ahead negative): 顾名思义，该正则匹配A，限制条件是A前面不是B

A(?!B)

想要匹配abc并且abc的前面不是123的表达式，应该这样：

/abc(?!123)/g

文本内容

abc123

abc456

123abc

后顾

后顾(Look behind positive): 匹配表达式A，限制条件A的后面是B

(?=B)A

想要匹配abc并且abc的后面是123的表达式

```
/(?<=123)abc/g
```

文本内容

abc123
abc456
123**abc**

负后顾

负后顾(Look behind negative): 匹配表达式A,限制条件是A的后面不是B

```
( ?<!B)A
```

想要匹配abc并且abc的后面不是123的表达式，应该这样：

```
/(?<!123)abc/g
```

文本内容

abc123
456**abc**
123abc

前瞻后顾

前瞻A(?=B) 负前瞻A(?!=B) 后顾(?<=B)A 负后顾(?<!B)A

正则表达式操作方法

在使用正则表达式的时候，是有很多个方法的，但是主要是集中在两个对象上面，一个就是 `RegExp` 正则对象，另一个就是 `String` 对象

方法	说明
<code>RegExp.prototype.test()</code>	根据正则表达式验证字符串
<code>RegExp.prototype.exec()</code>	根据正则表达式提取符合内容的字符串
<code>String.prototype.match()</code>	根据正则表达式在字符串中提取符合要求的字符串
<code>String.prototype.split()</code>	根据正则表达式来分割字符串
<code>String.prototype.search()</code>	根据正则表达式来搜索字符串的位置
<code>String.prototype.replace()</code>	根据正则表达式来替换字符串，原字符串不变，返回新字符串

test方法

该方法是正则表达式当中使用得最为频繁的一个方法，用于验证某一个字符串是否符合特定的规则，如果满足规则就返回true，否则就是false

```

/*
    验证用户名：必须是小写字母后面接任意非空字符，长度是6-10位
    验证年龄：必须是正整数
    验证性别：必须是男女
*/
var reg1 = /^[a-z]\S{5,9}$/;
reg1.test("a1234567d");           //true

var reg2 = /^\d+$/;
reg2.test("12");

var reg3 = /^[男女]$/;

```

exec提取方法

```

var str ="今天是9月10日，唉呀！今天又跑到香港去玩了，买了好多东西，购物花了我¥346.77
元，结果吃饭只能用美元，花了我$34.78元，坐港铁花了¥10元。";
//在上面的字符串里面，我们要提取所有的金额（美圆和人民币）

```

第一种操作

```

var str = "今天是9月10日，唉呀！今天又跑到香港去玩了，买了好多东西，购物花了我¥346.77
元，结果吃饭只能用美元，花了我$34.78元，坐港铁花了¥10元。";
//在上面的字符串里面，我们要提取所有的金额（美圆和人民币）

var reg = /(?<=[ ¥\$])\d+(?:\.\d+)?/g;
var result = "";           //提取的结果放在这个里面

//第一次提取                  //346.77
result = reg.exec(str);
console.log(result);

//第二次提取                  //34.78
result = reg.exec(str);
console.log(result);

//第三次提取                  //10
result = reg.exec(str);
console.log(result);

//第四次提取                  //null
result = reg.exec(str);

```

```
console.log(result);
```

通过上面的操作，我们得到了一个点，`exec` 只会每次提取一次，下次再提取的时候是在前一次结束的地方开始，一直到最后，直到提取完，最后就会得到 `null`

第二种方式：简化上面的操作

```
var str = "今天是9月10日，唉呀！今天又跑到香港去玩了，买了好多东西，购物花了我¥346.77元，结果吃饭只能用美元，花了我$34.78元，坐港铁花了¥10元。";  
//在上面的字符串里面，我们要提取所有的金额（美圆和人民币）  
  
var reg = /(?:[¥\$])\d+(?:\.\d+)?/g;  
var result = ""; //提取的结果放在这个里面  
  
while ((result = reg.exec(str)) != null) {  
    console.log(result);  
}
```

这一种正则提取的方式一般不用，用就是在高级场合。对于一般的提取我们有字符串的方法，而如果在提取的时候还要做复杂的操作，我们才会使用 `exec()`

match()提取

```
var str = "今天是9月10日，唉呀！今天又跑到香港去玩了，买了好多东西，购物花了我¥346.77元，结果吃饭只能用美元，花了我$34.78元，坐港铁花了¥10元。";  
//在上面的字符串里面，我们要提取所有的金额（美圆和人民币）  
var reg = /(?:[¥\$])\d+(?:\.\d+)?/g;  
  
// 提取一般用这个比较简单的方法  
// String.prototype.match  
var result = str.match(reg); // ['346.77', '34.78', '10']  
//这里要注意，操作的主体发生了变化，这里是字符串操作正则
```

`match` 方法会快事的提取符合要求的字符串，最终形成一个数组返回。它与我们的 `exec` 很相似，但是 `exec` 是可以实现更高级别的提取操作的

```
var str = "我的身份证号是425789199011150105, 张三的身份证号是12355420050405233x";  
//写一个正则，提取身份证号，提出信息以后，快速的形成以下的数据结构  
/*  
 [ {  
     IDCard: "425789199011150105",  
     birthday: "1990-11-15",  
 }
```

```
    sex:"女"
}, {
  IDCard:"12355420050405233x",
  birthday:"2005-05-05",
  sex:"男"
}]
*/
var reg1 = /\d{17}[\dx]/gi;
var result1 = str.match(reg1);           //['425789199011150105',
'12355420050405233x']

//上面的提取只是帮我们提取了身份证号，并不好形成生日，性别

var reg2 = /\d{6}(\d{4})(\d{2})(\d{2})\d{2}(\d)[\dx]/gi;
var result2 = "";
var arr = [];
while ((result2 = reg2.exec(str)) != null) {
  console.log(result2);
  var obj = {
    IDCard: result2[0],
    birthday: result2.slice(1, 4).join("-"),
    sex: result2[4] % 2 == 0 ? "女" : "男"
  }
  arr.push(obj);
}
console.log(arr);
```

注意：无论是使用 `exec()` 还是 `match()` 的提取，我们在当前阶段都要添加 `g` 的修饰符

split分割

之前的字符串对象里面，我们已经学习过了这个东西，那个时候我们使用的是字符串的方式去分割，现在我们要以正则表达式的方式去分割

```

var str = "get-element-by-id";
str.split("-"); //['get', 'element', 'by', 'id']

//除了使用字符串分割以外，我们还可以使用正则表达式来完成

var str2 = "我们H2204班的男生都很帅，身高都在180CM以上，年龄都在19岁以下，钱都有100W以上"；
//请将上面的字符串以数字的形式分割

var reg = /\d+/";
str2.split(reg);
//结果['我们H', '班的男生都很帅，身高都在', 'CM以上，年龄都在', '岁以下，钱都有', 'W以上']

```

search搜索

```

var str1 = "大家好，我是人见人爱的张三，我的身高是170CM!" ;
var str2 = "大家好，我是高富帅李四，我的身高是186CM!" ;
var str3 = "大家好，我是美女西施，我的身高是169CM!" ;
var str4 = "大家好，我是古典成熟高质量男性小乐乐，我的身高是178CM!" ;

//要求：找到数字的位置
str1.indexOf("170");
str2.indexOf("186");
str3.indexOf("169");
str4.indexOf("178");

//这样做很麻烦，因为如果中间有100句话，那么，我们的数字要写100次
var reg = /\d+/g;
str1.search(reg);
str2.search(reg);
str3.search(reg);
str4.search(reg);

//这个时候的参数就固定成了一个正则表达式

```

replace方法

在之前的字符串里面，我们已经学习过了 `replace`，它可以替换我们字符串中间内容 原字符串不变，返回新的字符串

```

var str1 = "我爱你，你爱我吗" ;
var result1 = str1.replace(/爱/g, "恨"); // '我恨你，你恨我吗'

```

上面的东西我们之前就已经接触过了，没有什么新颖的地方，重点在于 `replace()` 第二个参数。它的第二个参数可以写成一个回调函数

```
str1.replace(/爱/g, function(p){  
    //第一个参数p代表的是正则表达式匹配的内容  
    console.log(p);  
    console.log("我是回调函数"+Math.random());  
});  
  
var str2 = "我现在很穷，我只有19块钱，但是我的同桌是富二代，有100块钱" ;  
str2.replace(/\d+/g, function(p){  
    console.log(p);  
    console.log("我是回调函数"+Math.random());  
});
```

当正则表达式的替换里面有原子组的时候

```
var str = "大家好，今天是2022-08-31 14:48:30, 今天天气不好，有点冷" ;  
var reg = /(\d{4}-\d{2}-\d{2})\s+(\d{2}:\d{2}:\d{2})/g;  
  
str.replace(reg, function(p, g1, g2){  
    //第一步参数:p代表当前正则匹配的总体内容  
    //后面的参数就代表的是原子组编号:g1就是group1 g2就是group2  
    console.log(p)  
    console.log(g1);  
    console.log(g2);  
})
```

2022-08-31 14:48:30

2022-08-31

14:48:30

在这个回调函数里面，`return` 返回的东西就是要替换的内容

```

var str2 = "我现在很穷,我只有19块钱,但是我的同桌是富二代,有100块钱";
// 想将上面的数字换成xxx
var result = str2.replace(/\d+/g, function (p) {
    // console.log(p);
    return "x".repeat(p.length);
})

```

➤ result

：'我现在很穷,我只有xx块钱,但是我的同桌是富二代,有xxx块钱'

长属性名	短属性名	说 明
input	\$_	最近一次要匹配的字符串。Opera未实现此属性
lastMatch	\$&	最近一次的匹配项。Opera未实现此属性
lastParen	\$+	最近一次匹配的捕获组。Opera未实现此属性
leftContext	\$`	input字符串中lastMatch之前的文本
multiline	\$*	布尔值, 表示是否所有表达式都使用多行模式。IE和Opera未实现此属性
rightContext	\$'	Input字符串中lastMatch之后的文本

练习与作业

- 提取歌词,将歌词内容提取为一个数组, 歌词时间提取为一个数组

```

var musicLrc = ` [00:00.000]少年 - 梦然 (Miya)
[00:02.020]词: 梦然
[00:04.040]曲: 梦然
[00:06.060]编曲: 张亮
[00:08.090]制作人: 张亮/徐阁
[00:10.110]和声编写: 海青/梦然
[00:12.130]和声演唱: 海青/梦然
[00:14.160]Rap: 梦然
[00:16.180]混音工程师: 赵靖
[00:18.200]母带工程师: 赵靖
[00:20.230]监制: 梦然
`;

var reg1 = /(?:\[(\d{2}:\d{2})\.\d{3})(?=\])|/g;
var timeArr = musicLrc.match(reg1);

var reg2 = /(?:\[(\d{2}:\d{2})\.\d{3}\]).*/g;
var text = musicLrc.match(reg2);

```

2. 删除与某个字符相邻且相同的字符，比如 `fdaffdaakllllfjkkklja` 字符串处理之后成为 `fdafdklfjklja`

```
var str = "fdaffdaakllllfjkkklja";
//fdafdklfjklja
var reg = /([a-z])\1+/g;
var result = str.replace(reg, function(p,g1){
    // console.log(p,g1);
    return g1;
});
var result2 = str.replace(reg, "$1");
```

3. 现有数字，请将数据转换成格式（推荐使用正则表达式+replace的方式完成），如 12345678转换成 12,345,678

第一种解法

```
var str ="123456789";           //123,456,789
var reg1 = /\d{1,3}(?=(\d{3})+$)/g;

/*
var result = str.replace(reg1,function(p){
    // console.log(p);
    return p+",";
});
console.log(result);
*/


var result = str.replace(reg1, "$&, ");
console.log(result);
```

第二种解法

```
var num =123456789;           //123,456,789

//前提条件它是数字，不是字符串
var result = num.toLocaleString("en-US");
```

4. 写一个正则表达式，匹配1~15之间的任意数

```
var reg3 = /[1-9](?=<=1)[0-5])/;
//这个正则表达式是最典型的后顾的写法
```

5. 写一个方法，提取出一段话中的人民币金额与美元金额。如下所示

```
var str ="今天是9月10日，唉呀！今天又跑到香港去玩了，买了好多东西，购物花了我  
￥346.77元，结果吃饭只能用美元，花了我$34.78元，坐港铁花了￥10元。";  
var reg = /(?:[\\$\\￥])\\d+(\\.\\d+)?/g;  
var result = str.match(reg);
```

6. 给一个连字符串例如：get-element-by-id转化成驼峰形（推荐使用正则表达式+replace的方式完成）

```
var str = "get-element-by-id";  
//getElementById  
// - e E  
// - b B  
// - i I  
  
var reg = /-([a-z])/g;  
  
var result = str.replace(reg, function(p,g1){  
    //第一个参数p代表的就是正则表达式匹配的内容  
    // console.log(p,g1);  
    return g1.toUpperCase();  
});  
  
  
var result2 = str.replace(reg, "$1");
```

7. dgfhfgh254.45bhku289fgdhdy675gfh获取一个字符串中的数字字符，并按数组形式输出。
输出 [254,289,675]

```
var str = "dgfhfgh254.45bhku289fgdhdy675gfh";  
//[254,289,675]  
var reg1 = /(\d+)(\\.\\d+)?/g;  
  
var result = [];  
var temp = "";  
while ((temp = reg1.exec(str)) != null) {  
    // console.log(temp[1]);  
    result.push(+temp[1]);  
}  
console.log(result);
```

8. 写一个正则表达式，匹配班级的学号从 H22040001~H22049999

```
var reg = /^H2204(?!0000)\d{4}$/;
```

2级DOM事件

之前已经学习了0级DOM事件，0级DOM事件是以属性的形式存在的

```
<!DOCTYPE html>
<html lang="zh">

<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>0级事件</title>
</head>

<body>
    <button onclick="sayHello(this,event)" type="button" id="btn1">按钮
</button>
    <button type="button" id="btn2">按钮</button>
</body>
<script>
    // 0级事件绑定
    function sayHello(obj,event) {
        console.log(obj);
        console.log(event);
    }

    var btn2 = document.querySelector("#btn2");
    btn2.onclick = function(event){
        console.log(this);
        console.log(event);
    };
</script>

</html>
```

上面就是最基础的0级DOM事件，但是0级事件会有一些不足的地方

- 0级事件总是由内向外传播，不能改变方向
- 0级事件是以属性的形式存在，如果重复赋值，就会后面的覆盖前面的
- 0级事件是以属性的形式存在，有这个属性就有这个事件，没有这个属性就没有这个事件，所以如果想实现自定义事件是不可能的

在之前讲DOM的时候，我们提到过一个DOM的结构层次图

元素素材	第一层	第二层	第三层	第四层	第五层
div标签	HTMLDivElement	HTMLElement	Element	Node	EventTarget
p标签	HTMLParagraphElement	HTMLElement	Element	Node	EventTarget
input标签	HTMLInputElement	HTMLElement	Element	Node	EventTarget
document	HTMLDocument	Document		Node	EventTarget

- ▼ **[[Prototype]]: EventTarget**
- ▶ **addEventListener: f addEventListener()**
 - ▶ **dispatchEvent: f dispatchEvent()**
 - ▶ **removeEventListener: f removeEventListener()**
 - ▶ **constructor: f EventTarget()**
 - ▶ **Symbol(Symbol.toStringTag): "EventTarget"**
 - ▶ **[[Prototype]]: Object**

通过 DOM的结构层次分析，我们可以看到在DOM的最高层里有一个 **EventTarget**，这指的就是2级DOM事件

- **addEventListener()** 添加事件监听
- **removeEventListener()** 移除事件监听
- **dispatchEvent()** 派发事件，后期自定义事件会使用到【设计模型里面的观察者模式也会用到】

事件监听

0级事件是依赖于 **Element** 上面的属性的形式存在，而2级事件则是依赖于 **EventTarget** 上面的存在，0级事件的绑定是通过属性赋值来完成，而2级事件则是通过事件监听的方式来完成的

```
dom.addEventListener("事件名", function(event){  
    //代码体  
}, true/false);
```

在上面的语法规则里面，最后一个的参数代表事件的传播方向，默认是false代表事件由内向外传播，如果设置为true代表事件由外向内传播

我们可以通过上面的语法规则来完成一个简单的2级事件绑定

```
<body>  
    <button type="button" id="btn1">按钮</button>  
</body>  
<script>  
    var btn1 = document.querySelector("#btn1");  
    /*  
     *  
     * 事件冒泡  
     */  
    btn1.onclick=function(event){  
        console.log("我是0级事件");  
    }  
    /*  
     * 事件捕获  
     */  
    btn1.addEventListener("click", function (event) {  
        console.log("我是2级DOM事件");  
    });  
</script>
```

原来的0级事件里面事件有名是有 on 的，在2级DOM事件里面是没有的

事件取消

在之前在0级DOM事件里面，因为事件是以属性的形式存在，所以我们如果想取消某个事件，直接将这个事件的属性名赋值为 null 就可以了，如下所示

```
<body>  
    <div class="box"></div>  
    <button type="button" class="btn1">按钮</button>  
</body>  
<script>  
    var box = document.querySelector(".box");  
    var btn1 = document.querySelector(".btn1");  
    box.onclick = function (event) {  
        console.log("我是一个盒子")  
    }
```

```
btn1.onclick = function(event){
    console.log("取消了box的事件");
    //取消事件
    box.onclick = null;
}
</script>
```

但是这种方法在2级DOM事件里面是不行的，因为2级DOM事件是以事件监听的方式存在

```
<body>
    <div class="box"></div>
    <button type="button" class="btn1">按钮</button>
</body>
<script>
    var box = document.querySelector(".box");
    var btn1 = document.querySelector(".btn1");

    function aaa() {
        console.log("我是box里面的事件")
    }
    box.addEventListener("click", aaa);

    btn1.addEventListener("click", function () {
        //去移除box的事件
        console.log("移除box的事件");
        box.removeEventListener("click", aaa);
    })
</script>
```

如果要移除一个2级DOM的事件，我们就要使用 `removeEventListener()` 来完成

事件的传播方向

在默认情况下，事件总是由内向外进行传播的，同时在2级DOM事件里面，最后一个参数默认是 `false` 代表的是事件冒泡（事件冒泡：事件由内向外传播），最后一个参数如果是 `true` 代表事件捕获（事件捕获：事件由外向内传）

```
<body>
    <div class="box">
        <button type="button" id="btn1">按钮</button>
    </div>
```

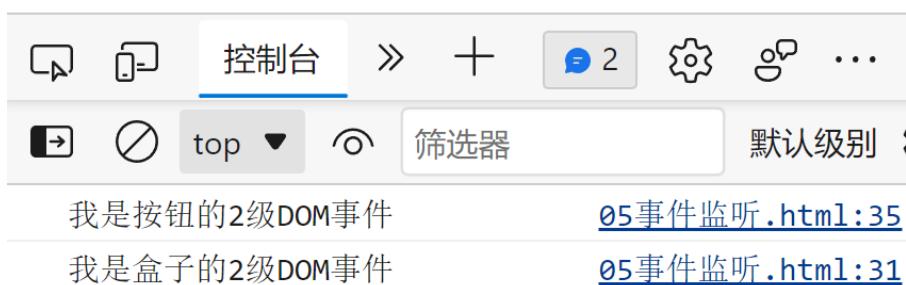
```

</body>
<script>
    var box = document.querySelector(".box");
    var btn1 = document.querySelector("#btn1");

    box.addEventListener("click", function(event){
        console.log("我是盒子的2级DOM事件");
    }, false); //事件由内向外

    btn1.addEventListener("click", function (event) {
        console.log("我是按钮的2级DOM事件");
    }, false); //事件由内向外
</script>

```



现在我们将最后一个参数换成 `true`

```

box.addEventListener("click", function(event){
    console.log("我是盒子的2级DOM事件");
}, true);

btn1.addEventListener("click", function (event) {
    console.log("我是按钮的2级DOM事件");
}, true);

```

我是盒子的2级DOM事件

我是按钮的2级DOM事件

2级事件的事件链

在0级DOM事件里面，因为事件是以属性的形式存在，所以不可能多次赋值，也就不可以多次绑定

```
<body>
    <button type="button" id="btn1" onclick="aaa()">按钮</button>
</body>
<script>
    function aaa() {
        console.log("我是aaa");
    }

    var btn1 = document.querySelector("#btn1");
    btn1.onclick = function (event) {
        console.log("我是bbb");
    }
</script>
```

如果在0级事件里面多次赋值，必然是后面覆盖前面，上面的代码在触发事件以后打印的结果是 "我是bbb"

2级事件是以监听的形式存在的，所以它不会在这个问题

```

<body>
    <button type="button" id="btn1">按钮</button>
</body>
<script>
    var btn1 = document.querySelector("#btn1");
    btn1.addEventListener("click", function(event){
        console.log("我是aaa");
    })

    btn1.addEventListener("click", function(event){
        console.log("我是bbb");
    })
</script>

```

样式 已计算 布局 事件侦听器 DOM 断点 »

上级 全部 Framework 侦听器

▼ click

- ▶ button#btn1 [删除] 10事件多次绑定.html:20
- ▶ button#btn1 [删除] 10事件多次绑定.html:16

队头

在使用2级事件的时候，它可以同是绑定多个，同时会根据你绑定的顺序来进行事件队列的管理【谁先绑定就谁先入队】。事件像队列一样的去执行，我们就把它看成是事件链



```

<body>
    <button type="button" id="btn1">按钮</button>
</body>
<script>
    var btn1 = document.querySelector("#btn1");

    btn1.addEventListener("click", function(event){
        console.log("我是bbb");
    })
    btn1.onclick=function(){
        console.log("我是0级事件");
    }
</script>

```

```
btn1.addEventListener("click", function(event){  
    console.log("我是aaa");  
})  
</script>
```

在上面的代码里面，我们可以得到2个点

1. 事件的执行顺序与绑定顺序有关，谁先绑定，谁就在事件队列的前面先执行
2. 0级事件与2级事件可以同时存在

断开事件链

在上个章节我们讲到了，2级事件是可以多次监听的，触发这个事件以后，这个事件绑定的方法就会依次向链条一样执行（事件队列）。但是我们可以在某一个地方把这个事件链断开

```
<body>  
    <button type="button" id="btn1">按钮</button>  
</body>  
<script>  
    var btn1 = document.querySelector("#btn1");  
  
    btn1.addEventListener("click", function(event){  
        console.log(event);  
        console.log("我是bbb");  
    })  
    btn1.onclick=function(event){  
        console.log("我是0级事件");  
        event.stopImmediatePropagation(); //断开事件链，并阻止事件传播  
    }  
    btn1.addEventListener("click", function(event){  
        console.log("我是aaa");  
    })  
</script>
```

说明：

在上面的代码里面，我们对btn1进行了3次事件绑定，但是在第2次绑定事件的时候我们调用了 `event.stopImmediatePropagation()` 来断开事件链，这个事件在传递到这里的时候就断开了（后面的事件绑定就不再执行了）。同时这里还要注意，它会阻止事件的冒泡与捕获

2级事件中的this

在事件方法里面， this到底是什么？

在2级事件里面，事件方法里面的 this 永远指向当前事件的DOM对象

第一种情况：最普通的情况，这一种情况的this指不到DOM对象

```
<body>
<button type="button" id="btn1" onclick="sayHello()">按钮</button>
</body>
<script>
    function sayHello(){
        console.log(this);
        //这里的this指向的window, 如何让这里的 this指向当前事件的DOM对象
    }
</script>
```

第二种情况

```
<body>
<button type="button" id="btn1" onclick="sayHello(this)">按钮</button>
</body>
<script>
    function sayHello(obj){
        console.log(this);
        console.log(obj);
        //这里的this指向的window, 如何让这里的 this指向当前事件的DOM对象
    }
</script>
```

第三种情况

```
<body>
<button type="button" id="btn1" onclick="sayHello.call(this)">按钮</button>
</body>
<script>
    function sayHello(){
        //在不传递参数的情况下，让这里的this指向当前事件对象
        console.log(this);
    }
</script>
```

这个时候我们通过 `call` 去改变了 `this` 的指向

第四种情况

```
<body>
<button type="button" id="btn1">按钮</button>
</body>
<script>
  var btn1 = document.querySelector("#btn1");
  btn1.onclick = function (event) {
    console.log(this);
    //这个时候this就指向当前的事件对象
  }
</script>
```

第五种情况：2级事件里面的this指向事件的DOM对象

```
<body>
  <button type="button" id="btn1">按钮</button>
</body>
<script>
  var btn1 = document.querySelector("#btn1");
  btn1.addEventListener("click", function (event) {
    console.log(this);
    //这里的this指向的是当前的事件对象
  });
</script>
```

2级事情的兼容性

2级事件是后面才出来的事件，所以之前的旧的浏览器就不支持，特别是 `IE8` 及以下，它是一种特殊的形式来表现的

w3c的标准	IE标准
<code>addEventListener</code>	<code>attachEvent</code>
<code>removeEventListener</code>	<code>detachEvent</code>

W3C标准

```
btn1.addEventListener("click",function (event){  
    console.log("我是W3C的2级DOM事件");  
})
```

低版本的IE

```
btn1.attachEvent("onclick",function (event){  
    console.log("我是IE里面的事件");  
});
```

为了就对这种兼容性，我们一般会使用条件注释的方式来导入JS的文件

```
<!--[if lt ie 9]>  
<script src="js/ie.js"></script>  
<![endif]-->  
  
<script src="js/w3c.js"></script>
```

取消事件的默认行为

在0级事件里面，如果要取消事件的默认行为，我们可以直接在事件方法的后面返回一个 `return false`。如下所示

```
<body>  
    <a href="http://www.baidu.com" onclick="sayHello();return false;">百度一下2222</a>  
    <hr>  
    <a href="http://www.baidu.com" id="a1">百度一下</a>  
</body>  
<script>  
    //事件先于响应  
    function sayHello(){  
        alert("你好啊");  
    }  
  
    //事件先于响应  
    var a1 = document.querySelector("#a1");  
    a1.onclick=function (event){  
        alert("你点击了我一下");  
  
        //取消事件的默认行为  
        return false;
```

```
    }
</script>
```

在2级DOM事件里面，怎么办呢？

```
<body>
  <a href="http://www.baidu.com" id="btn1">百度一下</a>
</body>
<script>
  var btn1 = document.querySelector("#btn1");
  btn1.addEventListener("click", function (event){
    console.log(event);
    alert("我是按钮1");
    //如果取消事件的默认行为
    //event.defaultPrevented 代表默认行为是否被阻止了，只读

    //阻止默认行为
    event.preventDefault();
  });
</script>
```

0级事件与2级事件的区别

1. DOM 的0级事件是以属性的形式存在的，不能多次赋值，而DOM的2级事件是通过 `addEventListener` 来实现监听，可以多次监听，所以2级事件会有一个事件链的概念
2. DOM的0级事件当中所有的事件名前面都有一个 `on`，2级而面去掉了前面的这个 `on`
3. DOM的0级事件在这里不能控制事件的传播方向，DOM的2级事件可以控制事件的传播方向
4. 0级事件如果要取消默认行为直接一个 `return false` 就可以了，而2级事件需要通过 `event.preventDefault()` 来完成
5. 0级事件如果想取消只需要将这个事件属性赋值为 `null` 就可以了，而2级事件如果想取消则要通过 `removeEventListener()` 去完成

HTML条件注释

普通的HTML注释形式是

```
<!-- 注释 -->
```

而IE5~IE9这5个版本的IE浏览器还另外支持一种特殊的if条件注释（感觉有点类似模板渲染时的语法结构）

```
<!--[if IE]> html语句 <![endif]-->
```

这样在处理IE浏览器兼容性问题的时候就可以把hack代码集中在一块了，或者其他意想不到的用途

if条件注释分为三种形式

1、是否IE（即：!）

```
<!--[if IE]> html代码 <![endif]-->
<!--[if !IE]> html代码 <![endif]-->
```

2、是哪个版本的IE（即：=）

```
<!--[if IE 6]> html代码 <![endif]-->
<!--[if IE 8]> html代码 <![endif]-->
```

3、是哪个区间的IE（即：<, <=, >, >=）

```
<!-- IE8以下版本的浏览器才会执行内部的html代码，如需要包含IE8则使用lte -->
<!--[if lt IE 8]> html代码 <![endif]-->
```

```
<!-- IE7以上版本的浏览器才会执行内部的html代码，如需要包含IE7则使用gte -->
<!--[if gt IE 7]> html代码 <![endif]-->
```

对于条件注释

IE5~IE9的视角是：

<!--[if IE]>	条件注释前缀
html代码	代码
<![endif]-->	条件注释后缀

其他浏览器的视角是：

```
<!--  
[if IE] html代码 <![endif]  
-->
```

普通注释前缀
字符串
普通注释后缀

那么如果想要if条件注释中所谓的html代码IE5~IE9虽然能识别但是不能执行，而其他浏览器也能识别并执行的话，可以这样写

```
<!--[if !IE]>--> html代码 <!--<![endif]-->
```

思路就是把条件注释语法结构的前缀和后缀分别给注释掉，这时

IE5~IE9的视角变成了：

```
<--[if IE]>  
--> html代码 <--  
<![endif]-->
```

条件注释前缀
代码
条件注释后缀

其他浏览器的视角则变成：

```
<!--[if IE]>-->  
html代码  
<!--<![endif]-->
```

一条普通注释
代码
一条普通注释

就都能识别出中间的代码了。

个人觉得使用条件注释的时候的一个注意点



```
<!--[if !IE]>-->  
<script src="js/jquery-3.2.1.min.js"></script>  
<!--<![endif]-->  
  
<!--[if IE]>  
<script src="js/jquery-1.12.4.min.js"></script>  
<![endif]-->
```



就是在每个浏览器中上边两个中只能使用一个的话一定得记得把IE9考虑进来，因为它也是能识别条件注释的（感觉IE9就是IE向现代高级浏览器过渡的东西，CSS3只支持一部分，但是之前IE专有的一些问题照样存在）

HTML5的多媒体技术

audio

```
<audio src="./assets/13.mp3" controls></audio>
<hr>
<audio controls>
  <source src="./assets/13.mp3" >
</audio>
```

属性

1. **controls** 是否在页面上面显示控制面板，它是一个单属性
2. **autoplay** 是否自动播放，它也是一个单属性【当前属性在谷歌浏览器里面第一次使用无效】
3. **currentSrc** 这个属性返回当前正在播放音乐的 **src** 地址
4. **currentTime** 当前音乐的进度时间，它以秒为单位
5. **duration** 当前音乐的总时间，它以秒为单位
6. **ended** 代表当前音乐是否已经播放结束，如果播放结束会触发 **onended** 事件
7. **loop** 它是一个单属性，用于设置当前音乐是否循环播放
8. **muted** 它是一个单属性，用于设置当前音乐是否静音
9. **volume** 用于设置当前音乐播放的音量，它的值是0-1
10. **playbackRate** 用于设置当前音乐播放的倍数，默认值是1
11. **paused** 代表当前音乐是否处理暂停状态，它是一个单属性，**true** 代表音乐暂停了，**false** 代表音乐正在播放
12. **networkState** 表示当前媒体的网络连接状态：0表示空，1表示正在加载，2表示正在加载元数据，3表示已经加载了第一帧，4表示加载完成
13. **readyState** 表示媒体是否已经就绪（可以播放了）。0表示数据不可用，1表示可以显示当前帧，2表示可以开始播放，3表示媒体可以从头到尾播放，4可用数据足以开始播放
14. **preload** 预先加载，**metadata** 元数据（音频的基本数据信息），**auto** 自动加载

事件

1. **oncanplay** 当音频已经在加载并且可以开始播放了就会触发这个事件
2. **onended** 当音频播放结束以后就会自动触发这个事件，如果当前的音频添加了 **loop** 属性则会进行单曲循环，永远不会触发这个 **onended** 事件
3. **onloadeddata** 当音频加载到了第一帧就会触发
4. **onplay** 当音乐开始播放的时候触发
5. **onplaying** 当音乐已经开始播放的时候触发
6. **onratechange** 当音频的播放速度被改变的时候就会触发这个事件
7. **onpause** 当音乐暂停的时候触发的事件
8. **onvolumechange** 当音乐的音量改变的时候触发
9. **onwaiting** 当播放网络音频时，网络情况比较差就会出现缓冲，需要等待加载的时候就会触发这个事件
10. **ontimeupdate** 当音乐在播放的时候会自动触发（通俗的说就是音乐播放的时间发生变化以后）

方法

1. **play()**
2. **pause()**

video

音频与视频是一样的，只是它的标签不一样而已，它使用 **video** 标签

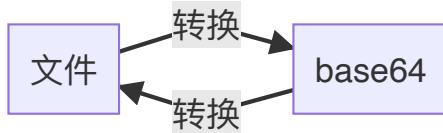
```
<video id="v1" src="./assets/v1.mp4" controls preload="auto"></video>
<!-- 或 -->
<!-- <video id="v1">
<source src="./assets/v1.mp4">
</video> -->
```

视频标签的属性与上面的音频标签的属性保持了一致的原则，扩展了一些新的东西

1. **poster** 当前视频的海报图片
2. **dom.requestFullScreen()** 请求全屏播放
3. **document.exitFullscreen()** 退出全屏播放
4. **dom.requestPictureInPicture()** 请求画中画播放
5. **document.exitPictureInPicture()** 退出画中画

文件与base64

Base64是网络上最常见的用于传输8Bit字节码的编码方式之一，Base64就是一种基于64个可打印字符来表示二进制数据的方法。【理论上来讲，base64可以表示任何数据，也就可以表示任何文件】



文件是可以转换成 `base64` 的，同样 `base64` 也可以转换成我们的文件。

如果要将文件转换成 `base64`，那么我们就需要使用一个特定的内置对象（这个对象是HTML5里面新出的）`FileReader`

```
<body>
    <input type="file" onchange="fileChange(this)">
</body>
<script>

    function fileChange(obj) {
        var file = obj.files[0];
        // 将文件转换成base64就非常简单
        var reader = new FileReader();
        // DataURL 就可以理解为base64
        reader.readAsDataURL(file);           //它会将这个文件转换成base64
        //不一定立即就得到结果，它会需要一定的时间
        reader.onload = function () {
            //onload代表的就是读取完成了，得到base64字符串
            console.log(reader.result);
        }
    }
</script>
```

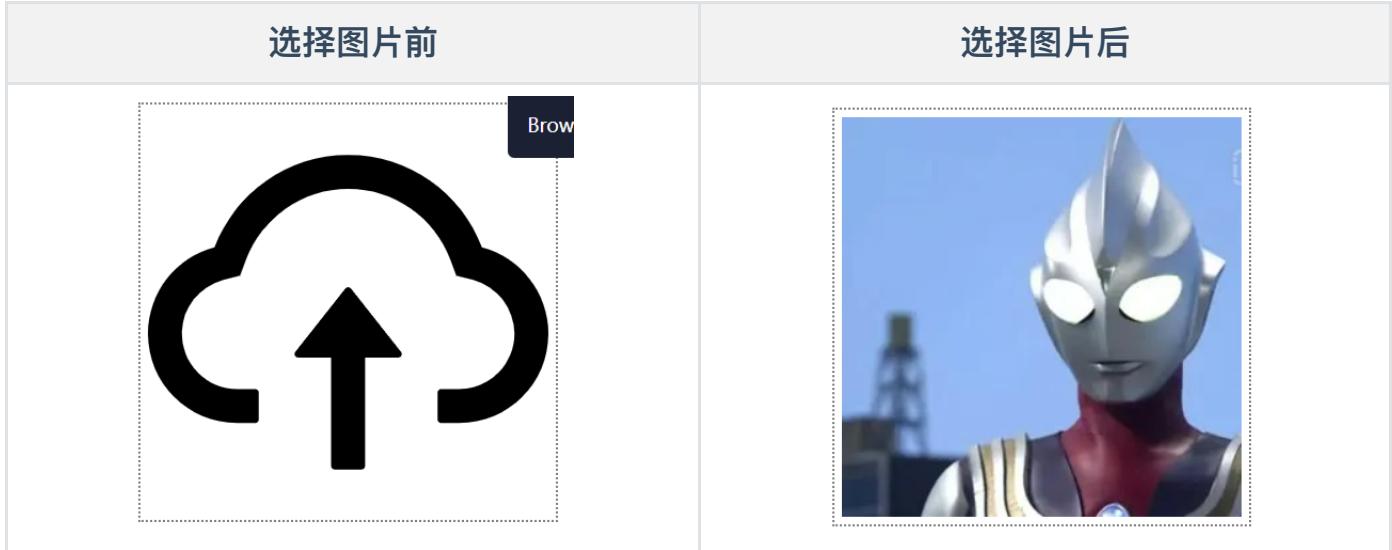
通过上面的代码，我们已经得到了 `base64` 的字符串。但是这个东西可以干什么？

base64可以做为url路径去使用

```
<body>
    
    <input type="file" id="f1" onchange="fileChange(this)">
```

```
</body>
<script>
    // 要求点击图片以后,弹出一个选择文件的对话框
    function imgClick() {
        //当图片被点击的时候,其实点击的是f1
        var f1 = document.querySelector("#f1");
        f1.click();
    }

    function fileChange(obj) {
        if (obj.files.length > 0) {
            var file = obj.files[0];
            console.log(file);
            var reg = /^image\/(jpe?g|webp|png|gif|svg|+xml)$/;
            if (reg.test(file.type)) {
                if(file.size / 1024 / 1024 > 2){
                    alert("图片不能大于2M")
                }
                else{
                    var reader = new FileReader();
                    reader.readAsDataURL(file);
                    reader.onload = function(){
                        var img1 = document.querySelector(".img1");
                        img1.src = reader.result;
                    }
                }
            }
            else {
                alert("请选择图片");
            }
        }
        //要求1:必须是图片
        //要求2:图片的大小不能超过2M
        //要求3:选中图片以后在img里面预览
    }
</script>
```



摄像头

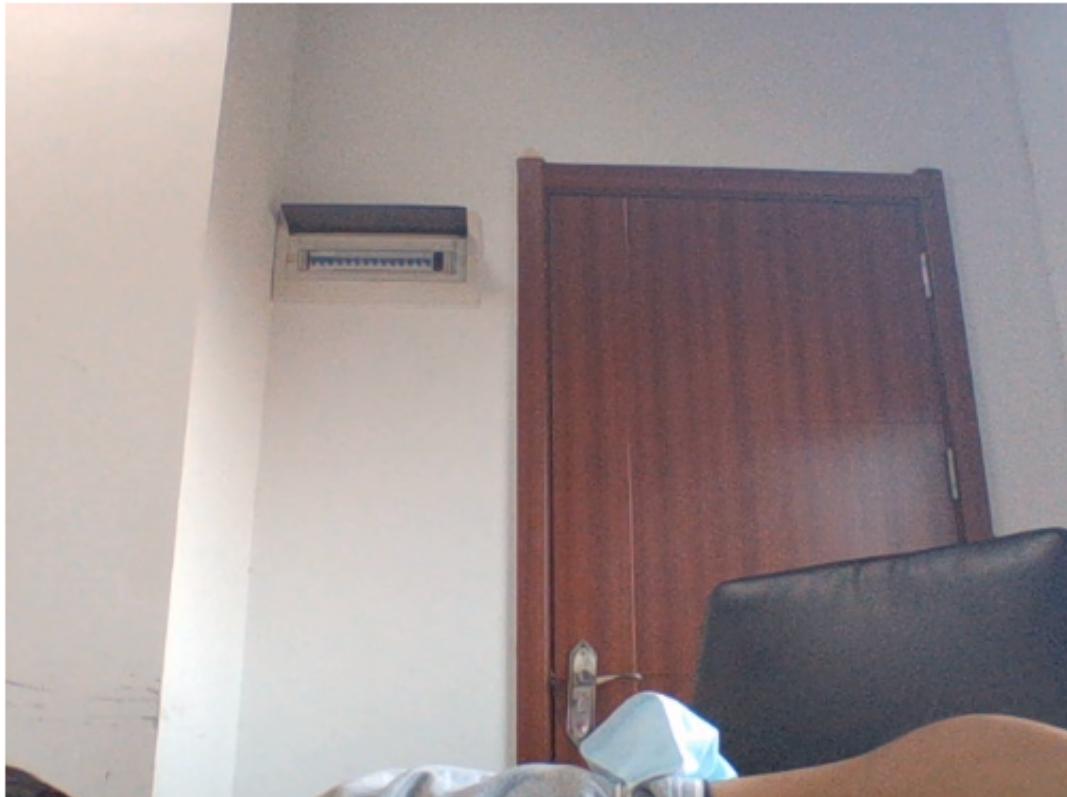
我们如果想要打开电脑上面的摄像头我们可以借助于浏览器去完成，而浏览器在DOM里一个专门的对象叫 `navigator`、在这个对象下面有一个方法叫 `getUserMedia`，它就可以实现这个功能

```

<body>
    <!-- src后面接一个路径 -->
    <video src="" id="v1" controls></video>
    <hr>
    <button onclick="openCamera()" type="button">开启摄像头</button>
</body>
<script>
    function openCamera() {
        navigator.getUserMedia({
            video: true,
            audio: false
        }, function(stream){
            //获取设备的信息成功
            // console.log(stream);
            var v1 = document.querySelector("#v1");
            // v1.src = stream;
            //stream不是一个路径,是一个对象 ,所以直接赋值给src就会报错
            // console.log(typeof stream);
            // 如果赋值进去的是一个对象 ,应该使用srcObject的属性
            v1.srcObject = stream;
            v1.play();

        }, function(error){
    
```

```
//获取设备的信息失败  
console.log(error);  
});  
}  
</script>
```



开启摄像头

canvas画布

canvas是html5里面新出的一个标签，它的中文意思是画布，程序员一把喜欢把它理解成一虚拟的屏幕

画布的创建

```
<canvas id="c1" width="400" height="400"></canvas>
```

画布非常特殊，它只能通过width/height来设置宽高，不能通过style来进行设置

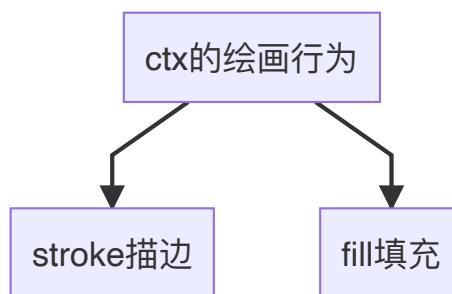
画笔的创建

画笔也叫绘画上下文，有了这个画笔以后，我们就可以在画布上面画任何自己所需要的东西了

```
/** @type {HTMLCanvasElement} */
var c1 = document.querySelector("#c1");
var ctx = c1.getContext("2d");
```

现在在的我们已经可以得到画笔 `ctx`，它是一个2d的画笔，有了这个画笔以后，我们就可以画任何我们所需要的东西了

但是在画东西之前，我有个事情要说清楚，`ctx`的画笔只有两种行为



描边出来的东西一定是空心的，填充出来的东西就一定是实心的

ctx基本方法与属性

画笔的操作是有很多种情况的，不同的情况对应的方法与属性也不一样

1. `font` 用于设置字体的大小与样式
2. `strokeText()` 描边一个空心的文本
3. `fillText()` 填充一个实心的文本
4. `strokeStyle` 设置描边的颜色
5. `fillStyle` 设置填充的颜色
6. `strokeRect()` 描边一个空心的矩形，一次成形
7. `fillRect()` 填充一个实心的矩形，一次成形

8. `clearRect` 清除一块矩形的区域
9. `rect()` 得到一个矩形的路径，后期可以通过描边 `stroke` 或 `fill` 来进行填充
10. `beginPath()` 开始一个新的路径。相当于把笔在墨池里面蘸一下墨水
11. `moveTo(x, y)` 将笔移动到一个指定的坐标
12. `lineTo(x, y)` 画一条线条指定的坐标
13. `stroke()` 对之前的路径进行描边
14. `fill()` 对之前的闭合区域进行填充
15. `lineWidth` 代表细节的粗细
16. `lineCap` 设置线条末端的形状
17. `textAlign` 用于设置文字的水平排列，它有 `left/center/right` 或 `start/center/end`
18. `textBaseline` 用于设置的垂直排列，它有 `top/middle/bottom/baseline`
19. `setLineDash([4, 10])` 设置线条为虚线
20. `arc(x, y, radius, start, end, direction)` 画一个弧度，3点钟方向为弧度的起点

渐变设置

这里的渐变的原理与我们之前CSS里面渐变的原理是相同的

线性渐变

```
// 第一步：先得到c1的画布
/** @type {HTMLCanvasElement} */
var c1 = document.querySelector("#c1");
var ctx = c1.getContext("2d");

// ctx.strokeRect(50,50,300,100);
ctx.rect(50, 50, 300, 50);
ctx.stroke();
//能够将这个颜色设置成渐变
var gradient = ctx.createLinearGradient(0, 0, 300, 0);
gradient.addColorStop(0, "red");
gradient.addColorStop(0.5, "orange");
gradient.addColorStop(1, "blue");
//这个gradient就是我们所设置的线性渐变色
ctx.fillStyle = gradient;
ctx.fill();
```



除了填充的时候可以使用渐变，我们线条描边的时候也可以使用渐变

```
ctx.beginPath();           //开始一个新路径
ctx.moveTo(0,300);
ctx.lineTo(400,300);
ctx.lineWidth = 50;
ctx.strokeStyle = gradient;
ctx.setLineDash([30,4]);
ctx.stroke();
```

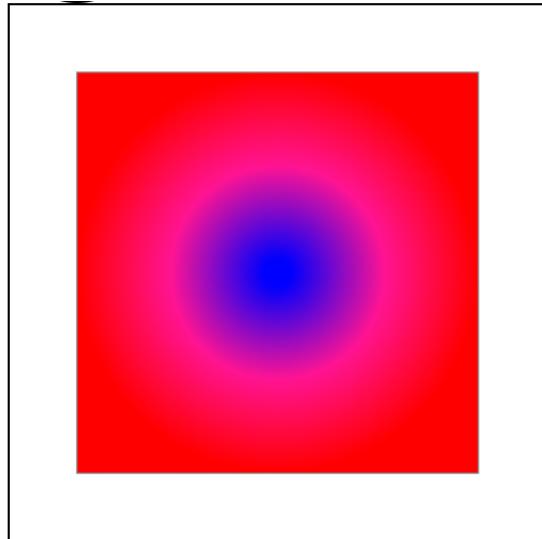


径向渐变

```
// 第一步：先得到c1的画布
/** @type {HTMLCanvasElement} */
var c1 = document.querySelector("#c1");
var ctx = c1.getContext("2d");

ctx.rect(50, 50, 300, 300);
ctx.stroke();

var gradient2 = ctx.createRadialGradient(200, 200, 10, 200, 200, 150);
gradient2.addColorStop(0, "blue");
gradient2.addColorStop(0.5, "deeppink")
gradient2.addColorStop(1, "red");
ctx.fillStyle = gradient2;
ctx.fill();
```

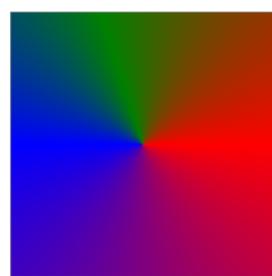


圆锥渐变

```
// 第一步：先得到c1的画布
/** @type {HTMLCanvasElement} */
var c1 = document.querySelector("#c1");
var ctx = c1.getContext("2d");

ctx.rect(50, 50, 300, 300);
ctx.stroke();

//第一步：先创建渐变色
var gradient3 = ctx.createConicGradient(0, 200, 200);
gradient3.addColorStop(0, "red");
gradient3.addColorStop(0.5, "blue");
gradient3.addColorStop(0.7, "green");
gradient3.addColorStop(1, "red");
ctx.fillStyle = gradient3;
ctx.fill();
```



绘制图片

图片的绘制分为2种情况，一种是静态绘制，一种是动态绘制。

绘制图片调用的方法是 `drawImage(图片对象, x, y, width, height)`

绘制静态的图片

静态图片的绘制指的是绘制页面上面已经加载过了的图片（静态绘制就是绘制一个页面上面已经存在的东西）

```
<body>
  
  <hr>
  <canvas id="c1" width="400" height="400"></canvas>
</body>
<script>
  /** @type {HTMLImageElement} */
  var img1 = document.querySelector("#img1");
  /** @type {HTMLCanvasElement} */
  var c1 = document.querySelector("#c1");
  var ctx = c1.getContext("2d");

  img1.onload = function () {
    //图片加载完成
    ctx.drawImage(img1, 0, 0, img1.clientWidth / 2, img1.clientHeight / 2);
  }
</script>
```

静态绘制是用得非常广泛的技术

场景一：实现video的视频截图并下载保存

```
<body>
  <video id="v1" src="assets/全班一起歌唱祖国太好听了.mp4" controls></video>
  <canvas id="c1"></canvas>
  <hr>
  <button type="button" onclick="takePhoto()">截图</button>
</body>
<script>
  /** @type {HTMLVideoElement} */
  var v1 = document.querySelector("#v1");
```

```

/** @type {HTMLCanvasElement} */
var c1 = document.querySelector("#c1");
var ctx = c1.getContext("2d");
v1.onloadedmetadata = function () {
    c1.width = v1.clientWidth;
    c1.height = v1.clientHeight;
}

/**
 * 截图
 */
function takePhoto() {
    // 本意是绘图，在这里是把视频里面的东西绘制到画面上面
    ctx.drawImage(v1, 0, 0, c1.width, c1.height);
    // 截图以后，将画布上面的图像信息转换成base64，DataURL指的就是base64
    var base64Str = c1.toDataURL("image/png");
    // 使用a标签下载
    var a = document.createElement("a");
    a.href = base64Str;
    a.download = "标哥的截图.png";
    a.click();
}
</script>

```

1. `drawImage()` 这个方法不仅可以绘制图片，还可以绘制视频的当前帧
2. `canvas` 可以调用 `toDataURL()` 将画布上面的转变成base64
3. `a` 是可以实现下载的

场景二：摄像头拍照

之前在讲video与浏览器对象的时候，我们讲过一点，可以利用浏览器对象打开摄像头，然后将报像头的数据对接到video上面



```

<body>
    <video id="v1" controls></video>
    <hr>
    <canvas id="c1" width="400" height="400"></canvas>
    <hr>

```

```
<button type="button" onclick="openCamera()">打开摄像头</button>
<button type="button" onclick="takePhoto()">拍照</button>
</body>
<script>
    /** @type {HTMLCanvasElement} */
    var c1 = document.querySelector("#c1");
    var ctx = c1.getContext("2d");
    /** @type {HTMLVideoElement} */
    var v1 = document.querySelector("#v1");

    //打开摄像头
    function openCamera() {
        navigator.getUserMedia({
            video: true,
            audio: false
        }, function (stream) {
            console.log("成功");
            v1.srcObject = stream;
            v1.play();
        }, function (error) {
            console.log("失败");
            console.log(error);
        });
    }
    //拍照下载
    function takePhoto(){
        if(!v1.paused){
            //将视频里面的数据绘制在画布上面
            ctx.drawImage(v1, 0, 0, c1.width, c1.height);
            // 将画布上面的东西转换成base64
            var base64Str = c1.toDataURL("image/png");
            // 用a标签下载
            var a = document.createElement("a");
            a.href = base64Str;
            a.download = "照片.png";
            a.click();
        }
        else{
            alert("请先打开摄像头");
        }
    }
</script>
```



[打开摄像头](#) [拍照](#)

绘制动态的图片

绘制一个页面上面已经存在的东西，我们静态绘制；动态绘制就是绘制一个页面上原本不存在的动态创建的东西

```
<body>
  <canvas id="c1" width="400" height="400"></canvas>
</body>
<script>
  // 希望画一张图片上去
  /** @type {HTMLCanvasElement} */
  var c1 = document.querySelector("#c1");
  var ctx = c1.getContext("2d");

  // 动画的创建一个图片的DOM对象
```

```

var img = new Image();
img.src = "./img/h_R0.png";
img.onload = function(){
    console.log("图片加载完成");
    ctx.drawImage(img, 50, 250);
}
</script>

```

画布的globalCompositeOperation属性

globalCompositeOperation 属性设置或返回如何将一个源（新的）图像绘制到目标（已有的）的图像上。

源图像 = 您打算放置到画布上的绘图。

目标图像 = 您已经放置在画布上的绘图。

默认值:	source-over
JavaScript 语法:	context.globalCompositeOperation="source-in";

属性值

值	描述
source-over	默认。在目标图像上显示源图像。
source-atop	在目标图像顶部显示源图像。源图像位于目标图像之外的部分是不可见的。
source-in	在目标图像中显示源图像。只有目标图像之内的源图像部分会显示，目标图像是透明的。
source-out	在目标图像之外显示源图像。只有目标图像之外的源图像部分会显示，目标图像是透明的。
destination-over	在源图像上显示目标图像。
destination-atop	在源图像顶部显示目标图像。目标图像位于源图像之外的部分是不可见的。
destination-in	在源图像中显示目标图像。只有源图像之内的目标图像部分会被显示，源图像是透明的。
destination-out	在源图像之外显示目标图像。只有源图像之外的目标图像部分会被显示，源图像是透明的。
lighter	显示源图像 + 目标图像。
copy	显示源图像。忽略目标图像。
xor	使用异或操作对源图像与目标图像进行组合。

画布的globalAlpha属性

这个属性可以设置全局透明度

```

/** @type{HTMLCanvasElement} */
var c1 = document.querySelector("#c1");
var ctx = c1.getContext("2d");
ctx.globalAlpha = 0.5;      //在绘画的时候将透明度设置为0.5
var img = new Image();
img.src = "./img/2018上.jpg";
img.onload = function(){
    ctx.drawImage(img,0,0,c1.width,c1.height);
}

```



画布的变换

1. `translate(x,y)` 将画布的原点移动到一个新的坐标
2. `rotate(弧度)` 将画布按一个弧度进行旋转，一个圆的弧度是 `Math.PI*2`；

画布的配置保存及还原

1. `save()` 将当前的配置保存起来，配置信息入栈
2. `restore()` 还原之前的配置，配置信息出栈

```

/** @type {HTMLCanvasElement} */
var c1 = document.querySelector("#c1");
var ctx = c1.getContext("2d");

// 1.请画一个 黑色的 18px 微软雅黑的 "标哥哥"
ctx.font = "18px 微软雅黑";
ctx.fillStyle = "black";
ctx.fillText("标哥哥",10,100);
ctx.save();      //保存之前的配置

// 2.请画一个 蓝色的32px 华康少女字体的 "标哥哥"
ctx.font = "32px 华康少女字体";
ctx.fillStyle = "blue";

```

```
ctx.fillText("标哥哥", 10, 200);

// 3.请画一个 黑色的 18px 微软雅黑的 “帅哥哥”
ctx.restore();      //还原到之前的配置
ctx.fillText("帅哥哥", 10, 300);
```

通过上面的例子，我们已经可以看到配置信息的保存与还原的操作



配置信息还可以多次保存

```
<body>
  <canvas id="c1" width="400" height="400"></canvas>
</body>
<script>
  /** @type {HTMLCanvasElement} */
  var c1 = document.querySelector("#c1");
  var ctx = c1.getContext("2d");

  ctx.font = "32px 微软雅黑";
  ctx.fillStyle = "black";
  ctx.fillText("第一次", 0, 50);
  ctx.save();           //第一次保存配置

  ctx.font = "16px 宋体";
  ctx.fillStyle = "red";
  ctx.fillText("第二次", 0, 100);
  ctx.save();           //第二次保存配置

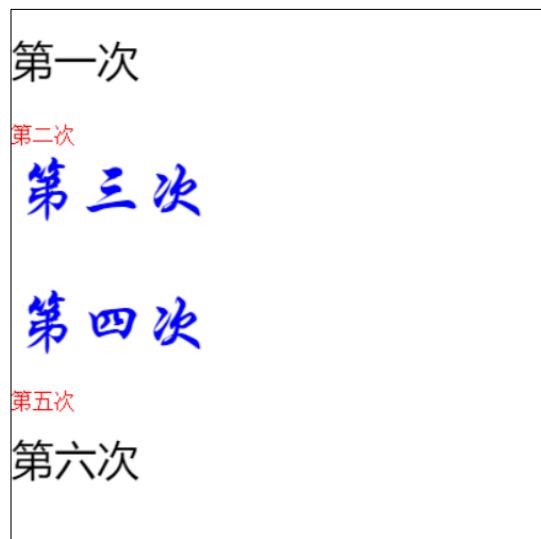
  ctx.font = "50px 华文行楷";
  ctx.fillStyle = "blue";
  ctx.fillText("第三次", 0, 150);
  ctx.save();           //第三次保存配置
```

```
//-----
ctx.restore();           //还原配置，配置信息出栈
ctx.fillText("第四次", 0, 250);

ctx.restore();
ctx.fillText("第五次", 0, 300);

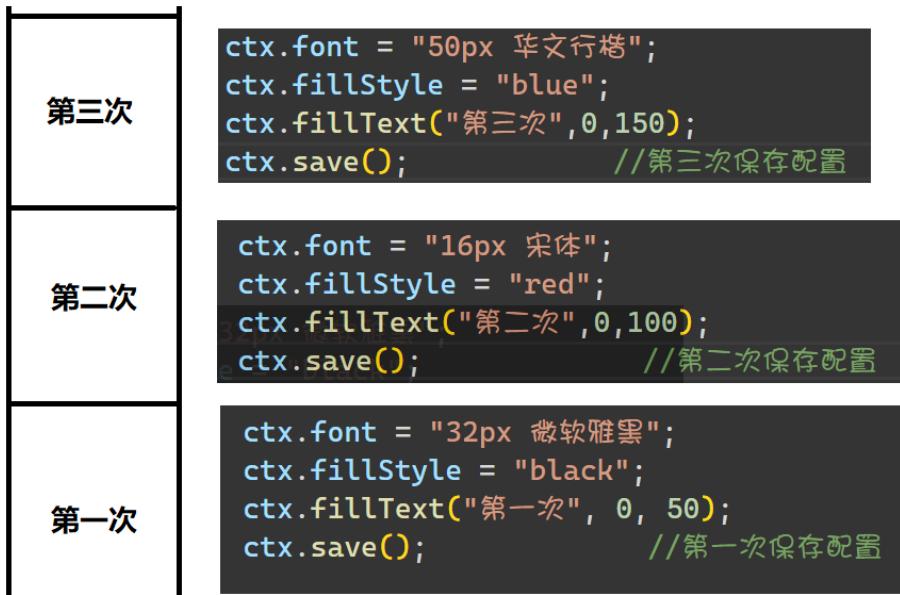
ctx.restore();
ctx.fillText("第六次", 0, 350);

</script>
```



每一次的save都会入栈

ctx配置信息的保存以是 **栈** 的形式在操作 每一次restore就会出栈



使用图像数据

在canvas里面，我们有一个方法，它可以使用我们canvas数据

- `getImageData()` 将画布里面的数据拿出来，它里面有一个rgba的数组
- `putImageData()` 将数据重新放回画布

```
<body>
  
  <hr>
  <canvas id="c1" width="400" height="400"></canvas>
</body>
<script>
  /** @type {HTMLCanvasElement} */
  var c1 = document.querySelector("#c1");
  var ctx = c1.getContext("2d");

  //第一步：先拿到这个图片
  /** @type {HTMLImageElement} */
  var img1 = document.querySelector("#img1");
  img1.onload = function () {
```

```

c1.height = img1.clientHeight;
ctx.drawImage(img1, 0, 0, c1.width, c1.height);
var imageData = ctx.getImageData(0, 0, c1.width, c1.height);
// imageData.data //颜色数组
for (var i = 0; i < imageData.data.length; i += 4) {
    var r = imageData.data[i];
    var g = imageData.data[i + 1];
    var b = imageData.data[i + 2];
    // 取三个颜色的平均数
    var avg = ~~((r + g + b) / 3);

    imageData.data[i] = avg;
    imageData.data[i + 1] = avg;
    imageData.data[i + 2] = avg;
}
//这个时候 imageData就替换完成,被我们处理好了
ctx.putImageData(imageData, 0, 0);

}

</script>

```





通过这个技术，我们还可以实现图片的“扣绿”

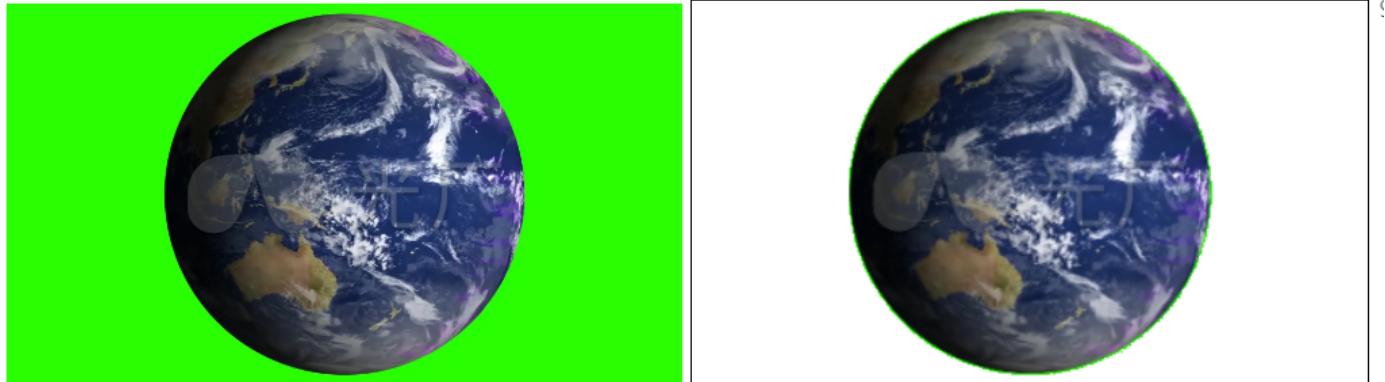
```

<body>
  
  <hr>
  <canvas id="c1" width="400" height="400"></canvas>
</body>
<script>
  /** @type {HTMLCanvasElement} */
  var c1 = document.querySelector("#c1");
  var ctx = c1.getContext("2d");

  //第一步：先拿到这个图片
  /** @type {HTMLImageElement} */
  var img1 = document.querySelector("#img1");
  img1.onload = function () {
    c1.height = img1.clientHeight;
    ctx.drawImage(img1, 0, 0, c1.width, c1.height);
    //第二步：获取画布的数据
    var imageData = ctx.getImageData(0, 0, c1.width, c1.height);
    for (var i = 0; i < imageData.data.length; i += 4) {
      var r = imageData.data[i];
      var g = imageData.data[i + 1];
      var b = imageData.data[i + 2];
      var a = imageData.data[i + 3];
      //第三步：判断绿色同，处理数据
      if (g > 250) {
        imageData.data[i + 3] = 0;
      }
    }
  }
</script>

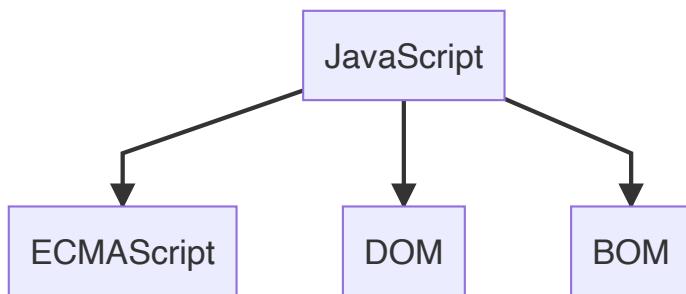
```

```
    }
    // 第四步：放回数据
    ctx.putImageData(imageData, 0, 0);
}
</script>
```



BOM基础

BOM的全称叫 **Browser Object Model** 浏览器对象模型，它是我们JS里面的一个组成部分



location对象

location是浏览器的内置对像，全称是 **window.location**，它指的是浏览器的地址栏

1. **href** 用于设置或获取浏览器地址栏里面的地址，如果是设置，则浏览器会跳转到新的网址
2. **hostname** 用于返回当前地址栏里面的主机名

<code>https://www.softteam.xin:9090/html_project</code>	<code>www.softteam.xin</code>
<code>http://127.0.0.1:5500/01.html</code>	<code>127.0.0.1</code>
<code>http://aaa.bbb.ccc/02.html</code>	<code>aaa.bbb.ccc</code>

3. **port** 返回当前地址栏里面的端口号

```
http://localhost:9998/02location.html      端口号9998  
http://www.softeem.xin:8090/                端口号8090  
https://www.baidu.com/                     端口号443  
http://www.softeem.com/web3/index.html     端口号80
```

https 开头的网址端口号默认是443, **http** 开头的端口号默认是 80

4. **host** 用于返回当前地址栏里面的 **hostname + port**

```
http://localhost:9998/02location.html  
主机地址 localhost:9998
```

5. **protocol** 返回当前地址栏里面的协议号, 它是 **http:** 或 **https:**
6. **origin** 返回当前地址栏的域的信息, 它是location三大属性之一, **origin=protocol+//+hostname+port**

```
http://localhost:9998/02location.html      域http://localhost:9998  
http://www.softeem.xin:8090/                域http://www.softeem.xin:8090/  
https://www.baidu.com/                   域https://www.baidu.com/  
http://www.softeem.com/web3/index.html    域http://www.softeem.com
```

这个域后期对我们的BOM编程限制非常大, 因为会有一个现象叫跨域【后面会讲, 也会遇到】

7. **hash** 返回或设置当前地址栏里面的哈希值, 它是location三大属性之一, hash值代表的就是地址栏 **#** 后面的东西 (后期的SPA开发里面用到)

```
http://localhost:9998/02location.html#div1  
//哈希 #div1
```

8. **search** 返回当前地址栏里面的 **?** 后面的东西, 它是location三大属性之一, 它常常用于跨页面传值

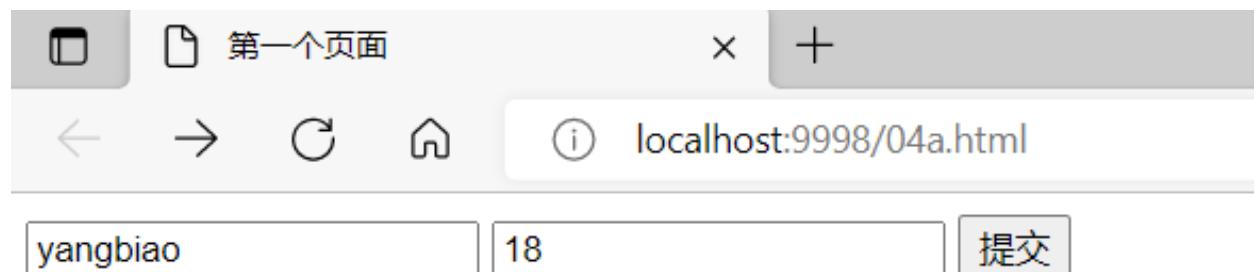
```
http://localhost:9998/03location.html?aaa=123  
//search就是?aaa=123
```

在讲到 **search** 属性的时候 就不得不提起BOM当中比较重要的一个对象叫 **URLSearchParams**

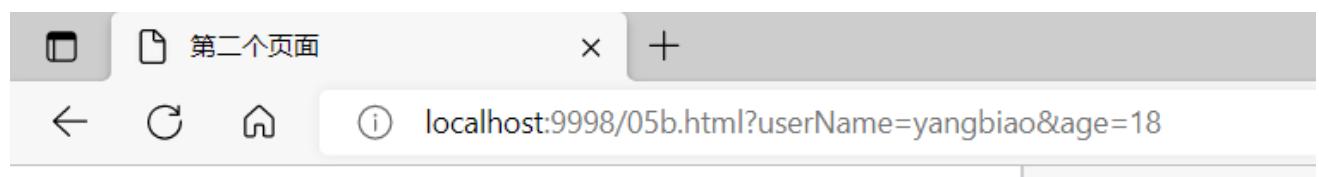
```
<body>  
  <input type="text" id="userName" placeholder="请输入账号">  
  <input type="text" id="age" placeholder="年龄">  
  <button type="button" onclick="toPage2()">提交</button>
```

```
</body>
<script>
    function toPage2() {
        var userName = document.querySelector("#userName").value;
        var age = document.querySelector("#age").value;
        // location.href = "05b.html";      这个代码只会让我们把页面跳转到
        05b, 但是数据没有过去

        location.href = "05b.html?userName=" + userName + "&age=" +
        age;
    }
</script>
```



现在我们回到 **05b.html** 这个页面



```
✓ location
< Location {ancestorOrigins: DOMStringList, href: 'http://localhost:9998/05b.html?us
  ▼ame.yangbiao&age=18', origin: 'http://localhost:9998', protocol: 'http:', host: 'l
  lhost:9998', ...} ⓘ
  ► ancestorOrigins: DOMStringList {length: 0}
  ► assign: f assign()
    hash: ""
    host: "localhost:9998"
    hostname: "localhost"
    href: "http://localhost:9998/05b.html?userName=yangbiao&age=18" ?
    origin: "http://localhost:9998"
    pathname: "/05b.html"
    port: "9998"
    protocol: "http:"
  ► reload: f reLoad()
  ► replace: f replace()
    search: "?userName=yangbiao&age=18" ?
  ► toString: f toString()
  ► valueOf: f valueOf()
    Symbol(Symbol.toPrimitive): undefined
  ► [[Prototype]]: Location
```

这个时候我们就可以看到地址栏的上面多了 `search` 部分，我们只需要拿到里面的值就可以了。如果要拿到里面的值就必须使用内置对象 `URLSearchParams`

```
// http://localhost:9998/05b.html?userName=yangbiao&age=18
var p = new URLSearchParams(location.search);
p.get("userName");           //yangbiao
p.get("age")                //18
```

9. `pathname` 代表当前浏览器地址栏里面的路径

```
http://localhost:9998/05b.html?userName=yangbiao&age=18
它的pathname就是/05b.html
```

10. `reload()` 重新加载当前地址，相当于刷新当前页面

11. `assign()` 载入一个新的网址，它的作用与 `href` 是一样的

```
location.assign("http://www.baidu.com");
location.href = "http://www.baidu.com";
```

上面的2个代码效果是一样的，没有区别

12. `replace()` 方法，替换当前地址栏里面的地址，也会跳到一个新的网页

`replace()` 这个方法是替换了地址栏里面东西，它退不回之前的页面

什么场景下面会使用`replace`

```
<body>
    <div>
        <input type="text" placeholder="请输入账号">
    </div>
    <div>
        <input type="password" placeholder="请输入密码">
    </div>
    <div>
        <button type="button" onclick="checkLogin()">登录</button>
    </div>
</body>
<script>
    function checkLogin(){
        alert("登录成功");
        // location.href = "http://www.baidu.com";
        // 如果通过href或assign去跳转页在，这样还是可以回到之前的页面
        location.replace("http://www.baidu.com");
    }
</script>
```

在上面的代码当中，如果是登录的场景里面，登录成功以后不应该再回到之前的页面，这个时候就阿使用 `replace` 来完成跳转

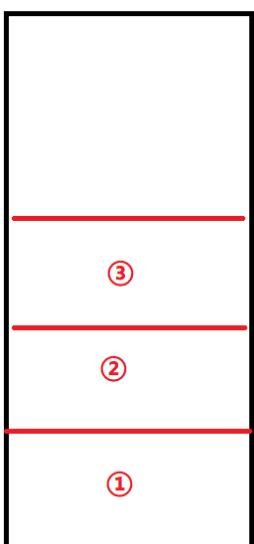
同时在支付的场景下面，如果支付成功或支付失败，都只能用 `replace` 来跳转

```
<body>
    <button type="button" onclick="payOrder()">确定支付</button>
</body>
<script>
    function payOrder(){
        if(confirm("确定要支付吗?")){
            // location.href = "07-1replace.html";
            // 支付成功以后，也不能退回到之前的页面
            location.replace("07-1replace.html");
        }
    }
</script>
```

关于href/assign与replace的原理是什么

为什么 `href/assign` 去跳转页面以后可以回到之前的页面，而 `replace` 不可以

history对象



history对象专门用于存储浏览器访问的历史记录

① `location.href = "http://localhost:9998/07replace.html";`

② `location.href = "http://www.baidu.com";`

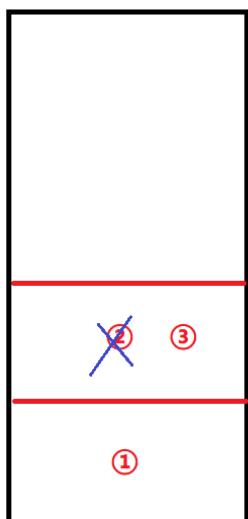
③ `location.assign("http://www.softeem.xin:8090/");`

现在后退，到了哪个页面

.

现在我们再来看 `replace` 的情况

history对象



① `location.href = "http://localhost:9998/07replace.html";`

② `location.assign("http://www.baidu.com");`

③ `location.replace("http://www.softeem.xin:8090");`

请注意这里的③用了`replace`, 它是的替换的意思，它会把②替换成③

如果这个时候再后退，应该是哪个页面？

history对象

history对象就是浏览器访问以后的历史记录对象

1. `length` 属性
2. `back()` 方法，相当于浏览器的后退功能
3. `forward()` 方法，相当于浏览器的前进功能
4. `go(step)` 方法，直接前进几步或后退几步，如果是负数就代表后退，如果是正数就代

表前进

navigator对象

这个对象代表当前的浏览器，它记录了当前浏览器的相关信息

1. `appVersion` 记录了当前浏览器的版本信息，后期我们可以通过这个版本信息来判断当前浏览器到底是哪个版本，如IE，或Chrome或火狐等
2. `maxTouchPoints` 当前浏览器是否支持多点触摸
3. `bluetooth` 获取当前设置的蓝牙信息
4. `connection` 返回当前设备的网络连接信息
5. `geolocation` 获取当前设备的位置，相当于地理定位

```
<body>
    navigator
        <button type="button" onclick="start()">开始定位</button>
</body>
<script>
    // 地理定位
    function start() {
        navigator.geolocation.getCurrentPosition(function (pos) {
            // 定位成功以后的回调
            console.log("定位成功");
            console.log(pos)
        }, function (error) {
            // 定位失败以后的回调
            console.log("定位失败");
            console.log(error)
        }, {
            // 是否启用高精度定位信息
            enableHighAccuracy: true
        })
    }
</script>
```

6. `getUserMedia()` 获取当前设备的多媒体信息，我们前面学过的打开电脑的摄像头与话筒用的就是这个

window对象

整个浏览器最高的对象就是window对象，这个对象是所有BOM的根对象，它里面有一些常用的方法跟大家说一下

1. `alert()` 弹出一个对话框，只有确定按钮
2. `confirm()` 弹出一个询问框，包含确定与取消按钮，如果点击确定则返回 `true`，如果点击取消就返回 `false`
3. `prompt()` 弹出一个输入框，用户可以输入，这个方法的返回值就是用户输入的内容，如果用户点击取消就返回 `null`

```
var x = prompt("请输入你心中最帅的那个人?");
```

localhost:9998 显示

请输入你心中最帅的那个人



4. `open(url:string, target:string, features?:string)` 打开一个新的页面

`url` 代表要打开的网址，`target` 代表打开的方式，这个与 `a` 标签的 `target` 保持一致，最后的 `features` 代表新打开的网页的样式

```
window.open("http://www.softeam.xin:8090", "_blank",
"width=375px, height=667px, left=200px, top=300px")
```

这个时候会在新的浏览器里面打开网页，打开的以后 `_target` 的方式打开，并且宽度和高度以及左边和上边的位置都设置

channelmode=yes no 1 0	是否要在影院模式显示 window。默认是没有的。仅限IE浏览器
directories=yes no 1 0	是否添加目录按钮。默认是肯定的。仅限IE浏览器
fullscreen=yes no 1 0	浏览器是否显示全屏模式。默认是没有的。在全屏模式下的 window，还必须在影院模式。仅限IE浏览器
height=pixels	窗口的高度。最小值为100
left=pixels	该窗口的左侧位置
location=yes no 1 0	是否显示地址字段。默认值是yes
menubar=yes no 1 0	是否显示菜单栏。默认值是yes
resizable=yes no 1 0	是否可调整窗口大小。默认值是yes
scrollbars=yes no 1 0	是否显示滚动条。默认值是yes
status=yes no 1 0	是否要添加一个状态栏。默认值是yes
titlebar=yes no 1 0	是否显示标题栏。被忽略，除非调用HTML应用程序或一个值得信赖的对话框。默认值是yes
toolbar=yes no 1 0	是否显示浏览器工具栏。默认值是yes
top=pixels	窗口顶部的位置。仅限IE浏览器
width=pixels	窗口的宽度。最小值为100

5. `close()` 关闭一个网页

跨页面作用技术 【重难点】

通过 `window.open()` 打开的页面叫父子页面

父子页面是通过 `open()` 打开以后的页面，父子页面可以相互的操作

01.html

```
<body>
  <h1 style="color: blue;">我是父页面</h1>
  <button type="button" onclick="a1()">打开子页面</button>
</body>
<script>
```

```
var childWindow = null;
function a1() {
    // 01的页面打开了02的页面
    // 那么01就是02的爹(父页面)
    childWindow = window.open("02.html");
    childWindow.document.querySelector("h2").innerText="我是你儿子啊，父亲在操作儿子";
    //这里的childWindow指代的就是就是我们的子页在的02的window对象
}
</script>
```

代码分析：

当我们通过 `window.open()` 打开一个页面以后，我们就可以返回一个 `window` 对象，这个对象就是新打开的页面的 `window` 全局对象

02.html

```
<!DOCTYPE html>
<html lang="zh">
<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>子页面</title>
</head>
<body>
    <h2 style="color: red;">我是子页面</h2>
</body>
<script>
    //如果在子页面里面要打爹
    console.log(window.opener);
</script>
</html>
```

当我们去点击父页面上面的按钮以后就会后期02这个页面，那么02的页面就是01的页面的子页面，父子之间是可以相互操作

`window.opener`; //通过这个可以找到父页面

localStorage本地存储

它的中文名称叫本地存储，它指的是浏览器的存储行为，我们可以在网页上面调用这个方法来实现对一个东西进行存储。它指的是浏览器的本地存储

1. `setItem(key, value)` 将一个值放在缓存当中

```
localStorage.setItem("userName", "yangbiao");
localStorage.setItem("age", "18");
```

2. `getItem(key)` 根据一个key从缓存当中取出值

```
localStorage.getItem("userName");      // "yangbiao"
localStorage.getItem("age");          // "18"
```

3. `removeItem(key)` 根据一个key从缓存当中删除某一项

```
localStorage.removeItem("userName");
```

4. `clear()` 清除 `localStorage` 的缓存

```
localStorage.clear();
```

`localStorage` 本身还是一个对象，它的每一个缓存就是一个属性名，所以缓存的操作也可以以对象的方式来完成

```
localStorage.setItem("nickName", "小小花");    // 放一个缓存
localStorage.nickName = "小小花";                // 以对象的形式来操作

localStorage.getItem("nickName");                 // "小小花"
localStorage.nickName;                           // "小小花"

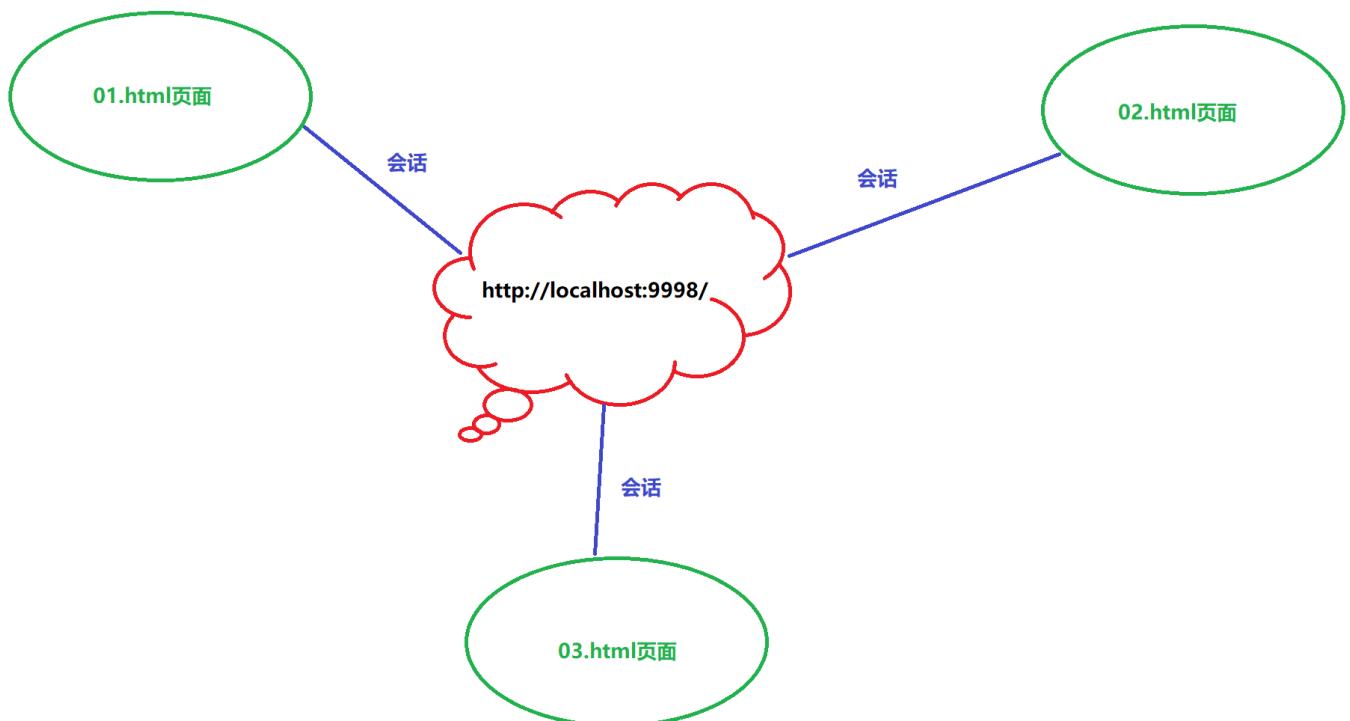
localStorage.removeItem("nickName");             // 删除缓存"小小花"
delete localStorage.nickName;
```

localStorage的注意事项及特点

1. `localStorage`可以跨页面共享值
2. `localStorage`在关闭浏览器以后还会保存
3. `localStorage`如果不手动清除会一直存在
4. `localStorage`不能跨域访问

上面的域是 `http://localhost:9998`，下面的域是 `http://127.0.0.1:8080`，两个域不相同，所以它们不能共享数据，也不能互相访问

sessionStorage会话存储



`sessionStorage`是一个会话存储，我们可以理解为它的缓存是在这一根蓝色的线上面的，它不在浏览器的本地，它所具备的方法与操作方式与`localStorage`保持一致

1. `setItem(key, value)` 赋缓存
2. `getItem(key)` 取缓存

3. `removeItem(key)` 删除一个缓存
4. `clear()` 清除`sessionStorage`的缓存

同时对象的操作方式也与`localStorage`一致

```
sessionStorage.setItem("age", "19");
sessionStorage.age = "19";
```

sessionStorage的注意事项及特点

虽然说`sessionStorage`与上面的学习过的`localStorage`操作方式保持一致，但是它们的特点完全不一样

1. `sessionStorage`不能跨页面共享数据，除非是父子页面
2. `sessionStorage`关闭浏览器以后数据会自动清除
3. `sessionStorage`不能跨域访问

cookie

cookie也是一个缓存，但是与上面的storage的缓存是完全不一样的，storage的缓存是在`window`对象下面，所以我们是使用`window.localStorage`和`window.sessionStorage`

但是`cookie`是在`document`对象下面，它的访问就是`window.document.cookie`

cookie也是一种缓存方案，但是它与前面缓存有一点区别，它可以上设置到期时间，到期以后自动清除，同时也要注意它的到期时间指的是GMT时间（GMT时间：格林尼治时间，也就是0时区的时间）

cookie的设置

```
document.cookie = "缓存名=缓存值";
document.cookie = "userName=biaogege";
```

筛选器											X	<input type="checkbox"/> 仅显示有问题的 Cookie
名称	值	Domain	Path	Expires / Max-Age	大...	H..	Secure	Sam...	Sa...	Parti...	Pri...	
userName	biaogege	localhost	/	会话	session	16					Med...	

我们可以看到，`userName`的cookie已经设置成功了，但是它的`Expires/Max-Age`指的是它的过期时间，仍然是会话，这是因为如果`cookie`在设置的时候不设置过期时间就默认以`session`为过期

设置带过期时间的cookie

```
document.cookie = "缓存名=缓存值;Expires=过期时间";
document.cookie = "userName=biaogege;Expires="+new Date("2022-09-07
12:00:00").toGMTString();
```

名称	值	Domain	Path	Expires / Max-Age	大...	H...	Secure	Same...	Sa...	Partit...	Pri...
userName	biaogege	localhost	/	2022-09-07T12:00:00.000Z	16						Med...
io	Kbb1my54XXBdfl...	localhost	/	会话	22	✓					Med...

cookie的取值

名称	值	Domain	Path	Expires / Max-Age	大小	H...	Secure	Same...	Sa...	Partit...	Pri...
age	18	localhost	/	2022-09-06T17:01:00.000Z	5						Medi...
userName	biaogege	localhost	/	2022-09-06T09:07:00.000Z	16						Medi...

现在cookie里面有2个值，怎么取出 cookie 里面的值呢？

```
document.cookie;           // 'userName=biaogege; age=18'
//如果在结果里面取得userName和age的值
function getCookieValue(cookieName) {
    var str = document.cookie + ";";
    var reg = new RegExp("(?=<=" + cookieName + "=).*?(?=;) ", "g");
    var result = str.match(reg);
    // 取到了就是一个数组，取不到就是null
    return result ? result[0] : null;
}
```

在上面的代码里面， cookie 的取值操作我们使用了正则表达式去完成

真正的开发里面，没有谁会自己手动的写正则取 cookie，因为有插件叫 js-cookie

```
<script src="js/js.cookie.js"></script>
<script>
    Cookies.get("userName");
    Cookies.get("sex");
    Cookies.get("age");

    Cookies.set("hobby", "123123123");

    Cookies.set("stuName", "张三峰", {
        Expires:new Date("2022-09-06 12:00:00").toGMTString()
    });

```

```
//删除某一个cookie  
Cookies.remove("stuName");  
</script>
```

cookie的注意事项及特点

1. cookie 是可以跨页面的
2. cookie 的 path 会隔离，子级的path可以访问父级的path，父级的访问不了子级的，同级别的 path 也是可以相互访问的
3. cookie 关闭浏览器以后不会自动消失，它的消失是根据过期时间来决定的，但是这个时间是GMT时间
4. cookie 是可以存储在 document 上面的，它会随着请求到达服务器，随着响应返回浏览器
5. cookie 是在大小限制，每个浏览器都不一样



一、浏览器允许每个域名所包含的cookie数：

Microsoft指出InternetExplorer8增加cookie限制为每个域名50个，但IE7似乎也允许每个域名50个cookie。

Firefox每个域名cookie限制为50个。

Opera每个域名cookie限制为30个。

Safari/WebKit貌似没有cookie限制。但是如果cookie很多，则会使header大小超过服务器的处理的限制，会导致错误发生。

注：“每个域名cookie限制为20个”将不再正确！

JSON

这里有一个注意事项，上面的3种缓存里面都有一个共同的缺点，它们只能存储字符串。如果现在有一个数组要存储进去，怎么办？如果有一个对象要存储进去，怎么办？

```

var arr = ["a", "b", "c", "d", "e"];
var userInfo = {
  userName: "张三",
  age: 18,
  sex: "男"
}
sessionStorage.setItem("arr", arr);
sessionStorage.setItem("userInfo", userInfo);

```

密钥	值
userInfo	[object Object]
arr	a,b,c,d,e

因为只能存储字符串，所以我们可以看到默认情况下 `arr.toString()` 了， `userInfo.toString()` 了

为了解决上面的问题，我们就有必要了解学习一下JSON

JSON (JavaScript Object Notation, JS对象简谱) 是一种轻量级的数据交换格式。它基于 ECMAScript (European Computer Manufacturers Association, 欧洲计算机协会制定的js规范) 的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

标哥说：“JSON是对象的字符串表示方法”

```

// 对象，也是数组
var arr = ["a", "b", "c", "d", "e"];
//JSON
var str = '["a", "b", "c", "d", "e"]';

//对象
var obj = {
  userName: "张三",
  age: 18
}
//JSON
var str2 = '{"userName": "张三", "age": 18}';

```

如果我们将一个对象变成了字符串以后，我们就可以将它以字符串的形式进行存储

其实在所有浏览器里面都有一个内置对象叫JSON

JSON.stringify()序列化

这个方法可以将JS的对象序列化成JSON字符串

```
//对象
var obj = {
    userName: "张三",
    age: 18
}
//我们现在想得到上面对象的JSON字符串，怎么办呢？
var str = JSON.stringify(obj);           //'{ "userName": "张三", "age": 18}'
```

JSON.parse()转化

这个方法可以将JSON字符串转化成JS对象

```
var str = '{"userName": "张三", "age": 18}';
var userInfo = JSON.parse(str);      //这里就得到了对象
```

URL对象

这个对象与我们之前所学习的 `location` 对象一样的，但是有一个点一定要讲那就是 `URLSearchParams`，它可以将 `search` 字符串转换成对象来操作，这一点在 `location` 也提到了

```
var str = "http://127.0.0.1:8080/07URL.html?userName=biaogege&age=18";
```

我们怎么样得到上面地址栏里面的 `userName` 和 `age` 的值呢？

第一种方式：使用我们之前所学习过的`URLSearchParams`

```
var str = "http://127.0.0.1:8080/07URL.html?userName=biaogege&age=18";
var p = new URLSearchParams("?userName=biaogege&age=18");
//现在的问题就是，你怎么拿？后面的东西      【待定：我们假设我们已经拿到？后面的东西】
var userName = p.get("userName");
var age = p.get("age");
```

在上面的这个方法里同，我们是假设我们拿到了 `?` 后面的字符串，但实际上我们没有拿到

第二种方式：使用URL对象

```

var str = "http://127.0.0.1:8080/07URL.html?userName=biaogege&age=18";
//第一步：直接将上面的str转变成一个url对象
var u = new URL(str);

var userName = u.searchParams.get("userName");
var age = u.searchParams.get("age");

```

```

Live reload enabled.

> str
< 'http://127.0.0.1:8080/07URL.html?userName=biaogege&age=18'
> u
< ▼ URL {origin: 'http://127.0.0.1:8080', protocol: 'http:', username: '', password: '', host: '127.0.0.1:8080', ...} ⓘ
  hash: ""
  host: "127.0.0.1:8080"
  hostname: "127.0.0.1"
  href: "http://127.0.0.1:8080/07URL.html?userName=biaogege&age=18"
  origin: "http://127.0.0.1:8080"
  password: ""
  pathname: "/07URL.html"
  port: "8080"
  protocol: "http:"
  search: "?userName=biaogege&age=18"
  ▶ searchParams: URLSearchParams {}      这里就直接有了URLSearchParams了
    username: ""
  ▶ [[Prototype]]: URL
>

```

URL.createObjectURL()

这个是重点，这个是超重点

25.4.3 对象URL

对象 URL 也被称为 blob URL，指的是引用保存在 `File` 或 `Blob` 中数据的 URL。使用对象 URL 的好处是可以不必把文件内容读取到 JavaScript 中而直接使用文件内容。为此，只要在需要文件内容的地方提供对象 URL 即可。要创建对象 URL，可以使用 `window.URL.createObjectURL()` 方法，并传入 `File` 或 `Blob` 对象。这个方法在 Chrome 中的实现叫 `window.webkitURL.createObjectURL()`，因此可以通过如下函数来消除命名的差异：

在之前的多媒体学习的时候，我们讲了文件与base64,将一个文件读取成 `base64` 格式，我们现在再来学习一下

```

<body>
  <img id="img1" alt="">
  <input type="file" onchange="fileChange(this)">
</body>
<script>
  function fileChange(obj) {

```

```

if (obj.files.length > 0) {
    var file = obj.files[0];
    var reg = /^image\/(jpe?g|bmp|gif|svg+xml|webp)$/;
    if (reg.test(file.type)) {
        //将图片读取成base64
        var reader = new FileReader();
        reader.readAsDataURL(file);
        reader.onload=function(){
            var img1 = document.querySelector("#img1");
            img1.src = reader.result;
        }
    }
    else {
        alert("请选择图片");
    }
}

```

</script>



```

<img id="img1" alt="data:image/jpeg;base64,/9j...uGOBMLjp1r08ror2N2ZV3a1j/9k
="> == $0

```

上面的代码就是我们将文件读取成了 `base64` 的格式，然后直接显示，这么做是没有问题的，但是我们也知道如果读取比较大的时候的时候（如视频video），这个时候就会变得非常慢

现在有了 `URL.createObjectURL()` 这一个方法以后，我们再次操作就会变得非常简单了

```

<body>
    <img id="img1" alt="">
    <input type="file" onchange="fileChange(this)">
</body>
<script>
    function fileChange(obj) {
        if (obj.files.length > 0) {
            var file = obj.files[0];
            var reg = /^image\/(jpe?g|bmp|gif|svg+xml|webp)$/;

```

```

        if (reg.test(file.type)) {
            var img1 = document.querySelector("#img1");
            //核心代码就这一句
            img1.src = URL.createObjectURL(file);
        }
        else {
            alert("请选择图片");
        }
    }
</script>

 == $0

```

这个时候的方式比上面读成 `base64` 的方式要快很多，也方便很多，直接将文件转转换成了一个对象URL

URL编码与解码

在浏览器里在，哪地址栏里面有中文的时候，它默认不会以中文来显示，而是一个“乱码”，这种“乱码”其实是一个特殊的加密编码

```
http://127.0.0.1:9998/03%E8%A7%86%E9%A2%91.html
```

在上面的网址里面，我们看到里面有一串乱码是因为里面有中文，在浏览器里面，浏览器有自带的路径加密与解密方式

1. `encodeURIComponent()` 将一个字符串进行URI的编码

```

encodeURIComponent("杨标");      // '%E6%9D%A8%E6%A0%87'
encodeURIComponent("abcde3r");   // 'abcde3r'

```

这个方法只对中文进行编码，对英文与数组是不会进行编码的

2. `decodeURIComponent()` 将一个字符串进行URI的解码

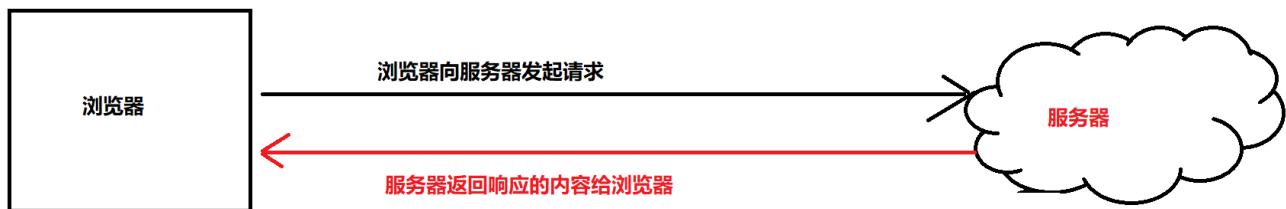
```
decodeURIComponent('%E6%9D%A8%E6%A0%87'); //杨标
```

同理，这个方法只对中文进行解码，英文与数组不进行操作

Ajax网络通讯

重点，难点，必用点

在Web端网络通讯方法有很多种



1. **ajax** 异步请求
2. **websocket** 双工套接字请求
3. **fetch** (后面ES6里面的)

现在我们主要的重点就放在Ajax上面

什么是Ajax

Ajax即Asynchronous Javascript And XML（异步JavaScript和XML）在2005年被Jesse James Garrett提出的新术语，用来描述一种使用现有技术集合的‘新’方法，包括：HTML或XHTML，CSS，JavaScript，DOM，XML，XSLT，以及最重要的 XMLHttpRequest。[3] 使用Ajax技术网页应用能够快速地将增量更新呈现在用户界面上，而不需要重载（刷新）整个页面，这使得程序能够更快地回应用户的操作

上面的一段话就是百度百科对于Ajax解释，**其实用标哥的话说就是一种浏览器向http/https服务器发起请求的一种方式，只是这种方式是一种异步的方式**

假设我们现在一个服务器地址，如下

```
http://www.softteam.xin:8888/public/musicData/musicData.json
```

请问如何向上面这个地址发起请求？

Ajax的创建过程

Ajax的核心是 XMLHttpRequest，它是浏览器的内置对象，用于发起网络请求，向 http/https 服务器发请求

场景：标哥（浏览器）想喝一杯奶茶，B27的楼底下就有一个蜜雪冰城的奶茶店（服务器），然后标哥不想自己去，决定让小岳岳跑一趟。这个时候我们就可以把小岳岳当成是 xhr

小岳岳就要去B27（地址）的奶茶店去买奶茶（发起求）

第一步：标哥把小岳岳叫过来

```
var xhr = new XMLHttpRequest();
```

第二步：标哥告诉小岳岳地址在哪里，怎么去

```
xhr.open("GET", url, true);
```

第三步：立即让小岳岳去买奶茶

```
xhr.send();
```

第四步：监控小岳岳的变化

```
xhr.addEventListener("readystatechange", function () {
  if (xhr.readyState == 4 && xhr.status == 200) {
    // 第五步：拿到服务器响应的结果
    // 它是一个JSON字符串
    var result = xhr.responseText;
    var obj = JSON.parse(result);
    console.log(obj);
  }
})
```

现在我们把上面的代码整理合并

```
var url = "http://www.softeam.xin:8888/public/musicData/musicData.json";
function getData() {
  //第一步：创建ajax的对象
  var xhr = new XMLHttpRequest();
  //第二步：建立连接
  // GET代表请求方式，后面我们还会讲到POST请求
  // url代表请求的服务器地址
  // true代表异步请求
  xhr.open("GET", url, true);
  //第三步：发送请求
  xhr.send();
  // 第四步：监听请求状态的变化
  xhr.addEventListener("readystatechange", function () {
    if (xhr.readyState == 4 && xhr.status == 200) {
      // 第五步：拿到服务器响应的结果
    }
  });
}
```

```
// 它是一个JSON字符串
var result = xhr.responseText;
var obj = JSON.parse(result);
console.log(obj);
}
})
}

}
```

Ajax对象的分析

Ajax请求的核心对象是 XMLHttpRequest，它是用来帮我们发请求的，我们要充分了解这个东西

属性

1. **readyState** 代表ajax请求状态的变化，它一共有0~5个值
 - 0. 初始化阶段，准备阶段
 - 1. 载入阶段，开始发起请求
 - 2. 载入完成，请求发送完成
 - 3. 解析阶段，服务器处理请求的阶段
 - 4. 完成，浏览器接收服务器响应的数据
2. **status** 代表ajax请求结果的http的状态码，200代表成功
 - 200代表成功
 - 304代表缓存
 - 403无权限
 - 404代表路径错误
 - 5xx代表服务器错误，服务不能执行一个正确的请求
3. **timeout** 设置请求超时的时间，如果请求超过多长时间得到不到响应，那么我们就判断它失败
4. **responseText** 请求成功以后服务器返回的文本信息
5. **responseType** 设置默认的响应数据类型，这个值默认是 **text**
 - **text** 设置响应的类型为文本格式
 - **json** 设置响应的类型为 **json** 格式，如果设置这个格式，后面在 **response** 的结果就会自动的调用 **JSON.parse()** 进行转换
 - **document** 返回一个网页格式的文件
 - **blob** 代表返回的是一个二进制的数据
 - **arraybuffer** 代表返回的格式是一个缓冲区数组

6. `response` 服务器响应的结果，它不一定是文本信息，当我们的 `responseType` 设置为 `text` 的时候，就可以直接使用 `responesText` 来得到结果，但是如果不是 `text` 的时候就需要通过 `response` 来得到结果

方法

1. `open(method, url, async)` 打开某一个链接

第一个参数 `method` 代表请求试，目前我们只学习了 `GET`，它常用的有 `GET/POST/PUT/DELETE/OPTIONS/PATCH` 等

第二个参数代表请求服务器的地址

第三个参数代表是否异步请求，默认是 `true` 【新版的浏览器只能设置为 `true`】

2. `send(params?)` 开始发送请求, `send`方法里面有参数的，只是在 `get` 里面没有参数
3. `setRequestHeader` 设置请求头（后面的POST请求会用到）

事件

1. `onreadystatechange` 请求状态的变化
2. `ontimeout` 请求超时以后触发
3. `onerror` 请求失败以后会触发
4. `onabort` 请求取消的时候
5. `onprogress` 请求的进度条发生变化的时候

Ajax的封装

```
/**  
 * Ajax请求的工具类  
 * @author 杨标  
 */  
  
var AjaxHelper = {  
    /**  
     * ajax发起get请求  
     * @param {string} url 请求的地址  
     * @param {Function} callBack 请求成功以后的回调函数  
     */  
    get: function (url, callBack) {  
        var xhr = this.init(callBack);  
        xhr.open("GET", url, true);  
        xhr.send();  
    }  
};
```

```
},
post: function (url, callBack) {
    var xhr = this.init(callBack);
    xhr.open("POST", url, true);
    xhr.send();
},
/***
 * 初始化
 * @param {Function} callBack
 * @returns {XMLHttpRequest} 创建好的xhr对象
 */
init: function (callBack) {
    var xhr = new XMLHttpRequest();
    xhr.addEventListener("readystatechange", function () {
        if (xhr.readyState == 4 && xhr.status == 200) {
            var result = xhr.response;
            if (typeof callBack === "function") {
                callBack(result);
            }
        }
    });
    return xhr;
}
}
```

同步与异步的概念

- 同步等待
- 异步执行

场景：

现在是下午，标哥讲了一天的课，要让小岳岳帮我去蜜雪冰城买杯奶茶，喝奶茶润一下嗓子再继续讲课

情况一：同步的场景

小岳岳拿着标哥给的5块钱屁颠屁颠的就跑了，标哥看着小岳岳离开的背影，心里久久不平静，它一定要等小岳岳把奶茶买回来，喝了以后再继续讲课

现在我们来分析一下

人物：标哥，小岳岳

事情：标哥要讲课，小岳岳要买奶茶

红色：标哥
蓝色：小岳岳

15:55

16:00 小岳岳在奶茶店买奶茶

16:10

标哥在讲课

标哥在等待小岳岳买奶茶

标哥喝了奶茶以后继续讲课

这种情况就是2个人在同一个时间线上面，它会造成时间阻塞

情况二：异步的情况

小岳岳拿着标哥的钱飞快的跑了出去，帮标哥买奶茶，标哥看着小岳岳这健步如飞的步伐，心中甚是感动，同时也看到了教室里面的学生看着标哥殷切的求知若渴的眼神，标哥不好意思停下来讲解，而是强忍着心中的疲惫继续讲课。不知过了多久，小岳岳回来了，标哥的课也讲得差不多了

红色：标哥
蓝色：小岳岳

15:55

16: 00

标哥继续讲课

16: 05

16:00

小岳岳买奶茶

16:05

这个时候2个时间线是不再同一个线上的，它们就是异步，不会造成时间阻塞

JSDoc注释

JSDOC的全称叫 `javascript document` 注释，它叫文档注释

在我们以前的注释里面，好像没有这个概念。在大型项目里面，文档注释是非常频繁的
以前学习注释的时候分为两种类型的注释，单行注释与多行注释

```
//单行注释  
/*  
多行注释  
*/
```

```
/**  
* 文档注释  
*/
```

文档头部相关的注释

```
/**  
* @author 作者  
* @description 测试文件  
* @date 2022-12-1  
* @version 1.0  
*/
```

方法类型注释

```
/**  
* 计算两个变量相加的和  
* @param {number} a 第一个值  
* @param {number} b 第二个值  
* @returns {number} 返回两个变量相加的值  
*/  
function add(a, b) {  
    var x = a + b;  
    return x;  
}
```

如果想注释复杂数据类型，可以用下面的方式

```
/**  
 * 遍历打印一个数组  
 * @param {Array} arr  
 */  
function printArr(arr){  
    arr.forEach(function(item, index, _arr){  
        console.log(item);  
    })  
}
```

如果是复杂数据类型，就写这个对象的构造函数

```
/**  
 * 这里用于演示特定类型的数组  
 * @param {Array<string>} arr  
 */  
function printArr(arr){  
    //arr是一个装了字符串的数组  
    arr.forEach(function(item){  
        item.toUpperCase();  
    })  
}  
  
/**  
 * 这里也可以注释字符串的数组  
 * @param {string[]} arr  
 */  
function printArr2(arr){  
    arr.forEach(function(item, index){  
        item.toUpperCase();  
    });  
}
```

变量类型的注释

```
/**  
 * @type {string}  
 */  
var x ;  
  
/** @type {string} */
```

```
var x ;  
  
/** @type {HTMLImageElement} */  
var img1 = document.querySelector("#img1");  
  
/** @type {HTMLInputElement} */  
var txt1 = document.querySelector("#txt1");  
  
/**  
 *  
 * @param {Array} arr  
 */  
function printArr(arr){  
    arr.forEach(function(**@type {String} */ item, index, _arr){  
        item.toUpperCase();  
    })  
}
```

默写总结

- 总结人：杨标
- 总结时间：2022年8月25日
- 默写班级：H2204班

第一次默写

默写日期：2022年7月6日

1. 写出计数器使用的3个属性

- 你要对谁计数，那么就在这个元素的外层添加 `counter-reset` 属性
- 你要对谁计数，那么就在这个元素上面添加 `counter-increment` 属性
- 你要把统计的结果放在什么地方，那么就使用伪元素 `::before`/`::after` 来追加 `counter()` 属性

```
/* 你要对谁计数，那么就在这个元素的外层添加`counter-reset`属性 */  
fieldset {  
    /* 重置一个计数器，也可以认为是定义一个计数器*/
```

```
        counter-reset: aaa 0;
    }

/* 你要对谁计数，那么就在这个元素上面添加`counter-increment`属性 */
fieldset>p {
    counter-increment:aaa 1 ;
}

/* 你要把统计的结果放在什么地方，那么就使用伪元素`::before/::after`来追加
`counter()`属性 */
fieldset>p::before{
    content: counter(aaa);
}
```

根据上面的案例，我们可以得到3个属性操作

- **counter-reset:计数器名子 初始值** 初始化一个计数器，并初始化这个计数器值
- **counter-increment:计数器名子 增量** 在统计数量的时候，使用哪个计数器，并设置增量为多少
- **counter(计数器)** 从计数器里面拿出当前计数的值

2. 写出溢出处理的属性名与属性值

```
overflow:hidden;
overflow:auto;
overflow:visible;
overflow:scroll;
/*同时，还要注意一下overflow-x, overflow-y的使用方法*/
```

3. 边框的属性以及边框线条类型的属性值

border-style /*边框属性*/

- **solid** 实线
- **dashed** 虚拟
- **dotted** 点线
- **double** 双线
- **groove** 线槽

4. list-style-type下面任意的4个属性值

- **none** 取消前面的符号
- **disc** 实心圆
- **circle** 空心圆
- **square** 实心矩形

- `upper-alpha` 大写英文字母
- `lower-alpha` 小写英文字母
- `upper-roman` 大写罗马文
- `lower-roman` 小写的罗马文
- `decimal` 数值
- `decimal-leading-zero` 数值，但是以0开始，如 `01`
- `cjk-heavenly-stem` 采用天干记数，如 甲、乙、丙、丁、戊、己、庚、辛、壬、癸
- `cjk-earthly-branch` 采用地支纪年，如 子、丑、寅、卯、辰、巳、午、未、申、酉、戌、亥

5. `list-style-position`的属性值

- `outside` 符号在li这个元素的外部【默认值】
- `inside` 符号在在li元素的内部

6. 圆角边框的属性名

```
border-radius      /*四个角*/  
border-top-left-radius;    /*左上*/  
border-top-right-radius;   /*右上*/  
border-bottom-right-radius; /*右下*/  
border-bottom-left-radius;  /*左下*/
```

7. 文本居中

```
text-align:center;
```

8. 字体变为倾斜的状态用什么属性与属性值

- `font-style:italic` 使用任何字
- `font-style:oblique` 使字体倾斜

上面的两个方法都可以让页面上面的文字产生倾斜的效果，但是原理是不一样的，`italic` 先到系统的字体库里面去看，看有没有倾斜，如果有倾斜，则使用字体的斜体；如果系统的字体库里面没有这个字体的斜体，它会直接让文字倾斜（也就是退而求其次，使用 `oblique` 来倾斜）

9. 字体加粗的CSS属性及属性值，字体变细的属性值

`font-weight` 用于设置字体的权重

- `normal` 字体体质正常
- `bold` 字体加粗
- `bolder` 字体还粗一点

- **lighter** 字体变细

正常下完整的字体应该是有9个等级，分别是 **100~900**,如果是完整字体，我们可以设置数字来表示字体的权重， **normal** 对应的是400

10. 文字阴影怎么使用

```
text-shadow:水平偏移 垂直偏移 阴影模糊 阴影颜色;  
text-shadow: 20px 30px 3px blue;
```

11. 如何自定义字体

```
/* 自定义字体 */  
@font-face {  
    /* 定义字体的名称 */  
    font-family: "bgg";  
    /* 说明字体在什么地方 */  
    src: url("fonts/HKSN.ttf");  
}  
.aaa {  
    font-family: "bgg";  
}
```

12. 单行文本溢出省略

```
p{  
    white-space:nowrap;  
    overflow:hidden;  
    text-overflow:ellipsis;  
}
```

13. 多行文本溢出省略【3行】

```
p{  
    display: -webkit-box;  
    /* 盒子里面的内容垂直排列 */  
    -webkit-box-orient: vertical;  
    /* 在第2行的时候省略掉 */  
    -webkit-line-clamp: 2;  
    /* 溢出的部分隐藏 */  
    overflow: hidden;  
}
```

14. background-size: 覆盖、包含

- `contain` 这个属性值是让背景图片完全显示在盒子里面，这样可能会有一个方向空出来
- `cover` 这个属性值是让背景图片完全覆盖住盒子，这样可能会有一个方向的图片被裁剪掉显示不出来

15. 设置左上角的圆角边框为30px

```
border-top-left-radius:30px;
```

16. CSS里面英文进行大小写转换用什么属性和属性值

`text-transform` 设置英文字母的大小写转换

- `uppercase` 大写字母
- `lowercase` 小写字母
- `capitalize` 单词的首字母

17. 文字大小什么CSS属性，有哪些单位？最小字体多少？默认字体多少？

- `px` 像素
- `pt` 字号
- `em` 一个父级元素【也是自己元素】的字体大小，`em`的全称是 `element` 元素
- `rem` 一个根元素字体的大小，它是一个响应式字体单位，它在 `html` 标签的 `font-size` 大小为标准
- `vw/vh` 响应式字体单位，全称叫 `viewport width` 和 `viewport height`

最小字体是12px,默认字体是16px或12pt

18. 文本首行缩进

```
text-indent:2em; /*首行缩进2个文字*/
```

19. 文字的装饰CSS，它由哪几个属性组成

`text-decoration` 文字的描述信息，它是一个简写的属性

- `text-decoration-line` 设置线条的位置
 - `underline` 下划线
 - `overline` 上划线
 - `line-through` 中划线（删除线）
 - `none` 不要设置任何线条
- `text-decoration-color` 设置的线条的颜色
- `text-decoration-style` 设置线条的类型
 - `solid` 实现

- `dashed` 虚线
- `dotted` 点线
- `double` 双线
- `wavy` 波浪线

三个属性合起来就变成了上面的 `text-decoration` 一个属性

20. CSS常用的颜色值类型有哪几种?

- 枚举色
- 十六进制色
- RGB三元色
- RGBA透明度颜色

这里要注意，RGBA是可以设置一个颜色的透明度，同时 `transparent` 是全透明的意思

21. a标签里面target的属性值有哪些?

- `_blank` 新容器打开链接
- `_self` 当前容器打开链接
- `_framename` 在指定的iframe里面打开
- `_parent` 在父级打开
- `_top` 在顶级容器打开

22. 表格里面，行合并、列合并

- 行合并 `span`
- 列合并 `colspan`

23. 在CSS外部样式表里面，有一个文件a.css,怎么导入到网页里面

```
<link rel="stylesheet" type="text/css" href="a.css">
<style>
  @import url("a.css");
</style>
```

24. 如何将网页设置为UTF-8(中文)

```
<meta charset="utf-8">
```

25. 写出HTML里面特殊的文字标记

```
&gt;  
&lt;  
&deg;  
&nbsp;  
&reg;  
&copy;
```

第二次默写

默写时间：2022年7月13日

1. 如何让后代元素主动的去继承父级元素的样式 【属性】
inherit
2. 相对定位， 绝对定位的属性值
 - 相对定位： relative
 - 绝对定位： absolute
3. 字体倾斜的2个单词

```
font-style:italic;  
font-style:oblique;
```

4. 块级元素让宽度适应内容

```
width:fit-content;
```

5. 背景图大小里面的覆盖、包含
 - 覆盖cover
 - 包含contain
6. 盒子模型由外向内4层分别是什么
margin-border-padding-content
7. 省略号单词
ellipsis
8. 英文单词进行大小写转换的属性，以及3个属性值
text-transform
uppercase
lowercase
capitalize

9. CSS层叠性里面，一个元素最终样式是有哪几部分样式来共同决定的
行内样式>内部样式块>默认样式>继承样式

10. 标准流布局里面，各种元素的水平居中方案

- 块级：margin:auto
- 行内，行内块级：外层块级或行内块这样上面加 text-align:center

11. 浮动脱流以后的元素居中方案

- 父级元素变 inline-block，再到父级设置 text-align:center
- 父级宽度设置 fit-content,然后 margin:auto

12. 定位脱流以后的元素水平居中方案

```
div{  
    left:0;  
    right:0;  
    margin:auto  
}
```

13. 如果恢复浮动以后的元素的父级元素高度

- 在浮动元素的后面添加块级元素clear:both;
- 在浮动元素的父级添加overflow
- 使用伪类::after

14. 什么样的元素类型不遵守标准盒子模型规范，是怎么不遵守的？

- 行内元素不遵守，margin-left/margin-right正常
- margin-top/margin-bottom没有
- padding-left/padding-right是正常
- padding-top/padding-bottom只能撑开自身，撑不开父级

15. 解决margin穿透有什么办法？

- border-top
- BFC:overflow,float,absolute

16. 一个元素宽度丢失了会是什么原因？

脱流，元素类型转换为inline

17. 绝对定位元素以谁为参照物进行定位

以父级的除static的定位为参照物
或父级trasnform为参照物

18. border-color如果将颜色设置成与内容color保持一致，使用什么属性值？

```
border:1px solid currentcolor;
```

主要使用的 `currentcolor` 这个属性值

19. 一个元素使用定位以后，会多出哪几个CSS属性？

`left/right/top/bottom/z-index`

20. 请列出权重值为10的CSS选择器

类，伪类，属性

21. 请选出`fieldset`元素下面的除了第2个`h2`标签的所有标签

```
fieldset *:not(h2:nth-of-type(2))
```

22. 请选出`ul`下面的`li`中前3项`li`

```
ul>li:nth-child(-n+3)
```

23. 多行文本溢出省略

```
display:-webkit-box;  
-webkit-box-orient:vertical;  
-webkit-line-clamp:2;  
overflow:hidden;
```

24. 滤镜的属性及常用的几个属性值（将页在统一调成灰色）

```
filter:blur(2px)  
filter:alpha(opacity=50);  
filter:hue-rotate(30deg)  
filter:grayscale(0.5);
```

25. `border-radius`四个或八个值分别代表什么？

```
border-radius: 50px 0px 0px 0px ; /*简写*/  
border-radius: 50px 0px 0px 0px / 50px 0px 0px 0px; /*完整写法*/
```

前面的4个值代表横轴半径，后面的3个值代表纵轴半径

```
border-radius: 150px 0px 0px 0px / 50px 0px 0px 0px;
```

第三次默写

默写时间：2022年7月20日

1. 列出不能执行过渡的属性 【至少写6个】

- `float`
- `display`
- `visibility`
- `overflow`
- `position`
- `z-index`
- `font-family`
- ``text-align/text-align-last``
- `cursor`

2. CSS3对不同浏览器兼容性添加的前缀是什么？

对于低版本的浏览器，我们需要添加特定的前缀

- 以谷歌为核心的浏览器，我们要添加 `-webkit-`
- 以IE为核心的浏览器则要添加 `-ms-`
- 火狐浏览器则是使用 `-moz-`
- 欧朋浏览器使用 `-o-`

3. CSS3里面过渡的4个属性分别是什么？

- `transition-property` 要执行过渡的属性
- `transition-duration` 要执行过渡的时间，它可以以 `s` 秒为单位，也可以以 `ms` 毫秒为单位】
- `transition-timing-function` 过渡的效果，也叫过渡的时间函数 【默认值是 `ease` 】
 - `linear` 匀速
 - `ease-in` 先慢后快， `ease` 单词是轻松的意思， `in` 是开始的意思
 - `ease-out` 先快后慢， `out` 代表结束的意思
 - `ease-in-out` 前后都很慢， 中间很快
 - 我们还可以在浏览器里面手动的编辑我们效果，如 `cubic-bezier(0.74, -1.06, 0.35, 2.34)`
- `transition-delay` 过渡等待 【默认值0s】

4. 变换的属性是什么？

`transform`

5. 设置变换的起点的属性是什么?

`transform-origin`

6. 位移, 缩放, 旋转, 变形的属性名分别是什么?

- 位移

`translate() / translateX() / translateY() / translateZ() / translate3d()`
`)`

- 缩放 `scaleX() / scaleY() / scaleZ() / scale3d() / scale()`

- 旋转 `rotateX() / rotateY() / rotateZ() / rotate() / rotate3d()`

- 变形 `skewX() / skewY() / skew()`

7. 如何设置元素变换以后背对用户时不可见?

`backface-visibility: hidden;`

8. 形成3D变换的条件是什么?

- 设置视距 `perspective: 距离大小`
- 开启3D空间 `transform-style: preserve-3d`

9. 多个变换结合时语法格式是什么样子的?

`transform: 变换1 变换2 ... ;`

10. 变换的注意事项有哪2个点?

- 所有的行内元素都不支持 `transform` 变换的, 除非转换成块级或行内块级元素
- 不要将 `background-color` 与 `preserve-3d` 一起使用

11. 线性渐变的语法格式?

`background-image: linear-gradient(to 方向, 颜色1 [开始位置] [结束位置], 颜色2 [开始位置] [结束位置] ...);`

12. 径向渐变的语法格式?

`background-image: radial-gradient([形状 at 横坐标 纵坐标], 颜色1 [开始位置] [结束位置] ...);`

13. 径向渐变的形状, 圆与椭圆?

- 圆 `circle`

- 椭圆 `ellipse`

14. 圆锥渐变的语法格式?

```
background:conic-gradient(颜色1 [开始位置] [结束位置], ...);
```

15. 定义动画的基本语法格式?

```
@keyframes 动画名称 {  
    from{}  
    to{}  
}
```

16. 在目前所学习的css里面, 有哪些以@开头的命令, 作用分别是什么?

- `@charset utf-8;` 设置字符串
- `@font-face` 自定义字体
- `@import url("a.css")` 导入外部的CSS文件
- `@keyframes` 定义动画
- `@media` 媒体查询

17. 使用动画的8个属性列出来?

- `animation-name` 动画的名称 【必填】
- `animation-duration` 动画执行一次的时间 【必填】
- `animation-iteration-count` 动画重复的次数 【默认值是1】, 如果希望动画一直重复执行, 则可以通过设置 `infinite` 无穷大来实现
- `animation-timing-function` 动画执行的时间函数 【默认值是 `ease`】, 如果希望动画匀速执行可以设置 `linear`, 这里面的属性值和 `transition-timing-function` 保持一致
- `animation-delay` 动画的等待时间 【默认值为0】
- `animation-direction` 动画执行的方向, 【默认值 `normal`】
 - `normal` 正常的
 - `reverse` 逆向的
 - `alternate` 正向与逆向交替运行
 - `alternate-reverse` 逆向与正向交替运行
- `animation-play-state` 动画的播放状态
 - `running` 运行状态 【默认值】
 - `paused` 暂停状态
- `animation-fill-mode` 动画在结束以后停留在什么状态

- **backwards** 回到开始状态
- **forwards** 停留在结束状态

18. 动画在停止的时候的状态有2个，分别是回到开始状态的和结束状态的？

animation-fill-mode 动画在结束以后停留在什么状态

- **backwards** 回到开始状态
- **forwards** 停留在结束状态

19. 动画在执行的时候的方向的属性值4个分别是什么？

- **normal** 正常的
- **reverse** 逆向的
- **alternate** 正向与逆向交替运行
- **alternate-reverse** 逆向与正向交替运行

20. 动画执行时无穷大的值是什么？

animation-iteration-count: infinite;

21. CSS里面如何定义全局变量，如何使用变量？

```
/*这里的伪类:root相当于html标签*/
:root{
    --a:100px;      /*定义变量*/
}
.div1{
    width:var(--a);      /*使用变量*/
}
```

22. CSS里面如何进行四则算术运算，有什么注意事项？

```
.div{
    width: calc(100px + 100px);      /*符号的左右必须要加上空格*/
}
```

23. 能够形成BFC的属性和属性值有哪些？

- 浮动元素，float 除 none 以外的值；
- 绝对定位元素，position (absolute, fixed)；
- display 为以下其中之一的值 inline-block, table-cell, table-caption, flex 弹性盒子；
- overflow 除了 visible 以外的值 (hidden, auto, scroll)
- 元素加上display:flow-root (没有副作用，但是兼容性差。display:flow-root属性是css新增的属性，专门用来触发BFC，不干别的。)

24. 5种定位分别是什么？

- 相对定位 `relative`
- 绝对定位 `absolute`
- 固定定位 `fixed`
- 静态定位 `static`
- 粘性定位 `sticky`

25. 在CSS里面，滤镜的属性和属性值？

```
filter:blur(2px)
filter:alpha(opacity=50);
filter:hue-rotate(30deg)
filter:grayscale(0.5);
```

第四次默写

2022年8月10日

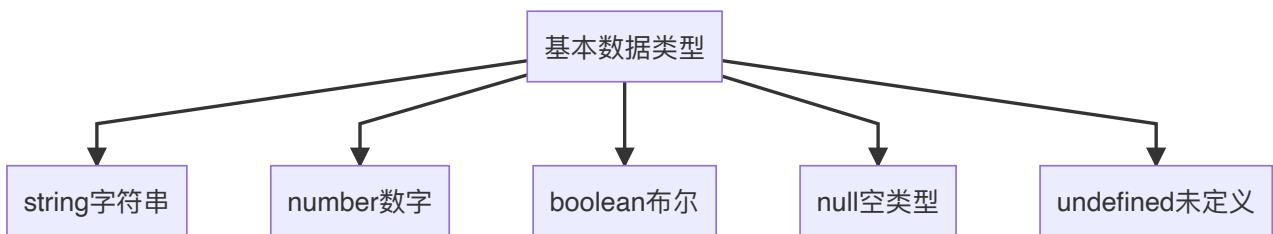
1. 二进制转十进制

```
var a = 10110 ;
parseInt(a,2);      //parseInt得到的结果一定是10进制
```

2. 下面代码结果是什么？

```
Number(null);      //0
Number(undefined); //NaN
```

3. JS基本数据类型有哪些？



4. 下面代码运行结果是什么？

```
console.log("0" == false);      //true;
```

5. 下面代码运行结果是什么?

```
console.log(2%0); //NaN
```

6. 下面代码运行结果是什么?

```
var x = ['abcde', 123456];
var y = typeof typeof x[1]; // "string"
```

7. 下面代码结果是什么?

```
var arr = ["a", "b", "c", "d", "e"] ;
arr.slice(2, 4);
console.log(arr); // ["a", "b", "c", "d", "e"]
//slice()方法提取数组的元素，原数组不会改变
```

8. 相等操作符, 关系运算符, 布尔运算符, 赋值运算符, 算术运算符 优先级从高到低排列

- . () []
- 一元操作符 ++, --
- 算术运算符 [先乘除, 后加减]
- 关系操作符 >, <, >=, <=, ==, !=
- 逻辑操作符 [先非, 再与, 最后或]
- 三目运算符
- 赋值运算符

9. 字符串转数字有哪些方法

```
var a = "123";
parseInt(a);
parseFloat(a);
var b = +a; // 一元操作符转换
Number(a); // 默认的隐式转换
```

10. 使用条件表达式找出最大的变量

```
var a = 1, b=2, c=3;

//第一种
var max = (a > b ? a : b) > c ? (a > b ? a : b) : c;

//第二种
// var max = a > b ? (成立以后a,c比较) : (不成立b,c比较);
var max = a > b ? (a>c?a:c) : (b>c?b:c);
```

11. JS里面有哪些值会被认为是false

```
//javascript一定要注意6个明确的false条件
//空字符串 "", false, 0, NaN, null, undefined
```

12. 退出循环中断循环的关键字是什么?

答: 退出循环的关键字是 **break**;

13. 函数的返回值使用什么关键字?

答: 函数里面通过 **return** 返回

14. 定义数组的方法有哪些?

```
var arr = new Array();
var arr = new Array(2);           //定义长度
var arr = new Array(1,3,"a");    //静态初始化
```

15. arr.length是多少?

```
var arr = [2,1,,3,7,];
//在定义数组的时候 , 最后的一个逗号会被忽略, 但是中间的逗号是正常的, 所以arr数组的
长充是5
```

16. 数组的栈方法

- **push()** 入栈
- **pop()** 出栈

17. 数组的队列方法

- **unshift()** 插队
- **shift()** 出队

18. 使用什么方法可以得到一个相同的互不影响的数组

这个题目应该是要使用深拷贝的，但是如果数组里同存放的是基本数据类型，则可以通过下面的方法

```
var arr = [1,2,4,"a","b"];
var arr1 = arr.slice();      //第一种方式
var arr2 = arr.concat();    //第二种方式
```

19. 数组里面 `splice` 方法怎么使用，参数各代表什么？

```
arr.splice(开始索引, 删除元素个数, 要放入的新元素....);
```

20. `forEach`与`map`迭代的区别在哪里？

答：`forEach`与`map`都是执行的静态遍历，但是`map`可以接回调函数的返回值，将每次回调函数的返回值变成一个新的数组

21. 如何检测一个变量是否是数组

```
var arr = new Array();
arr instanceof Array;      //第一种方式
Array.isArray(arr);        //第二种方法
```

22. 函数中实参数组叫什么？

答：函数的实参数组是`arguments`

23. 写出下面代码的运行结果

```
var a=true,b=false,c=null,d=undefined;
"" || b&&c;           //false
"" &&b&&c;           //"
"" &&b || c;           //null
! "" || b&&c;         //true
!d || b&&c;           //true
```

24. 写出下面代码的运行结果

```
var arr = new Array("3.1");           //正常定义数组，里面有一个元素"3.1"
var arr2 = new Array(-1);             //报错，长度不能为-1
var arr3 = new Array(Number("1"));    //正常定义数组，长度为1
```

25. 请写出"A", "a", "0"的ascii码

0011		01/0		0101		0110		0111	
3		4		5		6		7	
十进制	字符								
48	0	64	@	80	P	96	`	112	p
49	1	65	A	81	Q	97	a	113	q
50	2	66	B	82	R	98	b	114	r
51	3	67	C	83	S	99	c	115	s
52	4	68	D	84	T	100	d	116	t
53	5	69	E	85	U	101	e	117	u
54	6	70	F	86	V	102	f	118	v
55	7	71	G	87	W	103	g	119	w
56	8	72	H	88	X	104	h	120	x
57	9	73	I	89	Y	105	i	121	y
58	:	74	J	90	Z	106	j	122	z
59	;	75	K	91	[107	k	123	{
60	<	76	L	92	\	108	l	124	
61	=	77	M	93]	109	m	125	}
62	>	78	N	94	^	110	n	126	~
63	?	79	O	95	_	111	o	127	△

第五次默写

1. 写出下面的结果

```
console.log([] + []);           // ""
console.log({} + []);           // "[object Object]"
console.log([] == ![]);         //true      注意推断过程
```

2. new关键字干了什么事？

要创建 Person 的新实例，必须使用 new 操作符。以这种方式调用构造函数实际上会经历以下 4 个步骤：

- (1) 创建一个新对象；
- (2) 将构造函数的作用域赋给新对象（因此 this 就指向了这个新对象）；
- (3) 执行构造函数中的代码（为这个新对象添加属性）；
- (4) 返回新对象。

3. 如果获取对象所有的属性名

```
Object.getOwnPropertyNames(obj);    //获取obj对象的所有属性名
```

4. 数据属性有哪4个特征（特性）

1. 数据属性

数据属性包含一个数据值的位置。在这个位置可以读取和写入值。数据属性有 4 个描述其行为的特性。

- `[[Configurable]]`: 表示能否通过 `delete` 删除属性从而重新定义属性，能否修改属性的特性，或者能否把属性修改为访问器属性。像前面例子中那样直接在对象上定义的属性，它们的这个特性默认值为 `true`。
- `[[Enumerable]]`: 表示能否通过 `for-in` 循环返回属性。像前面例子中那样直接在对象上定义的属性，它们的这个特性默认值为 `true`。
- `[[Writable]]`: 表示能否修改属性的值。像前面例子中那样直接在对象上定义的属性，它们的这个特性默认值为 `true`。
- `[[Value]]`: 包含这个属性的数据值。读取属性值的时候，从这个位置读；写入属性值的时候，把新值保存在这个位置。这个特性的默认值为 `undefined`。

5. 构造函数与普通函数的区别

- 一个函数如果以 `new` 去调用那么它就是构造函数，如果只是通过 `函数名+()` 这种形式调用它就是普通函数
- 普通函数的返回值是通过 `return` 来完成的，而构造函数的返回值是自动返回的
【注意返回对象与返回普通类型的区别，这一点其实很少遇到，但还是要注意】
- 构造函数里面的 `this` 指向了当前构造函数所创建的对象，而普通函数里面的 `this` 指向了浏览器的全局对象 `window`
- 构造函数里面的 `this` 指向了当前构造函数所创建的对象，而普通函数里面的 `this` 指向了浏览器的全局对象 `window` 【这个知识点在后面的DOM里面会讲到】

6. 下面代码的运行结果？

```
function a(){}
a instanceof Function;           //true
//这个题目其实充分的说明了，函数也是对象，因为 var fun = new Function() 这也可以创建对象，
//同时要注意 new出来的就一定是对象
```

7. 定义函数的方法有哪些？

```
var a = new Function();          //第一种方式
function b(){...}                //第二种方式
var c = function (){}           //第三种方式
```

8. 解释一下什么是回调函数？

答：回调函数就是把函数当成参数传递到另一个函数里面。回调函数也叫回调方法，它可以解决当函数的返回值无法回，或有些逻辑需要分段执行的时候

9. 解释一下什么是深拷贝，什么是浅拷贝？

答：深拷贝就是深入到内存的堆里面去拷贝，这个拷贝出来的2个对象就完全互不影响；浅拷贝只是拷贝了栈里面的值

```
var obj1 = {
    userName: "张珊",
    sex: "男",
    computer: {
        type: "国产",
        price: 3999,
        name: "RedmiBook16"
    },
    boyFriends: [ "男朋友1号", "男朋友2号", "备胎1号", "2号鱼" ]
}

//现在想对上面的对象实现完整的拷贝，怎么办呢？

//我给你一个对象，你还我一个互不影响 的相同对象
function deepCopy(oldObj) {
    if (typeof oldObj != "object" || oldObj == null) {
        //说明不是对象，或是null，直接返回
        return oldObj;
    }
    //第一步：创建对象
    var newObj = Array.isArray(oldObj) ? [] : {};
    var arr = Object.getOwnPropertyNames(oldObj);
    arr.forEach(function (item) {
        newObj[item] = deepCopy(oldObj[item]);
    });
    return newObj;
}

var obj2 = deepCopy(obj1);
```

10. 列出数组所有的迭代方法

ECMAScript 5 为数组定义了 5 个迭代方法。每个方法都接收两个参数：要在每一项上运行的函数和（可选的）运行该函数的作用域对象——影响 `this` 的值。传入这些方法中的函数会接收三个参数：数组项的值、该项在数组中的位置和数组对象本身。根据使用的方法不同，这个函数执行后的返回值可能会影响方法的返回值。以下是这 5 个迭代方法的作用。

- ❑ `every()`：对数组中的每一项运行给定函数，如果该函数对每一项都返回 `true`，则返回 `true`。
 - ❑ `filter()`：对数组中的每一项运行给定函数，返回该函数会返回 `true` 的项组成的数组。
 - ❑ `forEach()`：对数组中的每一项运行给定函数。这个方法没有返回值。
 - ❑ `map()`：对数组中的每一项运行给定函数，返回每次函数调用的结果组成的数组。
 - ❑ `some()`：对数组中的每一项运行给定函数，如果该函数对任一项返回 `true`，则返回 `true`。
- 以上方法都不会修改数组中的包含的值。

11. 代码运行的结果是什么？

```
var arr = ["a", "b", "c"];
arr.forEach(function(item, index){
    if(index==1){
        arr.push("d");
    }
    console.log(item);
});

//打印a,b,c 原因：forEach执行的是静态遍历
```

12. //写代码，在第二个“b”元素的后面插入一个“标哥”

```
var arr = ["a", "b", "c", "d", "e", "b", "g", "b", "c"];
//代码如下
//第一步：先找出第1个“b”的位置
var index = arr.indexOf("b");
//第二步：再找出第2个b的位置
var index2 = arr.indexOf("b", index);
//第三步：插入
arr.splice(index2+1, "b");
```

13. 在数组里面有哪些方法在执行以后不会影响原数组？

- `concat()`
- `slice()`
- 迭代方法 `forEach/map/filter/every/some`
- 归并方法 `reduce/reduceRight`
- `indexOf/lastIndexOf/toString()/join()`

14. 对象调用属性的方式有哪些？

```
对象.属性  
对象["属性"];
```

15. 请说明一下obj对象里面有几个属性，属性值分别是什么？

```
var a = "bgg";
var b = a;
var obj = {
  [a]:"hello",           //中括号代表的是变量，所以[a]相当于"bgg"
  [b]:"world"           // [b]也相当于"bbb"
}
//两个属性不同，所以只有1个属性值，为"world"；
```

16. 如果删除对象的某一个属性?

```
delete 对象.属性;
```

17. 通过什么方法可以获取对象属性的描述信息

```
Object.getOwnPropertyDescriptor(对象);
```

18. 如果判断对象是否具备某一个属性

```
/*第一种方法*/
属性 in 对象          //判断自己或原型父级上面有没有这个属性

/*第二种方式*/
对象.hasOwnProperty(属性) //只判断自己是否包含这个属性
```

19. 以下代码输出什么?

```
var name = "sex"
var company = {
  name: "qdywxs",
  age: 3,
  sex: "男"
}
console.log(company[name]);      //这里的name没有加引号，是一个变量代
表"sex"，结果就是"男"
console.log(company.name);     //这里的name是一个普通属性，结果就
是"qdywxs";
```

20. 对象在执行加法与减法的时候分别是怎么操作的?

答：对象在执行加法，则字符串优先，先调用 `toString()`。

对象在执行减法操作的时候，数字优先，先调用 `valueOf()`，调用失败则会再次调用 `toString()`

21. call/apply有什么区别，作用是什么？

答： `call/apply` 可以改变函数内部 `this` 的指向， `call` 有呼叫谁来调用方法的意思， `apply` 有申请谁过来调用方法的意思

22. 以下代码结果是什么？

```
var arr = [];
arr.push==Array.prototype.push;
//结果为true,始终要注意一个点，一个对象的__proto__应该等于它的构造函数的
prototype
```

23. 如何判断一个变量是整数？

```
//第一种方法
Number.isInteger(a);
//第二种方法
parseInt(a)===a;
//第三种方法
a%1==0;
```

24. 如何判断一个变量它是NaN？

```
Number.isNaN(a);
```

25. 怎么定义一个变量a， 才会有下面的结果

```
a==1&&a==2&&a==3 //结果为true
```

```
//答案如下：这里要注意，在执行判断操作的时候，要执行数字优先的原则，所以会优先调用
valueOf的方法
//这里我们就应该想到要使用对象了
var a = {
    i:0;
    valueOf:function(){
        i++;
        return i;
    }
}
```

第六次默写

默写时间：2022年8月25日

1. DOM当中获取子级元素的属性是什么？

答：`children` 属性可以获取当前元素的子级元素，它是一个 `HTMLCollection` 类型的伪数组，它是一个动态数组

2. `innerHTML`与`innerText`的区别是什么？

答：`innerHTML`在获取的时候它可以将标签与内容同时获取到，而`innerText`在获取的时候是不可以获取到标签的，只能获取到文本内容

同时在设置值的时候，`innerHTML` 在设置标签的时候会生成页面的元素，而`innerText`则只会将设置的HTML标签以文本的形式显示出来

3. `innerHTML`与`outerHTML`的获取是什么？

答：`innerHTML`是获取当前内部的网页标签及内容，`outerHTML`还可以获取自身的标签及内部的标签内容

4. `classList`里面，新增样式，删除样式，判断样式是否存在及切换式的方法是什么？

```
classList.add("样式");
classList.remove("样式");
classList.contains("样式");
classList.toggle("样式");
```

5. `insertAdjacentElement()`插入元素的时候，分别是在哪4个位置插入的？

```
//开始之前 beforeBegin
//开始之后 afterBegin
//结束之前 beforeEnd
//结束之后 afterEnd
```

6. 如何克隆一个相同的DOM结点（包含子节点）；

答：通过 `cloneNode(true)` 可以得到一个相同的结点

7. DOM操作里面如何获取一个HTML标签当中的属性？如果设置一个HTML标签当中的属性？

答：可以通过 `getAttribute("属性名")` 来获取属性，可以通过 `setAttribute("属性名", "属性值")` 来设置属性

8. 什么是单属性？你见过哪些单属性？单属性在DOM当中的属性值是什么？

答：单属性就是那些只有属性名而没有属性值的（或者可以理解为属性名与属性值相同的），常见的单属性有 `checked/readonly/disabled/required` 等，单属性的值在 DOM 中表现的值为 `true/false`

9. 鼠标单击的事件是什么？鼠标双击的事件是什么？鼠标右键菜单的事件是什么？

答：单击事件 `onclick`，双击事件 `ondblclick`，鼠标右键菜单事件
`oncontextmenu`；

10. 元素获取焦点的事件是什么？元素失去焦点的事件是什么？

答：元素获取焦点为 `onfocus`，元素失去焦点为 `onblur`

11. 键盘按下的事件是什么？键盘松开的事件是什么？

答：键盘按下的事件是 `onkeydown`，键盘松开的事件是 `onkeyup`

12. `keypress`事件与`keydown`事件有什么区别？

答：`keypress` 只针对字符按键才可以触发，功能按键如 f1, ctrl 等不可以触发，而 `keydown` 是针对所有的按键都可以触发

13. 表单元素里面值`value`改变以后的事件是什么？

答：表单元素的值改变了以后，触发的是 `onchange` 事件

14. 表单元素里面输入框输入的时候触发的事件是什么？

答：表单元素里面输入框在输入的时候会不停的触发 `oninput` 事件

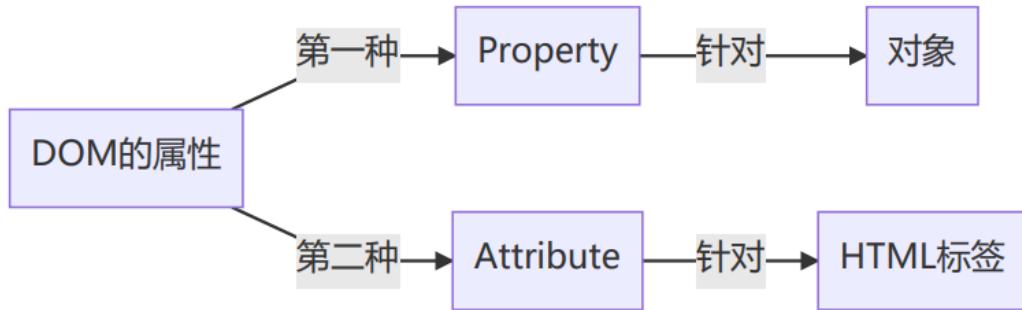
15. `HTMLCollection`与`NodeList`的区别是什么？

答：`HTMLCollection` 它是一个动态集合，在进行删除操作的时候要注意沙漏效应，而 `NodeList` 是一个静态集合

16. 如果在DOM操作当中删除一个元素？

答：DOM 操作里面，删除元素的方法有 2 种，第一种是通过 `removeChild()` 来实现删除一个子元素，另一个是 `remove()` 来删除自身

17. 说明一下DOM操作里面Property与Attribute的区别？



18. 数学对象Math里面如何生成20~100（包含20与100）的随机数？**重点**

这个问题已经出现了很多次了，思路应该是这样的，首先生成1个0~80之间的随机数，再加上20，这样随机数的范围就变在20~100了

```
var num = ~(Math.random() * 81) + 20;
```

19. 字符串对象String中split()方法有什么作用

```
var str = "get-element-by-id";
str.split("-")
//结果是什么?
```

答：split方法是字符串包装对象String里面的方法，它可以将字符串以指定的字符隔开，上面代码的意思就是将字符串用 - 来进行隔开，所以最后的结果就是 ["get", "element", "by", "id"]

20. 数学对象Math里面，`Math.round(-7.5)` 的结果是什么？

答：这里要注意负数针对四舍五入的情况，我们要执行“五舍六入”的情况进行

21. 日期对象Date里面，如何获取当前日期的时间戳？

答：这是一个很常见的面试题，也是一个很常见的应用题

```
//第一种方法
var a = Date.now();
//第二种方法
var d = new Date();
d.getTime(); //得到时间戳
//第三种方法
var d = new Date();
d.valueOf();
```

22. 字符串对象里面如何将"123"变成"00123"，将"456"变成"00456"？使用什么方法？

答：在字符串对象里面，我们有一个方法叫 `padStart(maxLength, fillString)` 它可以将字符串补齐到指定的长度

```
"123".padStart(5,0); // "00123";
"456".padStart(5,0); // "00456";
```

23. 字符串对象里面，截取字符串的方法有哪些，它们怎么使用的？如”hello world”希望截取以后得到”ell”

答：在字符串截取的操作当中，有3种方法

- `slice(start,end)` 这一种方法是模拟了数组的操作方法，它可以实现负数做为参数，如果是负数做为参数可以倒数
- `substring(start,end)` 这一种方法与上面的方法是一样的，从开始索引截取到结束索引，不包含结束索引，但是要注意，这里不推荐使用负数做为参数
- `substr(start,length)` 这一种方法比较特殊，它的第二个参数是要截取的长度

```
var a = "hello world";
a.slice(2,5);
a.substring(2,5);
a.substr(2,3);
```

24. 内置对象里面，`window.isNaN()` 与 `Number.isNaN()` 的区别是什么？

答：先看下面的方法

```
Number.isNaN(value:any);
window.isNaN(value:number);
```

`window.isNaN()` 这个判断方法它只接收 `number` 类型，如果我们传入的参数不是数值，它会默认调用 `Number` 方法做一次隐式类型转换，`Number("a")` 得到的结果 `Nan`，所以最后 `window.isNaN("a")` 检测的结果就是 `true`

第七次默写

默写时间：2022年9月7日

1. `location`对象中的`origin`是由什么组成的？

答：`origin`由`protocol`, `hostname`及端口号组成。【这里要注意跨域的概念】

2. `location`中的`search`是什么？

答：`search`指的是地址栏 `?` 及后面的东西

3. 在这一段字符串里面，如何获取userName及age的值

```
var str = "http://www.softteam.xin:8090/1.html?  
userName=biaoogege&age=18";  
var u = new URL(str);  
var userName = u.searchParams.get("userName");  
var age = u.searchParams.get("age");
```

4. 0级事件里面如何阻止事件的默认行为？2级事件如何阻止事件的默认行为？

答：0级事件通过 `return fasle` 来阻止事件的默认行为；2级事件要通过 `event.preventDefault()` 来完成阻止事件的默认行为

5. 取消事件传播（取消事件冒泡）怎么办？

```
event.cancelBubble=true;  
event.stopPropagation();
```

6. 事件的绑定者与事件的触发者分别是什么

答：事件的绑定者是 `event.currentTarget`，事件的触发者 `event.target`

7. 2级事件里面通过什么方式取消事件的监听？

答：通过 `removeEventListener()` 取移除事件的监听

8. 2级事件的事件链如何断开？

答：`event.stopImmediatePropagation`

9. 2级事件里面如果改变事件的传播方向？

答：在 `addEventListener()` 进行事件监听的时候，最后一个参数是false代表事件冒泡，最后一个参数是true代表事件捕获

10. offsetWidth/clientWidth/scrollWidth分别指的是什么？

答：offsetWidth相当于宽度+内边距+边框

clientWidth相当于宽度+内边距

scrollWidth指的是元素在没有滚动条的情况下宽度

11. offsetX/clientX/pageX/screenX分别指的是什么？

答：offsetX指鼠标距离事件触发者的横坐标

clientX指鼠标距离浏览器的横坐标

pageX指鼠标距离页面的横坐标

screenX指鼠标距离屏幕的横坐标

12. 鼠标按下的事件是什么？如何判断鼠标按下的是左键还是右键？

答：鼠标按下的事件是 `onmousedown`，在鼠标事件对象里面有一个button属性用来表述鼠标按下的键是哪一个，0代表左键，1代表中键，2代表右键

13. localStorage与sessionStorage的区别是什么

1. localStorage可以跨页面共享值
 2. localStorage在关闭浏览器以后还会保存
 3. localStorage如果不手动清除会一直存在
 4. **localStorage不能跨域访问**
1. sessionStorage不能跨页面共享数据，除非是父子页面
 2. sessionStorage关闭浏览器以后数据会自动清除
 3. **sessionStorage不能跨域访问**

14. localStorage与cookie的区别是什么？

cookie的特点及特点

1. cookie 是可以跨页面的
2. cookie 的 path 会隔离，子级的path可以访问父级的path，父级的访问不了子级的，同级别的 path 也是可以相互访问的
3. cookie 关闭浏览器以后不会自动消失，它的消失是根据过期时间来决定的，但是这个时间是GMT时间
4. cookie 是可以存储在 document 上面的，它会随着请求到达服务器，随着响应返回浏览器
5. cookie 是在大小限制，每个浏览器都不一样

15. 如何json字符串转为JS对象，如何将JS对象序列化为json字符串

```
JSON.parse();  
JSON.stringify();
```

16. 正则表达式中学过的3个修饰符是什么？各代表什么意思？

g代表全局
i代表忽略大小写
m代表换行

17. 正则表达式里面的前瞻，负前瞻，后顾，负后顾怎么写？

前瞻: A(?=B);
负前瞻: A(?!B);
后顾: (?<B)A
负后顾: (?<!B)

18. 正则表达式里面, 数字用什么表示, “或”用什么表示

数字:\d表示或[0-9]表示
或: 可以使用|表示, 也可以使用原子表的[]表示

19. 写一个正则表达式, 用来匹配整数或小数, 要注意: 这里有可能会有负数?

```
var reg = /^-?\d+(\.\d+)?$/;
```

20. 正则表达式里面, 1次或多次用什么? 0次或1次用什么? 0次或多次用什么?

1次或多次使用+, 也可以使用{1,}
0次或1次使用?, 也可以使用{0,1}
0次或多次使用*, 也可以使用{0,}

21. Ajax里面用来监听状态变化的事件是什么? 哪一个属性是用来表示状态的值的? 状态为多少表示请求已返回?

答: 用来监听状态变化的事件是 `onreadystatechange` 事件, `readyState` 用来表示状态的变化, 当状态为4时代表请求已返回

22. 音频标签audio里面, 哪个事件是播放时间改变以后会触发的事件, 哪个事件是开始播放的时候会触发的? 哪个事件是缓冲的时候触发的? 哪个事件是暂停的时候触发的?

时间改变的事件: `ontimeupdate`
播放事件: `onplay`或`onplaying`
缓冲: `onwaiting`
暂停: `onpause`

23. 视频标签里面哪个属性用于设置video的“海报”? 如何开启中全屏? 如何开启画中画播放?

`poster`属性用来设置海报
`video.requestFullScreen()`开启全屏
`video.requestPictureInPicture()`开启画中画

24. 如果将下面的file对象转化为base64格式

```
var file = obj.files[0];
var reader = new FileReader();
reader.readAsDataURL(file);
reader.onload = function(){
    console.log(reader.result); //打印base64
}
```

25. 如何获取div1元素上面的“张三”,“10”这2个属性值?

```
<div id="div1" data-user-name="张三" data-userAge = "10"></div>
<script>
    var div1 = document.querySelector("#div1");
    div1.dataset.userName;
    div1.dataset.userage;
    //这里要注意驼峰命名的情况下要转义
</script>
```