07_Javascript_DOM操作

历史

什么是DOM？简单说就是一套对文档的内容进行抽象和概念化的方法。

最早的DOM（0级）

只是提供了操作表单和图片的手段

```
document.images[2] //文档中的第三个图片
document.forms['details'] //文档中名为details的表单
```

DHTML（0级）

实际上就是HTML，CSS，和JavaScript技术组合的术语。

此时JavaScript加入了更多操作文档其他元素的方法，但是当时分两个派系：网景派系和微软派系。他们使用JavaScript操作文档的方法各不相同，所以程序员需要针对两个浏览器编写不同的代码。并且需要编写探嗅器去分辨在哪个浏览器执行。

DOM标准（1级）

W3C抓住时机利用双方优点推出了标准化DOM。各大浏览器开始遵从W3C标准制定了“第一级的DOM”。

如下语法

```
var xpos=document.getElementById('myelement').style.left;
```

W3C推出的标准化DOM可以让任何一种程序设计语言对使用任何一种标记编程语言编写出来的任何一份文档进行操控。

所以DOM是W3C推出的一套标准API，他与你使用什么语言无关，可以用PHP或Python同样去操作DOM。

W3C对DOM的定义是：“一个与系统平台和编程语言无关的接口，程序和脚本可以通过这个接口动态地访问和修改文档的内容，结构和样式。”

目前所有主流浏览器已经95%以上采用DOM标准，并且持续更高的支持率。所以这是我们前端工程师的福音，我们终于可以实现“编写一次，随处运行”的梦想了。

简介

DOM中的D就是document。当创建一个网页并且把它加载到web浏览器中时，DOM就在幕后悄然而生，它把你编写的网页文档转换成一个文档对象。

DOM中的O即是对象的含义。

DOM中的M是模型的意思，模型你可以把它看成是一个有结构的东西，我们这里表示成一棵家谱树，在这棵树中我们用parent,child,sibling来表示关系。

eg

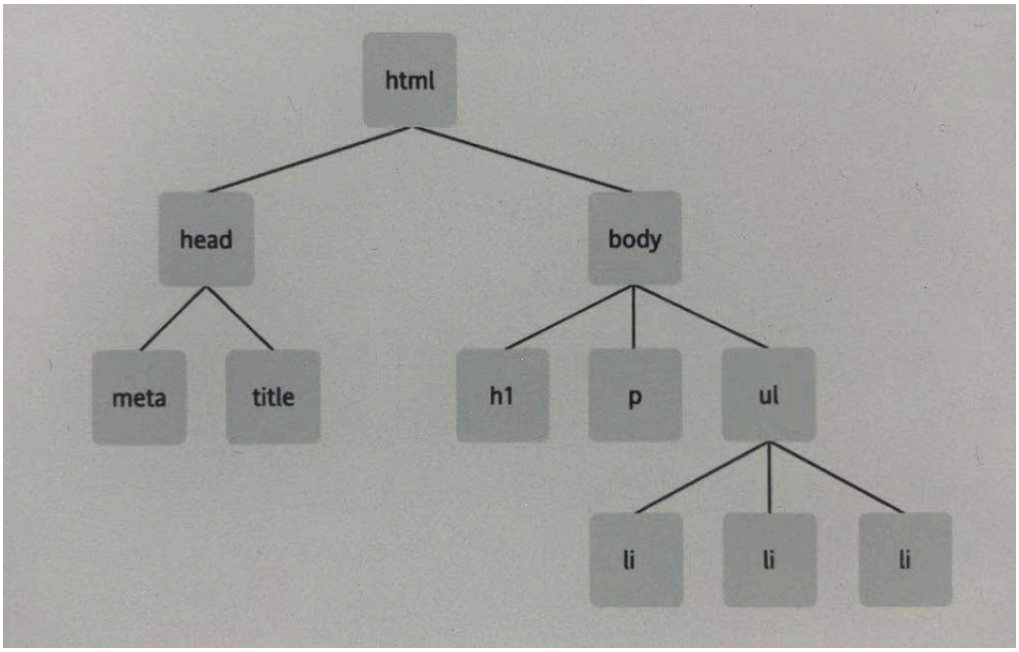
```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Shopping list</title>
</head>

<body>
    <h1>What to buy</h1>
    <p title="a gentle reminder">Donot forget to buy this stuff.</p>
    <ul id="purchases">
        <li>A tin of beans</li>
        <li class="sale">Cheese</li>
        <li class="sale important">Milk</li>
    </ul>
</body>

</html>
```

该文档的模型可以表示如下：



我们如果可以用上图的关系来描述其中所有成员的关系，那么我们的文档模型就描述出来了，不过我们用更专业的术语叫他节点树，而不是家谱树。

节点

文档是由节点构成的。

节点类型包括：

- 元素节点
- 文本节点
- 属性节点
- 文档节点（document）
- 注释节点

常见节点类型：

- 元素节点
- 文本节点
- 属性节点

元素节点

一个HTML标签就是一个元素节点，元素节点像文档结构一样可以嵌套，节点的名字就是标签的名字，例如p标签就是p元素节点。

文本节点

标签中的文本信息就是文本节点，通常他会被包含在元素节点内容，但不是所有的元素节点中都有文本节点。

属性节点

属性节点对元素节点做出更加具体的描述，例如很多元素节点都有title这个属性，这个属性就是属性节点，由于属性也是定义在开始标签中的，所以属性节点也包含在元素节点中。并非所有的元素都包含着属性，但所有的属性都被元素包含。

获取元素

有三种方法可以获取元素节点

通过ID

通过标签名字

通过class

getElementById（id）

根据id获取元素节点，返回一个节点对象

getElementsByTagName(tag)

根据标签名字获取一组元素节点，返回一个节点对象集合。

> getElementByTagName("*")他的参数可以是通配符*，表示获取当前文档的所有元素节点。

eg:

```
var shopping=document.getElementById("purchases");
var items=shopping.getElementByTagName("*");
```

以上代码可以获取到id为purchases元素节点内的所有子节点元素。

getElementsByClassName(class)

根据类名获取一组元素节点，返回一个节点对象集合。

还可以找到多个类名的元素

```
var items=shopping.getElementByClassName("important sale");
```

并且类名顺序不重要，并且也可以包含其他类名，只要有这两个类名都能包含到。

querySelector()

根据选择器获取第一个元素
document.querySelector(".类名") //需要前面的点
document.querySelector("#id") //需要前面的#
document.querySelector("标签名")

querySelectorAll()

根据选择器获取所有元素

document.querySelectorAll(".类名")
document.querySelectorAll("标签名")

特殊的可以直接获取的元素

获取body元素
  document.body
获取html元素
  document.documentElement
获取head元素
  document.head
获取title元素
  document.title

操作元素

学会了如何获取元素，我们下面讨论元素获取过后，如何操作

操作元素节点的内容

元素对象.innerHTML
  获取的所有的包括html标签

元素对象.innerText
  只获取文本，不包括html标签

元素对象.outerHTML
  当前元素本身全部获取到

元素对象.textContent
  只获取文本，不包括html标签，在设置内容时, 如果内容有html标签字符串, 只会原样输出. 如果有空格, 也会原样输出, 在获取内容时, 只能获取文本, 也能获取到多余的空格

表单元素对象.value
  获取表单元素的value值

示例代码：

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

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>元素操作</title>
</head>

<body>
    <p>我在<b>p标签</b>里</p>
    <div>我在div内</div>
    <input type="text" value="我是单行文本" />
    <textarea>我是多行文本</textarea>
</body>

</html>
<script>
    var p = document.querySelector('p');
    var div = document.querySelector('div');
    var input = document.querySelector('input');
    var area = document.querySelector('textarea');

    console.log(p.innerHTML);//我在<b>p标签</b>里
    console.log(p.innerText);//我在p标签里
    console.log(p.outerHTML);//<p>我在<b>p标签</b>里</p>
    console.log(div.innerHTML);//我在div内
    console.log(input.value);//我是单行文本
    console.log(area.innerHTML);//我是多行文本
    console.log(area.value);//我是多行文本
</script>
```

操作元素节点的样式

元素对象.style.css属性名 = "属性值"
如果css属性名带有短横线 "-" ,则须做以下处理
如：background-color要写成backgroundColor

注意:

0. 通过js方式设置的样式设置在了行间

1. 通过js方式给元素设置样式, 只能通过style

2. 通过style方式设置的样式, 可以被style方式获取到, 而通过css语法设置的样式, 无法通过style获取到!!!!!

如何获取非行间的css样式?

```
getComputedStyle(元素)

返回值：computedStyle对象，里面存储了所有的css样式

获取到computedStyle对象的某个值
对象.样式名
对象["样式名"]
```

```
div1.style.width = "200px";
div1.style.height = "200px";
div1.style.backgroundColor = "red";
console.log(div1.style.width, div1.style.height, div1.style.backgroundColor,
div1.style.border);

var cStyleObj = getComputedStyle(div1);
console.log(cStyleObj);
```

```
div1                                                    02_DOM基础01.html:194
200px 200px red                                         02_DOM基础01.html:263
                                                        02_DOM基础01.html:266
  CSSStyleDeclaration {0: "align-content", 1: "align-items", 2: "align-self", 3: "alignment-baseline", 4: "ani
  mation-delay", 5: "animation-direction", 6: "animation-duration", 7: "animation-fill-mode", 8: "animation-it
  eration-count", 9: "animation-name", 10: "animation-play-state", 11: "animation-timing-function", 12: "appea
  rance", 13: "backdrop-filter", 14: "backface-visibility", 15: "background-attachment", 16: "background-blend
  -mode", 17: "background-clip", 18: "background-color", 19: "background-image", 20: "background-origin", 21:
  "background-position", 22: "background-repeat", 23: "background-size", 24: "baseline-shift", 25: "block-siz
  e", 26: "border-block-end-color", 27: "border-block-end-style", 28: "border-block-end-width", 29: "border-bl
  ock-start-color", 30: "border-block-start-style", 31: "border-block-start-width", 32: "border-bottom-color",
  33: "border-bottom-left-radius", 34: "border-bottom-right-radius", 35: "border-bottom-style", 36: "border-bo
  ttom-width", 37: "border-collapse", 38: "border-end-end-radius", 39: "border-end-start-radius", 40: "border-
  image-outset", 41: "border-image-repeat", 42: "border-image-slice", 43: "border-image-source", 44: "border-i
▸ mage-width", 45: "border-inline-end-color", 46: "border-inline-end-style", 47: "border-inline-end-width", 4
  8: "border-inline-start-color", 49: "border-inline-start-style", 50: "border-inline-start-width", 51: "borde
  r-left-color", 52: "border-left-style", 53: "border-left-width", 54: "border-right-color", 55: "border-right
  -style", 56: "border-right-width", 57: "border-start-end-radius", 58: "border-start-start-radius", 59: "bord
  er-top-color", 60: "border-top-left-radius", 61: "border-top-right-radius", 62: "border-top-style", 63: "bor
  der-top-width", 64: "bottom", 65: "box-shadow", 66: "box-sizing", 67: "break-after", 68: "break-before", 69:
  "break-inside", 70: "buffered-rendering", 71: "caption-side", 72: "caret-color", 73: "clear", 74: "clip", 7
  5: "clip-path", 76: "clip-rule", 77: "color", 78: "color-interpolation", 79: "color-interpolation-filters",
  80: "color-rendering", 81: "column-count", 82: "column-gap", 83: "column-rule-color", 84: "column-rule-styl
  e", 85: "column-rule-width", 86: "column-span", 87: "column-width", 88: "content", 89: "cursor", 90: "cx", 9
  1: "cy", 92: "d", 93: "direction", 94: "display", 95: "dominant-baseline", 96: "empty-cells", 97: "fill", 9
  8: "fill-opacity", 99: "fill-rule", …}
```

元素对象.className = "类名"

因为属性名class在JS中为关键字，所在此处写做className

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>操作样式</title>
    <style>
        div {
            width: 300px;
            height: 300px;
            background-color: aqua;
            transition: 5s;
```

```
        }

        .red {
            background-color: red;
        }

        .green {
            background-color: green;
        }

        .size {
            width: 500px;
            height: 500px;
        }
    </style>
    <script>
        window.onload = function() {
            //document.querySelector('div').style.backgroundColor = "red";
            document.querySelector('div').className += " red";
            //console.log(document.getElementsByClassName('size red'));
        }
    </script>
</head>

<body>
    <div class="size">

    </div>
</body>

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

操作元素节点的属性

获取元素对象属性：
  方法1：元素对象.属性名
    主要用来操作内置属性
  方法2：元素对象.getAttribute('属性名')
    主要用来操作自定义属性
设置元素对象属性：
  方法1：元素对象.属性名 = '属性值'
  方法2：元素对象.setAttribute('属性名','属性值')

> 如果设置的属性原先不存在，则创建该属性并赋值

移除元素对象属性:
  元素对象属性.removeAttribute('属性名')

H5中自定义属性

H5中规定自定义属性以 data- 开头
　可以通过getAttribute()、setAttrbute()、removeAttribute()来操作自定义属性
H5新增以下方式操作自定义属性
　元素对象.dataset.属性名
　元素对象.dataset['属性名']
　IE11开始支持

位置与尺寸相关操作

offset系列

offset（偏移量）,可以动态得到该元素的位置、大小等

offset系列属性	作用
element.offsetParent	返回作为该元素带有定位的父级元素 如果父级都没有定位则返回body
element.offsetTop	返回元素相对带有定位父元素上方的偏移
element.offsetLeft	返回元素相对带有定位父元素左边框的偏移
element.offsetWidth	返回自身包括padding 、 边框、内容区的宽度，返回数值不带单位
element.offsetHeight	返回自身包括padding、边框、内容区的高度，返回数值不带单位

offsetWidth与style.width的区别

offset

- offset 可以得到任意样式表中的样式值
- offset 系列获得的数值是没有单位的
- offsetWidth 包含padding+border+width
- offsetWidth 等属性是只读属性，只能获取不能赋值
- > 所以，我们想要获取元素大小位置，用offset更合适

style

- style 只能得到行内样式表中的样式值
- style.width 获得的是带有单位的字符串
- style.width 获得不包含padding和border 的值
- style.width 是可读写属性，可以获取也可以赋值
- > 所以，我们想要给元素更改值，则需要用style改变

client系列

client系列的相关属性用来获取元素可视区的相关信息。
通过client系列的相关属性可以动态的得到该元素的边框大小、元素大小等。

client系列属性	作用
element.clientTop	返回元素上边框的大小
element.clientLeft	返回元素左边框的大小
element.clientWidth	返回自身包括padding 、 内容区的宽度，不含边框，返回数值不带单位
element.clientHeight	返回自身包括padding 、 内容区的高度，不含边框，返回数值不带单位

scroll系列

scroll 系列的相关属性可以动态的得到该元素的大小、滚动距离等

scroll系列属性	作用
element.scrollTop	返回被卷去的上侧距离，返回数值不带单位
element.scrollLeft	返回被卷去的左侧距离，返回数值不带单位
element.scrollWidth	返回自身实际的宽度，不含边框，返回数值不带单位
element.scrollHeight	返回自身实际的高度，不含边框，返回数值不带单位

注意: 获取非行元素的尺寸和位置

- 尺寸
  - 内尺寸
    - clientWidth = width + 左右padding
    - clientHeight = height + 上下padding
  - 外尺寸
    - offsetWidth = width + 左右padding + 左右border
    - offsetHeight = height + 上下padding + 上下border

html+css代码

```
#box {
    width: 500px;
    height: 500px;
    border: 5px solid red;
    background-color: lightblue;
    margin: 30px;
    position: relative;
  }
  #big {
    width: 300px;
    height: 400px;
    background-color: yellow;
    padding: 15px;
    margin: 30px;
    border: 0 solid black;
    border-width: 1px 2px 3px 4px;
    overflow: auto;
  }
  #small {
    width: 200px;
    height: 600px;
    background-color: red;
  }
<div id="box">
  <div id="big">
```

```
    <div id="small"></div>
  </div>
</div>
```

获取元素的尺寸

```
console.log(big.clientWidth, big.clientHeight);
console.log(big.offsetWidth, big.offsetHeight);
```

```
330 430                                   02_DOM基础02.html:62
336 434                                   02_DOM基础02.html:63
```

- 位置
  - 内位置
    - clientLeft: 元素左padding的外边缘距离自身左border的外边缘之间的距离(border-left-width)
    - clientTop:元素上padding的外边缘距离自身上border的外边缘之间的距离(border-top-width)
  - 外位置
    - offsetLeft: 元素的左边框的外边缘距离离它最近的, 设置了非static定位的祖先元素的左边框的内边缘
    - offsetTop: 元素的上边框的外边缘距离离它最近的, 设置了非static定位的祖先元素的上边框的内边缘

```
console.log(big.clientLeft, big.clientTop);
console.log(big.offsetLeft, big.offsetTop);
```

```
4 1                                       02_DOM基础02.html:78
30 30                                     02_DOM基础02.html:79
```

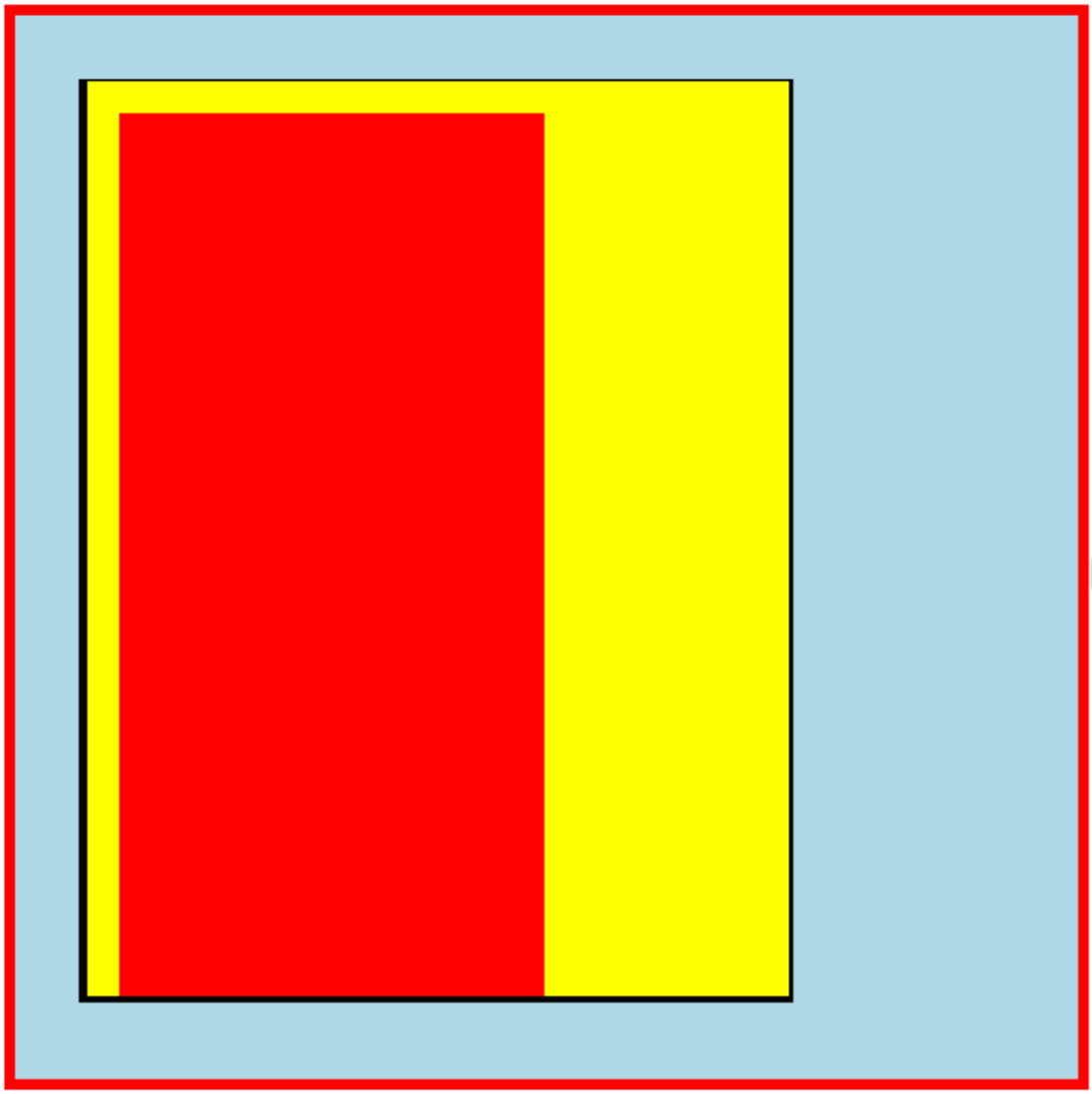
- 滚动尺寸
  - scrollWidth： 可滚动元素里面可视内容的盒子宽+滚动元素的左右padding
  - scrollHeight： 可滚动元素里面可视内容的盒子高+滚动元素的上下padding
- 滚动位置
  - scrollLeft：元素在水平方向滚动到的位置
  - scrollTop：元素在竖直方向滚动到的位置

```
console.log(big.scrollWidth, big.scrollHeight);
console.log(big.scrollLeft, big.scrollTop);
```

```
330 630                                   02_DOM基础02.html:90
0 0                                       02_DOM基础02.html:91
```

滚动事件onscroll， 当滚动条发生移动时触发

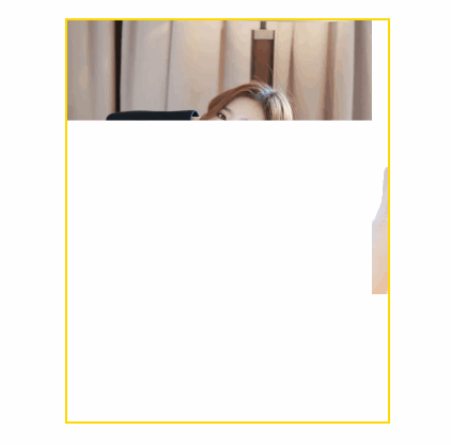
```
big.onscroll = function(){
  if(this.scrollTop > 100) {
    this.style.backgroundColor = "blue";
  }else {
    this.style.backgroundColor = "yellow";
  }
}
```



注意:以上所有的属性， 只有scrollLeft和scrollTop是读写性的， 其余的全是只读的

案例

无限大图滚动



思路: 通过JS的计时器设置非常短的时间间隔, 每次触发时, 修改 承载所有图片的父级元素 的left值, 每次的改变量要小, 其实是通过计时器模拟的动画效果

难点: 如何实现无限轮播效果

核心代码:

```
<div id="box">
<div id="content">
    <img src="img/1.png" alt="" />
    <img src="img/2.jpeg" alt="" />
    <img src="img/3.jpeg" alt="" />
    <img src="img/4.png" alt="" />
    <img src="img/5.jpeg" alt="" />
    <img src="img/6.png" alt="" />
    //在最后添加一张假的第一张图
    <img src="img/1.png" alt="" />
</div>
</div>
// js动画
// 声明变量记录content的left值
var leftV = 0;
var timer = setInterval(moveContent, 20);
function moveContent(){
    leftV-=5;
    content.style.left = leftV + "px";
    // 当偏移到最后一张时, 将其拉回到第一张
    if(leftV <= -1800) {
        leftV = 0;
    }
}
```

无限大图滚动按钮版

思路: 图片切换的逻辑还是通过JS的计时器设置非常短的时间间隔, 每次触发时, 修改 承载所有图片的父级元素 的left值, 每次的改变量要小, 其实是通过计时器模拟的动画效果. 左右按钮在点击时需要对应修改left值, 使之变大或者减小达到分别控制左右偏移的效果

难点: 如何实现左右按钮的点击偏移和无限轮播效果

核心代码:

```
left.onclick = leftClick;
right.onclick = rightClick;

// 声明变量存储当前是第几张图片 0表示第1张
var index = 0;

// 左边按钮点击事件
function leftClick(){
    // 当到第一张图时, 瞬间拉回到最后一张假图
    if(index == 0) {
        index = 6;
    }
    // 做JS动画, 实现换图效果
    // 声明一个变量记录本次换图的偏移量(-350)
    var lx = 0;
    var lTimer = setInterval(function(){
        lx-=5;
        contentDiv.style.left = index * -350 - lx + "px";
        if(lx <= -350) {
            // 本次换图结束, 清除计时器
            clearInterval(lTimer);
            index--;
        }
    }, 20);
};
// 右边按钮点击事件
function rightClick(){
    // 当到最后一张假图时, 瞬间拉回到第1张
```

```
    if(index == 6) {
        index = 0;
    }
    // 做JS动画, 实现换图效果
    // 声明一个变量记录本次换图的偏移量(-350)
    var rx = 0;
    var rTimer = setInterval(function(){
        rx-=5;
        contentDiv.style.left = index * -350 + rx + "px";
        if(rx <= -350) {
            // 本次换图结束, 清除计时器
            clearInterval(rTimer);
            index++;
        }
    }, 20);
};
```