10_Javascript_事件

事件介绍

javascript是以事件驱动为核心的一门语言

- 事件的三要素

  事件源、事件、事件驱动程序

  比如：我用手去按开关，灯亮了。这件事情里，事件源是：开关。事件是：按。事件驱动程序是：灯的开和关

注册事件

给元素添加事件，称为注册事件或绑定事件

- 0级DOM事件(传统方式)
  利用on开始的事件
  onclick = function(){}
  同一元素的同一事件只能注册一次
- 2级DOM事件(监听方式)
  addEventListener('click', function(){}, true)
  第三个参数true为捕获阶段
  第三个参数false或省略为冒泡阶段
  ie9以后支持
  同一元素的同一事件可以注册多次

删除事件

- 0级DOM事件
  元素对象.onclick = null;
- 2级DOM事件
  元素对象.removeEventListener('click',函数名)

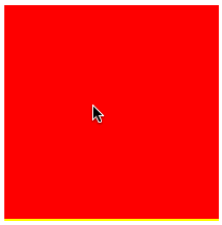
常见事件

鼠标事件

事件	说明
onclick	鼠标点击某个对象
ondblclick	鼠标双击某个对象
onmouseover	鼠标被移到某元素之上
onmouseout	鼠标从某元素移开
onmousemove	鼠标被移动
onmousedown	某个鼠标按键被按下
onmouseup	某个鼠标按键被松开

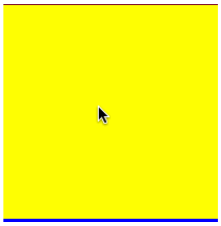
- 鼠标左键单击事件 onclick

```
div1.onclick = function(e){
  this.style.backgroundColor = randomColor();
}
```



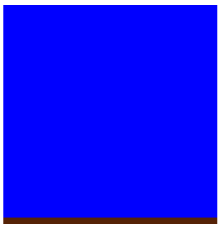
- 鼠标左键双击事件 ondblclick

```
div2.ondblclick = function(){
  this.style.backgroundColor = randomColor();
}
```



- 鼠标移入移出事件
  - 移入 onmouseover/onmouseenter
  - 移出 onmouseout/onmouseleave

```
div3.onmouseover = function(){
  this.style.backgroundColor = randomColor();
};
div3.onmouseout = function(){
  this.style.backgroundColor = randomColor();
};
```

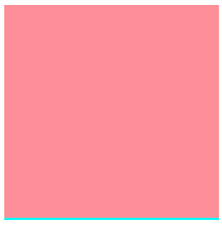


- 鼠标移动事件 onmousemove

```
div4.onmousemove = function(){
  this.style.backgroundColor = randomColor();
}
```

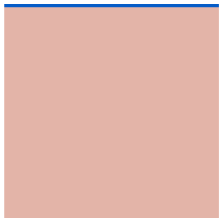
- 鼠标按下事件 onmousedown

```
div5.onmousedown = function(e){
  this.style.backgroundColor = randomColor();
};
```



- 鼠标抬起事件 onmouseup

```
div6.onmouseup = function(){
  this.style.backgroundColor = randomColor();
}
```

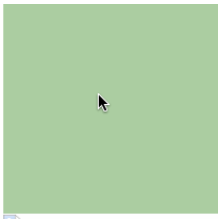


- 鼠标右键单击事件 oncontextmenu

```
div7.oncontextmenu = function(){
  this.style.backgroundColor = randomColor();
}
```

- 鼠标滚轮事件 onmousewheel

```
div8.onmousewheel = function(){
  this.style.backgroundColor = randomColor();
}
```



键盘事件

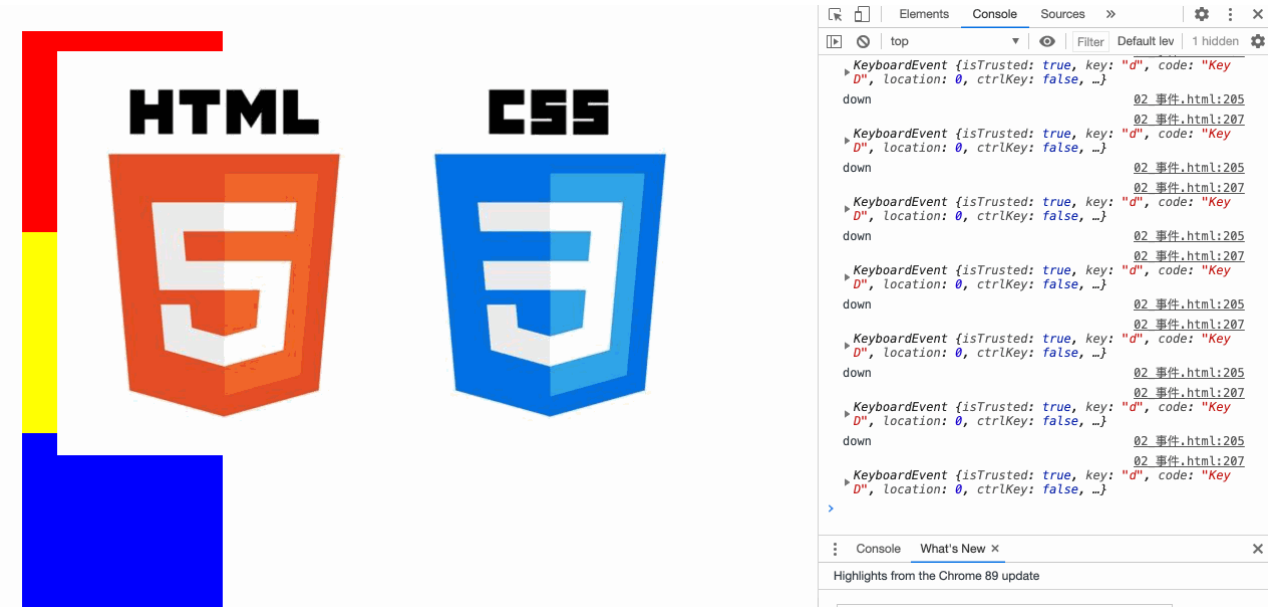
事件	说明
onkeypress	键盘的某个键被按下
onkeydown	键盘的某个键被按下或按住
onkeyup	键盘的某个键被松开

输入内容时按键执行顺序：
　onkeydown->onkeypress->输入->onkeyup
onkeypress无法响应中文输入，也无法响应系统功能键而onkeydown可以

- 键盘按下事件 onkeydown
  - 我们可以通过事件对象的keyCode属性, 获取用户按的是哪个键

```
var posX = 0;
```

```
var posY = 0;
window.onkeydown = function(e){
  console.log("down");
  var even = e || event;
  console.log(even);
  switch (even.keyCode){
    case 87:{
      posY -= 5;
      break;
    }
    case 83: {
      posY += 5;
      break;
    }
    case 65: {
      posX -= 5;
      break;
    }
    case 68: {
      posX += 5;
      break;
    }
  }
  img1.style.left = posX + "px";
  img1.style.top = posY + "px";
};
```



- 键盘抬起 onkeyup

```
window.onkeyup = function(){
  console.log("up");
}
```

- 键盘按下事件 onkeypress

```
window.onkeypress = function(e){
  var even = e || event;
  console.log(even);
}
```

光标事件

事件	说明
onfocus	元素获得焦点
onblur	元素失去焦点

表单事件

事件	说明
onsubmit	提交按钮被点击
onreset	重置按钮被点击
onselect	文本内容被选定
onchange	用户改变表单元素的内容

- 内容修改事件 onchange

```
i1.onchange = function(){
  console.log("内容变化了!");
};
```

- 获得焦点事件 onfocus

```
i1.onfocus = function(){
  console.log("获得焦点!");
}
```

- 失去焦点事件 onblur

```
i1.onblur = function(){
  console.log("失去焦点!");
};
```

- 内容输入事件 oninput

```
i1.oninput = function(){
  console.log("内容输入了!");
}
```

- 数据提交事件 onsubmit 该事件绑定给form表单

```
f.onsubmit = function(){
  console.log("数据提交了!");
}
```

事件流

事件发生时会在元素节点之间按照特定的顺序传播，这个传播过程即DOM事件流

DOM事件流分为3个阶段：

- 捕获阶段

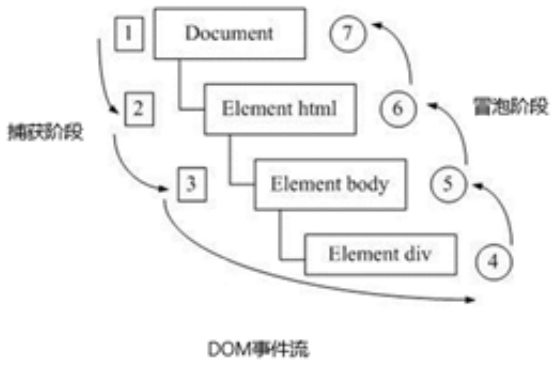
IE最早提出，事件逐级向下传播。当某个节点触发事件后, 系统会从根节点开始, 一直向下寻找, 直至找到真正触发该事件的节点. 沿途在直系继承树上绑定了相同事件的节点会按照顺序依次触发.

- 当前目标阶段

  真正触发事件的节点, 触发事件.

- 冒泡阶段

  网景最早提出，事件逐级向上传播。真正触发事件的节点触发完毕之后, 会沿着直系继承树一直向上传递该事件, 一直传递到根节点, 沿途绑定了相同事件的节点会依次触发.



DOM事件流

JS代码只能执行捕获或冒泡其中的一个阶段

- 0级DOM事件只能得到冒泡阶段
- 2级DOM事件可以通过addEventListener第三个参数来确定捕获还是冒泡
  实际开发中更多关注的是事件冒泡，很少用到事件捕获
  有些事件是没有冒泡的，比如onblur、onfocus、onmouseenter、onmouseleave

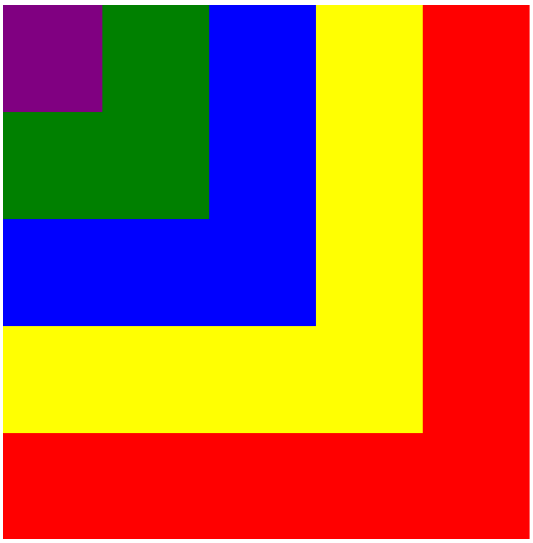
事件冒泡代码

```
<style type="text/css">
* {
  margin: 0;
  padding: 0;
}
#div1 {
  width: 500px;
  height: 500px;
  background-color: red;
}
#div2 {
  width: 400px;
  height: 400px;
  background-color: yellow;
}
#div3 {
  width: 300px;
  height: 300px;
  background-color: blue;
}
```

```
#div4 {
  width: 200px;
  height: 200px;
  background-color: green;
}
#div5 {
  width: 100px;
  height: 100px;
  background-color: purple;
}
```

```
<div id="div1">
  <div id="div2">
    <div id="div3">
      <div id="div4">
        <div id="div5"></div>
      </div>
    </div>
  </div>
</div>
```

```
div1.onclick = function(){alert("div1")};
div2.onclick = function(){alert("div2")};
div3.onclick = function(){alert("div3")};
div4.onclick = function(){alert("div4")};
div5.onclick = function(){alert("div5")};
```



事件捕获代码

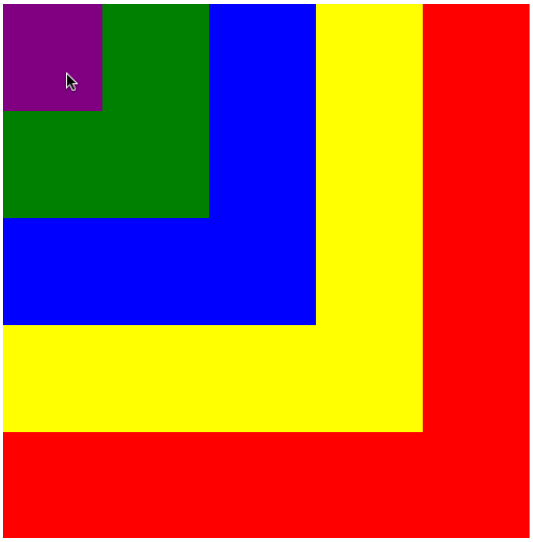
通过addEventListener的第三个参数, 可以将js默认的事件传播方式修改为捕获

```
<style type="text/css">
```

```css
* {
  margin: 0;
  padding: 0;
}
#div1 {
  width: 500px;
  height: 500px;
  background-color: red;
}
#div2 {
  width: 400px;
  height: 400px;
  background-color: yellow;
}
#div3 {
  width: 300px;
  height: 300px;
  background-color: blue;
}
#div4 {
  width: 200px;
  height: 200px;
  background-color: green;
}
#div5 {
  width: 100px;
  height: 100px;
  background-color: purple;
}
```

```html
<div id="div1">
  <div id="div2">
    <div id="div3">
      <div id="div4">
        <div id="div5"></div>
      </div>
    </div>
  </div>
</div>
```

```js
var isCapture = true;
div1.addEventListener("click", function(){alert("div1")}, isCapture);
div2.addEventListener("click", function(){alert("div2")}, isCapture);
div3.addEventListener("click", function(){alert("div3")}, isCapture);
div4.addEventListener("click", function(){alert("div4")}, isCapture);
div5.addEventListener("click", function(){alert("div5")}, isCapture);
```



事件对象

事件触发发生时就会产生事件对象，并且系统会以实参的形式传给事件处理函数。

当一个事件被触发之后, 会产生一系列与该事件有关的详细的数据信息, JS将它们全部存储在了一个叫"event"的 事件对象里, 我们直接使用即可

事件件对象本身的获取存在兼容问题
　标准浏览器中是浏览器给方法传递的参数，只需要定义形参 e 就可以获取到。
　在 IE6~8 中，浏览器不会给方法传递参数，如果需要的话，需要到 window.event 中获取查找。

```
element.onclick = function(e) {
  //兼容678
  e = e || window.event
 }
```

鼠标事件对象

- 以可视窗口为参照
  e.clientX
  　e.clientY
- 以文档页面为参照(包含滚动条滚动距离)
  e.pageX IE9+
  　e.pageY IE9+
- 以电脑屏幕为参照
  e.screenX
  　e.screenY

键盘事件对象

- e.keyCode 返回按键的ASCII值

事件对象常见的属性和方法

事件对象属性方法	说明
e.target	返回触发事件的对象 标准
e.srcElement	返回触发事件的对象 非标准 ie6-8使用
e.type	返回事件的类型 比如 click mouseover 不带on
e.cancelBubble	该属性阻止冒泡 非标准 ie6-8使用
e.returnValue	该属性 阻止默认事件（默认行为） 非标准 ie6-8使用 比如不让链接跳转
e.preventDefault()	该方法 阻止默认事件（默认行为） 标准 比如不让链接跳转
e.stopPropagation()	阻止冒泡 标准

e.target 和 this 的区别

this 是事件绑定的元素（绑定这个事件处理函数的元素）。
e.target 是事件触发的元素
通常情况下terget 和 this是一致的，
但有一种情况不同，那就是在事件冒泡时（父子元素有相同事件，单击子元素，父元素的事件处理函数也会被触发执行）
这时候this指向的是父元素，因为它是绑定事件的元素对象，
而target指向的是子元素，因为他是触发事件的那个具体元素对象。
比如给ul绑定了click事件，当点击里面的li时，this表示ul而e.target表示li

阻止默认行为

因为JS已经为一些元素添加了默认事件, 如a链接的点击事件, document的鼠标滚轮事件, 此时, 如果我们将来为我们的元素也添加了相同的事件, 系统的默认事件会阻碍我们的事件执行或者会阻碍我们写出的效果, 所以我们也有必要阻止默认事件的执行

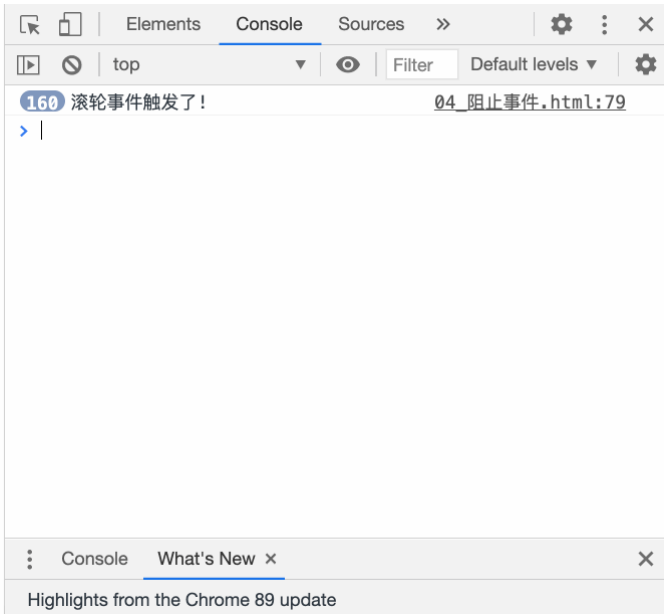
- 普通浏览器
  e.preventDefault();
- 低版本浏览器 ie678
  e.returnValue = false;
- 我们可以利用return false 也能阻止默认行为没有兼容性问题
  return false只能针对传统注册方式有效，对addEventListener注册的事件无效

阻止默认事件的执行：通过事件对象, 去调用preventDefault()

```css
#div6 {
  width: 200px;
  height: 200px;
  background-color: pink;
  margin-top: 500px;
}
```

```html
<div id="div6"></div>
```

```js
div6.onmousewheel = function(e){
  var even = e || event;
  even.preventDefault();
  console.log("滚轮事件触发了!");
}
```



右键菜单事件

oncontextmenu
阻止该事件默认行为可以实现禁用右键菜单

选中文本事件

onselectstart
阻止该事件默认行为可以实现禁止选中文本

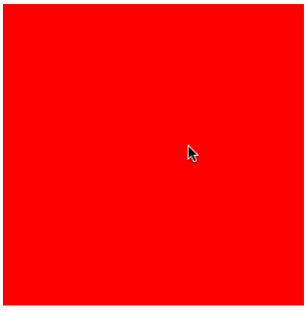
判断鼠标滚轮方向

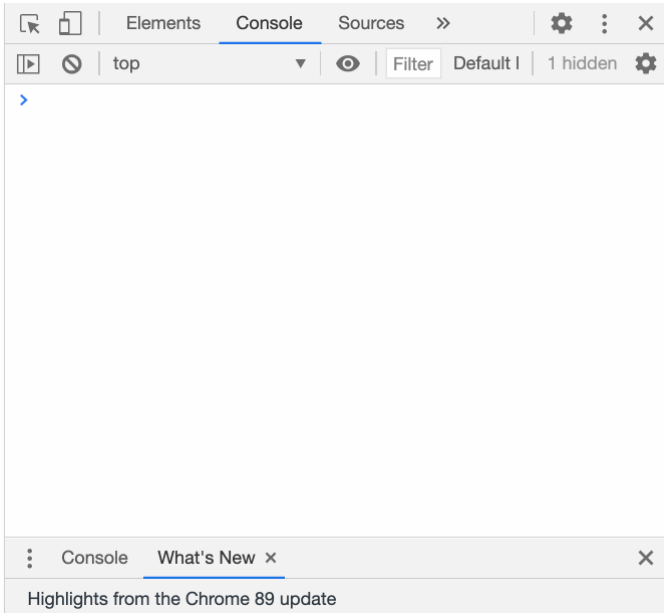
鼠标滚轮事件onmousewheel, 事件对象的wheelDelta属性存储滚轮的方向

wheelDelta>0 滚轮向上

wheelDelta<0 滚轮向下

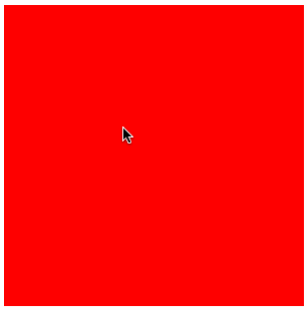
```
div1.onmousewheel = function(e){
  var even = e || event;
  if(even.wheelDelta > 0){
    console.log("上!");
  }else {
    console.log("下!");
  }
}
```

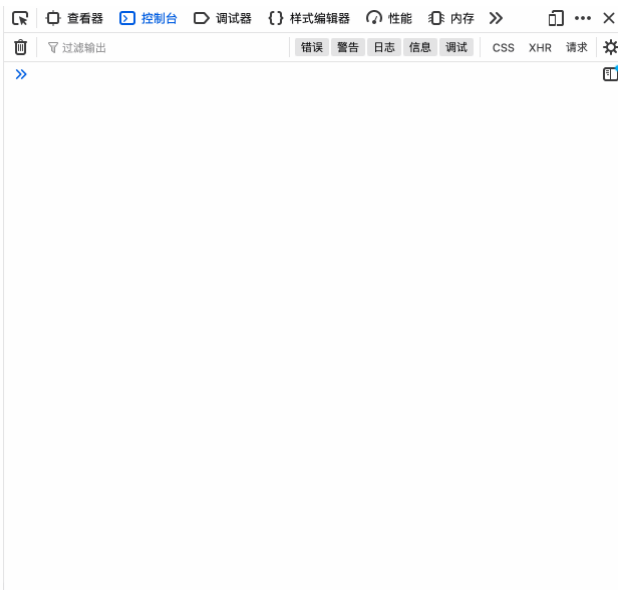




火狐浏览器不支持onmousewheel事件, 如果要兼容火狐, 需要通过"事件监听"的形式 绑定事件

```
div1.addEventListener("DOMMouseScroll", function(e){
  var even = e || event;
  console.log(even);
}, false);
```





阻止事件冒泡

因为JS默认的事件传播的特点, 如果继承树上有相同的事件, 会一并触发, 这会造成事件调用的混乱和问题, 所以, 我们有必要阻止事件冒泡

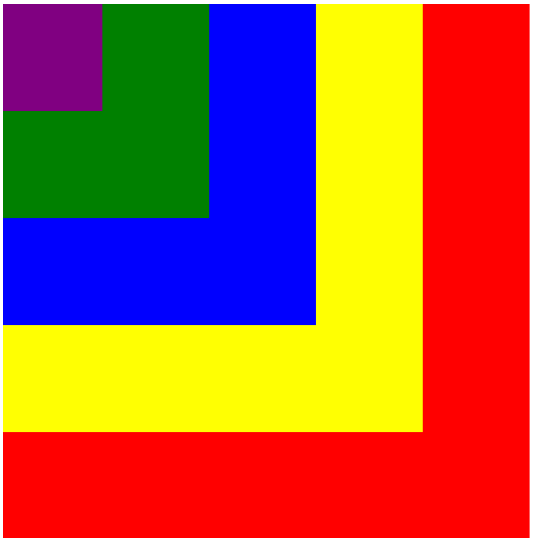
阻止事件冒泡的传播: 通过当前事件触发之后的事件对象, 去调用stopPropagation()

- 标准写法
  e.stopPropagation()
- IE6-8中的非标准写法
  e.cancelBubble = true

```
#div1 {
  width: 500px;
  height: 500px;
  background-color: red;
}
#div2 {
  width: 400px;
  height: 400px;
  background-color: yellow;
}
#div3 {
  width: 300px;
  height: 300px;
  background-color: blue;
}
#div4 {
  width: 200px;
  height: 200px;
  background-color: green;
}
#div5 {
  width: 100px;
  height: 100px;
  background-color: purple;
}
```

```
<div id="div1">
  <div id="div2">
    <div id="div3">
      <div id="div4">
        <div id="div5"></div>
      </div>
    </div>
  </div>
</div>
```

```
div1.onclick = function(){alert("div1")};
div2.onclick = function(){alert("div2")};
div3.onclick = function(){alert("div3")};
div4.onclick = function(){alert("div4")};
div5.onclick = function(e){
  var even = e || event;
  even.stopPropagation();
  alert("div5");
};
```



事件委托（委派）

事件委托也称为事件代理，在 jQuery 里面称为事件委派
事件委托原理
　给父节点添加事件， 利用事件冒泡影响每一个子节点
事件委托的作用
　只给父节点添加事件 ，不需要给每一个子节点添加事件，从而提高了程序的性能。
利用e.target来获取操作的子节点

轮播图

- **轮播图思路:**
  - 点击左右按钮, 换图 且 换分页器
  - 点击分页器, 换分页器 且 换图
  - 自动轮播 (每隔一段时间 调用 右按钮的点击事件)
  - 鼠标移入, 停止自动轮播, 显示左右按钮
  - 鼠标移出, 开启自动轮播, 隐藏左右按钮
- **重点:**
  - 分别把换图 和 换分页器 各自封装成两个函数
  - 换图和换分页器之间是要相互影响的, 让这两个逻辑共同依赖同一个变量

全局所需要的变量

```
// 声明一个变量记录当前是第几张图片
var nowIndex = 0;
// 声明一个变量记录将要切换到第几张图片
var goIndex = 0;
// 声明一个变量存储换一次图的偏移量
var x = 0;
// 声明一个存储一次换图的计时器
var changeImageTimer;
```

更换图片的函数

难点: 在于该函数需要在点击左右按钮, 自动轮播和点击分页器时都需要被调用, 如何在该函数调用时区分这三种行为, 因为这三种操作所影响的偏移元素需要偏移的方向和距离都是不同的

解决办法: 通过goIndex和nowIndex的关系判断当前的操作行为

```
function changeImage(flag) {
  changeImageTimer = setInterval(function() {
    x += flag == "auto" ? 5 : (flag == "page" ? 50 : 10);
    content.style.left = nowIndex * -300 + x * (goIndex > nowIndex ? -1 : 1) +
"px";
    // 一次换图结束, 清除计时器
    if(x == Math.abs(goIndex - nowIndex) * 300) {
      clearInterval(changeImageTimer);
      // 重置x
      x = 0;
      // 每次动画执行结束, 重新绑定按钮的点击事件
      setTimeout(function() {
        left.onclick = leftBtnClick;
        right.onclick = rightBtnClick;
      }, 200);
    }
  }, 10);
}
```

按钮点击事件

难点: 左按钮或者右按钮在点击的时候都会有各自的一个临界情况, 左按钮的临界点是0, 右按钮的临界点的是最后一张, 在点击按钮时, 记录当前下标和将来下标的两个变量都需要发生变化

```
function leftBtnClick() {
  // 每次点击之后就让点击事件失效
  this.onclick = null;
  if(goIndex == 0) {
    goIndex = 6;
    // 立刻把content拽回到最后一张
    content.style.left = "-1800px";
  }
  nowIndex = goIndex;
  goIndex--;
  changeImage("btn");
  changePaginations();
};

function rightBtnClick(clickFlag) {
    // 每次点击之后就让点击事件失效
    this.onclick = null;
    if(goIndex == 6) {
```

```
    goIndex = 0;
    // 立刻把content拽回到第一张
    content.style.left = "0px";
  }
    nowIndex = goIndex;
    goIndex++;
    changeImage(clickFlag);
    changePaginations();
};
```

跟换分页器的函数

难点: 需要找到即将被切换的分页器, 同时将上一次的分页器重置; 还要处理最后一张假图时的情况

解决办法: 通过goIndex变量即可知晓即将切换的分页, 上一次的分页器只需要将所有的重置即可

```
function changePaginations() {
  for(var i = 0; i < allLis.length; i++){
    if(i == goIndex) {
      allLis[i].style.backgroundColor = "darkblue";
    }else {
      allLis[i].style.backgroundColor = "lightcyan";
    }
  }
  // 用来处理到最后一张假图时的情况
  if(goIndex == 6) {
    allLis[0].style.backgroundColor = "darkblue";
  }
}
```

自动轮播效果

难点: 在鼠标移出可视区域时开启自动轮播, 在移入可视区域时停止自动轮播

解决办法: 给可视区域添加鼠标移入移出事件监听用户行为

遇到的问题: 我们之前自动轮播效果时通过计时器来进行切换图的, 但是我们通过操作会发现当页面处于非激活状态下时, 页面渲染被停止但是计时器还是在执, 所以当页面再次处于激活态, 计时器会出现混乱, 我们可以通过requestAnimationFrame解决此问题

```
wrapper.onmouseover = function() {
  left.style.opacity = 1;
  right.style.opacity = 1;
  isAuto = false;
}
wrapper.onmouseout = function() {
  left.style.opacity = 0;
  right.style.opacity = 0;
  isAuto = true;
```

```
  requestAnimationFrame(autoPlay);
}
// 自动播放
var flag = 0;
var isAuto = true;
requestAnimationFrame(autoPlay);
function autoPlay() {
  if(!isAuto) {
    return;
  }
  flag++;
  if(flag == 100) {
    rightBtnClick("auto");
    flag = 0;
  }
  requestAnimationFrame(autoPlay);
}
```