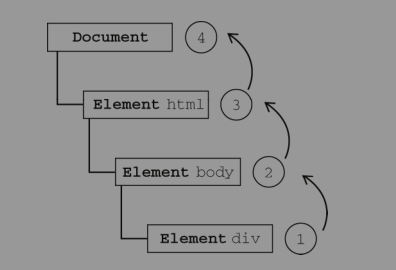
### 1. IE事件流和DOM事件流的区别?

事件流概念:事件流描述的是从页面中接收事件的顺序。IE 的事件流是事件冒泡流，而 Netscape Communicator 的事件流是事件捕获流。

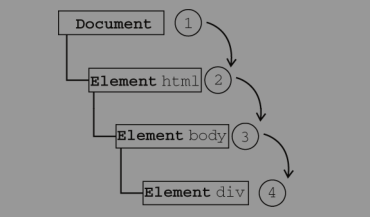
事件冒泡:IE 的事件流叫做事件冒泡（event bubbling），即事件开始时由最具体的元素（文档中嵌套层次最深的那个节点）接收，然后逐级向上传播到较为不具体的节点（文档）。

![](/images/domevent/event bubbling.png)



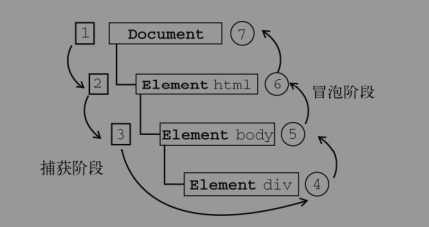
事件捕获:事件捕获的思想是不太具体的节点应该更早接收到事件，而最具体的节点应该最后接收到事件。事件捕获的用意在于在事件到达预定目标之前捕获它。

![](/images/domevent/event capturing.png)



DOM事件流:“DOM2级事件”规定的事件流包括三个阶段：事件捕获阶段、处于目标阶段和事件冒泡阶段。首先发生的是事件捕获，为截获事件提供了机会。然后是实际的目标接收到事件。最后一个阶段是冒泡阶段，可以在这个阶段对事件做出响应。

![](/images/domevent/DOM event.png)



ps:多数支持 DOM事件流的浏览器都实现了一种特定的行为；即使“DOM2 级事件”规范明确要求捕获阶段不会涉及事件目标，但 IE9、Safari、Chrome、Firefox 和 Opera 9.5 及更高版本都会在捕获阶段触发事件对象上的事件。结果，就是有两个机会在目标对象上面操作事件。IE9、Opera、Firefox、Chrome 和 Safari 都支持 DOM 事件流；IE8 及更早版本不支持 DOM 事件流。

```JavaScript

e=window.event||e;

if(e.stopPropagation){

e.stopPropagation();

}else{

e.cancelBubble=true;

}

```

```JavaScript

//e.target------>最开始触发该事件的事件源--谷歌支持,火狐支持

//window.event.srcElement---->最开始触发该事件的事件源--IE8的写法

//e.currentTarget--->当前触发该事件的事件对象--谷歌支持,火狐支持

//window.event.currentTarget-->当前触发该事件的事件对象--IE8写法

```

```JavaScript

/\* 事件有三个阶段:

\* 事件捕获阶段

\* 事件目标阶段

\* 事件冒泡阶段

\* 事件阶段有一个属性:这个属性是需要通过:事件参数对象.eventPhase来获取的

\* 如果这个属性的值是:

\* 1-----捕获阶段

\* 2-----目标阶段

\* 3-----冒泡阶段

e.type=====>获取的是当前触发该事件的事件类型(事件名字.没有on) \*/

```

### 2. IE和标准之间有哪些需要写的兼容性代码?

第一段:textContent(标准)与innerText(IE)

```javaScript

//设置任意一个元素的文本内容

function setInnerText(element,text) {

//该属性在浏览器中不支持

if(typeof element.textContent=="undefined"){

element.innerText=text;

}else{

//浏览器支持

element.textContent=text;

}

}

//获取任意一个元素的文本内容

function getInnerText(element) {

if(typeof element.textContent=="undefined"){

return element.innerText;

}else{

return element.textContent;

}

}

DOM 节点

//获取节点的属性------IE8中会忽略空白节点---认为是元素

//获取元素的属性------IE8中不能使用

//获取ul的父级节点

console.log(my$("uu").parentNode);

//获取ul的父级元素

console.log(my$("uu").parentElement);

//获取ul中的子级节点

console.log(my$("uu").childNodes);

//获取ul中的子级元素

console.log(my$("uu").children);

//获取ul中的第一个子级节点

console.log(my$("uu").firstChild);

//获取ul中的第一个子元素

console.log(my$("uu").firstElementChild);

//获取ul中最后一个子节点

console.log(my$("uu").lastChild);

//获取ul中最后一个子元素

console.log(my$("uu").lastElementChild);

//获取某个li的前一个兄弟节点

console.log(my$("two").previousSibling);

//获取某个li的前一个兄弟元素

console.log(my$("two").previousElementSibling);

//获取某个li的后一个兄弟节点

console.log(my$("two").nextSibling);

//获取某个li的后一个兄弟元素

console.log(my$("two").nextElementSibling);

节点中三个属性:

\* nodeType:如果是标签(1),如果是属性(2),如果是文本(3)

\* nodeName:如果是标签(标签的名字,大写的),如果是属性(属性的名字,小写的),如果是文本(#text)

\* nodeValue:标签--null,属性--属性的值,文本--文本内容

```

第二段:previousElementSibling和previousSibling;

```javaScript

//获取当前元素前一个元素

function getPreviousElement(element) {

if(element.previousElementSibling){

return element.previousElementSibling;

}else{

var ele=element.previousSibling;

while (ele&&ele.nodeType!==1){

ele=ele.previousSibling;

}

return ele;

}

}

//获取当前元素的后一个元素nextElementSibling和nextSibling

function getNextElement(element) {

if(element.nextElementSibling){

return element.nextElementSibling;

}else{

var ele=element.nextSibling;

while(ele&&ele.nodeType!==1){

ele=ele.nextSibling;

}

return ele;

}

}

//获取第一个子元素firstElementChild和firstChild

function getFirstElementByParent(parent) {

if(parent.firstElementChild){

return parent.firstElementChild;

}else{

var ele=parent.firstChild;

while (ele&&ele.nodeType!==1){

ele=ele.nextSibling;

}

return ele;

}

}

//获取最后一个子元素lastElementChild和lastChild

function getLastElementByParent(parent) {

if(parent.lastElementChild){

return parent.lastElementChild;

}else{

var ele=parent.lastChild;

while(ele&&ele.nodeType!==1){

ele=ele.previousSibling;

}

return ele;

}

}

//获取兄弟元素siblings

function getsiblings(ele) {

if(!ele)return;//判断当前的ele这个元素是否存在

var elements=[];//定义数组的目的就是存储当前这个元素的所有的兄弟元素

var el=ele.previousSibling;//当前元素的前一个节点

while (el){

if (el.nodeType===1){//元素

elements.push(el);//加到数组中

}

el=el.previousSibling;

}

el=ele.nextSibling;

while(el){

if(el.nodeType===1){

elements.push(el);

}

el=el.nextSibling;

}

return elements;

}

```

第三段:

```JavaScript

function getScroll() {

return {

top:window.pageYOffset||document.body.scrollTop||document.documentElement.scrollTop || 0,

left:window.pageXOffset||document.body.scrollLeft||document.documentElement.scrollLeft || 0

};

}

```

第四段:

引入:

```JavaScript

console.log(my$("dv").offsetLeft);//作废了

元素计算后的样式属性值----真正的设置的值(设置了left值,但是没有脱标)

谷歌支持,IE8不支持

window.getComputedStyle(元素,null).属性名字

IE8支持,谷歌不支持

元素.currentStyle.属性的名字

```

```JavaScript

//获取任意一个元素的某一个样式属性的值

function getStyle(element,attr) {

//判断浏览器是否支持这个属性,就相当于判断浏览器是否支持这个方法

if(window.getComputedStyle){

return window.getComputedStyle(element,null)[attr];

//null的位置是一个伪数组

}else{

return element.currentStyle[attr];

}

};

```

第五段:

```JavaScript

//谷歌和火狐都支持 事件参数对象---e,IE8不支持.IE8没有这个参数.

//低版本的火狐是不支持window.event的

//谷歌是支持window.event的,ie8支持.

e=window.event||e;

```

第六段:

```JavaScript

//e.clientX和e.clientY获取的是鼠标针对可视区的横纵坐标

//e.pageX和e.pageY----获取的是相对页面的坐标

//e.pageX和e.pageY在IE8中不支持,在ie8里面只能算,没有对应的属性支持.

//pageYOffset和pageXOffset是ie8里的写法,获取的是卷曲出去的值

//对象

var evtTools={

//该函数的返回值是一个事件参数对象

getEvent:function (e) {

return window.event||e;

},

//获取的是相对于页面可视区域的横坐标

getClientX:function (e) {

return this.getEvent(e).clientX;

},

//获取的是相对于页面可视区域的纵坐标

getClientY:function (e) {

return this.getEvent(e).clientY;

},

//获取的是向左卷曲出去的横坐标---相对于页面

getScrollLeft:function () {

return window.pageXOffset||document.body.scrollLeft||document.documentElement.scrollLeft||0;

},

//获取的是向上卷曲出去的纵坐标---相对于页面

getScrollTop:function () {

return window.pageYOffset||document.body.scrollTop||document.documentElement.scrollTop||0;

},

//获取的是相对于页面的横坐标----(卷曲出去的横坐标+可视区域的横坐标)

getPageX:function (e) {

return this.getEvent(e).pageX?this.getEvent(e).pageX:this.getScrollLeft()+this.getClientX(e);

},

//获取的是相对于页面的纵坐标----(卷曲出去的纵坐标+可视区域的纵坐标)

getPageY:function (e) {

return this.getEvent(e).pageY?this.getEvent(e).pageY:this.getScrollTop()+this.getClientY(e);

}

};

```

第七段:获取可视区域的宽度兼容代码

```JavaScript

//响应式布局原理:

//可视区域的宽,高度

//可视区域的宽度

function getClient() {

return window.innerWidth||document.body.clientWidth||document.documentElement.clientWidth;

}

//高度同理

```

第八段:绑定事件与解绑事件

为元素绑定事件

```JavaScript

/\* addEventListener()

\* 参数有三个

\* 1,事件的类型(事件的名字)

\* 2,事件处理函数(匿名函数,命名函数)

\* 3,false

\* 浏览器支持情况:谷歌支持,火狐支持,IE11也支持

\* this---当前触发该事件的对象

\* attachEvent()

\* 参数有两个

\* 1,事件的类型(事件的名字,有on)

\* 2,事件处理函数(匿名函数,命名函数)

\* 浏览器支持情况:谷歌不支持,获取不支持,IE11不支持,IE8支持

this---window \*/

```

为任意的一个元素注册多个的事件

```JavaScript

//为任意的一个元素注册多个的事件

//元素注册事件:元素,事件名字,事件处理函数(事件处理程序)

function addEvent(element,type,fn) {

//先获取该元素的事件

var oldEvent=element["on"+type];

//判断该元素之前有没有注册过事件

if(typeof oldEvent!="function"){

element["on"+type]=fn;

}else{

//注册了

element["on"+type]=function () {

oldEvent();

fn();

};

}

}

```

绑定事件的兼容代码

```JavaScript

//对象.addEventListener("click",匿名函数,false);

//对象.attachEvent("onclick",匿名函数);

//为任意一个元素绑定多个相同的事件

function addEventListener(element,type,fn) {

//判断浏览器是否支持这个方法

if(element.addEventListener){

element.addEventListener(type,fn,false);

}else if(element.attachEvent){

element.attachEvent("on"+type,fn);

}else{

element["on"+type]=fn;

}

}

```

为元素解绑事件的兼容代码

```JavaScript

//为元素解绑事件的兼容代码---元素,--事件名字--对应的事件处理函数的名字

function removeEventListener(element,type,fn) {

if(element.removeEventListener){

element.removeEventListener(type,fn,false);

}else if(element.detachEvent){

element.detachEvent("on"+type,fn);

}else{

element["on"+type]=null;

}

}

```

第九段:事件冒泡与捕获

兼容代码

```JavaScript

e=window.event||e;

if(e.stopPropagation){

e.stopPropagation();

}else{

e.cancelBubble=true;

}

```

```JavaScript

//e.target------>最开始触发该事件的事件源--谷歌支持,火狐支持

//window.event.srcElement---->最开始触发该事件的事件源--IE8的写法

//e.currentTarget--->当前触发该事件的事件对象--谷歌支持,火狐支持

//window.event.currentTarget-->当前触发该事件的事件对象--IE8写法

```

```JavaScript

/\* 事件有三个阶段:

\* 事件捕获阶段

\* 事件目标阶段

\* 事件冒泡阶段

\* 事件阶段有一个属性:这个属性是需要通过:事件参数对象.eventPhase来获取的

\* 如果这个属性的值是:

\* 1-----捕获阶段

\* 2-----目标阶段

\* 3-----冒泡阶段

e.type=====>获取的是当前触发该事件的事件类型(事件名字.没有on) \*/

```

第十段:H5中标签的兼容问题

```JavaScript

<!--只有ie浏览器能识别-->

<!--[if lte IE 9]>

<script src="html5shiv.min.js"></script>

<![endif]-->

```

```javaScript

<!--条件注释 在IE9以下 加载-->

<!--[if lt IE 9]>

<script src="lib/html5shiv/html5shiv.min.js"></script>

<script src="lib/respond/respond.min.js"></script>

<![endif]-->

```

<!--

html5shiv IE8 支持语义化标签

respond IE8 支持媒体查询

-->

<!-- 警告：respond你不能使用file://这个种形式打开网页 http:// -->

原理是:

```JavaScript

//html5中的标签想要在IE8中支持,先创建元素,设置css的display:block

```