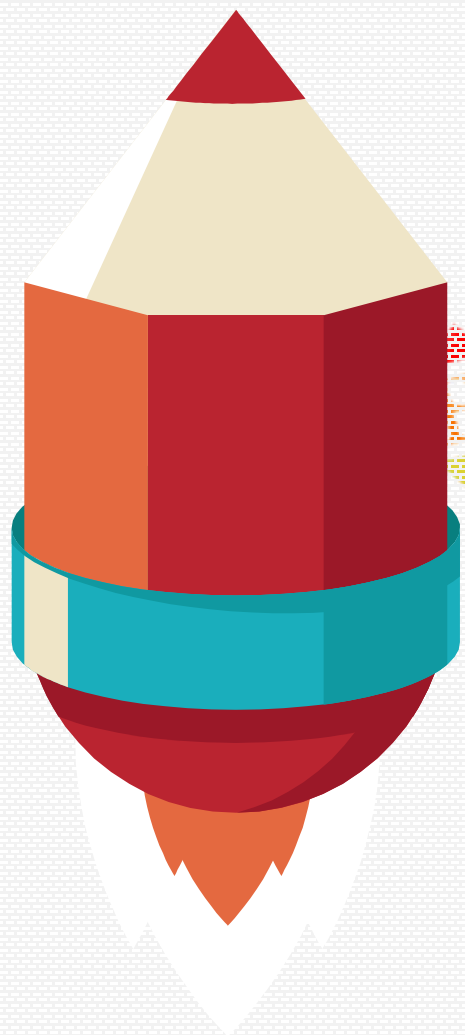




第20课：BOM终章

■ 主讲老师：万章



四知

移动端事件

DOM变动事件

文本事件

其他事件类型

组合式拖拽事件



移动端事件

移动端事件的简介

iOS 版Safari 为了向开发人员传达一些特殊信息，新增了一些专有事件。因为iOS 设备既没有鼠标

也没有键盘，所以在为移动Safari 开发交互性网页时，常规的鼠标和键盘事件根本不够用。随着Android中的WebKit 的加入，很多这样的专有事件变成了事实标准，导致W3C 开始制定Touch Events 规范（参见<https://dvcs.w3.org/hg/webevents/raw-file/tip/touchevents.html>）。以下介绍的事件只针对触摸设备

触摸式设备也是拥有鼠标事件的

移动端的主要事件

具体来说，有以下几个触摸事件。

`touchstart`：当手指触摸屏幕时触发；即使已经有一个手指放在了屏幕上也会触发。

`touchmove`：当手指在屏幕上滑动时连续地触发。在这个事件发生期间，调用`preventDefault()`可以阻止滚动。

`touchend`：当手指从屏幕上移开时触发。

`touchcancel`：当系统停止跟踪触摸时触发。关于此事件的确切触发时间，文档中没有明确说明。

上面这几个事件都会冒泡，也都可以取消。虽然这些触摸事件没有在DOM 规范中定义，但它们却是以兼容DOM 的方式实现的。因此，每个触摸事件的`event` 对象都提供了在鼠标事件中常见的属性：`clientX`、`clientY`、`screenX`、`screenY`、`detail`、`altKey`、`shiftKey`、`ctrlKey` 和`metaKey`。

移动端事件的触发对象

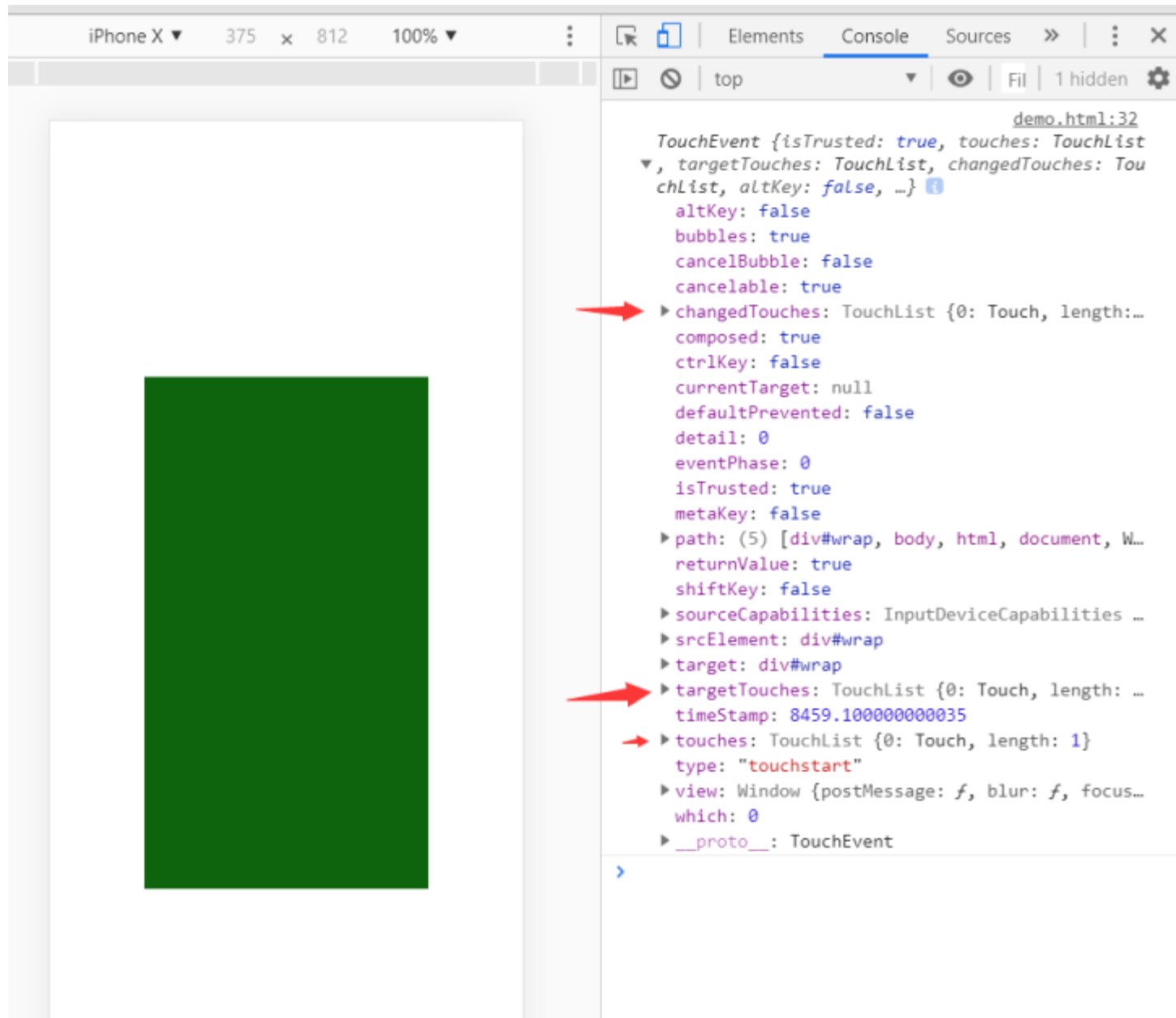
除了常见的DOM属性外，触摸事件还包含下列三个用于跟踪触摸的属性。

touches：表示当前跟踪的触摸操作的Touch对象的数组。

targetTouches：特定于事件目标的Touch 对象的数组。

❑ **changedTouches**：表示自上次触摸以来发生了什么改变的Touch 对象的数组。

注意：移动端屏幕大多支持多点触控，所以触摸事件的触发对象可能会有多个



移动端事件对象的位置属性

每个Touch 对象包含下列属性

clientX: 触摸目标在视口中的x 坐标。

clientY: 触摸目标在视口中的y 坐标。

identifier: 标识触摸的唯一ID。

pageX: 触摸目标在页面中的x 坐标。

pageY: 触摸目标在页面中的y 坐标。

screenX: 触摸目标在屏幕中的x 坐标。

screenY: 触摸目标在屏幕中的y 坐标。

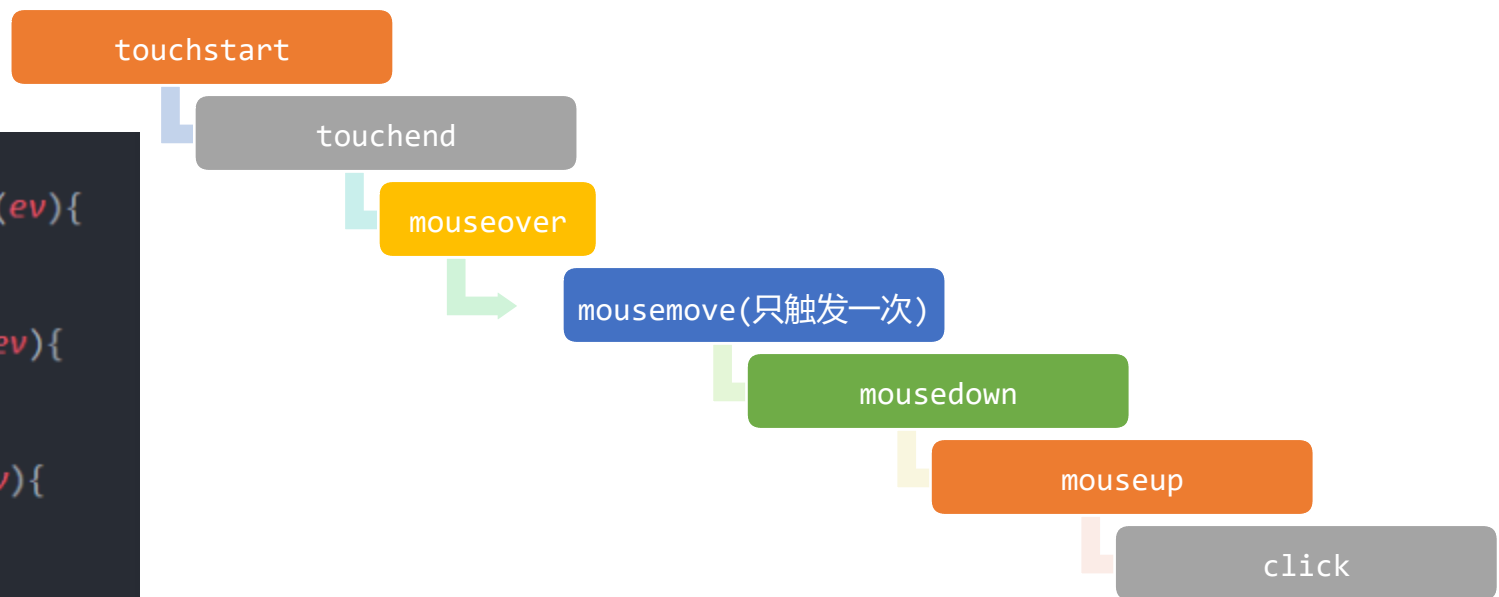
target: 触摸的DOM 节点目标。

使用这些属性(该位置属性与PC的位置属性相同)可以跟踪用户对屏幕的触摸操作

```
bubbles: true
cancelBubble: false
cancelable: true
changedTouches: TouchList {0: Touch, lengt...
composed: true
ctrlKey: false
currentTarget: null
defaultPrevented: false
detail: 0
eventPhase: 0
isTrusted: true
metaKey: false
path: (5) [div#wrap, body, html, document,...
returnValue: true
shiftKey: false
sourceCapabilities: InputDeviceCapabilitie...
srcElement: div#wrap
target: div#wrap
targetTouches: TouchList
  ▼ 0: Touch
    clientX: 226
    clientY: 263
    force: 1
    identifier: 0
    pageX: 226
    pageY: 263
    radiusX: 11.5
    radiusY: 11.5
    rotationAngle: 0
    screenX: 287
    screenY: 494
    target: div#wrap
    __proto__: Touch
  length: 1
  __proto__: TouchList
timestamp: 3097.8100000002087
touches: TouchList {0: Touch, length: 1}
type: "touchstart"
view: Window {postMessage: f, blur: f, foc...
which: 0
__proto__: TouchEvent
```

移动端事件的触发顺序1

```
30 let wrap=document.querySelector("#wrap");
31 wrap.addEventListener("touchstart",function(ev){
32     console.log("手指触摸");
33 })
34 wrap.addEventListener("touchmove",function(ev){
35     console.log("手指滑动");
36 })
37 wrap.addEventListener("touchend",function(ev){
38     console.log("手指抬起");
39 })
40 wrap.addEventListener("mouseover",function(ev){
41     console.log("鼠标悬浮");
42 })
43 wrap.addEventListener("mousedown",function(ev){
44     console.log("鼠标按下");
45 })
46 wrap.addEventListener("mouseup",function(ev){
47     console.log("鼠标抬起");
48 })
49 wrap.addEventListener("mousemove",function(ev){
50     console.log("鼠标滑动");
51 })
52 wrap.addEventListener("click",function(ev){
53     console.log("鼠标点击");
54 })
55
```



	Elements	Console	Sources	»	⋮	×
	top		Fil	2 hidden	⚙	
手指触摸		demo.html:32				
手指抬起		demo.html:38				
鼠标悬浮		demo.html:41				
鼠标滑动		demo.html:50				
鼠标点击		demo.html:44				
鼠标抬起		demo.html:47				
鼠标点击		demo.html:53				

移动端事件的触发顺序2

Elements	Console	Sources	»	⋮	×
🔍	🔍	top	▼	👁	File 2 hidden ⚙
手指触摸 demo.html:32					
133 手指滑动 demo.html:35					
手指抬起 demo.html:38					
>					

手指滑动事件一旦触发,将不会在激活鼠标事件

```
30 let wrap=document.querySelector("#wrap");
31 wrap.addEventListener("touchstart",function(ev){
32     console.log("手指触摸");
33 })
34 wrap.addEventListener("touchmove",function(ev){
35     console.log("手指滑动");
36 })
37 wrap.addEventListener("touchend",function(ev){
38     console.log("手指抬起");
39 })
40 wrap.addEventListener("mouseover",function(ev){
41     console.log("鼠标悬浮");
42 })
43 wrap.addEventListener("mousedown",function(ev){
44     console.log("鼠标按下");
45 })
46 wrap.addEventListener("mouseup",function(ev){
47     console.log("鼠标抬起");
48 })
49 wrap.addEventListener("mousemove",function(ev){
50     console.log("鼠标滑动");
51 })
52 wrap.addEventListener("click",function(ev){
53     console.log("鼠标点击");
54 })
55
```



DOM变动事件

DOM变动事件之删除节点事件

在使用removeChild()或replaceChild()或remove()
从DOM中删除某节点会触发删除节点事件

Elements	Console	Sources	»	:	×
top	▼	File	4 hidden	⚙	
	DOM节点被删除事件	demo.html:39			
	DOM节点从网页上被删除	demo.html:42			
	DOM节点从网页上被删除	demo.html:45			
	DOM节点树更新	demo.html:36			

```
<body>
  <div id="wrap">
    <div class="test"></div>
  </div>
  <script>
    let wrap=document.querySelector("#wrap");
    let test=document.querySelector(".test");
    let body=document.querySelector("body");
    body.addEventListener("DOMSubtreeModified",function(ev){
      console.log("DOM节点树更新");
    })
    wrap.addEventListener("DOMNodeRemoved",function(ev){
      console.log("DOM节点被删除事件");
    })
    wrap.addEventListener("DOMNodeRemovedFromDocument",function(ev){
      console.log("DOM节点从网页上被删除");
    })
    test.addEventListener("DOMNodeRemovedFromDocument",function(ev){
      console.log("DOM节点从网页上被删除");
    })
    wrap.remove();
  </script>
</body>
```

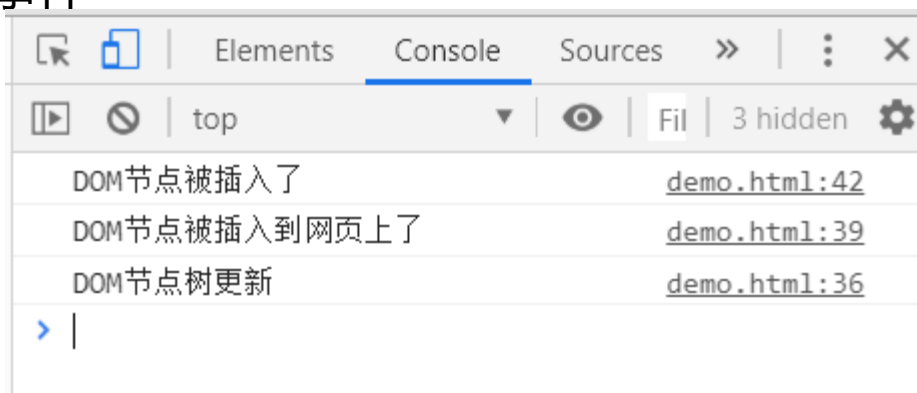
被删除节点触发DOMNodeRemoved
(DOM节点被删除)事件

被删除节点的父节点触
发DOMSubtreeModified (DOM树更新)事件

被删除的节点和被删除节点的子节点会触
发DOMNodeRemovedFromDocument (DOM节点从网页上被删
除)事件

DOM变动事件之插入节点事件

在使用appendChild()、replaceChild()或insertBefore()向DOM中插入节点时会触发插入事件



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

  </div>
  <script>
    let wrap=document.querySelector("#wrap");
    let test=document.createElement("div");
    let body=document.querySelector("body");
    wrap.addEventListener("DOMSubtreeModified",function(ev){
      console.log("DOM节点树更新");
    })
    test.addEventListener("DOMNodeInsertedIntoDocument",function(ev){
      console.log("DOM节点被插入到网页上了");
    })
    test.addEventListener("DOMNodeInserted",function(ev){
      console.log("DOM节点被插入了");
    })
    wrap.appendChild(test);
  </script>
</body>
```

被插入的节点触发DOMNodeInserted (DOM节点被插入)事件

被插入了节点的父节点触发DOMSubtreeModified (DOM树更新)事件

被插入的节点会触发DOMNodeInsertedIntoDocument (DOM节点已经被插入到网页上)事件



文本事件

文本事件

“DOM3 级事件”规范中引入了一个新事件，名叫textInput。

根据规范，当用户在可编辑区域(任意可输入区域)中输入字符时，就会触发这个事件。这个事件与keypress事件有所不同

区别之一就是任何可以获得焦点的元素都可以触发keypress 事件，但只有可编辑区域才能触发textInput事件。

区别之二是textInput 事件只会在用户按下能够输入实际字符的键时才会被触发，而keypress事件则在按下那些能够影响文本显示的键时也会触发（例如退格键）。

文本事件

```
<input type="text" id="wrap">
<script>
  let wrap=document.querySelector("#wrap");

  wrap.addEventListener("textInput",function(ev){
    console.log(ev);
  })
</script>
```

由于textInput 事件主要考虑的是字符，因此它的event 对象中还包含一个data 属性，这个属性的值就是用户输入的字符（而非字符编码）。换句话说，用户在没有按下上档键的情况下按下了S 键，data 的值就是"s"，而如果在按住上档键时按下该键，data 的值就是"S"。



The screenshot shows the browser's developer console with the 'Console' tab selected. The top bar indicates the file is 'demo.html:33'. The console log shows a 'TextEvent' object with the following properties:

- `isTrusted`: true
- `data`: "1" (highlighted with a red arrow)
- `view`: Window {postMessage: f, blur: f, focus...
- `detail`: 0
- `sourceCapabilities`: null
- `bubbles`: true
- `cancelBubble`: false
- `cancelable`: true
- `composed`: true
- `currentTarget`: null
- `defaultPrevented`: false
- `eventPhase`: 0
- `isTrusted`: true
- `path`: (5) [input#wrap, body, html, document, ...]
- `returnValue`: true
- `sourceCapabilities`: null
- `srcElement`: input#wrap
- `target`: input#wrap
- `timestamp`: 3677.059999999983
- `type`: "textInput"
- `view`: Window {postMessage: f, blur: f, focus...
- `which`: 0
- `__proto__`: TextEvent



其他事件对象

其他事件对象

属性/方法	类 型	读/写	说 明
bubbles	Boolean	只读	表明事件是否冒泡
cancelable	Boolean	只读	表明是否可以取消事件的默认行为
currentTarget	Element	只读	其事件处理程序当前正在处理事件的那个元素
defaultPrevented	Boolean	只读	为 true 表示已经调用了 preventDefault() (DOM3级事件中新增)
detail	Integer	只读	与事件相关的细节信息
eventPhase	Integer	只读	调用事件处理程序的阶段: 1表示捕获阶段, 2表示“处于目标”, 3表示冒泡阶段

```
demo.html:33
MouseEvent {isTrusted: true, screenX: 258, screenY: 508, clientX: 197, clientY: 277, ...}
  altKey: false
  bubbles: true
  button: 0
  buttons: 0
  cancelBubble: false
  cancelable: true
  clientX: 197
  clientY: 277
  composed: true
  ctrlKey: false
  currentTarget: null
  defaultPrevented: false
  detail: 1
  eventPhase: 0
  fromElement: null
  isTrusted: true
  layerX: 197
  layerY: 277
  metaKey: false
  movementX: 0
  movementY: 0
  offsetX: 122
  offsetY: 74
  pageX: 197
  pageY: 277
  path: (5) [div#wrap, body, html, document, ...]
  relatedTarget: null
  returnValue: true
  screenX: 258
  screenY: 508
  shiftKey: false
  sourceCapabilities: InputDeviceCapabilities
  srcElement: div#wrap
  target: div#wrap
  timeStamp: 646.0549999992509
  toElement: div#wrap
  type: "click"
  view: Window {postMessage: f, blur: f, foc...}
  which: 1
```

其他事件对象

属性/方法	类 型	读/写	说 明
<code>preventDefault()</code>	Function	只读	取消事件的默认行为。如果cancelable是true，则可以使用这个方法
<code>stopImmediatePropagation()</code>	Function	只读	取消事件的进一步捕获或冒泡，同时阻止任何事件处理程序被调用（DOM3级事件中新增）
<code>stopPropagation()</code>	Function	只读	取消事件的进一步捕获或冒泡。如果bubbles为true，则可以使用这个方法
<code>target</code>	Element	只读	事件的目标
<code>trusted</code>	Boolean	只读	为true表示事件是浏览器生成的。为false表示事件是由开发人员通过JavaScript创建的（DOM3级事件中新增）
<code>type</code>	String	只读	被触发的事件的类型
<code>view</code>	AbstractView	只读	与事件关联的抽象视图。等同于发生事件的window对象

其他事件对象eg

例：preventDefault()，阻止默认事件目标元素在该事件下默认执行的行为

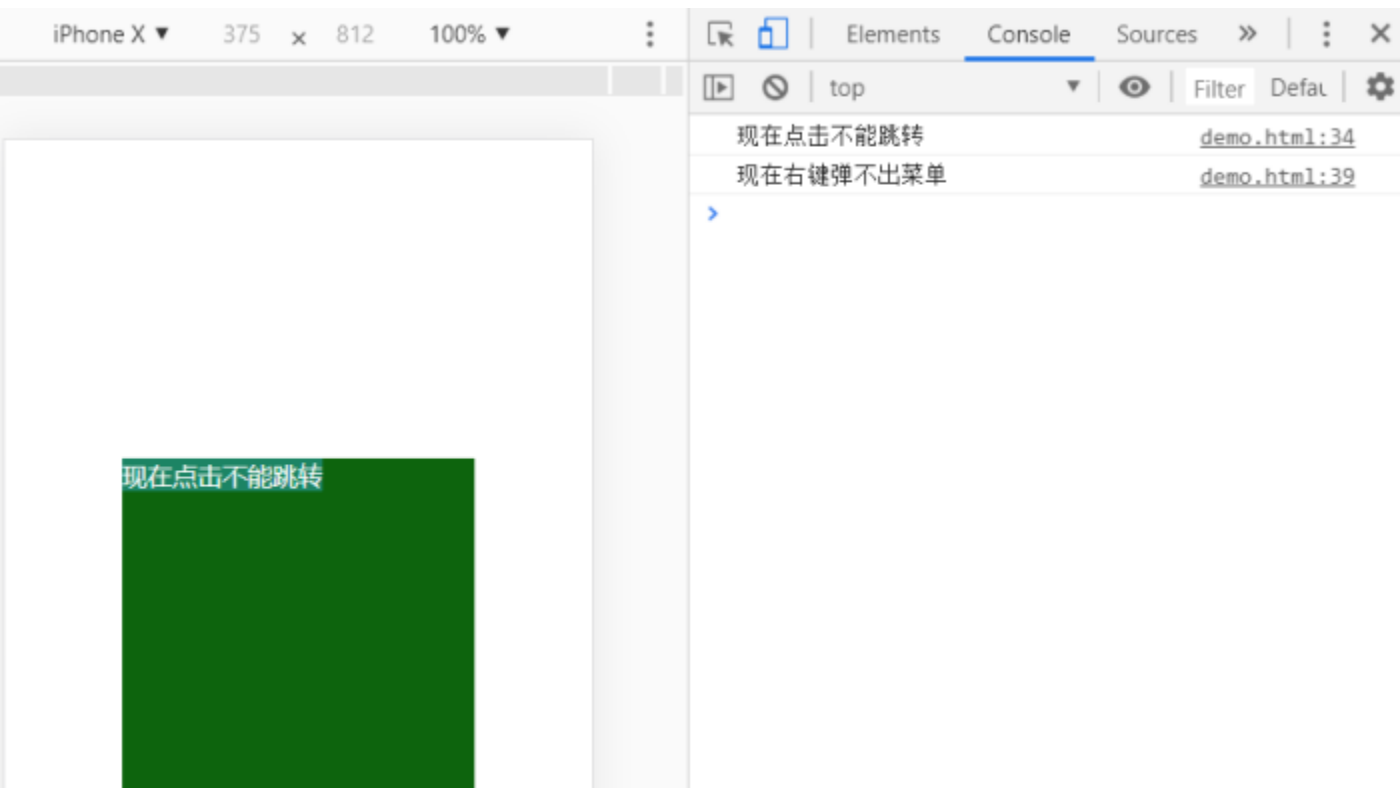
浏览器的常见默认行为

a标签可以进行跳转

右键弹出菜单

点击submit可以进行提交

... ..



```
<a id="wrap">现在点击不能跳转</a></div>
<script>
  let wrap=document.querySelector("#wrap");

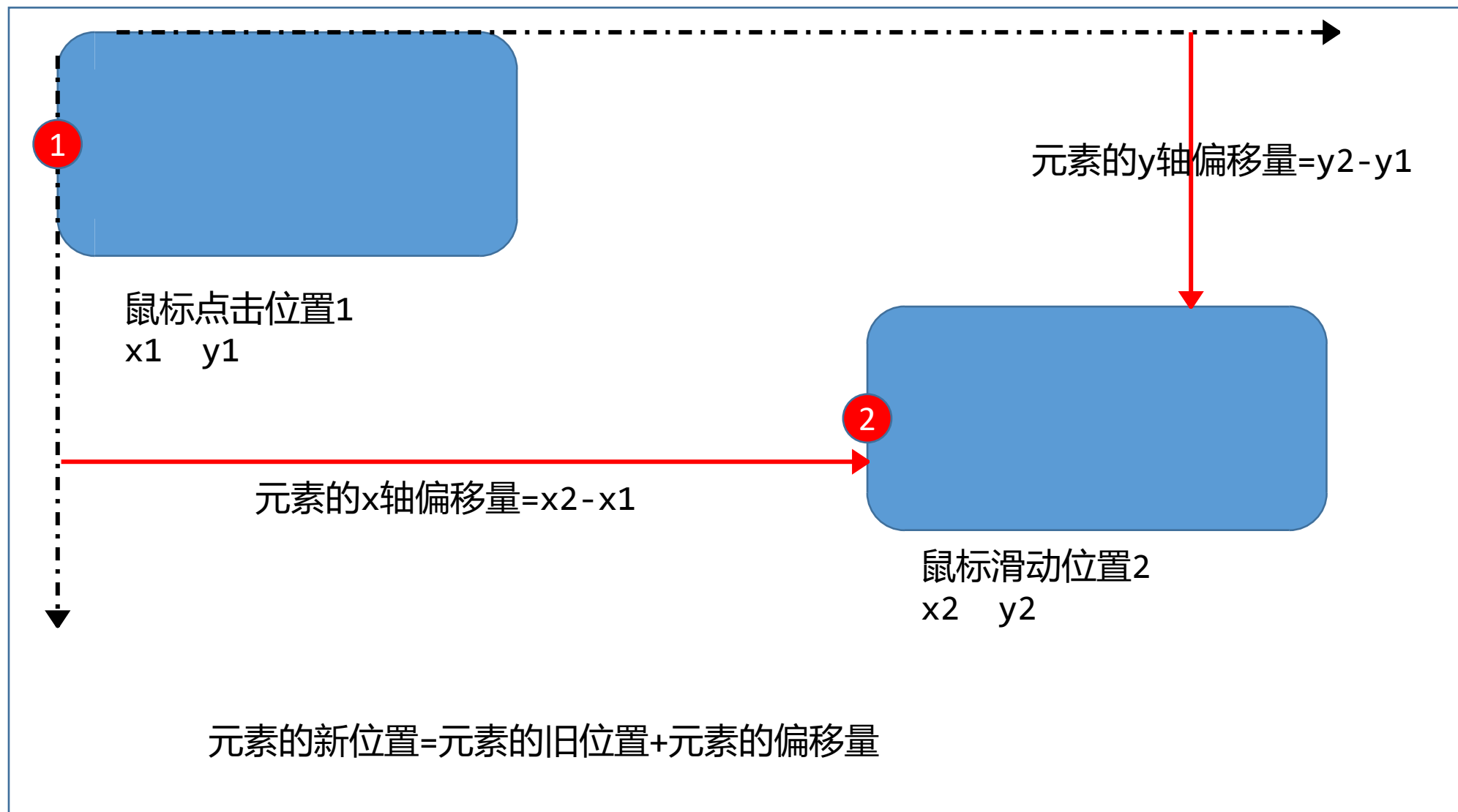
  wrap.addEventListener("click",function(ev){
    ev.preventDefault();
    console.log("现在点击不能跳转");
  })

  document.oncontextmenu=function(ev){
    ev.preventDefault();
    console.log("现在右键弹不出菜单");
  }
</script>
```



组合式拖拽事件

核心概念模型





核心概念模型

