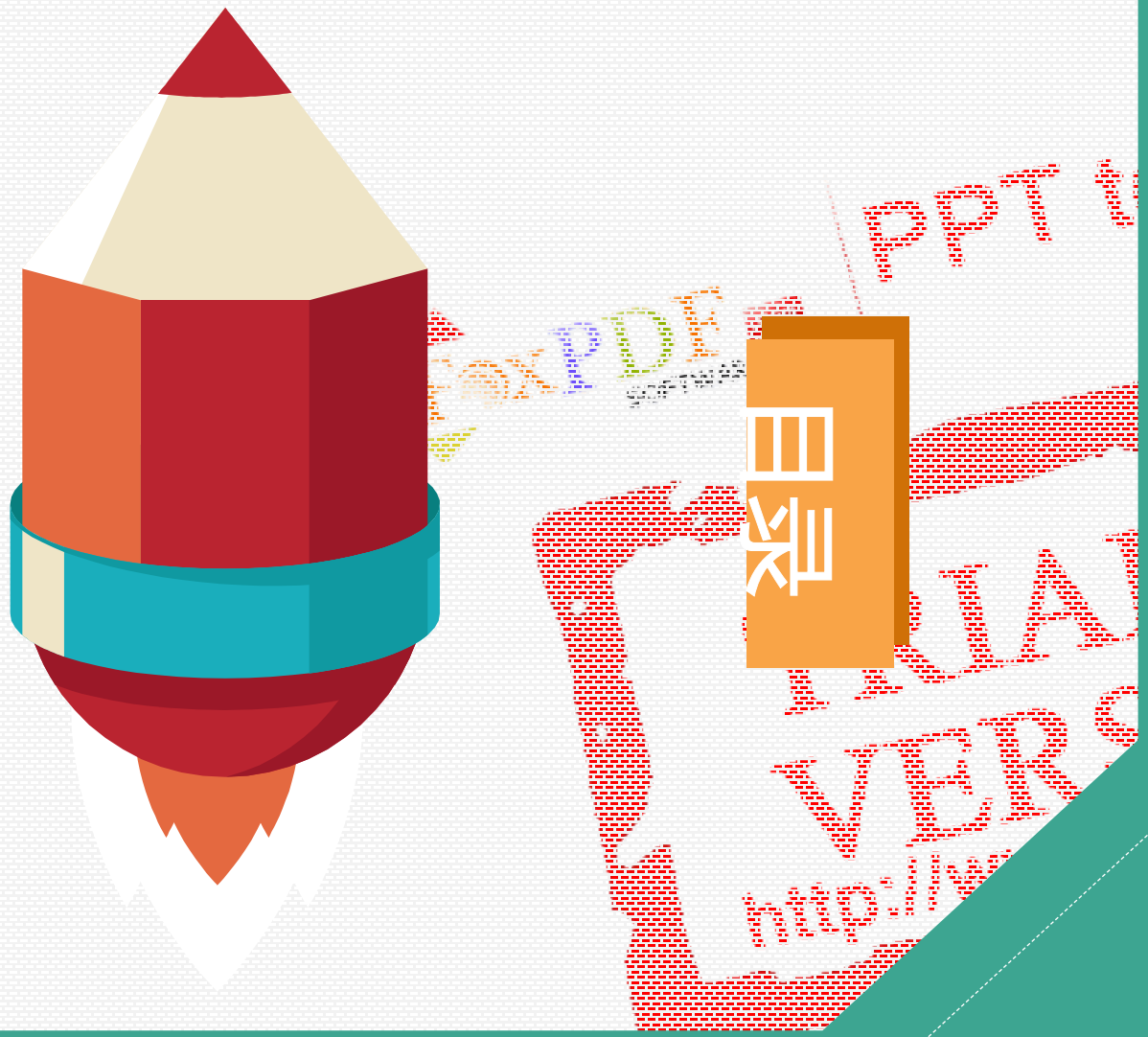
第18课：BOM中级

主讲老师：万章



- 什么是BOM
- 浏览器窗口设置
- location对象
- history对象
- 事件冒泡和事件捕捉

什么是BOM

什么是BOM

BOM 的核心对象是window，它表示浏览器的一个实例。在浏览器中，window 对象有双重角色,它既是通过JavaScript 访问浏览器窗口的一个接口，又是ECMAScript 规定的Global(全局) 对象。

这意味着在网页中定义的任何一个对象、变量和函数，都以window 作为其Global 对象，因此有权访问parseInt()等方法。

全局作用域

```
var age = 29;
function sayAge() {
        alert(this.age);
}
alert(window.age); //29
sayAge(); //29
window.sayAge(); //29
```

我们在全局作用域中定义了一个变量age 和一个函数sayAge()，它们被自动归在了window 对象名下。于是，可以通过window.age 访问变量age，可以通过window.sayAge()访问函数sayAge()。由于sayAge()存在于全局作用域中，因此this.age 被映射到window.age，最终显示的仍然是正确的结果。

```
var age = 29;
window.color = "red";
delete window.age;//返回false
delete window.color; //返回true
alert(window.age); //29
alert(window.color); //undefined
```

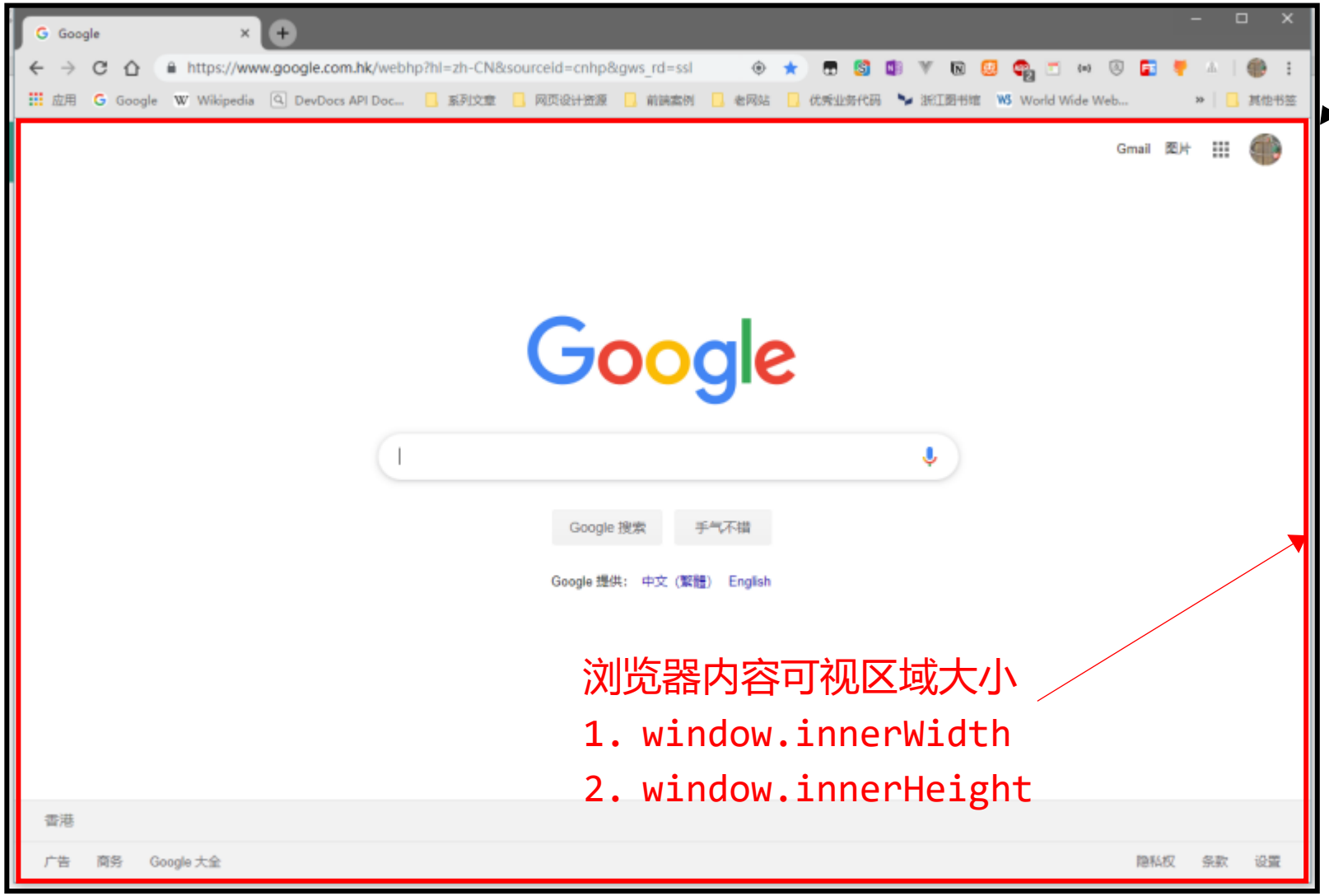
抛开全局变量会成为window 对象的属性不谈，定义全局变量与在window 对象上直接定义属性还是有一点差别：
全局变量不能通过delete 操作符删除，而直接在window 对象上的定义的属性可以

```
var age = 29;
let abc=22;
```

let指令创造的变量不会挂在window对象名下!!! !!!

浏览器窗口设置

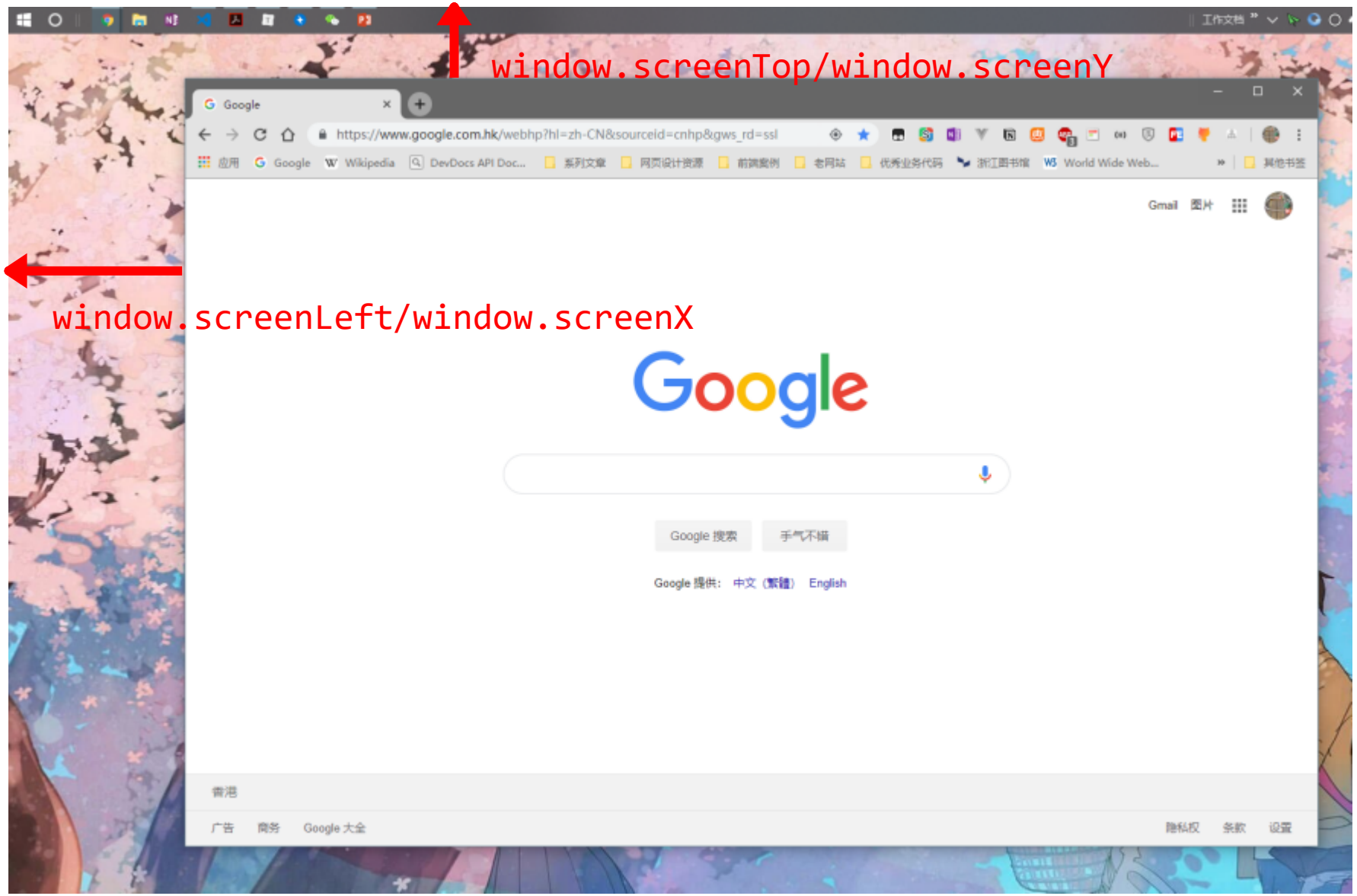
浏览器窗口大小获取



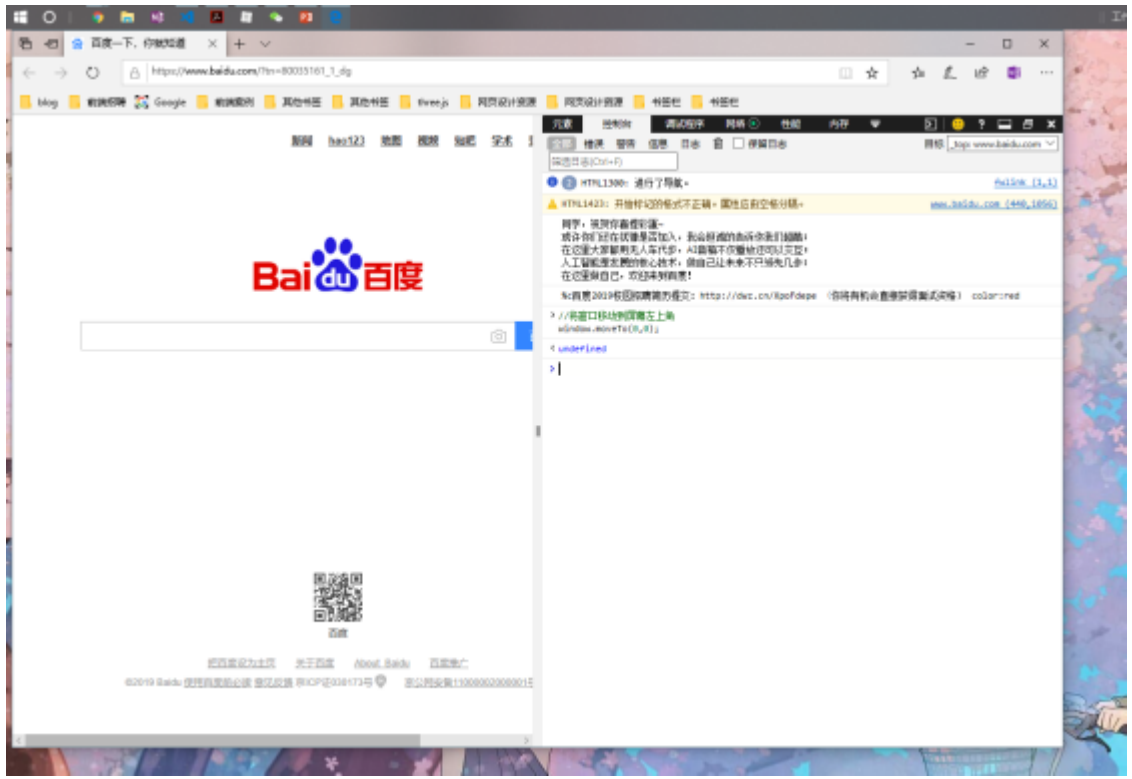
浏览器软件界面可视区域大小

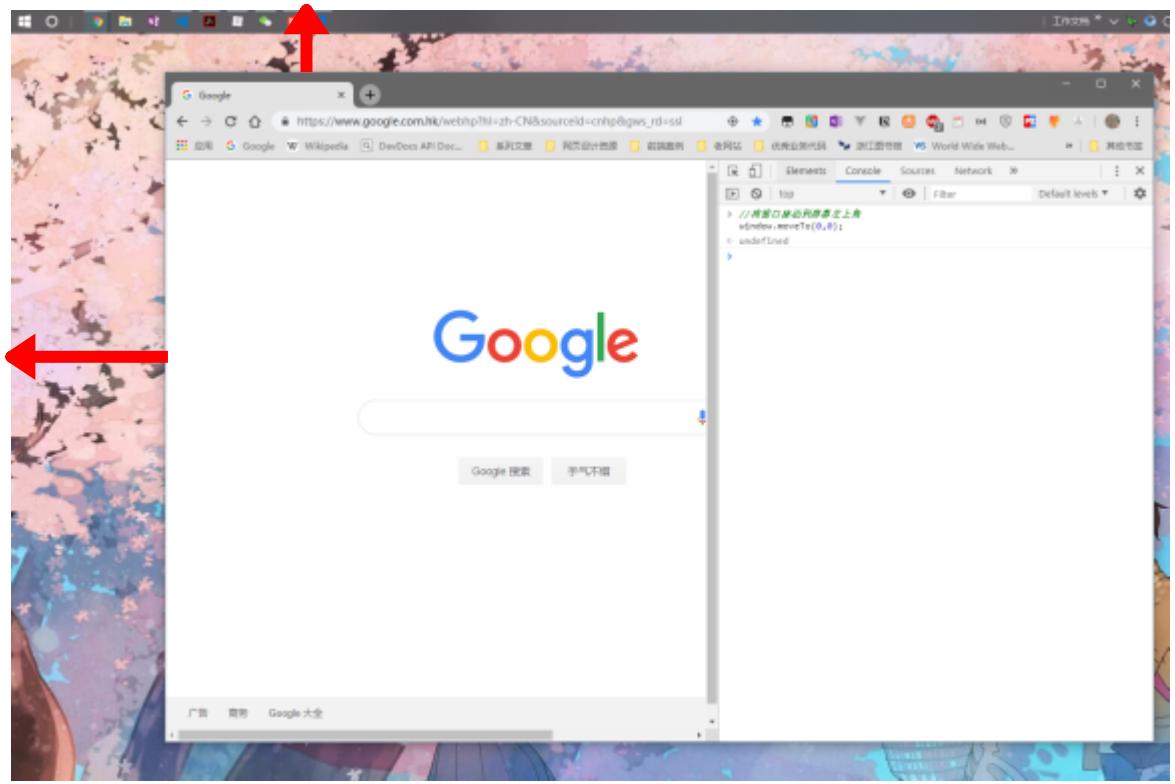
1. window.outerWidth
2. window.outerHeigh
   t

浏览器窗口位置获取


window.screenTop/window.screenY
window.screenLeft/window.screenX
Google
https://www.google.com.hk/webhp?hl=zh-CN&sourceid=cnhp&gws_rd=ssl
Gmail
图片
Google 搜索
手气不错
Google 提供：
中文（繁體）
English
香港
广告
商务
Google 大全
隐私权
条款
设置

浏览器窗口位置设置





window.moveTo()接收的是新位置的*x* 和*y* 坐标值

window.moveBy()接收的是在水平和垂直方向上移动的像素数

!!!!!!此方法当前谷歌系浏览器基本都是默认禁用的,所以调用后没啥效果

```
//将窗口移动到屏幕左上角
window.moveTo(0, 0);
//将窗向下移动100 像素
window.moveBy(0, 100);
//将窗口移动到(200,300)
window.moveTo(200, 300);
//将窗口向左移动50 像素
```

浏览器窗口之打开窗口window.open()

```
window.open(url, target, attr, boolean);
```

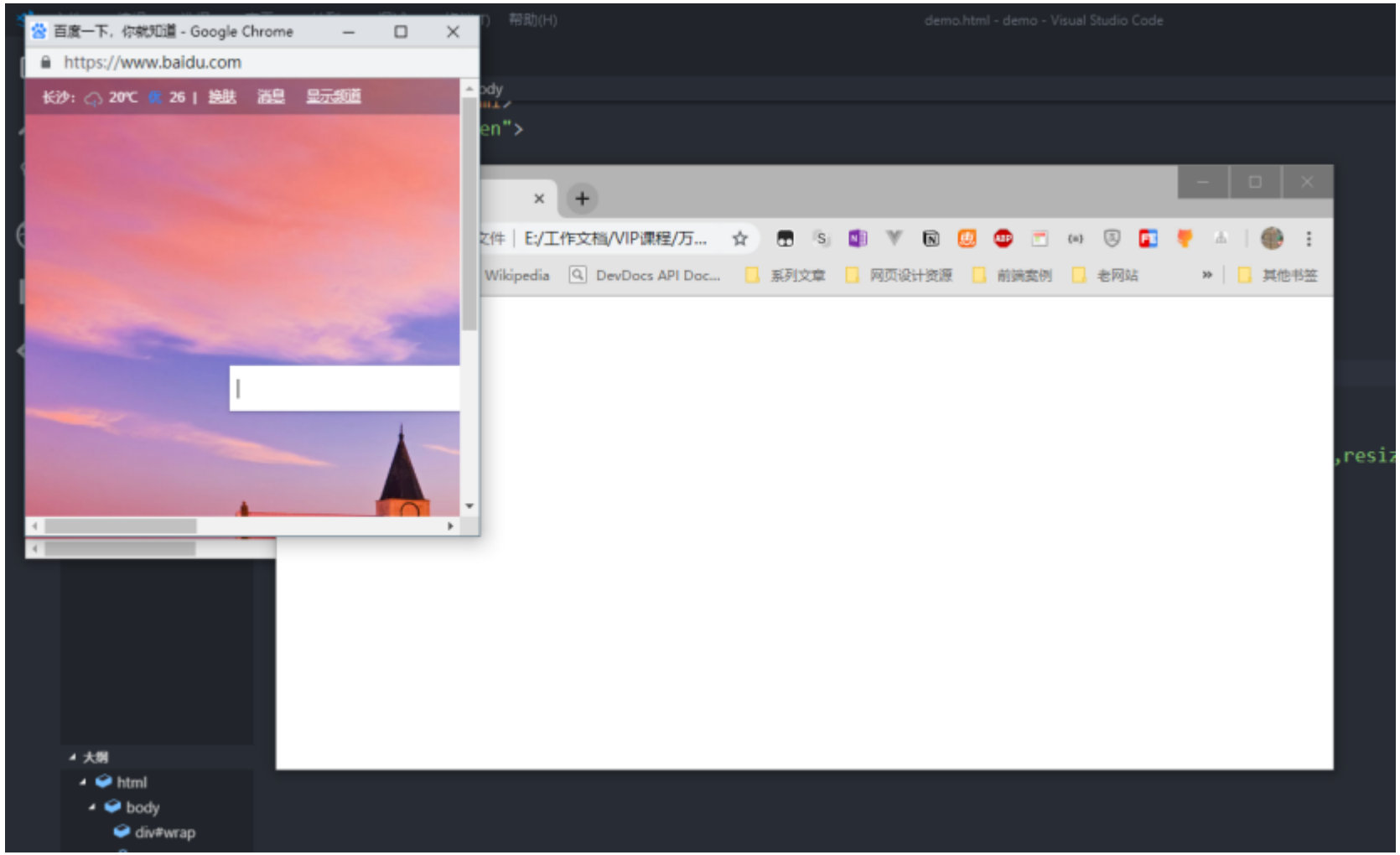
参数有四个:

1. 第一个是url， 你想要打开的网页的网址
2. 第二个目标地址,是你想在哪里打开这个这网页，参数可以是：_self 框架元素、parent 框架元素的父元素、top 框架元素的顶级父元素 或blank新网页，或是某个iframe的框架名称，
3. 第三个新窗口特性(该特性部分浏览器的安全控制规则不让使用，具体特性可以间附表)
4. 第四个参数表

设　置	值	说　明
fullscreen	yes或no	表示浏览器窗口是否最大化。仅限IE
height	数值	表示新窗口的高度。不能小于100
left	数值	表示新窗口的左坐标。不能是负值
location	yes或no	表示是否在浏览器窗口中显示地址栏。不同浏览器的默认值不同。如果设置为no，地址栏可能会隐藏，也可能会被禁用（取决于浏览器）
menubar	yes或no	表示是否在浏览器窗口中显示菜单栏。默认值为no
resizable	yes或no	表示是否可以通过拖动浏览器窗口的边框改变其大小。默认值为no
scrollbars	yes或no	表示如果内容在视口中显示不下，是否允许滚动。默认值为no
status	yes或no	表示是否在浏览器窗口中显示状态栏。默认值为no
toolbar	yes或no	表示是否在浏览器窗口中显示工具栏。默认值为no
top	数值	表示新窗口的上坐标。不能是负值
width	数值	表示新窗口的宽度。不能小于100

新窗口特性表

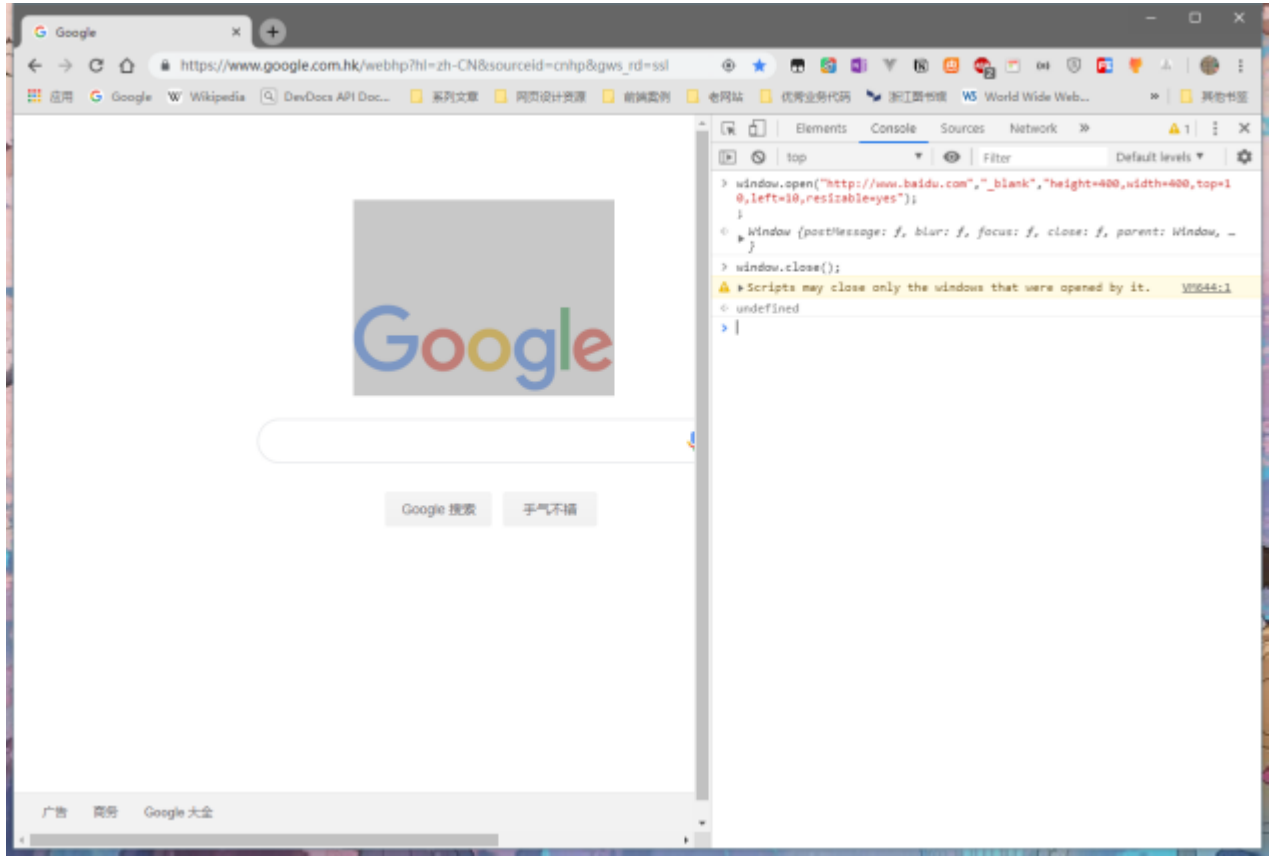
浏览器窗口之打开窗口window.open()

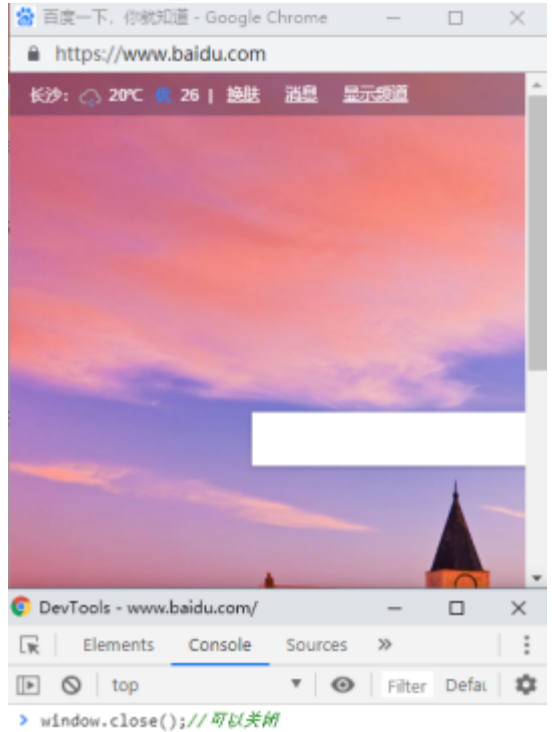


有些浏览器在默认情况下可能不允许我们针对主浏览器窗口调整大小或移动位置，但却允许我们针对通过window.open()创建的窗口调整大小或移动位置

```
window.open("http://www.baidu.com","_blank","height=400,width=400,top=10,left=10,resizable=yes");
```

浏览器窗口之关闭窗口window.close()





牢记,浏览器主窗口目前绝大多数浏览器是不允许用户通过js来关闭的，只有用户自己创造的小窗口才可以用window.close关闭

location对象

loacation对象

location 是最有用的BOM对象之一，它提供了与当前窗口中加载的文档有关的信息，还提供了一些导航功能。

事实上，location 对象是很特别的一个对象，因为它既是window对象的属性，也是document对象的属性；换句话说，window.location和document.location引用的是同一个对象。

属 性 名	例　子	说　明
hash	"#contents"	返回URL中的hash（#号后跟零或多个字符），如果URL中不包含散列，则返回空字符串
host	"www.wrox.com:80"	返回服务器名称和端口号（如果有）
hostname	"www.wrox.com"	返回不带端口号的服务器名称
href	"http://www.wrox.com"	返回当前加载页面的完整URL。而location对象的toString()方法也返回这个值
pathname	"/WileyCDA/"	返回URL中的目录和（或）文件名
port	"8080"	返回URL中指定的端口号。如果URL中不包含端口号，则这个属性返回空字符串
protocol	"http:"	返回页面使用的协议。通常是http:或https:
search	"?q=javascript"	返回URL的查询字符串。这个字符串以问号开头

loacation对象的字符串参数查询

```
function getSearchMsg() {
    if (location.search.length > 0) {
        let qs = location.search, //获取当前网页的搜索参数
            items = qs.split("&"), //将搜索参数的每一组参数都独立成一个数组的数组项目
            args = {}, //存储后期经过信息转码后的, 参数名/参数值
            item = null, //预存中间数据的临时数组
            name, //预存中间数据的参数名称
            value; //预存中间数据的参数值
        for (let i = 0; i < items.length; i++) {
            item = items[i].split("="); //讲每一组参数再次分割
            name = window.decodeURIComponent(item[0]); //获得每一组参数的参数名称, 并将名称进行转码
            value = window.decodeURIComponent(item[1]); //获得每一组参数的参数值, 并将值进行转码
            args[name] = value; //将得到的转码后的参数名称和值存储在对象中
        }
        return args;
    }
}
```

通过上面的属性可以访问到location 对象的大多数信息，但其中访问URL 包含的查询字符串的属性并不方便。尽管location.search 返回从问号到URL 末尾的所有内容，但却没有办法逐个访问其中的每个查询字符串参数。为此，可以像下面这样创建一个函数，用以解析查询字符串，然后返回包含所有参数的一个对象

loacation对象的位置操作

```
//将当前网页的地址跳转到百度
location.href = "http://www.baidu.com";
//假设初始URL 为http://www.wrox.com/WileyCDA/
//将URL 修改为"http://www.wrox.com/WileyCDA/#section1"
location.hash = "#section1";

//将URL 修改为"http://www.wrox.com/WileyCDA/?q=javascript"
location.search = "?q=javascript";

//将URL 修改为"http://www.yahoo.com/WileyCDA/"
location.hostname = "www.yahoo.com";

//将URL 修改为"http://www.yahoo.com/mydir/"
location.pathname = "mydir";

//将URL 修改为"http://www.yahoo.com:8080/WileyCDA/"
location.port = 8080;
```

每次修改location 的属性（hash 除外），页面都会以新URL 重新加载。

history对象

history对象

history 对象保存着用户上网的历史记录，从窗口被打开的那一刻算起。因为history是window对象的属性，因此每个浏览器窗口、每个标签页乃至每个框架，都有自己的history对象与特定的window 对象关联。出于安全方面的考虑，开发人员无法得知用户浏览过的URL(可以想象一下如果允许的话或发生什么事情)。

不过，借由用户访问过的页面列表，同样可以在不知道实际URL 的情况下实现后退和前进

```
//后退一页
history.go(-1);
//前进一页
history.go(1);
//前进两页
history.go(2);
```

也可以给go()方法传递一个字符串参数，此时浏览器会跳转到历史记录中包含该字符串的第一个位置—可能后退，也可能前进，具体要看哪个位置最近。如果历史记录中不包含该字符串，那么这个方法什么也不做，例如：

```
//跳转到最近的wrox.com 页面
history.go("wrox.com");
//跳转到最近的nczonline.net 页面
history.go("nczonline.net");
```

另外，还可以使用两个简写方法back()和forward()来代替go()。顾名思义，这两个方法可以模仿浏览器的“后退”和“前进”按钮

事件冒泡和事件捕捉

事件冒泡

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）。

```
<!DOCTYPE html>
<html>

<head>
        <title>Event Bubbling Example</title>
</head>

<body>
        <div id="myDiv">Click Me</div>
</body>

</html>
```

如果你单击了页面中的 元素，那么这个click 事件会按照如下顺序传播：

(1) 被点击

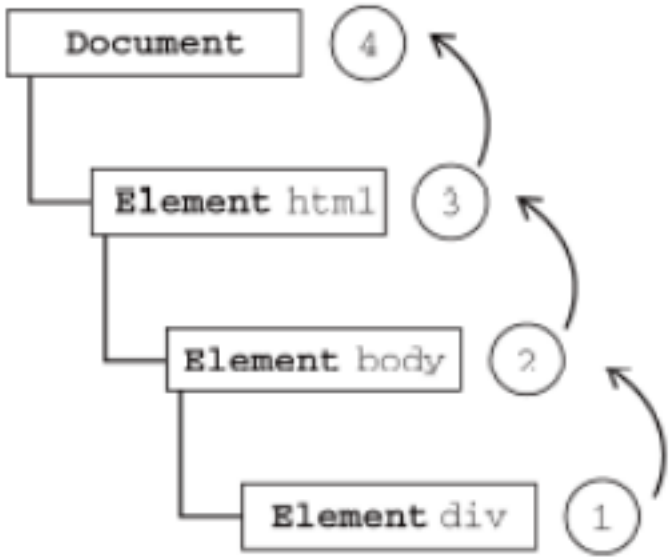
(2) <body>被点击

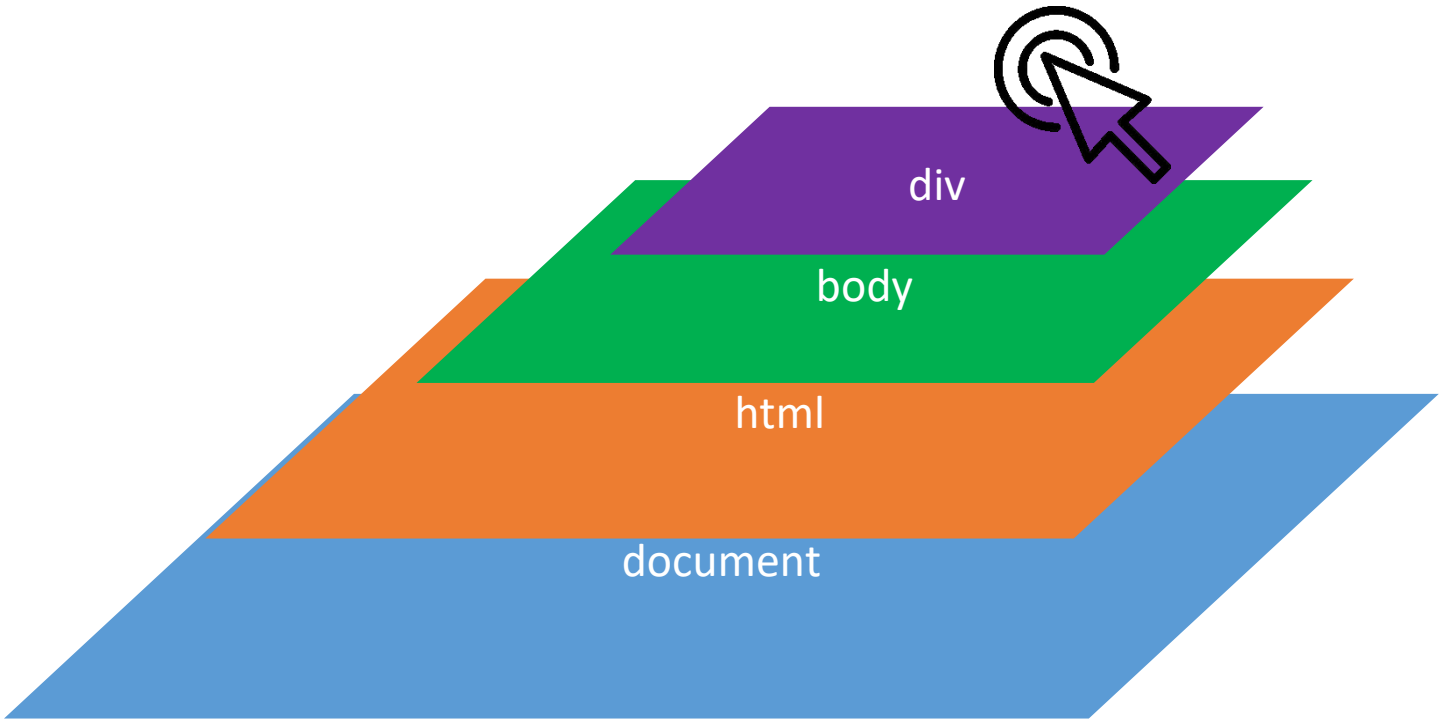
(3) <html>被点击

(4) document被点击

也就是说，click 事件首先在 元素上发生，而这个元素就是我们单击的元素。然后，click事件沿DOM树向上传播，在每一级节点上都会发生，直至传播到document对象

事件冒泡





举个例子：

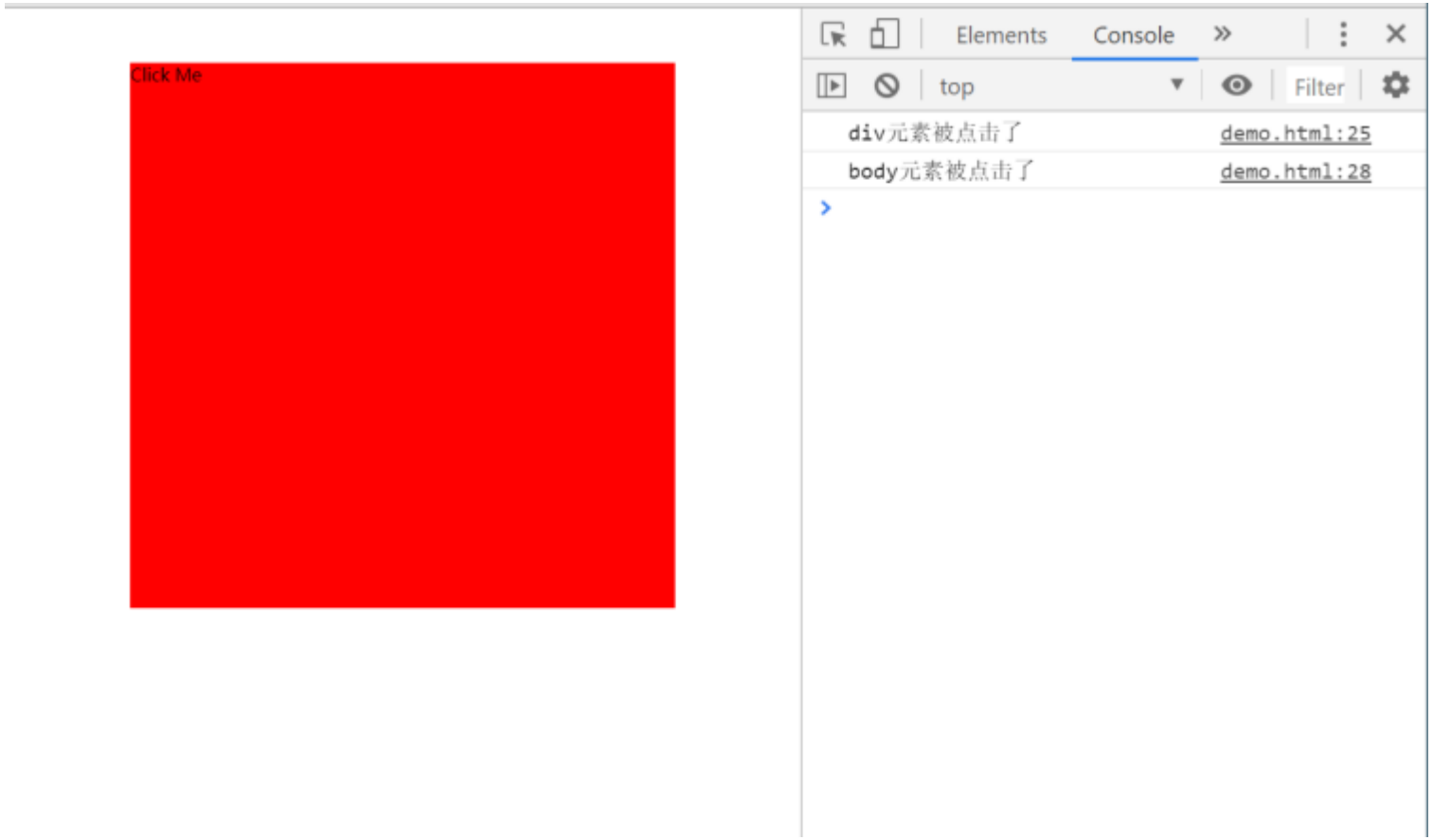
有人戳了一下你的鼻子

那么相当于你的脸被戳了

那么相当于你的头被戳

那么相当于你被戳了

事件冒泡




```html
<head>
    <title>Event Bubbling Example</title>
    <style>
        body{
            margin: 0;
        }
        #myDiv{
            width: 500px;
            height: 500px;
            background-color: red;
            margin: 50px auto;
        }
    </style>
</head>

<body>
    <div id="myDiv">Click Me</div>
    <script>
        let divEle=document.querySelector("#myDiv");
        let bodyEle=document.querySelector("body");
        divEle.onclick=function(){
            console.log("div元素被点击了");
        }
        bodyEle.onclick=function(){
            console.log("body元素被点击了");
        }
    </script>
</body>
```

只要点击到了div元素， 那么div和body元素绑定的事件都会起作用

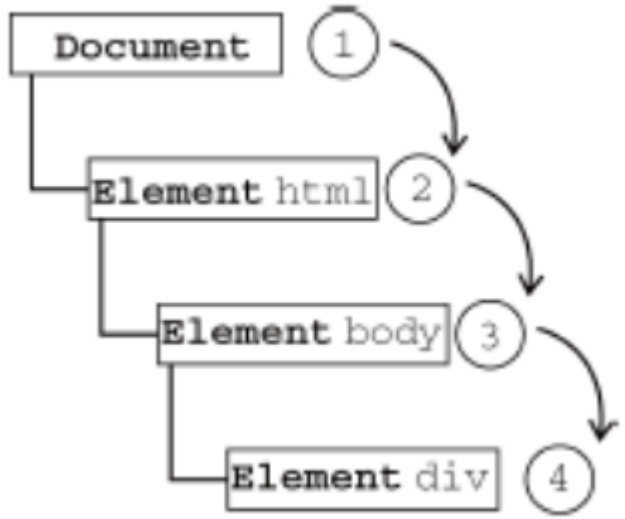
事件捕获

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。

仍以前面的HTML 页面作为演示事件捕获的例子，那么单击 元素就会以下列顺序触发click 事件。

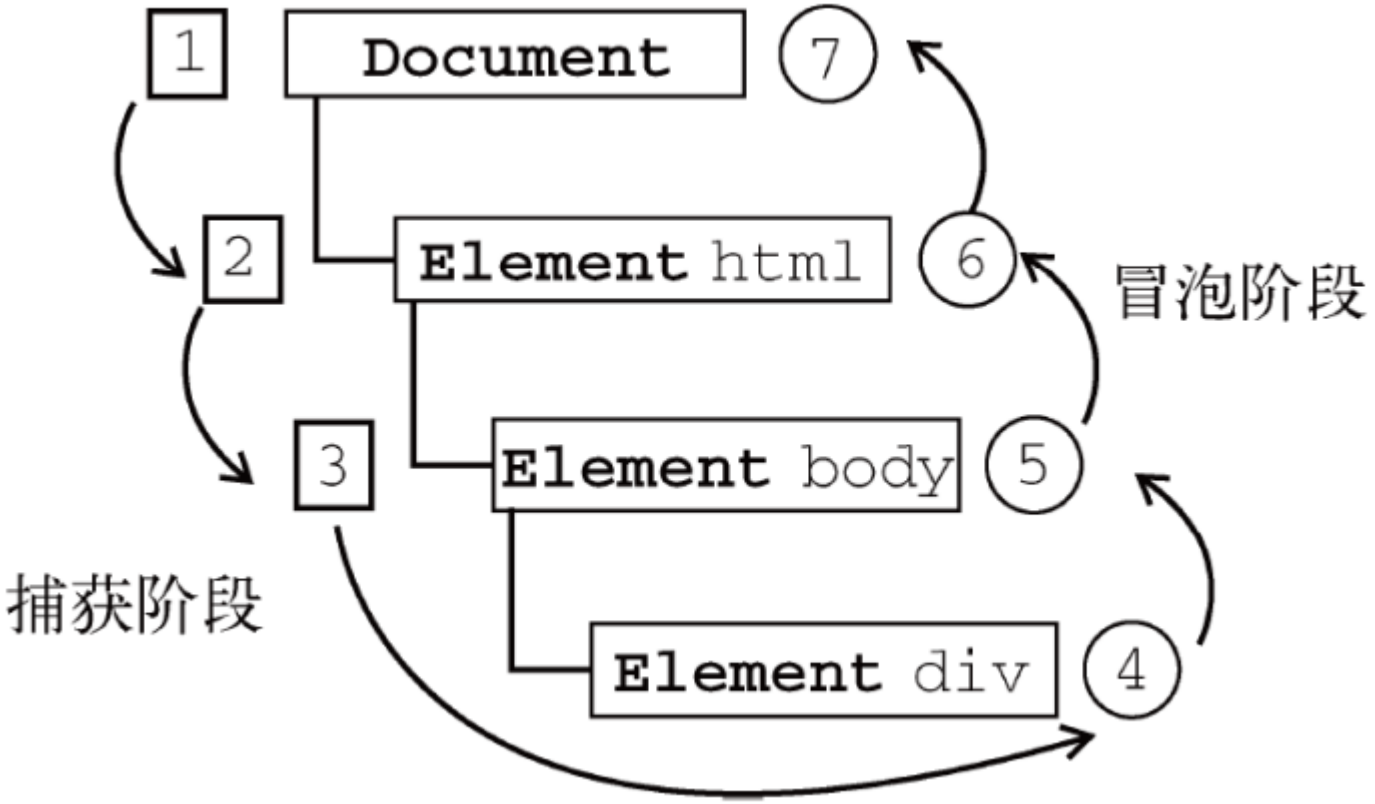
(1) document
(2) <html>
(3) <body>
(4)

在事件捕获过程中，document 对象首先接收到click 事件，然后事件沿DOM 树依次向下，一直传播到事件的实际目标，即 元素



DOM事件流

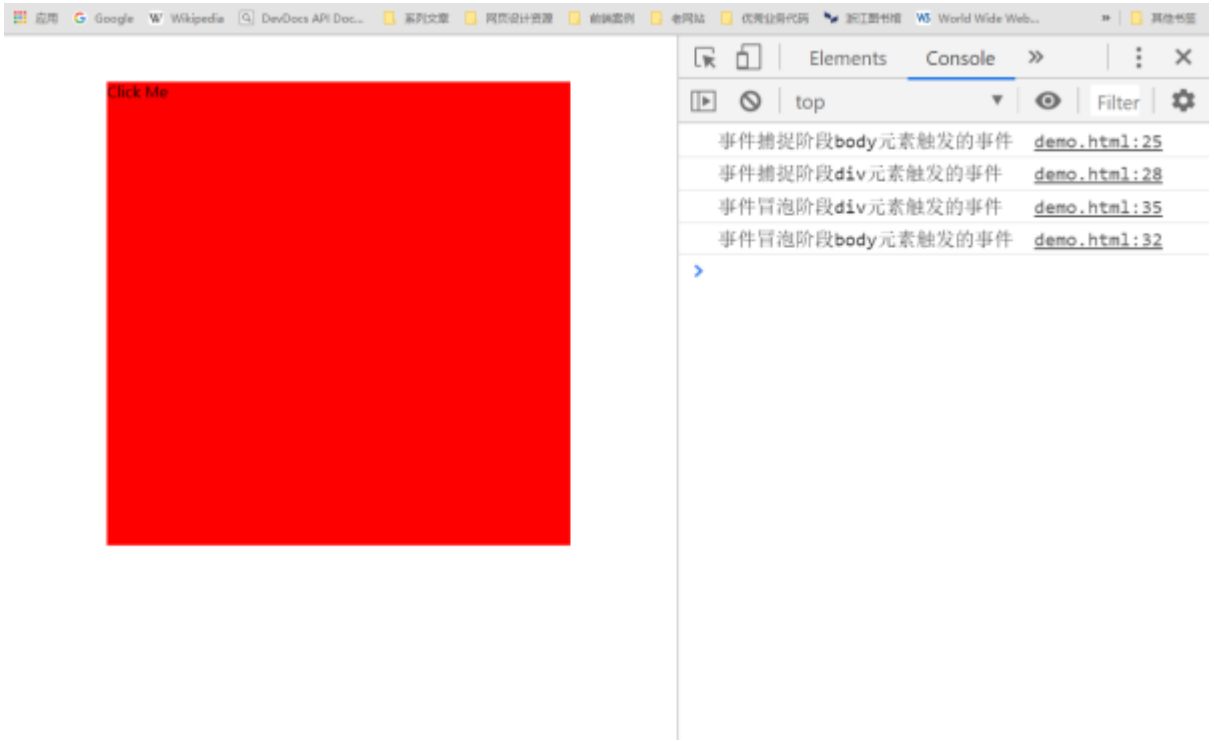
DOM2级事件”规定的事件流包括三个阶段：事件捕获阶段、处于目标阶段和事件冒泡阶段。
首先发生的是事件捕获，为截获事件提供了机会。然后是实际的目标接收到事件。最后一个阶段是冒泡阶段，可以在这个阶段对事件做出响应。以前面简单的HTML 页面为例，单击 元素会按照下图所示顺序触发事件。



事件监听方法.addEventListrner()

```
element.addEventListener(event, function(){}, boolean);
```

- 第一个参数是事件的类型（如 click,mousedown,mouseup,mousemove等等）
- 第二个参数是事件触发后调用的函数
- 第三个参数是个布尔值。默认是false（冒泡阶段执行函数）true(捕获阶段产生)




```html
<head>
    <title>Event Bubbling Example</title>
    <style>
        body{
            margin: 0;
        }
        #myDiv{
            width: 500px;
            height: 500px;
            background-color: red;
            margin: 50px auto;
        }
    </style>
</head>

<body>
    <div id="myDiv">Click Me</div>
    <script>
        let divEle=document.querySelector("#myDiv");
        let bodyEle=document.querySelector("body");
        bodyEle.addEventListener("click",function(){
            console.log("事件捕捉阶段body元素触发的事件")
        },true)
        divEle.addEventListener("click",function(){
            console.log("事件捕捉阶段div元素触发的事件")
        },true)

        bodyEle.addEventListener("click",function(){
            console.log("事件冒泡阶段body元素触发的事件")
        },false)
        divEle.addEventListener("click",function(){
            console.log("事件冒泡阶段div元素触发的事件")
        },false)
    </script>
</body>
```