中山大學
SUN YAT-SEN UNIVERSITY

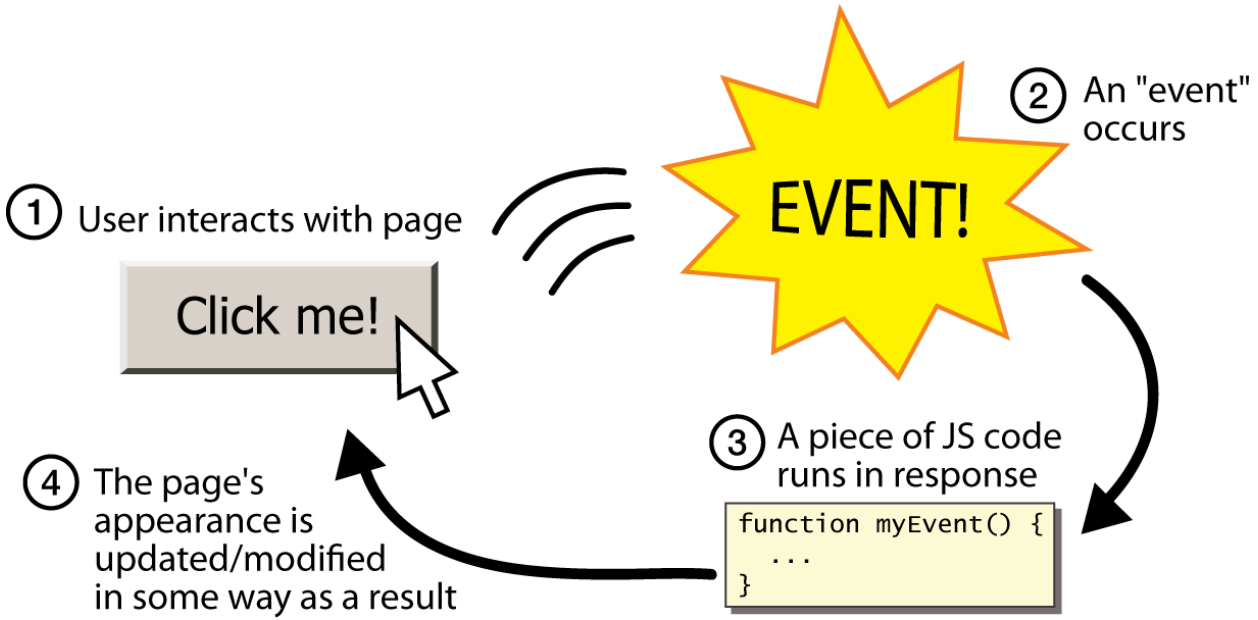
Lecture 06.
More JavaScript & DOM Basics

Modern Web Programming
(http://my.ss.sysu.edu.cn/wiki/display/WEB/ **supported by Deep Focus**)

School of Data and Computer Science, Sun Yat-sen University

Part I DOM

Event-driven programming



- you are used to programs start with a main method (or implicit main like in PHP)
- some programs instead wait for user actions called events and respond to them
- **event-driven programming**: writing programs driven by user events

<button>

the canonical clickable UI control (inline)

```
<button>Click me!</button>
```
HTML

Click me!

output

- button's text appears inside tag; can also contain images
- To make a responsive button or other UI control:
  - choose the control (e.g. button) and event (e.g. mouse click) of interest
  - write a JavaScript function to run when the event occurs, that is **event handler**
  - attach the function to the event on the control

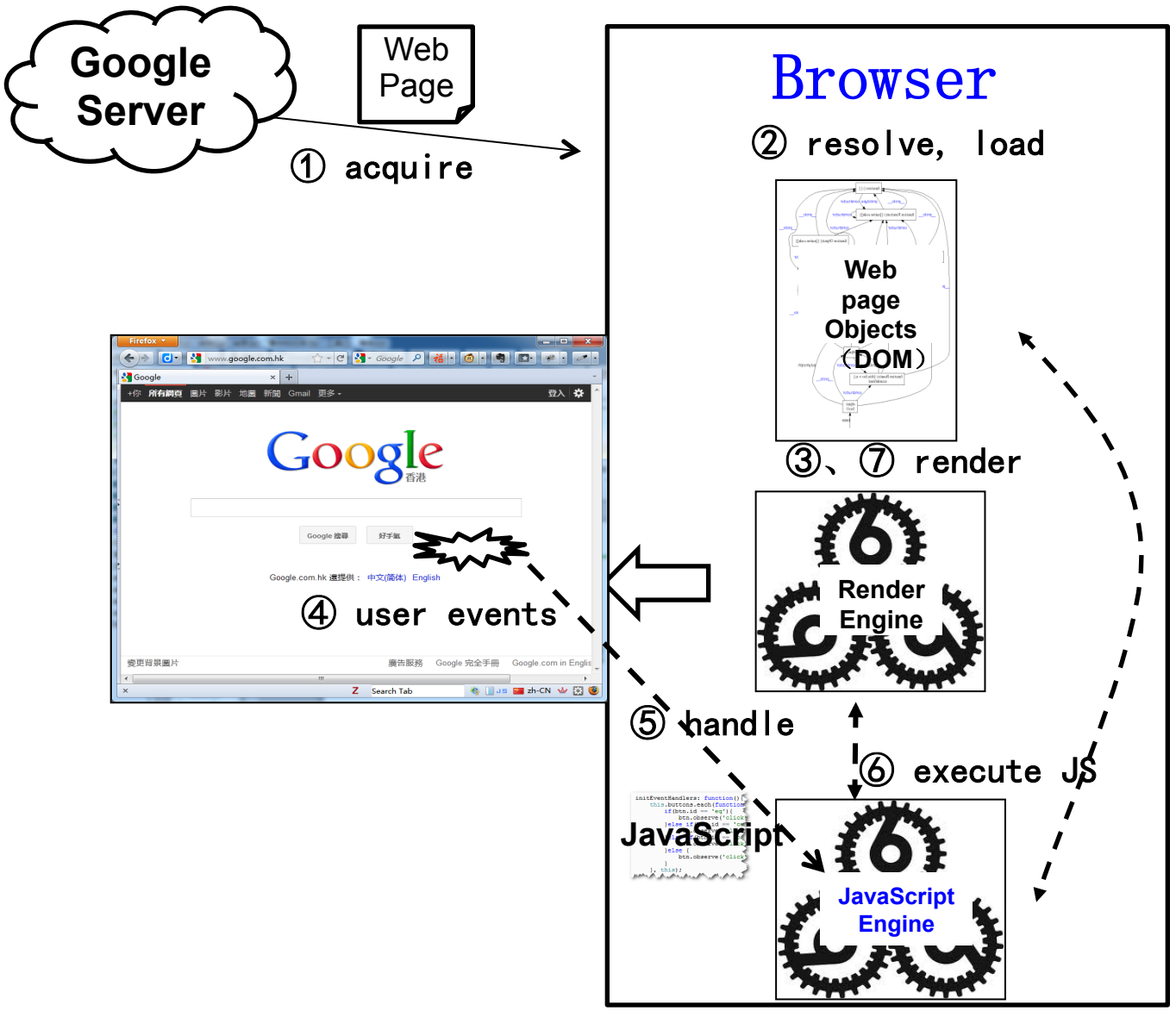
Manipulate Web page objects, which, how?

- the most important task of a event handler is to manipulate objects represented Web page.

Outline

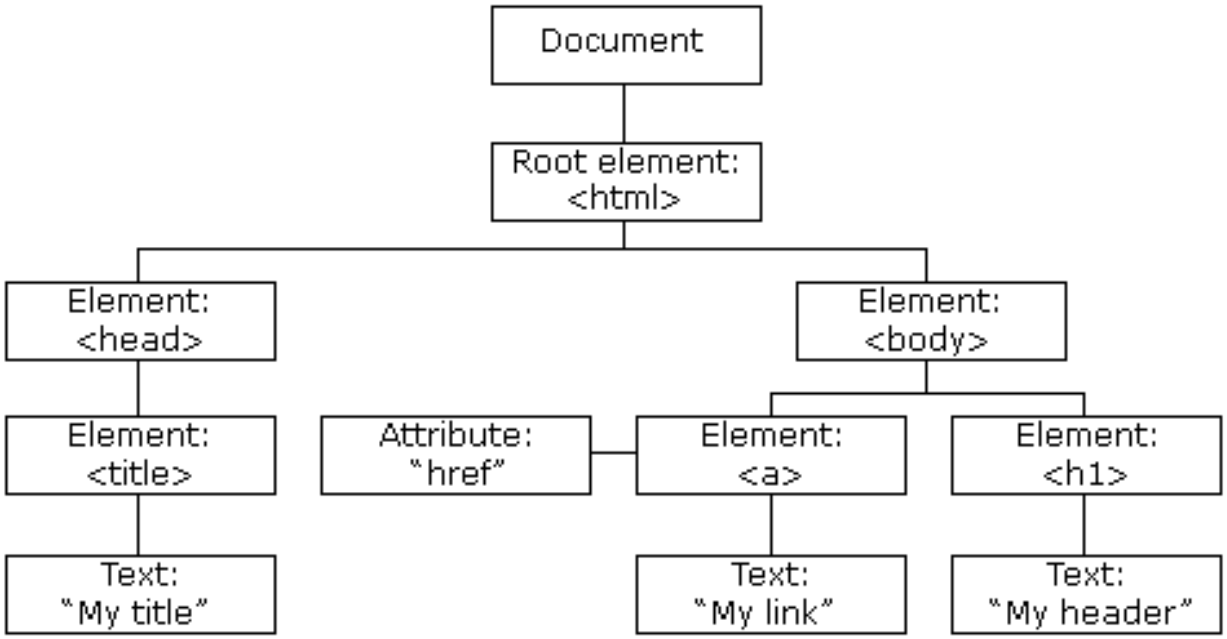
- Event-driven JavaScript
- **DOM basic**
- DOM element
- DOM tree
- DOM & BOM Object

Working mechanism of browsers



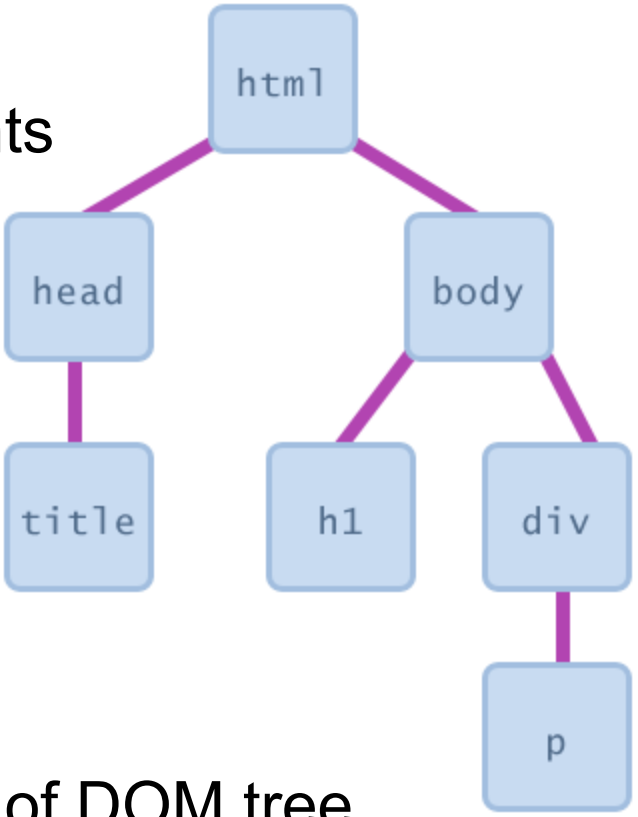
W3C DOM

- The **Document Object Model** is a **platform- and language-neutral interface** that will allow programs and scripts to **dynamically access and update** the **content, structure and style** of documents. The document can be further processed and the **results** of that processing can be **incorporated back into the presented page.**



Document Object Model (DOM)

a set of JavaScript objects that represent each element on the page



- most JS code manipulates elements on an HTML page
- we can examine elements' state
  - e.g. see whether a box is checked
- we can change state
  - e.g. insert some new text into a div
- we can change styles
  - e.g. make a paragraph red
- we can even change the structure of DOM tree
  - e.g. append a new paragraph

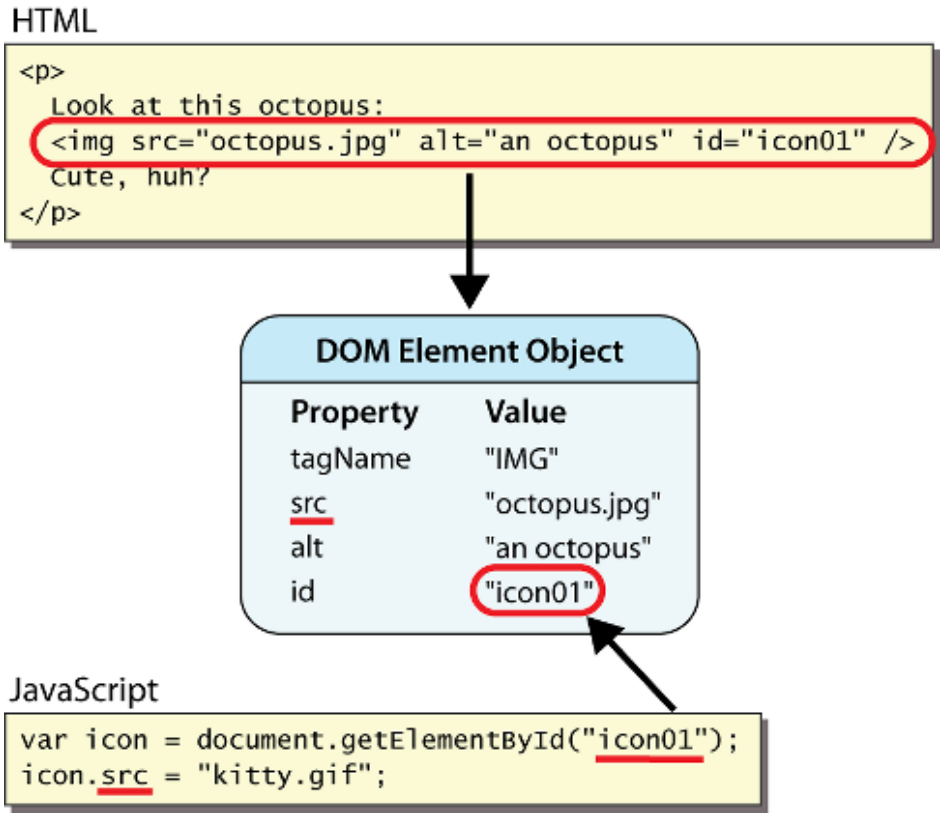
Outline

- Event-driven JavaScript
- DOM basic
- **DOM element**
- DOM tree
- DOM & BOM Object

DOM element

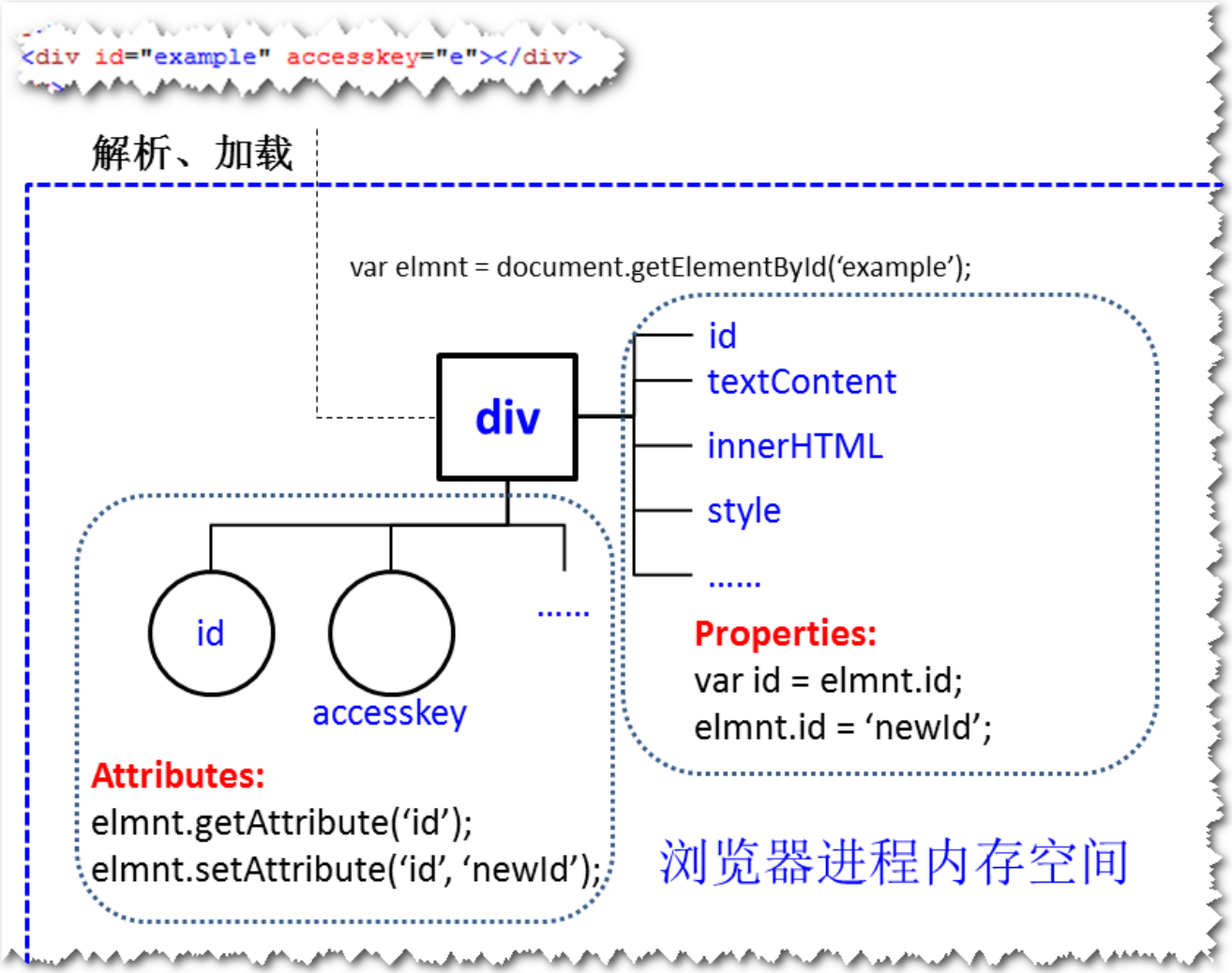
- every element on the page has a corresponding DOM object

- access/modify the properties of the DOM object with *objectName.attributeName*



- in fact, browsers evaluate a Web page into corresponding DOM objects at runtime

DOM Element



解析、加载
var elmnt = document.getElementById('example');
div
id
textContent
innerHTML
style
.....
Properties:
var id = elmnt.id;
elmnt.id = 'newId';
id
accesskey
.....
Attributes:
elmnt.getAttribute('id');
elmnt.setAttribute('id', 'newId');
浏览器进程内存空间

Manipulate text of DOM elements

- innerHTML vs. textContent

源代码 8-1 innerHTML 与 textContent 对比

```
------------------------ HTML ------------------------------
……
<div id="example" accesskey="e">
    <p>文本内容1</p><p>文本内容2</p>
</div>
<br/>
<button onclick="changTextContent();">改变textContent</button>
<br />
<button onclick="changeInnerHTML();">改变innerHTML</button>
……
---------------------- JavaScript --------------------------
window.onload = function(){
  var div_example = document.getElementById("example");
  alert(div_example.textContent); //文本内容1文本内容2
  alert(div_example.innerHTML); //<p>文本内容1</p><p>文本内容2</p>

  changTextContent = function(){
    div_example.textContent = "<p>新文本内容</p>"
  }

  changeInnerHTML = function(){
    div_example.innerHTML = "<p>新文本内容</p>"
  }
}
```

Improved calculator

源代码 8-2 改进计算器错误信息显示方式

```
---------------------------- HTML ----------
......
<div id = "calculator">
    <div id = "error"></div>
    <div id="output_field">
......
---------------------------- CSS -----------
......
div#error{color:red;}
......
---------------------- JavaScript ----------------------
......
function showError(message){
    document.getElementById("error").textContent = message;
}
......
```

Using value

value 属性与网页内容：对于一些由 value 属性定义其显示内容的 HTML 元素，例如：**文本框，各种按钮等，不能使用 textContent 和 innerHTML**，而应该通过其 DOM 元素的 value property 来获取/改变其显示内容。例如，计算器例程中的表达式和计算结果的显示，就是通过给 output_field 文本框的 value 属性赋值得以实现的，参见源代码 8-3。

源代码 8-3 文本框使用 value property 示例

```
------------------------- HTML -----------------------------
……
<div id = "calculator">
    <div id = "error"></div>
    <div id="output_field">
……
---------------------- JavaScript ------------------------
……
function updateOutput(data){
  document.getElementById("output").value = data;
}
……
```

Adjust style

- 表 8-1 部分 DOM style property 名与 CSS 属性名对照表

DOM style property 名	CSS 属性名	值示例
backgroundColor	background-color	“#ff00dd”
border	border	“1px solid red”
color	color	“red”
cssFloat	float	“left”
fontWeight	font-weight	“bold”
fontSize	font-size	“12pt”
zIndex	z-index	“456”

DOM style property 名与 CSS 属性名不同： DOM style property 的名称使用驼峰式大小写（CamelCase），而 CSS 属性名使用短横线分隔多个英文单词。此外，float 是 JavaScript 的保留字，因此 CSS 的 float 属性，在 DOM style property 里面用 cssFloat 表示。

Adjust style

源代码 8-4 使用 DOM style property 改变网页外观示例

HTML

```
……
<div id="poem">
    <h1>将进酒</h1>
    <h2>李白</h2>
    <p>
        君不见黄河之水天上来，奔流到海不复还。 <br/>
        君不见高堂明镜悲白发，朝如青丝暮成雪。 <br/>
        ……<br/>
        与尔同消万古愁。
    </p>
    <button id="changeButton">加下划线</button>
</div>
……
```

JavaScript

```
……
document.getElementById("changeButton").onclick = function(){
    document.getElementById("poem").style.textDecoration =
'underline';
}
……
```

DOM style properties are strings

DOM style properties 是字符串类型： DOM style property 中，类似 fontSize、borderWidth、padding 等属性都是有单位（em、pt、px）的字符串，而不是数值。误将数值直接赋值给这些 properties，是使用中常见的错误，参考源代码 8-5。

- 源代码 8-5 style property 是字符串，错误和正确的赋值方式示例

```
-------------------------- 错误做法 ----------------------------
document.getElementById("poem").style.fontSize = 24;

document.getElementById("poem").style.fontSize += 5;

-------------------------- 正确做法 ----------------------------
document.getElementById("poem").style.fontSize = "24px";

var poem = document.getElementById("poem");
var fontSize = parseInt(poem.style.fontSize);
poem.style = (fontSize + 5) + "px"; // 假定使用px作为单位
```

Unobtrusive CSS

源代码 8-6 非侵入式 CSS 与使用 DOM 元素的示例 1

```
------------------------- CSS  -----------------------------
.changed { text-decoration: underline;}
---------------------- JavaScript  --------------------------
……
document.getElementById("changeButton").onclick = function(){
    document.getElementById('poem').className = 'changed';
}……
```

源代码 8-7 非侵入式 CSS 与使用 DOM 元素的示例 2

```
------------------------- CSS  -----------------------------
.changed { text-decoration: underline;}
---------------------- JavaScript  --------------------------
……
document.getElementById("changeButton").onclick = function(){
    var poem = document.getElementById('poem');
    var className = poem.className;
    if(className.indexOf('changed') < 0) poem.className += '
changed';
}……
```

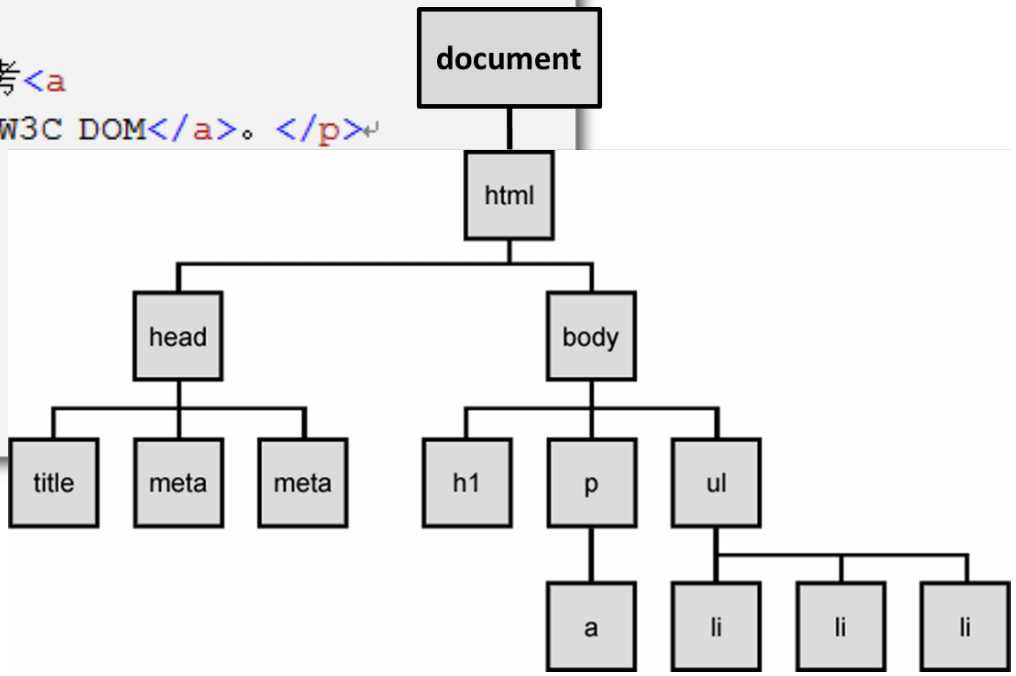
Outline

- Event-driven JavaScript
- DOM basic
- DOM element
- **DOM tree**
- DOM & BOM Object

DOM tree

源代码 8-8 HTML 文档运行时加载成为 DOM 树

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>DOM树</title>
</head>
<body>
    <h1>DOM树</h1>
    <p>DOM树是网页内存对象模型，参考<a
href="http://www.w3.org/DOM/">W3C DOM</a>。</p>
    <ul>
        <li>DOM节点</li>
        <li>导航与遍历</li>
        <li>选择元素</li>
    </ul>
</body>
</html>
```

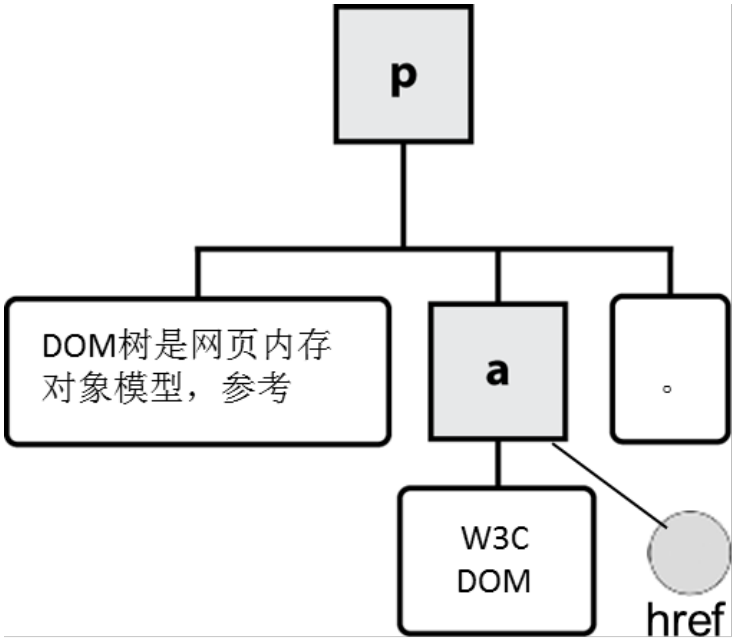


DOM Node

- Node types:
  - document node
  - element node
  - text node
  - attribute node
  - comment node

源代码 8-9 HTML 文档运行时加载成为 DOM 树（包括属性、文本节点）

```
……
<p>DOM树是网页内存对象模型，参考<a href="http://www.w3.org/DOM/">W3C
DOM</a>。</p>
……
```



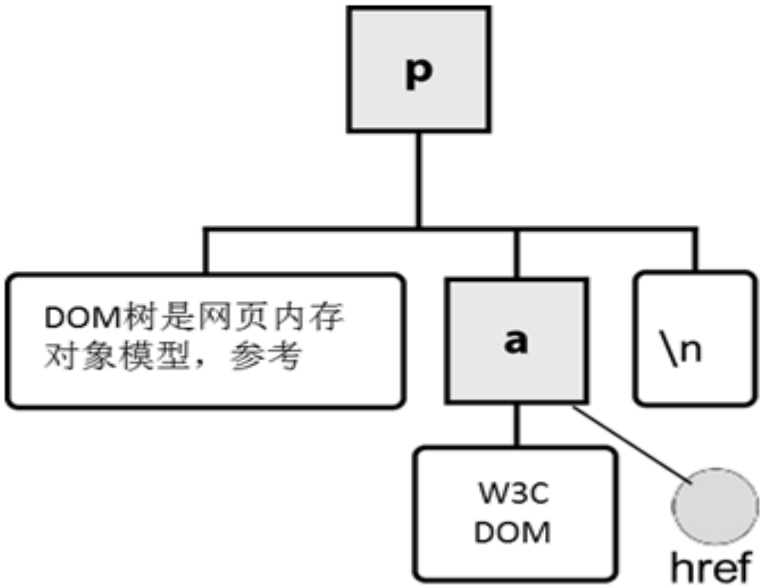
DOM node features

- Document node and element nodes may have child nodes
- Text nodes and attributes nodes can not

Blank spaces in a HTML file turn into Text

源代码 8-10 HTML 文档运行时加载成为 DOM 树（空白内容保留在文本节点内）

```
……
<p>DOM树是网页内存对象模型，参考<a href="http://www.w3.org/DOM/">W3C
DOM</a>
</p>
……
```



DOM node properties

表 8-2 DOM 节点 properties

property 名	描述
nodeName	节点名（只读），元素节点名为 HTML 标签名（全大写），属性节点名为 HTML 属性名，文本节点名始终为"#text"，文档节点名为"#document"
nodeValue	节点值，元素节点值为 null，属性节点值为 HTML 属性值，文本节点值为文本内容
nodeType	节点类型（只读），元素节点为 1，属性节点为 2，文本节点为 3，文档节点为 9，注释节点为 8
parentNode	父节点
previousSibling	前一个兄弟节点
nextSibling	后一个兄弟节点
firstChild	第一个子节点
lastChild	最后一个子节点
childNodes	所有子节点的数

源代码 8-11 DOM 节点 properties 示例

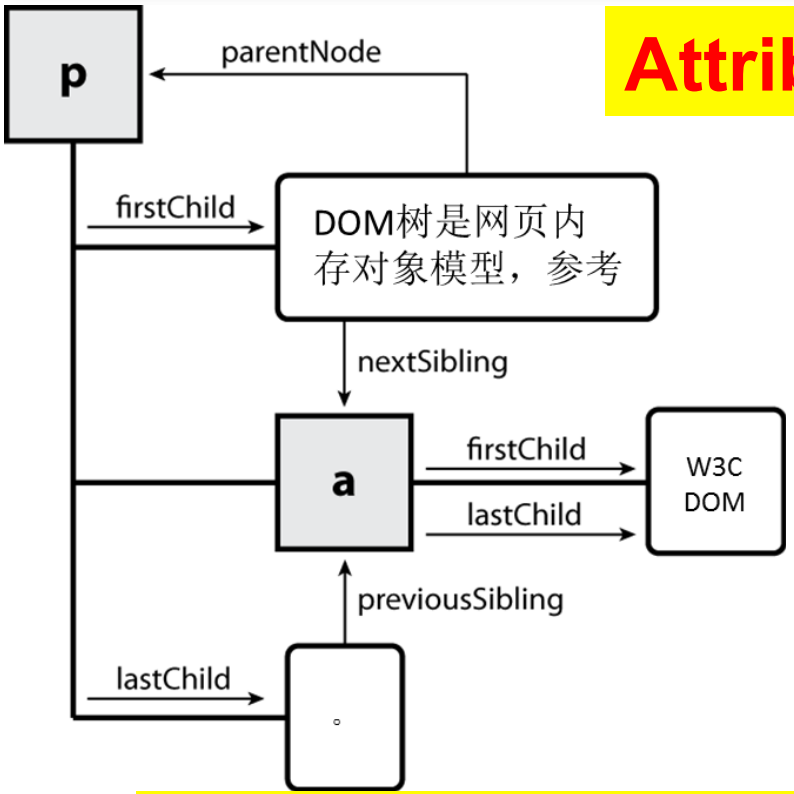
```
------------------------- HTML ----------------
<li id="dom-node">DOM节点</li>
……
---------------------- JavaScript -------------
var node = document.getElementById("dom-node");
alert(node.nodeName);     // LI
alert(node.nodeType);     // 1
alert(node.nodeValue);    // undefined

var child = node.firstChild;
alert(child.nodeName);    // #text
alert(child.nodeType);    // 3
alert(child.nodeValue);   // DOM节点
```

DOM traversal

源代码 8-9 HTML 文档运行时加载成为 DOM 树（包括属性、文本节点）

```
……
<p>DOM树是网页内存对象模型，参考<a href="http://www.w3.org/DOM/">W3C DOM</a>。</p>
……
```



Attributes are not included!

源代码 8-12 使用 DOM 节点 properties 进行 DOM 树遍历与导航示例

```
// 得到DOM p节点，参考8.4.4节
var p = document.getElementsByTagName('p')[0];

var textNode1 = p.firstChild.nextSibling.firstChild;
var textNode2 = p.childNodes[1].childNodes[0];
alert(textNode1.nodeValue);          // W3C DOM
alert(textNode2.nodeValue);          // W3C DOM
alert(textNode1 === textNode2);      // true

alert(textNode1.firstChild);             // null
alert(textNode1.lastChild);              // null
alert(textNode1.nextSibling);            // null
alert(textNode1.previousSibling);        // null
alert(textNode1.childNodes.length);      // 0
```

Be careful with black spaces, they are text nodes too!

Selecting DOM elements

源代码 8-14 使用 document 对象的方法选择 DOM 元素示例

```
------------------------- HTML ------------------------------
……
<body>
<div>
    <h1>将进酒</h1>
    <h2>李白</h2>
    <p>
        君不见黄河之水天上来，奔流到海
        ……
    </p>
    <form action="#">
        请选择文本颜色：
        <label><input type="radio
</label>
        <label><input type="radio
</label>
        <label><input type="radio
黑</label><br />
    </form>
……
---------------------- JavaScri
window.onload = function(){
  var radios = getAllRadios();
  for (var i = radios.length - 1
    radios[i].onclick = changeColor;
  }
}
                使用
```

将进酒

```
--------------------使用properties ------------------------
function getAllRadios(){
}
    // document.body就是HTML body元素，参考图 8-4和8.5.2a)节
    // 难以理解childNodes[1].childNodes[7]到底是什么
    var form = document.body.childNodes[1].childNodes[7];
    var radios = [];
    // 繁复的筛选
    var nodes = form.childNodes;
    for(var i = 0; i < nodes.length; i++) {
        if(nodes[i].nodeName != 'LABEL') continue;
        radions.push(nodes[i].firstChild);
    }
    return radios;
}

function changeColor(event){
    // 难以理解childNodes[1].childNodes[5]到底是什么
    var paragraph = document.body.childNodes[1].childNodes[5];
    // event.target指向用户点击的元素，参考8.6.1节
    paragraph.style.color = event.target.getAttribute('value');
}
```

Selecting DOM elements

表 8-3 document 对象的 DOM 元素选择方法

方法名	描述
getElementById	返回 id property/attribute 为给定值的 DOM 元素
getElementsByTagName	返回给定标签名对应的所有 DOM 元素
getElmentsByName	返回具有给定 name attribute 的所有 DOM 元素

还。

```
    </p>
    <form action
        请选择文本
        <label><
</label>
        <label><
</label>
        <label><
黑</label><br />
    </form>
```

```
---------------------- 使用document方法 ----------------------
function getAllRadios(){
    return document.getElementsByName("color");
}

function changeColor(event){
    var paragraph = document.getElementsByTagName('p')[0];
    paragraph.style.color = event.target.getAttribute('value');
}
```

```
window.onload = function(){
  var radios = getAllRadios();
  for (var i = radios.length - 1; i >= 0; i--) {
    radios[i].onclick = changeColor;
  }
}
```

Altering structure

- create nodes

源代码 8-15 创建新 DOM 节点

```
var newParagraph = document.createElement("p");
alert(newParagraph.nodeName);          // P
var newText = document.createTextNode("君不见高堂明镜悲白发");
alert(newText.nodeName);               // #text
alert(newText.nodeValue);              // 君不见高堂明镜悲白发
newParagraph.setAttribute("title", "新段落");
alert(newParagraph.getAttribute("title")); // 新段落
```

Can't be seen until it is appended/inserted into current DOM tree!

Altering structure

表 8-4 document 对象的 DOM 元素附加、插入、删除、替换方法

方法名	描述
appendChild(node)	将 node 附加成为当前节点的子节点，成为其 lastChild
insertBefore(new, old)	将 new 插入成为当前节点的子节点，为 old 节点的 previousSilbling
removeChild(node)	从当前节点的子节点当中删除 node
replaceChild(new, old)	将当前节点的 old 子节点，替换为 new

Altering structure

源代码 8-16 使用 document 对象的方法添加/删除 DOM 节点示例

HTML

```
.....
<div>
    <h1>将进酒</h1>
    <h2>李白</h2>
    <p>
        君不见黄河之水天上来，奔流到海不复还。
    </p>
</div>
.....
.newPara
```

JavaScript

```
window.onload = function(){
  newParagraph = document.createElement("p"); // 新诗句段落，用全局变量存储
  var newText = document.createTextNode("君不见高堂明镜悲白发，朝如青丝暮成雪");
  newParagraph.className = "newParagraph";
  newParagraph.appendChild(newText);
  document.getElementsByTagName('button')[0].onclick = toggleNewParagraph;
}

function toggleNewParagraph(){
  var poem = document.getElementsByTagName('div')[0];
    var paragraphs = document.getElementsByTagName('p');
    if(paragraphs.length > 1){
        poem.removeChild(newParagraph);
    }else{
      var oldParagraph = document.getElementsByTagName('p')[0];
      poem.insertBefore(newParagraph, oldParagraph.nextSibling);
    }
}
```

Outline

- Event-driven JavaScript
- DOM basic
- DOM element
- DOM tree
- **DOM & BOM Object**

BOM & DOM

- Objects created by browsers, and exposed their JS API
- In fact, browsers expose more than DOM objects

BOM 无标准、有共识：不同于 DOM，迄今为止还没有关于 BOM 的统一标准。不过，由于不同浏览器之间的渊源关系和业界的共识，绝大部分的浏览器都开放了表 8 5 中这些对象，并且普遍有相同（类似）的方法。

BOM

表 8-5 BOM 对象一览表

对象名	描述
window	浏览器用于显示网页的窗口
document	浏览器窗口内当前的网页，DOM 树的根（**即是 BOM 成员，又是 DOM 成员**）
location	当前网页的 URL
navigator	浏览器本身
screen	浏览器当前占据的屏幕区域
history	浏览器用户访问历史

window

表 8-6 window 对象常用方法列表

对象名	描述
alert、confirm、prompt	弹出对话框（参考相应章节）
setInterval 、 setTimeout 、 clearInterval、clearTimeout	定时器（参考定时器一节）
open(url, name, options) 、 close()	打开新的浏览器窗口，关闭当前浏览器窗口[10]
print()	打印当前网页
focus()、blur()	使当前浏览器窗口获得焦点、使当前浏览器失去焦点[11]
scrollBy(dx, dy)、scorllTo(x, y)	将浏览器窗口内页面纵向滚动 dx（负值为向上），横向滚动 dy（负值向左）；将浏览器窗口内页面滚动到(x, y)座标

Open new window (tab)

源代码 8-17 使用 winow 对象 open 方法打开浏览器新窗口（标签页）示例

```
------------------------- HTML -----------------------------
.....
        <button>打开新窗口</button>
.....
----------------------- JavaScript -------------------------
window.onload = function(){
  var button = document.getElementsByTagName('button')[0];
  button.onclick = function(){
    window.open(docuemt.URL, "新窗口"); //参考表 8-7
  }
}
```

document properties

表 8-7 document 对象 properties 列表

property 名	描述
cookie	当前网页有效的所有 cookie，以名值对的形式返回
domain	提供当前网页的 Web 服务器之域名
referrer	前一个网页，用户从那里点击链接访问了当前网页
title	当前网页的 title
URL	
anchors	
links	
forms	
images	

源代码 8-18 使用 document 对象 properties 示例

```
window.onload = function(){
  var images = document.images;
  for (var i = 0; i < images.length; i++) {
    images[i].onclick = showName;
  }
}

function showName(event){
    var name = getName(event.target.src);
    alert(name);
}

function getName(url){
    return url.substring(url.lastIndexOf("/") + 1, url.length);
}
```

document methods

表 8-8 document 对象动态写入网页文档方法列表

方法	描述
open()	打开输出流，准备写入文档内容
write(content)	在输出流当中写入网页内容
writeln(content)	在输出流当中写入网页内容，每次一行
close()	关闭当前输出流，将其内容显示到浏览器窗口中

```
<body>
    <div>
        <script>document.writeln("<p>动态写入的内容
</p>");</script>
    </div>
</body>
……
```

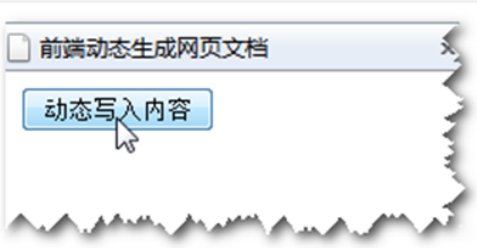

```
<body>
        <div>
                <script>document.writeln("<p>动态写入的内容</p>");</script>
        </div>
</body>
</html>
```

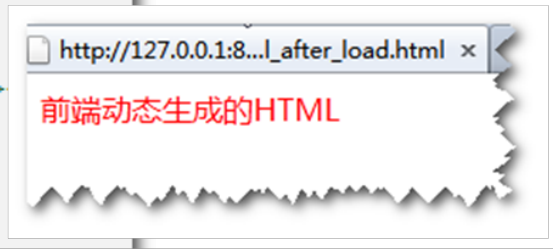
Don’t write a loaded document

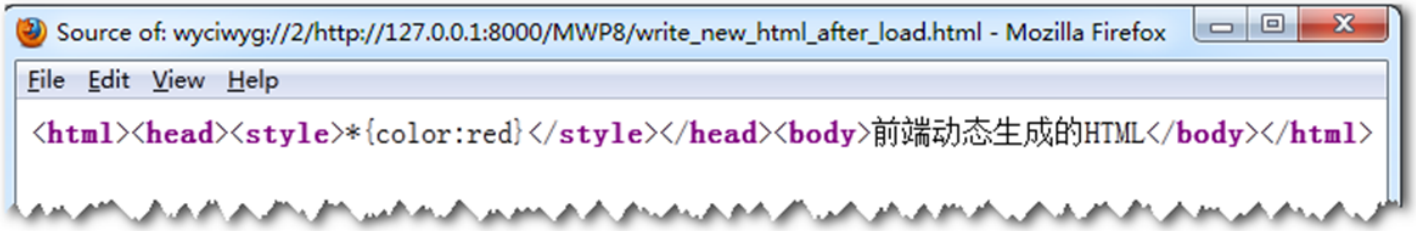
源代码 8-20 使用 document 对象方法前端动态写入网页文档示例（网页加载完成后）

```
---------------------------- HTML ------------------------------
……
<body>
    <div>
        <button>动态写入内容</button>
    </div>
</body>
……
----------------------- JavaScript -------------------------
window.onload = function(){
  document.getElementsByTagName("button")[0].onclick =
writeNewHTML;
}

function writeNewHTML()
  document.open();
  var txt="<html><head>                    le></head>
前端动态生成的HTML</body>
  document.write(txt);
  document.close();
)
```







location

表 8-9 location 对象 properties 列表（参考第一章 URL）

方法	描述
hre	
pro	
hos	
hos	
por	
pat	
has	
sea	

源代码 8-21 使用 screen 对象方法获取用户屏幕信息示例

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>screen对象用法示例</title>
<script>
    function showLocationInfo(){
        document.writeln("protocol: " + location.protocol + " <br />");
        document.writeln("host: " + location.host + " <br />");
        document.writeln("ho
        document.writeln("po
        document.writeln("pa
        document.writeln("ha
        document.writeln("se
    }
```

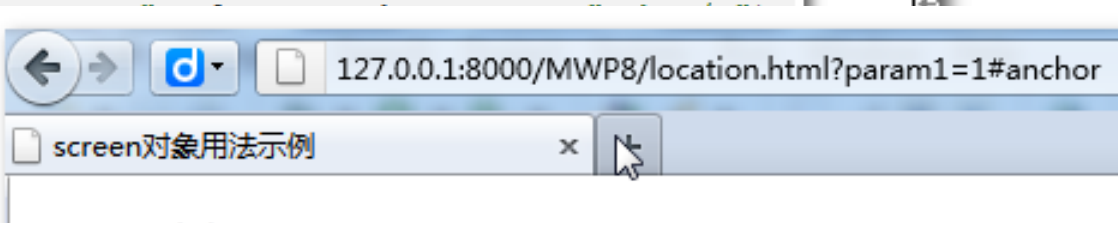


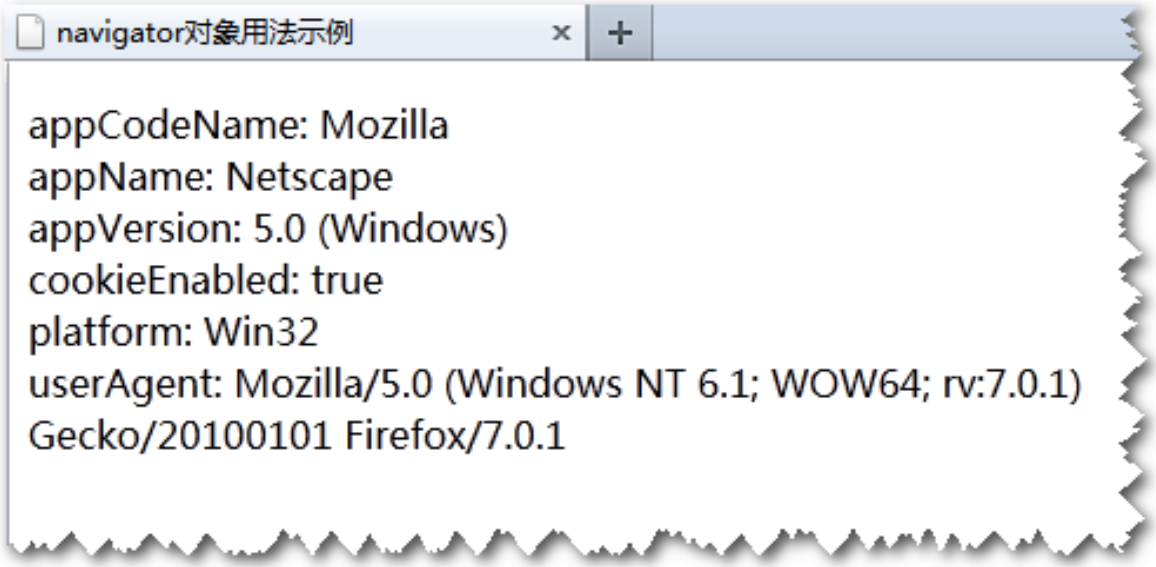
表 8-10 location 对象方法列表

方法	描述
reload()	向服务器重新请求当前网页
assign(url)	在当前窗口（标签页）打开给定的网页（url）
replace(url)	与 assign 相同，在当前窗口（标签页）打开给定的网页（url）

navigator

表 8-11 navigator 对象 properties 列表

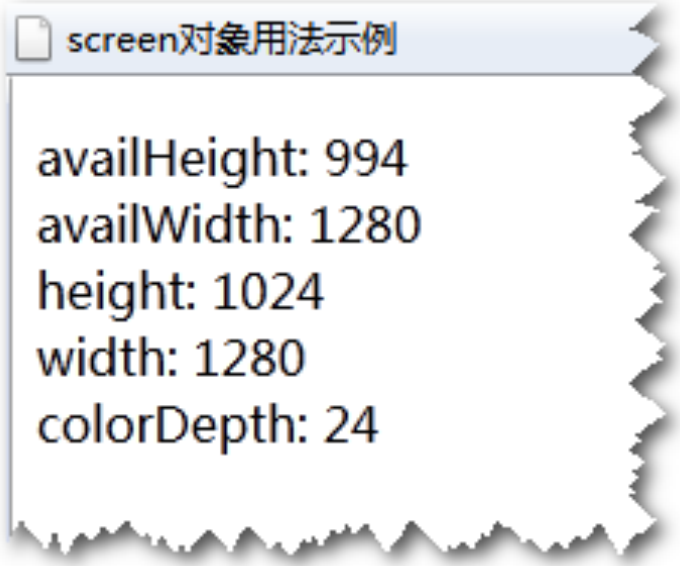
property 名	描述
appCodeName	浏览器的代码名称
appName	浏览器名
appVersion	浏览器版本
cookieEnabled	是否支持 cookie
platform	浏览器是在什么平台下编译的
userAgent	浏览器在和 Web 服务器通信时，HTTP(s)报文头部 user-agent 字段的值



screen

表 8-12 screen 对象 properties 列表

property 名	描述
availHeight	用户屏幕的高度（不包括 Windows 的任务栏）
availWidth	用户屏幕的宽度（不包括 Windows 的任务栏）
height	用户屏幕的整个高度（包括 Windows 的任务栏）
width	用户屏幕的整个宽度（包括 Windows 的任务栏）
colorDepth	用户屏幕的颜色解析度，每个像素点使用多少位数字来表示



history

history.length, urls visited in the current window(tab)

表 8-13 history 对象方法列表

方法	描述
back()	在当前网页的访问历史列表上，返回访问上一个网页
forward()	在当前网页的访问历史列表上，返回访问下一个网页
go(step)	在当前网页的访问历史列表上，返回访问从当前网页起第 step 个网页，step 为正数向前计算，为负数向后倒退

Exercises

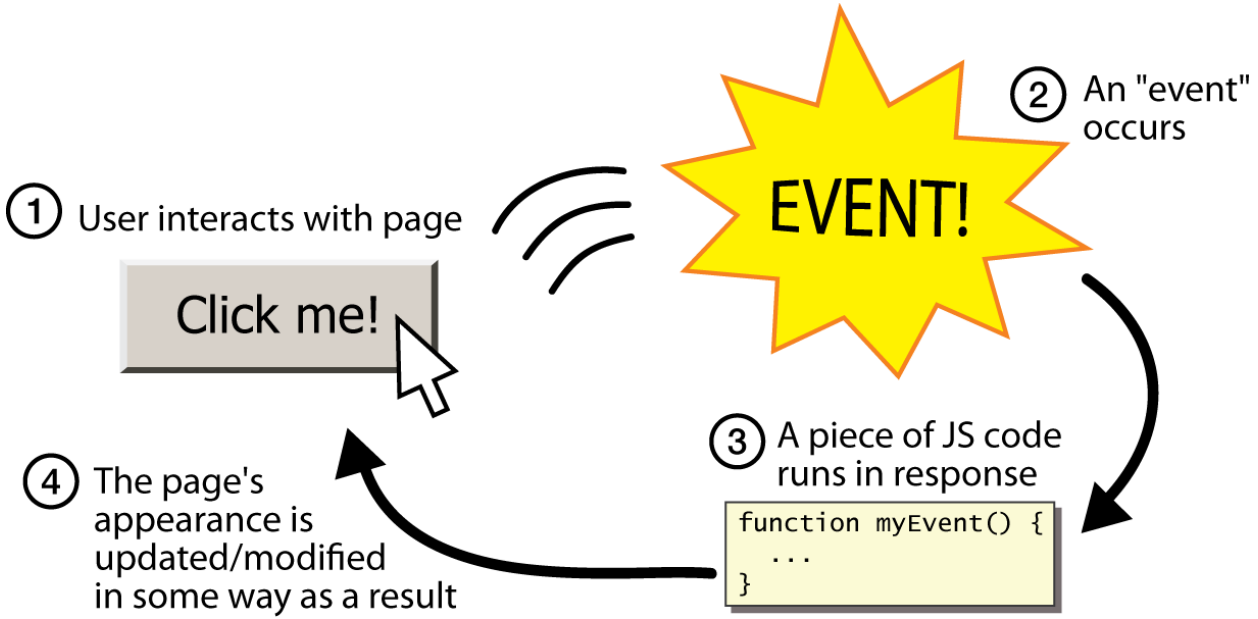
- write a html page showing your favorite movies (at least 3) on a unordered list
- make the color of the movie names turns from black to red one by one every 10 seconds
- add a button to the page, which pops up messages of reversed names of all movies listed when clicking
  - using DOM functions

Further Readings

- W3School DOM node reference
  http://www.w3school.com/dom/dom_node.asp/
- W3School DOM tutorial
  http://www.w3schools.com/htmldom/
- Quirksmode DOM tutorial
  http://www.quirksmode.org/dom/intro.html
- Prototype Learning Center
  http://www.prototypejs.org/learn
- How prototype extends the DOM
  http://www.prototypejs.org/learn/extensions

Part II DOM Events

Event-driven programming

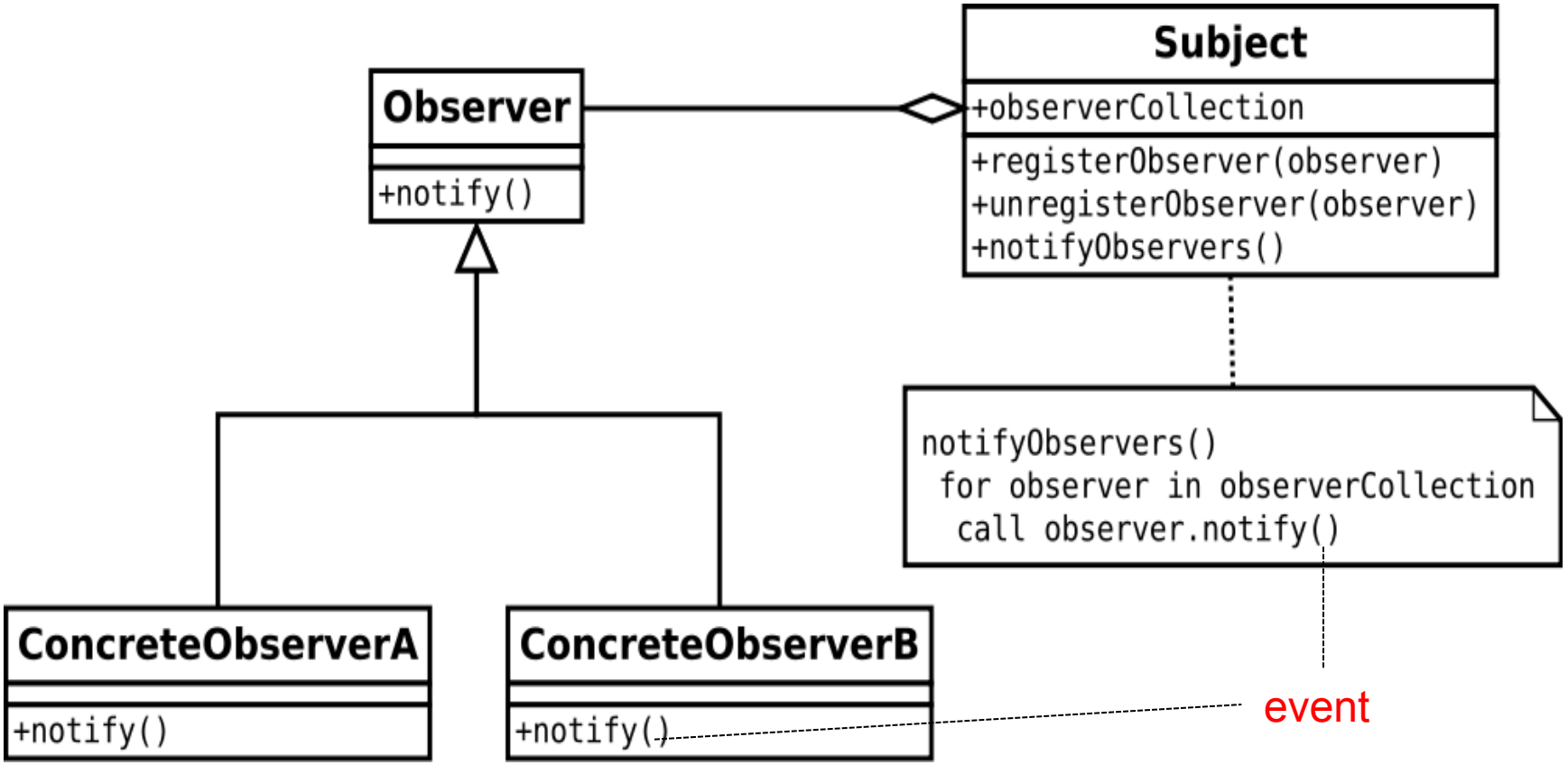


- **event-driven programming**: writing programs driven by user events

Outline

- **Observer pattern**
- Event object
- DOM 2 event flow
- Event handling
- Timer
- Unobtrusive JavaScript & onload event

Observer pattern



A subject may have **multiple** observers

Outline

- Observer pattern
- **Event object**
- DOM 2 event flow
- Event handling
- Timer
- Unobtrusive JavaScript & onload event

Event object

表 8-14 event 对象常用属性[14]

方法	描述
type	事件名称，例如点击事件为'click'
target	发生事件的 DOM（BOM）元素
currentTarget	其事件处理器正在执行的 DOM（BOM）元素
eventPhase	事件当前的阶段（capture、target、bubbling，参考 8.6.3 节）
altKey	事件发生时，键盘上的“Alt”键是否被按下
ctrlKey	事件发生时，键盘上的“Ctrl”键是否被按下
shiftKey	事件发生时，键盘上的“Shift”键是否被按下
button	事件发生时，用户鼠标哪个键被按下
clientX	事件发生时，鼠标在浏览器窗口的横坐标，浏览器左上角为原点
clientY	事件发生时，鼠标在浏览器窗口的纵坐标，浏览器左上角为原点
screenX	事件发生时，鼠标在屏幕上的横坐标，屏幕左上角为原点
screenY	事件发生时，鼠标在屏幕上的纵坐标，屏幕左上角为原点

The keyword this

```
function showThis(){
  alert(this);
  alert(this.name);
}

var a = {name:'example'};
a.showThis = showThis;

showThis();
window.showThis();
a.showThis();
```

- this refers to the object on which it was called.

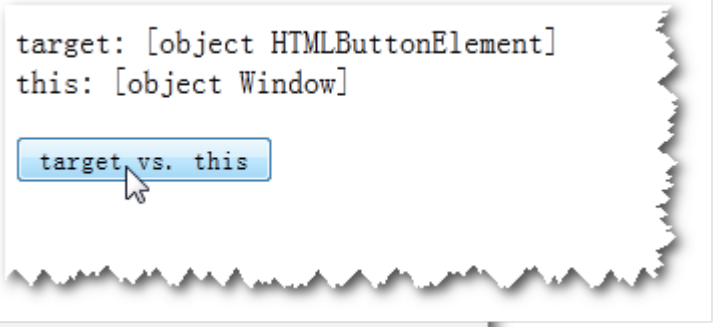
this vs. event.target

源代码 8-25 event.target vs. this 示例

```
------------------------- HTML -------
……
    <pre id="message"></pre>
    <button>target vs. this</button>
……
------------------------ JavaScript ---
window.onload = function() {
    document.getElementsByTagName('button')[0].onclick =
showTargetAndThis;
}

function showTargetAndThis(event) {
    var paragraph = document.getElementById("message");
    var eventInfo = createTargetAndThisInfo(event);
    var textNode = document.createTextNode(eventInfo);
    paragraph.appendChild(textNode);
}

function createTargetAndThisInfo(event) {
    var message = "target: " + event.target + "\n";
    message += "this: " + this;
    return message;
}
```



Add event handlers/listeners

1. 给 HTML 元素增加事件处理器属性，事件处理器名称为“on 事件名”，参见源代码 8-26（方法 1）
2. 给 DOM 元素事件处理器 property 赋值，事件处理器 property 名称为“on 事件名”，参见源代码 8-26（方法 2）
3. **使用 DOM 元素的添加事件处理器方法 addEventListener**，方法第 1 个参数为事件名，第 2 个参数是事件处理器，参见源代码 8-26（方法 3）

源代码 8-26 不同方法添加事件处理器示例

```
---------------------- 方法1（HTML）----------------------------

<button onclick="alert('Hello world!')">……

---------------------- 方法2(JavaScript) ----------------------

button.onclick = function(){alert('Hello world!');};

---------------------- 方法3(JavaScript) ----------------------

button.addEventListener('click', function(){alert('Hello
world');});
```

Add event handlers/listeners

源代码 8-27 添加/移除事件处理器示例

```
------------------------- HTML -----------------------------
......
        <p></p>
        <button>变!</button>
......
------------------------ JavaScript ------------------------
window.onload = function(){
    var button = document.getElementsByTagName("button")[0];
    button.addEventListener('click',showClickTimes);
    button.addEventListener('click',handler1);
}

var count = 0;

function showClickTimes(event){
    var paragraph = document.getElementsByTagName("p")[0];
    count++;
    paragraph.textContent = "第" + count + "次点击";
}

function handler1(event){
    alert("去年今日此门中，人面桃花相映红");
    event.target.removeEventListener('click', handler1);
    event.target.addEventListener('click',handler2);
}

function handler2(event){
    alert("人面不知何处去，桃花依旧笑春风");
    event.target.removeEventListener('click', handler2);
    event.target.addEventListener('click', handler1);
}
```

Outline

- Observer pattern
- Event object
- **DOM 2 event flow**
- Event handling
- Timer
- Unobtrusive JavaScript & onload event

Events on multiple elements

HTML 文档和 DOM 都是树型结构，因此当一个事件发生的时候，它既发生在当前元素上，也发生在当前元素的父元素和祖先元素上。例如，源代码 8-28 所示的网页，运行时如果用户点击了按钮，那么点击的事件也同样发生在 form、div 和 body 元素上。

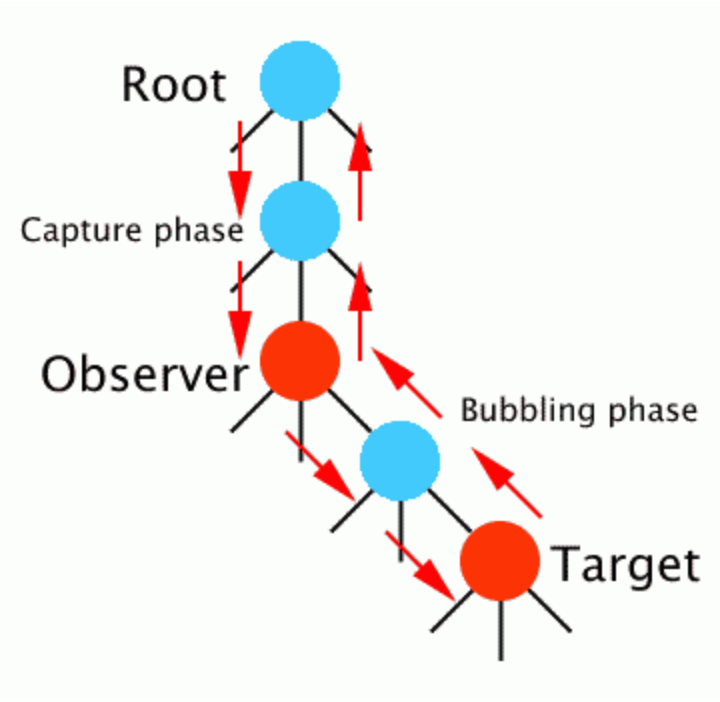
源代码 8-28 DOM 树与事件

```
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<div>
    <form action="#">
        <input type="button" value="点击"/>
    </form>
</div>
</body>
</html>
```

- When an event happens? In which order it propagates? And when it comes to an end?

Event flow

- each event has a target, which can be accessed via event

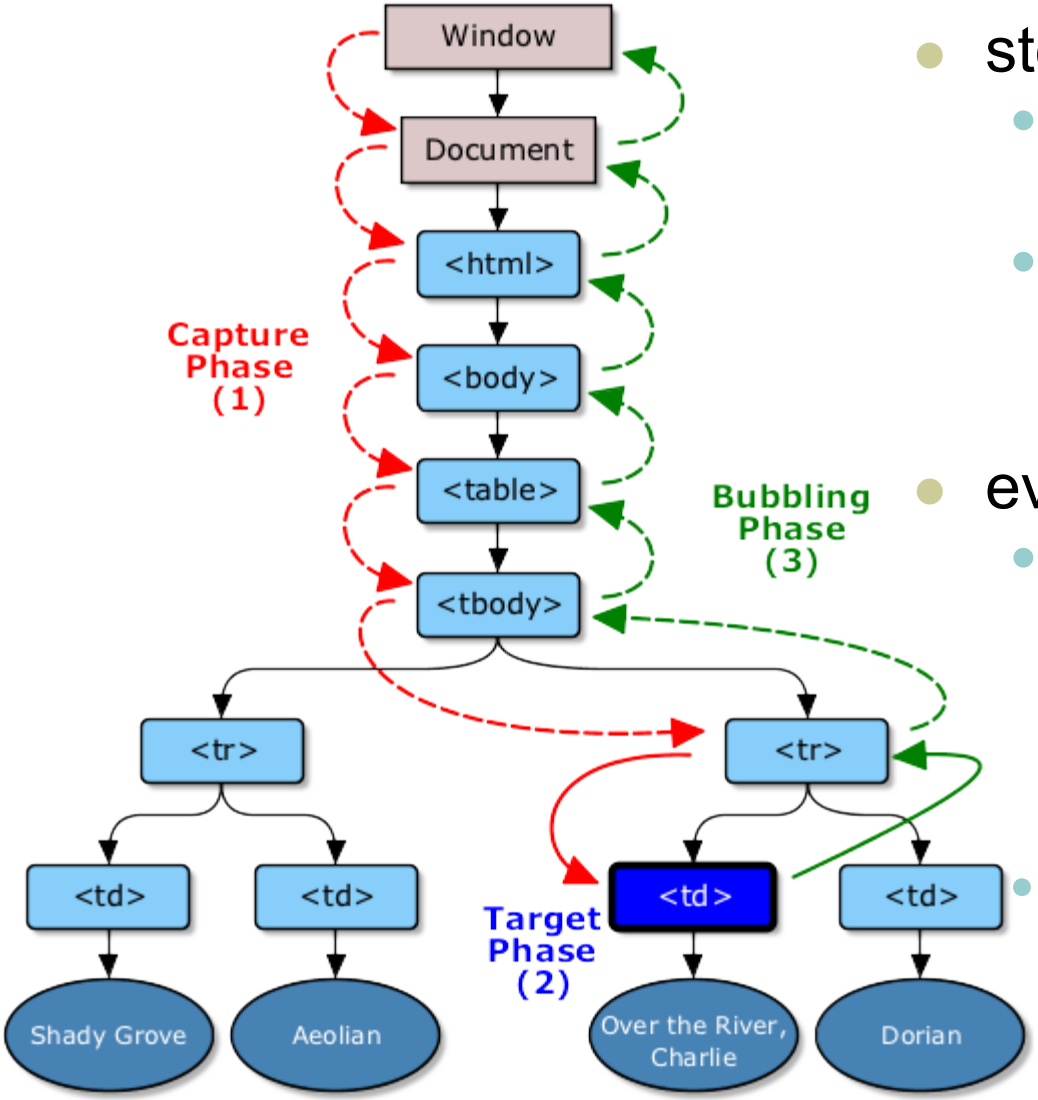



```
element.onclick = handler(e);
function handler(e){
    if(!e) var e = window.event;
    // e refers to the event
    // see detail of event
    var original = e.eventTarget;
}
```

- each event originates from the browser, and is passed to the DOM
- DOM propagates the event in 3 phases:
  - **capture** phase, **target** phase, **bubbling** phase (some events have no bubbling phase, i.e. load event of document element.)
  - register a capture phase handler: (IE early than IE9 can't do this)

```
element.addEventListener('click',handler,true);
```

Event flow



- stopping event propagation
  - by throwing any exception inside an event handler
  - by calling `event.stopProgagation();` inside a handler
- event cancelation
  - canceling default action (i.e. navigating to a new page when clicking on a hyperlink): `event.preventDefault();`
  - 源代码 8-29 DOM事件传播示例

Outline

- Observer pattern
- Event object
- DOM 2 event flow
- **Event handling**
- Timer
- Unobtrusive JavaScript & onload event

DOM 2 Event Types

- UI event types:
  - DOMFocusIn, DOMFocusOut, DOMActivate
- Mouse event types:
  - click, mousedown, mouseup, mouseover, mousemove, mouseout
- Key event types: (not in DOM 2, but will in DOM 3)
- Mutation events:
  - DOMSubtreeModified, DOMNodeInserted, …
- HTML event types:
  - load, unload, abort, error, select, change, submit, reset, focus, blur, resize, scroll

- more details

Useful event types

abort	blur	change	click	dblclick	error
keydown	keypress	keyup	load	mousedown	mousemove
mouseover	mouseup	reset	resize	select	submit
focus	mouseout	unload			

- **problem**: events are tricky and have incompatibilities across browsers reasons: fuzzy W3C event specs; IE disobeying web standards; etc.
- **solution**: using a JavaScript library which encapsulates these incompatibilities.

Mouse events

Clicking

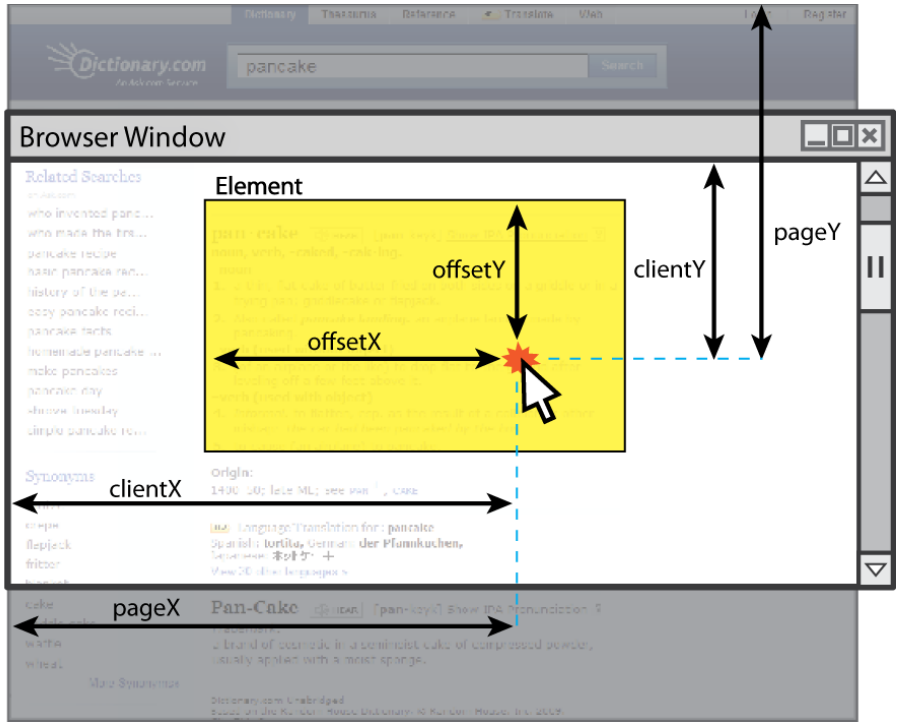
click	user presses/releases mouse button on this element
dblclick	user presses/releases mouse button twice on this element
mousedown	user presses down mouse button on this element
mouseup	user releases mouse button on this element

movement

mouseover	mouse cursor enters this element's box
mouseout	mouse cursor exits this element's box
mousemove	mouse cursor moves around within this element's box

Mouse event objects

- The event parameter passed to a mouse event handler has the following properties:



property/ method	description
clientX, clientY	coordinates in browser window
screenX, screenY	coordinates in screen
offsetX, offsetY*	coordinates in element
pageX, pageY*	coordinates in entire web page

* non-standard properties

Outline

- Observer pattern
- Event object
- DOM 2 event flow
- Event handling
- **Timer**
- Unobtrusive JavaScript & onload event

Timer events

method	description
setTimeout(*function*, *delayMS*);	arranges to call given function after given delay in ms
setInterval(*function*, *delayMS*);	arranges to call function repeatedly every *delayMS* ms
clearTimeout(*timerID*); clearInterval(*timerID*);	stops the given timer so it will not call its function

- both setTimeout and setInterval return an ID representing the timer
  - this ID can be passed to clearTimeout/Interval later to stop the timer

setTimeout example

```
<button onclick="delayMsg();">Click me!</button>
<span id="output"></span>
```

```
function delayMsg() {
  setTimeout(booyah, 5000);
  $("output").innerHTML = "Wait for it...";
}

function booyah() {    // called when the timer goes off
  $("output").innerHTML = "BOOYAH!";
}
```

JS

Click me!

output

setInterval example

```
var timer = null;  // stores ID of interval timer

function delayMsg2() {
  if (timer == null) {
    timer = setInterval(rudy, 1000);
  } else {
    clearInterval(timer);
    timer = null;
  }
}

function rudy() {   // called each time the timer goes off
  $("output").innerHTML += " Rudy!";
}
```

JS

Click me!

output

Passing parameters to timers

```
function delayedMultiply() {
  // 6 and 7 are passed to multiply when timer goes off
  setTimeout(multiply, 2000, 6, 7);
}
function multiply(a, b) {
  alert(a * b);
}
```

JS

Click me

output

- any parameters after the delay are eventually passed to the timer function
  - doesn't work in IE6; must create an intermediate function to pass the parameters

Common timer errors

- many students mistakenly write () when passing the function

```
setTimeout(booyah(), 2000);
setTimeout(booyah, 2000);

setTimeout(multiply(num1 * num2), 2000);
setTimeout(multiply, 2000, num1, num2);
```
JS

- what does it actually do if you have the () ?
- it calls the function immediately, rather than waiting the

Outline

- Observer pattern
- Event object
- DOM 2 event flow
- Event handling
- Timer
- **Unobtrusive JavaScript & onload event**

Unobtrusive JavaScript

- JavaScript event code seen previously was *obtrusive*, in the HTML; this is bad style
- now we'll see how to write *unobtrusive* JavaScript code
  - HTML with minimal JavaScript inside
  - uses the DOM to attach and execute all JavaScript functions
- allows separation of web site into 3 major categories:
  - content (HTML) - what is it?
  - presentation (CSS) - how does it look?
  - behavior (JavaScript) - how does it respond to user interaction?

Obtrusive event handlers (bad)

```
<button id="ok" onclick="okayClick();">OK</button>
```
HTML

```
// called when OK button is clicked
function okayClick() {
  alert("booyah");
}
```
JS

output

- this is bad style (HTML is cluttered with JS code)
- goal: remove all JavaScript code from the HTML body

Attaching an event handler in JavaScript code

```
// where element is a DOM element object
element.event = function;
```
JS

```
$("ok").onclick = okayClick;
```
JS

OK

output

- it is legal to attach event handlers to elements' DOM objects in your JavaScript code
  - notice that you do **not** put parentheses after the function's name
- this is better style than attaching them in the HTML
- Where should we put the above code?

When does my code run?

```HTML
  <head>
    <script src="myfile.js" type="text/javascript"></script>
  </head>

  <body> ... </body>
```

```JS
// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```

- your file's JS code runs the moment the browser loads the script tag
  - any variables are declared immediately
  - any functions are declared but not called, unless your global code explicitly calls them

A failed attempt at being unobtrusive

```HTML
<head>
  <script src="myfile.js" type="text/javascript"></script>
</head>

<body>
  <div><button id="ok">OK</button></div>
```

```JS
// global code
$("ok").onclick = okayClick;   // error: $("ok") is null
```

- problem: global JS code runs the moment the script is loaded
- script in head is processed before page's body has loaded
  - no elements are available yet or can be accessed yet via the DOM
- we need a way to attach the handler after the page has loaded...

The window.onload event

```JS
// this will run once the page has finished loading
function functionName() {
  element.event = functionName;
  element.event = functionName;
  ...
}

window.onload = functionName;    // global code
```

- we want to attach our event handlers right after the page is done loading
  - there is a global event called window.onload event that occurs at that moment
- in window.onload handler we attach all the other handlers to run when events occur

An unobtrusive event handler

```html
<!-- look Ma, no JavaScript! -->
<button id="ok">OK</button>
```
HTML

```js
// called when page loads; sets up event handlers
function pageLoad() {
  $("ok").onclick = okayClick;
}

function okayClick() {
  alert("booyah");
}

window.onload = pageLoad;  // global code
```
JS

OK

output

Common unobtrusive JS errors

- many students mistakenly write () when attaching the handler

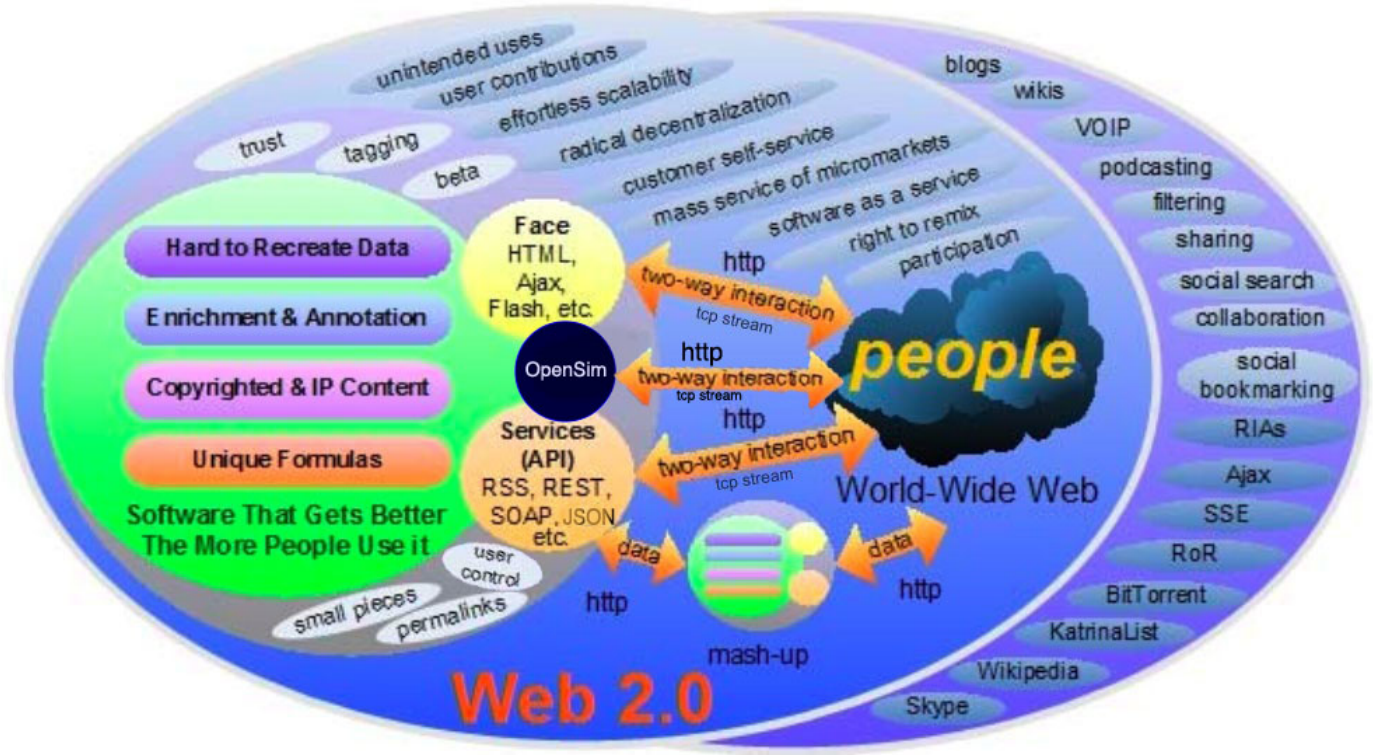
```
window.onload = pageLoad();
window.onload = pageLoad;

okButton.onclick = okayClick();
okButton.onclick = okayClick;
```
JS

- event names are all lowercase, not capitalized like most variables

```
window.onLoad = pageLoad;
window.onload = pageLoad;
```
JS

Thank you!


unintended uses
user contributions
effortless scalability
radical decentralization
customer self-service
mass service of micromarkets
software as a service
right to remix
participation
trust
tagging
beta
blogs
wikis
VOIP
podcasting
filtering
sharing
social search
collaboration
social bookmarking
RIAs
Ajax
SSE
RoR
BitTorrent
KatrinaList
Wikipedia
Skype
Hard to Recreate Data
Enrichment & Annotation
Copyrighted & IP Content
Unique Formulas
Software That Gets Better The More People Use it
Face
HTML, Ajax, Flash, etc.
OpenSim
Services (API)
RSS, REST, SOAP, JSON etc.
http
two-way interaction
tcp stream
people
World-Wide Web
user control
small pieces
permalinks
data
mash-up
Web 2.0