



中山大學  
SUN YAT-SEN UNIVERSITY

# Lecture 13 More JavaScript and DOM

**SE-805 Web 2.0 Programming**

(<http://my.ss.sysu.edu.cn/wiki/display/W2PSC/Home> , supported by Google;  
using some slides of & inspired by Marty Stepp's CSE 190 M courseware)

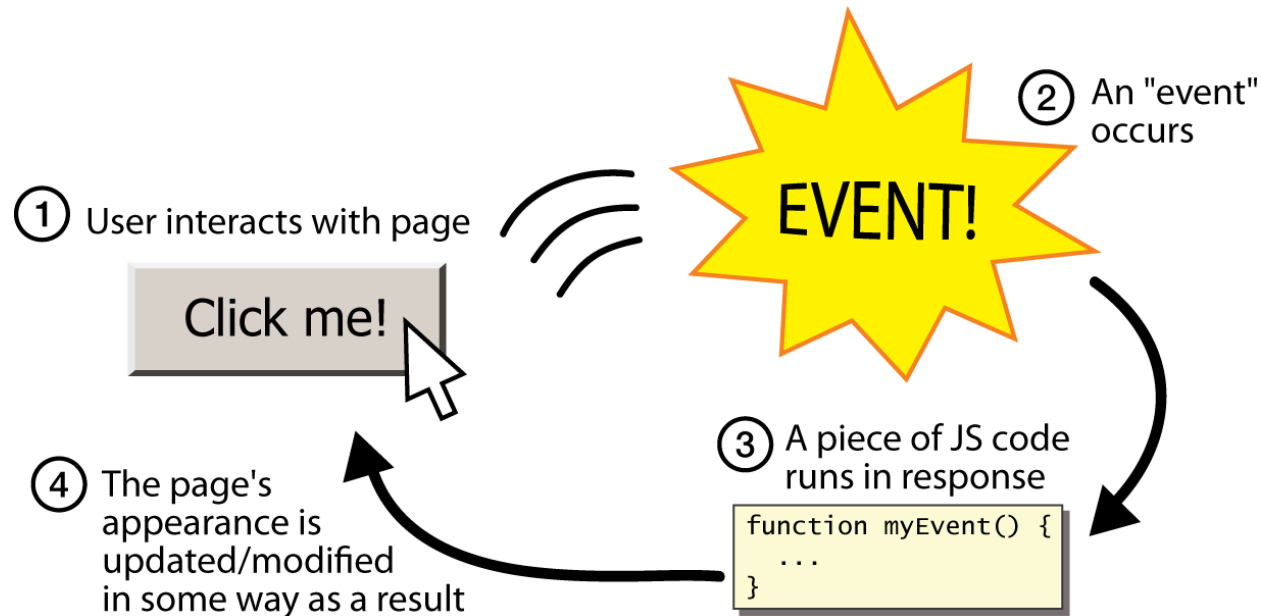
School of Software, Sun Yat-sen University

# Outline

---

- **Event-driven JavaScript**
- DOM basic
- Prototype and DOM
- Timer


# 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)*

<code>&lt;button&gt;Click me!&lt;/button&gt;</code>	HTML
	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
  - attach the function to the event on the control

# Event handlers

<code>&lt;element attributes onclick="function() ;"&gt;...</code>	HTML
<code>&lt;button onclick="myFunction() ;"&gt;Click me!&lt;/button&gt;</code>	HTML
	output

- JavaScript functions can be set as event handlers
  - when you interact with the element, the function will execute
- onclick is just one of many event HTML attributes we'll use
- event handlers never execute until the events they handled occur

# Outline

---

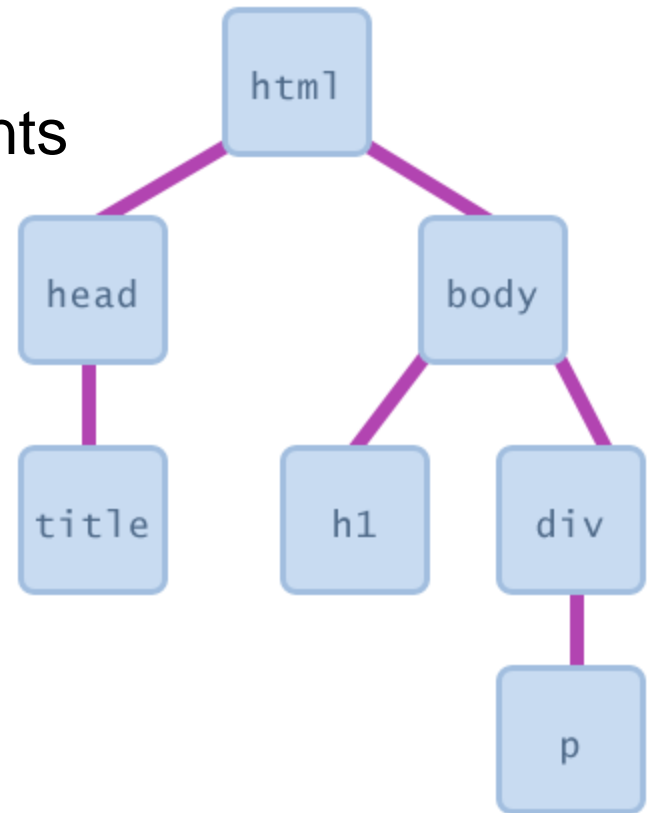
- Event-driven JavaScript
- **DOM basic**
- Prototype and DOM
- Timer

# 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



# DOM element

- every element on the page has a corresponding **DOM** object

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

HTML

```
<p>
  Look at this octopus:
  
  Cute, huh?
</p>
```

DOM Element Object

Property	Value
tagName	"IMG"
<u>src</u>	"octopus.jpg"
alt	"an octopus"
id	"icon01"

JavaScript

```
var icon = document.getElementById("icon01");
icon.src = "kitty.gif";
```

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



# Accessing elements: **document.getElementById**

```
var name = document.getElementById("id"); JS
```

```
<button onclick="changeText();">Click me!</button>
<span id="output">replace me</span>
<input id="textbox" type="text" /> HTML
```

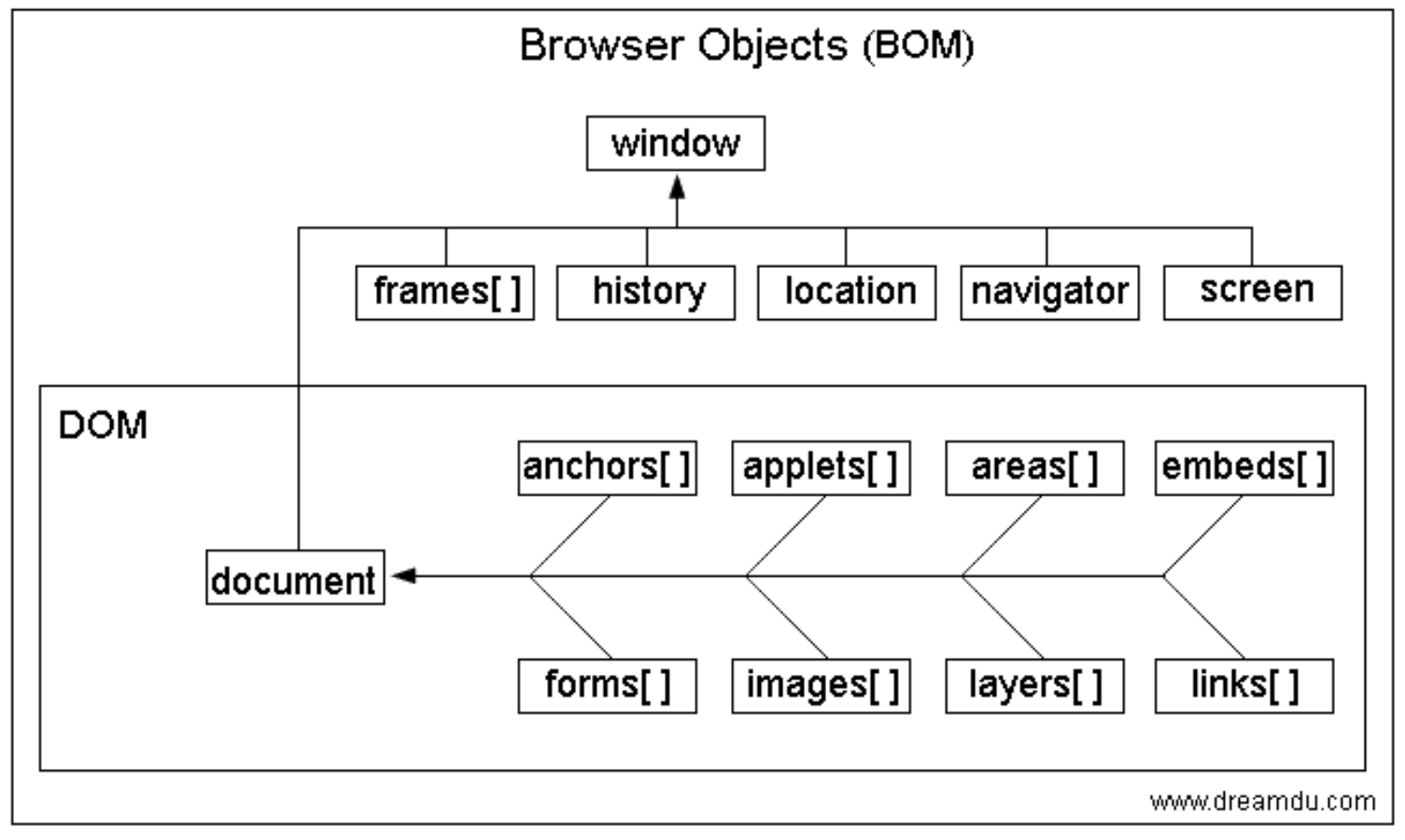
```
function changeText() {
  var span = document.getElementById("output");
  var textBox = document.getElementById("textbox");
  textBox.value = span.innerHTML;
  span.innerHTML = "Hello, how are you?";
} JS
```

Click me! replace me  output

- **document.getElementById** returns the **DOM** object for an element with a given **id**
- can change the text inside most elements by setting the **innerHTML** property
- can change the text in form controls by setting the **value** property

# Essential of DOM

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



# Outline

---

- Event-driven JavaScript
- DOM basic
- **Prototype and DOM**
- Timer

# Problems with JavaScript

---

- JavaScript is a powerful language, but it has many flaws:
- the DOM can be clunky to use
  - **document.getElementById** , more than 20 key strikes!
- the same code doesn't always work the same way in every browser
  - code that works great in Firefox, Safari, ... will fail in IE and vice versa
- many developers work around these problems with hacks (checking if browser is IE, etc.)

# Prototype framework

---

```
<script src="http://ssw2p.3322.org/public/scripts/prototype/prototype-1.6.0.3.js" type="text/javascript"></script>
```

JS

```
<!-- or link to Prototype home site -->  
<script src="http://prototypejs.org/assets/2008/9/29/prototype-1.6.0.3.js" type="text/javascript"></script>
```

JS

- the Prototype JavaScript library adds many useful features to JavaScript:
  - many useful extensions to the DOM
  - added methods to String, Array, Date, Number, Object
  - improves event-driven programming
  - many cross-browser compatibility fixes
  - makes Ajax programming easier (seen later)

# The \$ function

---

```
$ ("id")
```

*JS*

- returns the **DOM** object representing the element with the given **id**
- short for **document.getElementById("id")**
- often used to write more concise **DOM** code:

```
$ ("footer").innerHTML = $ ("username").value.toUpperCase (); JS
```

# DOM object properties

```
<div id="main" class="foo bar">
  <p>Hello, <em>very</em> happy to see you!</p>
  
</div>
```

HTML

Property	Description	Example
tagName	element's HTML tag	<code>\$("main").tagName</code> is "DIV"
className	CSS classes of element	<code>\$("main").className</code> is "foo bar"
innerHTML	content inside element	<code>\$("main").innerHTML</code> is " <p>Hello, <em>ve...
src	URL target of an image	<code>\$("icon").src</code> is "images/borat.jpg"

# DOM properties for form controls

```
<input id="sid" type="text" size="7" maxlength="7" />
<input id="frosh" type="checkbox" checked="checked" /> Freshman? HTML
```

 ☒ Freshman?

output

Property	Description	Example
value	the text in an input control	<code>\$("sid").value</code> could be "1234567"
checked	whether a box is checked	<code>\$("frosh").checked</code> is true
disabled	whether a control is disabled (boolean)	<code>\$("frosh").disabled</code> is false
readOnly	whether a text box is read-only	<code>\$("sid").readOnly</code> is false



# Abuse of **innerHTML**

---

```
// bad style!
var paragraph = document.getElementById("welcome");
paragraph.innerHTML = "<p>text and <a href='page.html'>link</a>"; JS
```


- **innerHTML** can inject arbitrary HTML content into the page
- however, this is prone to bugs and errors and is considered **poor style**
- we forbid using **innerHTML** to inject **HTML** tags; **inject plain text only**
  - (later, we'll see a better way to inject content with HTML tags in it)

# Adjusting styles with the DOM

```
<button id="clickme">Color Me</button>
```

```
window.onload = function() {
  document.getElementById("clickme").onclick = changeColor;
};
function changeColor() {
  var clickMe = document.getElementById("clickme");
  clickMe.style.color = "red";
}
```

JS



output

Property	Description
<u>style</u>	lets you set any CSS style property for an element

- contains same properties as in **CSS**, but with **camelCasedNames** examples: **backgroundColor**, **borderLeftWidth**, **fontFamily**

# Common DOM styling errors

---

- many students forget to write `.style` when setting styles

```
var clickMe = document.getElementById("clickme");  
clickMe.color = "red";  
clickMe.style.color = "red";
```

JS

- style properties are capitalized like `This`, not like `this`

```
clickMe.style.font-size = "14pt";  
clickMe.style.fontSize = "14pt";
```

JS

- style properties must be set as strings, often with units at the end

```
clickMe.style.width = 200;  
clickMe.style.width = "200px";  
clickMe.style.padding = "0.5em";
```

JS

- write exactly the value you would have written in the CSS, but in quotes

# Unobtrusive styling

---

```
function okayClick() {  
  this.style.color = "red";  
  this.className = "highlighted";  
}
```

JS

```
.highlighted { color: red; }
```

CSS

- well-written **JavaScript** code should contain as little **CSS** as possible
- use **JS** to set **CSS** **classes/IDs** on elements
- define the styles of those **classes/IDs** in your **CSS** file

# Outline

---

- Event-driven JavaScript
- DOM basic
- Prototype and DOM
- **Timer**

# Timer events

---

method	description
<u><code>setTimeout(function, delayMS);</code></u>	arranges to call given function after given delay in ms
<u><code>setInterval(function, delayMS);</code></u>	arranges to call function repeatedly every <i>delayMS</i> ms
<u><code>clearTimeout(timerID);</code></u> <u><code>clearInterval(timerID);</code></u>	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

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

# Summary

---

- Event-driven JavaScript
  - EDP, button, event handlers
- DOM basic
  - DOM, DOM Element, Accessing elements
  - BOM & DOM
- Prototype and DOM
  - JS problems, prototype, \$
  - DOM object properties (for form controls)
  - innerHTML, style, common errors
- Timer
  - timer events, setTimeout, setInterval
  - passing parameters, common errors

# 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
  - using Prototype.js functions

# Further Readings

---

- W3School DOM node reference  
[http://www.w3school.com/dom/dom\\_node.asp/](http://www.w3school.com/dom/dom_node.asp/)
- W3School DOM tutorial  
<http://www.w3schools.com/html/dom/>
- 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>

# Thank you!

