中山大學
SUN YAT-SEN UNIVERSITY

Lecture 13
More JavaScript and DOM

SE-805 Web 2.0 Programming (supported by Google)

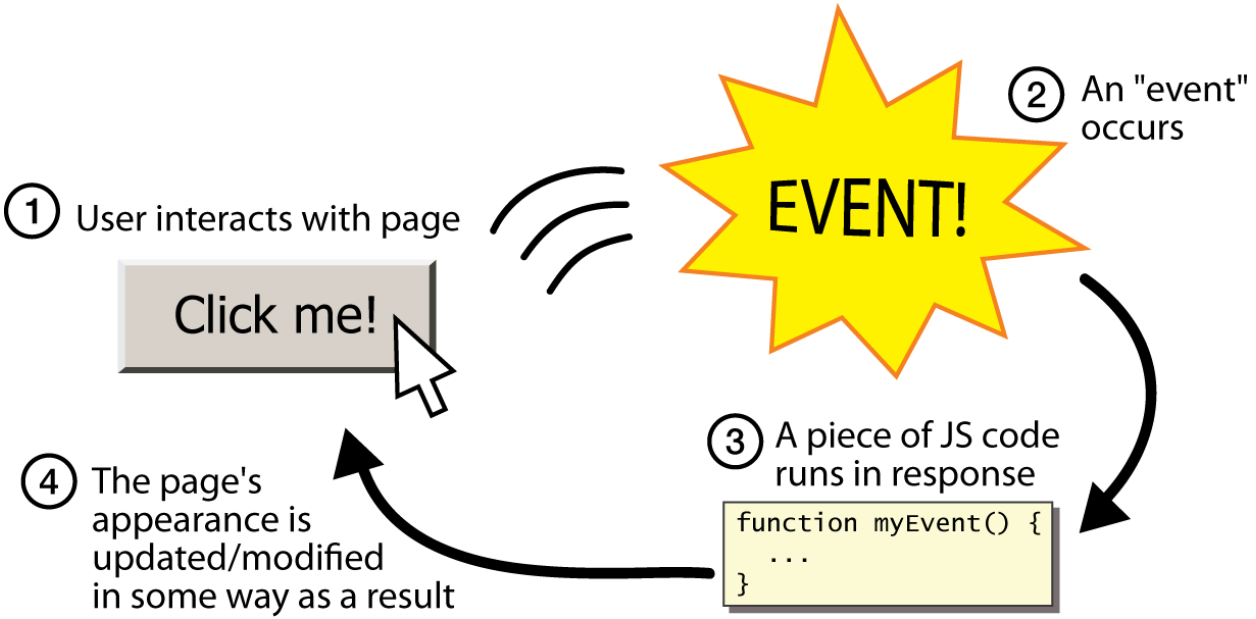
http://my.ss.sysu.edu.cn/courses/web2.0/

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)

```
<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
  - attach the function to the event on the control

Event Handlers

```
<element attributes onclick="function();">...
```
HTML

```
<button onclick="myFunction();">Click me!</button>
```
HTML

Click me!

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

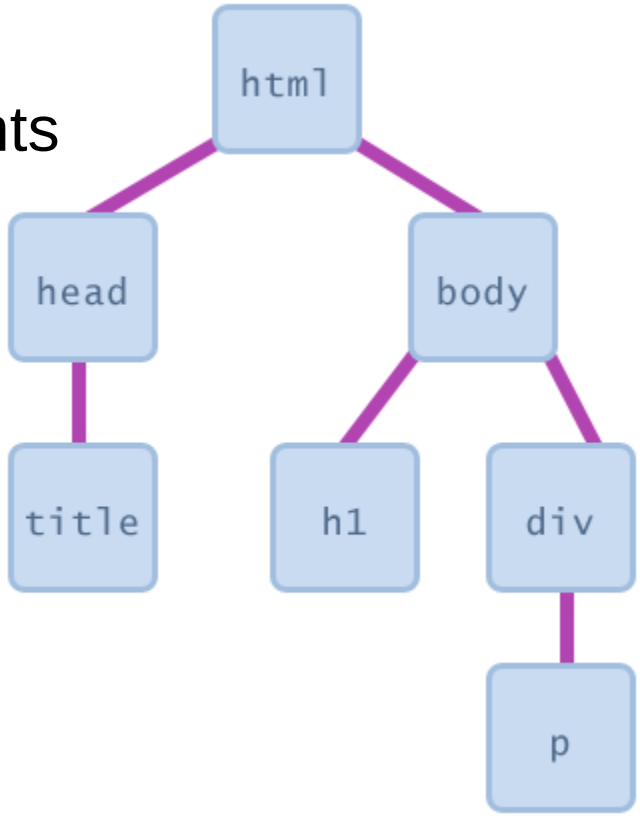
Outline

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

Document Object Model (DOM)

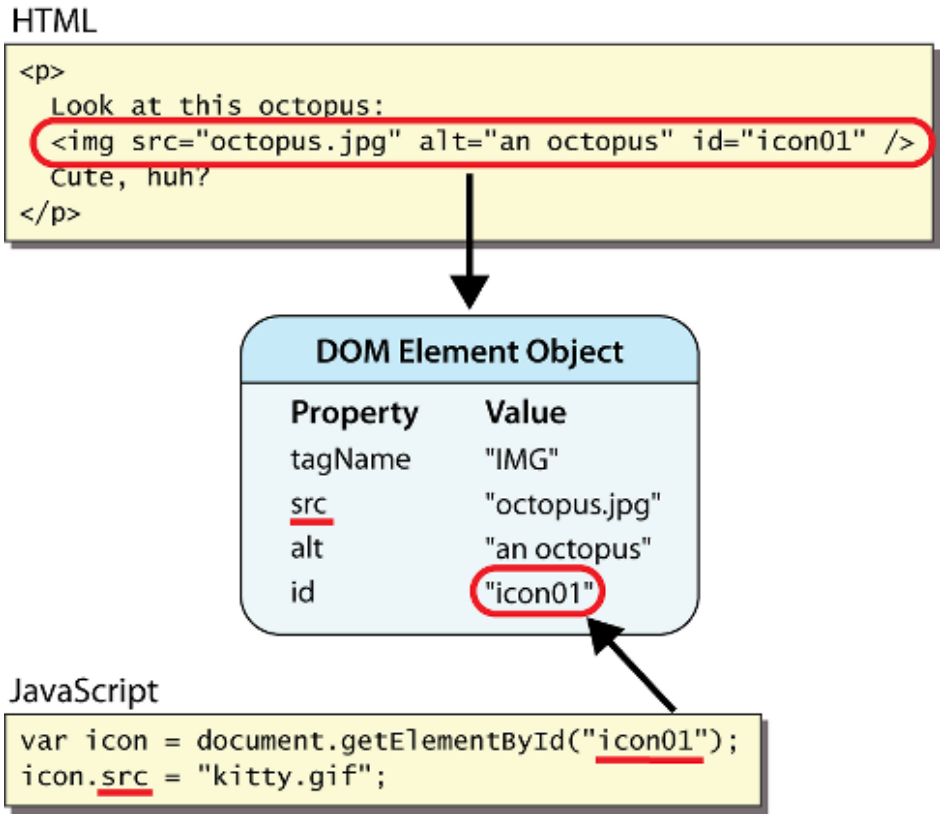
A set of JavaScript objects that represent each element on the page

- 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*

- 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

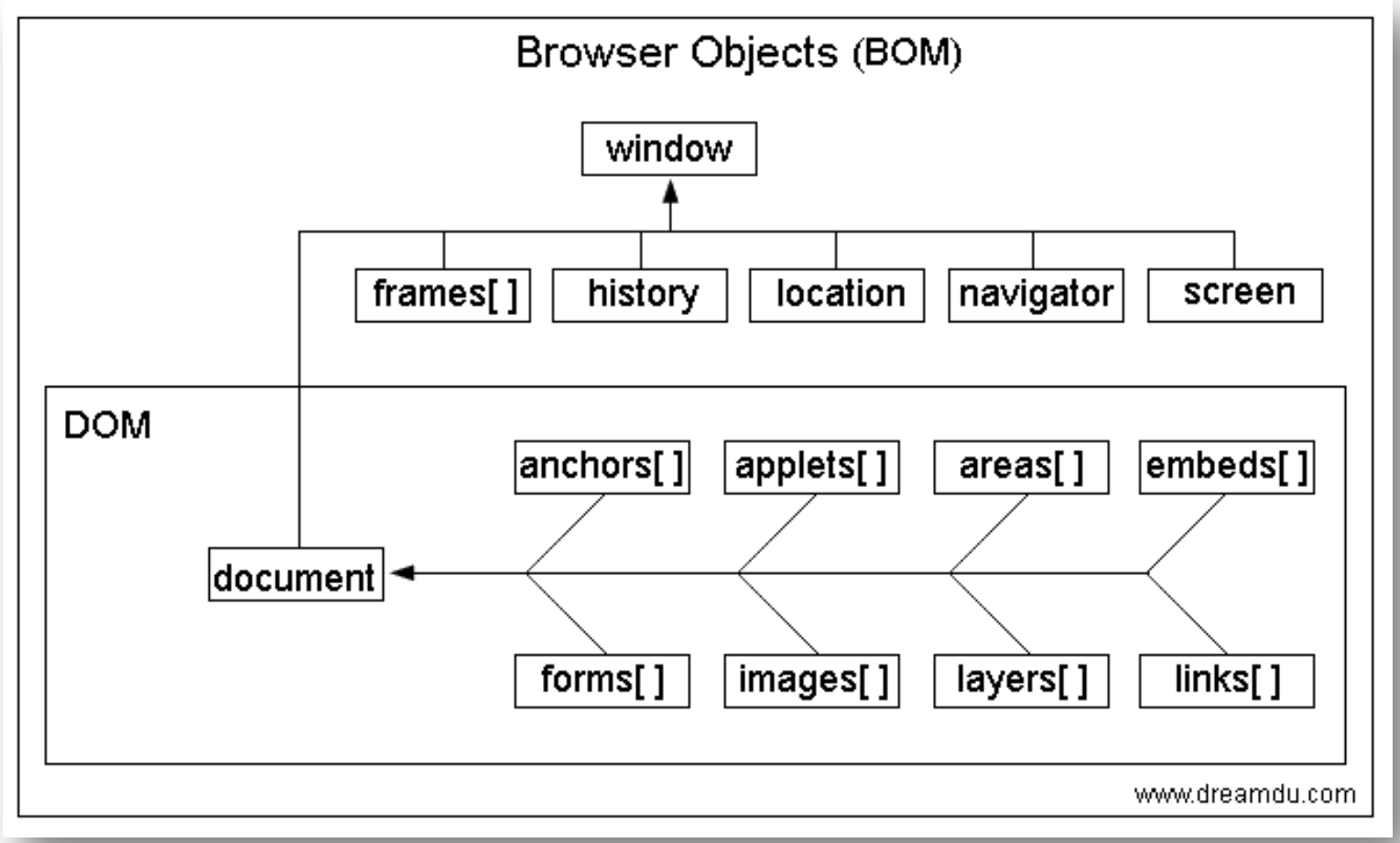
Click me! replace me

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")
```

- 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();
```

DOM Object Properties

```html
<div id="main" class="foo bar">
  <p>Hello, <em>very</em> happy to see you!</p>
  <img id="icon" src="images/borat.jpg" alt="Borat" />
</div>
```
HTML

Property	Description	Example
`tagName`	element's HTML tag	`$("main").tagName is "DIV"`
`className`	CSS classes of element	`$("main").className is "foo bar"`
`innerHTML`	content inside element	`$("main").innerHTML is "\n <p>Hello, <em>ve...`
`src`	URL target of an image	`$("icon").src 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	`$("sid").value` **could be** `"1234567"`
`checked`	whether a box is checked	`$("frosh").checked` **is** `true`
`disabled`	whether a control is disabled (boolean)	`$("frosh").disabled` **is** `false`
`readOnly`	whether a text box is read-only	`$("sid").readOnly` **is** `false`

Abuse of innerHTML

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

- 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

Color Me

output

Property	Description
style	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 likeThis, 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
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

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) as an 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/
- 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

Thank you!