FORM EVENTS

Form Events

Event	Description
onblur	A form element has lost focus (that is, control has moved to a different element, perhaps due to a click or Tab key press.
onchange	Some <input>, <textarea> or <select> field had their value change. This could mean the user typed something, or selected a new choice.
onfocus	Complementing the onblur event, this is triggered when an element gets focus (the user clicks in the field or tabs to it)
onreset	HTML forms have the ability to be reset. This event is triggered when that happens.
onselect	When the users selects some text. This is often used to try and prevent copy/paste.
onsubmit	When the form is submitted this event is triggered. We can do some pre–validation when the user submits the form in JavaScript before sending the data on to the server.

Validating Forms

You mean pre-validating right?

Writing code to prevalidate forms on the client side will reduce the number of incorrect submissions, thereby reducing server load.

There are a number of common validation activities including email validation, number validation, and data validation.

Form Submission

Example

```
function(pass,e){
    var pass = document.getElementById("pw").value;
    if(pass==""){
       alert ("enter a password");
       e.preventDefault();
    }
 }
```

LISTING 6.16 validating a password to not be blank

Validating Forms

Empty field

```
function(fieldValue,e){
    if(fieldValue==null || fieldValue== ""){
        // the field was empty. Stop form submission
        e.preventDefault();
        // Now tell the user something went wrong
        alert("you must enter a username");
    }
}
```

LISTING 6.18 A simple validation script to check for empty fields

Validating Forms

Empty field

If you want to ensure a checkbox is ticked, use code like that below.

```
if (inputField.type=="checkbox"){

        if (inputField.checked)

                //Now we know the box is checked

}
```

Submitting Forms

Submitting a form using JavaScript requires having a node variable for the form element. Once the variable, say, formExample is acquired, one can simply call the submit() method:

formExample.**submit();**

This is often done in conjunction with calling **preventDefault()** on the onsubmit event.

THE DOCUMENT OBJECT MODEL (DOM)

The DOM

Document Object Model

JavaScript is almost always used to interact with the HTML document in which it is contained.

This is accomplished through a programming interface (API) called the **Document Object Model (DOM).**

DOM is a live structure.

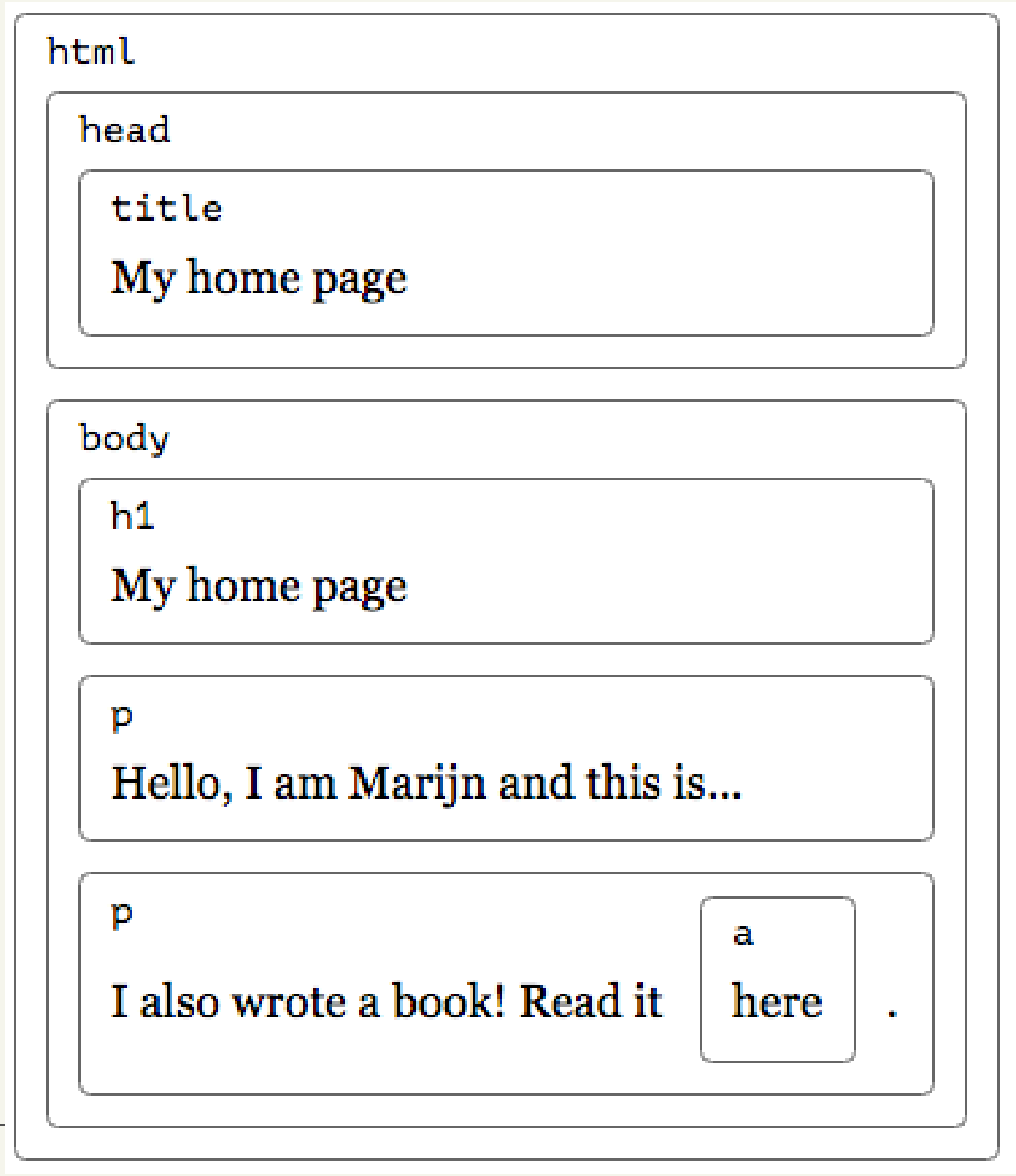
According to the W3C, the DOM is a:

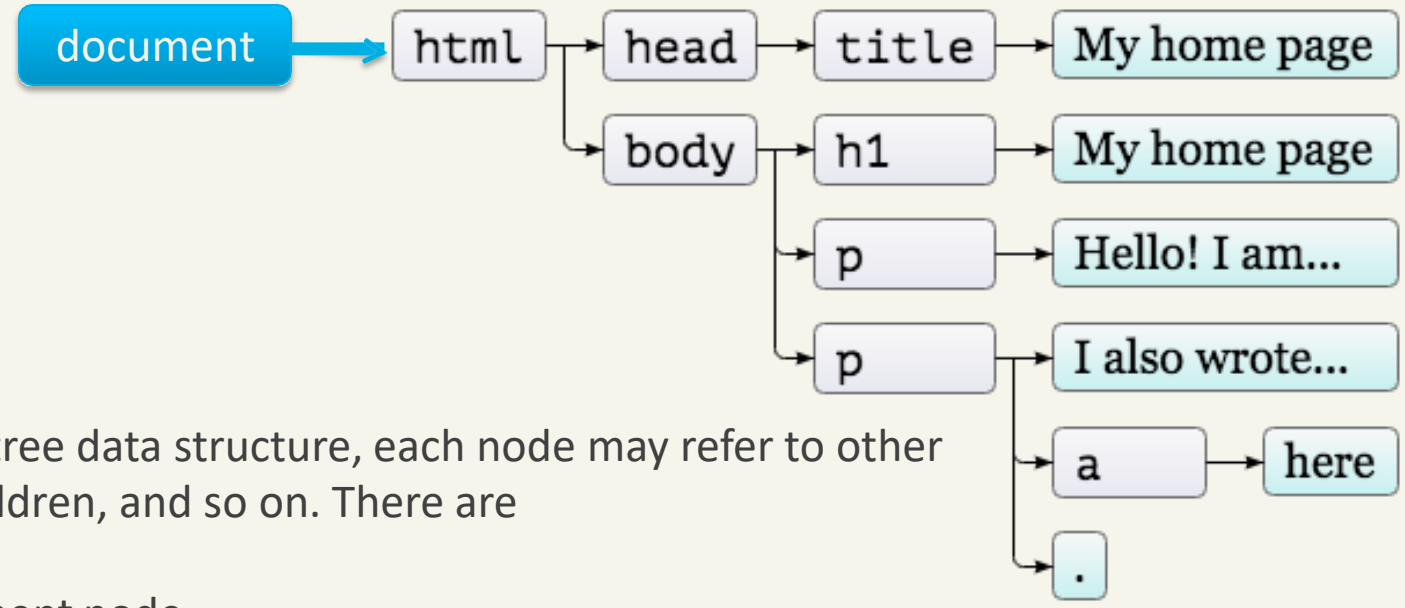
Pla$orm(and language(neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.

```
<!doctype html>
<html>
  <head>
    <title>My home page</title>
  </head>
  <body>
    <h1>My home page</h1>
    <p>Hello, I am Marijn and this is my home page.</p>
    <p>I also wrote a book! Read it
      <a href="http://eloquentjavascript.net">here</a>.</p>
  </body>
</html>
```

For each box, there is an object, which we can interact with to find out things such as what HTML tag it represents and which boxes and text it contains.

This representation is called the Document Object Model, or DOM for short.





DOM is a tree data structure, each node may refer to other nodes, children, and so on. There are

- Document node,
- documentElement node,
- element nodes,
- text nodes,
- attribute nodes,
- comment nodes.

They are all specialization of the Node object.

Document Object

The **DOM document object** is the root JavaScript object representing the entire HTML document.

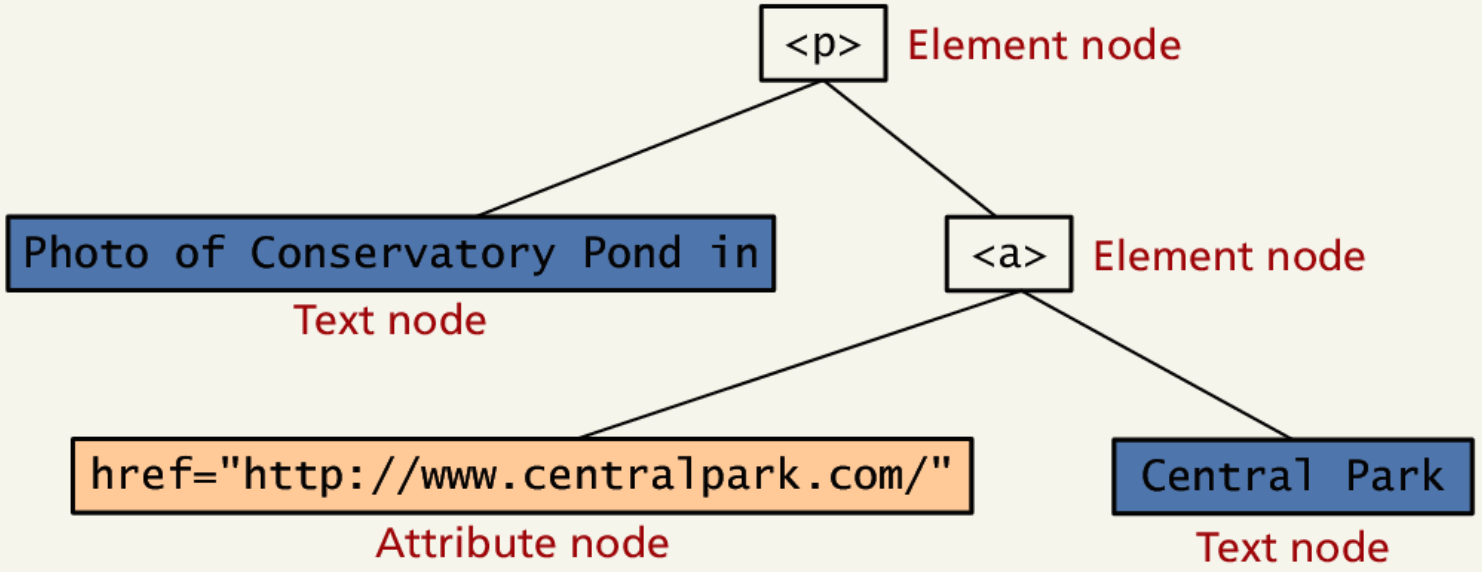
It contains some properties and methods that we will use extensively in our development and is globally accessible as **document**.

```
// specify the doctype, for example html

var a = document.doctype.name;

var docEl = document.documentElement; //html element

var head = document.head; //head delement

var body= document.body; //body element
```

DOM Nodes

Element, text and attribute nodes

```
<p>Photo of Conservatory Pond in
   <a href="http://www.centralpark.com/">Central Park</a>
</p>
```



nodeType Property

Oggetto	Proprietà nodeType	Valore
Element	ELEMENT_NODE	1
Text	TEXT_NODE	3
Document	DOCUMENT_NODE	9
Comment	COMMENT_NODE	8
Attr	ATTRIBUTE_NODE	2
DocumentType	DOCUMENT_TYPE_NODE	10

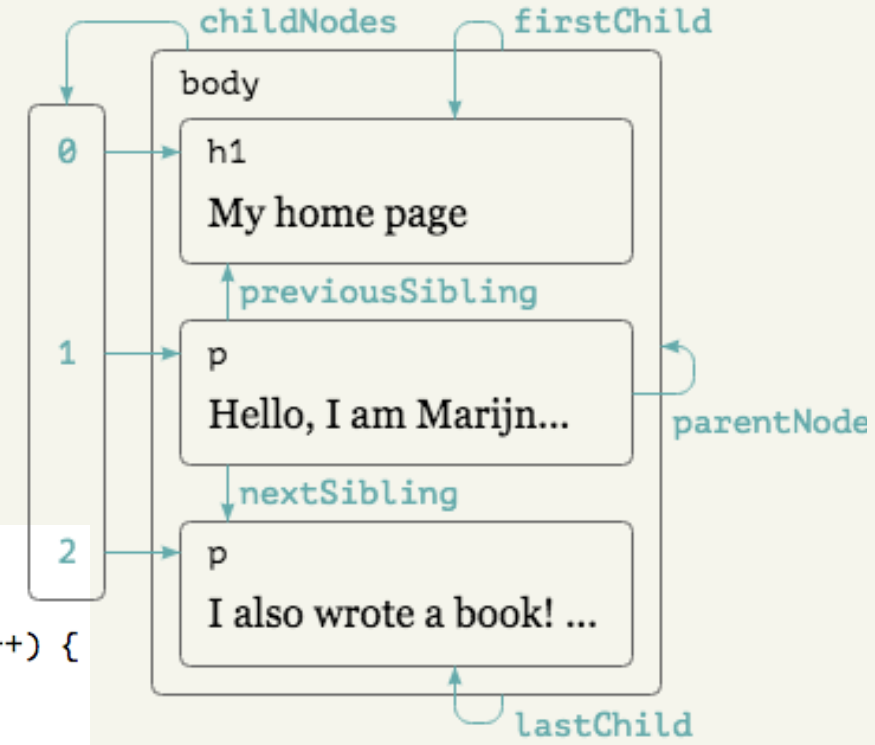
DOM Nodes

Essential Node Object properties

Property	Description
attributes	Collection of node attributes
childNodes	A NodeList of child nodes for this node
firstChild	First child node of this node.
lastChild	Last child of this node.
nextSibling	Next sibling node for this node.
nodeName	Name of the node
nodeType	Type of the node
nodeValue	Value of the node
parentNode	Parent node for this node.
previousSibling	Previous sibling node for this node.

Moving through the DOM

In addition to parent and child links, there are many other pointers to allow to walk the tree, such as first and last child, next and previous sibling




```
function talksAbout(node, string) {
  if (node.nodeType == document.ELEMENT_NODE) {
    for (var i = 0; i < node.childNodes.length; i++) {
      if (talksAbout(node.childNodes[i], string))
        return true;
    }
    return false;
  } else if (node.nodeType == document.TEXT_NODE) {
    return node.nodeValue.indexOf(string) > -1;
  }
}

console.log(talksAbout(document.body, "book"));
// → true
```

Accessing nodes

getElementById(), getElementsByTagName()

Document Object

Document Object Methods

Method	Description
createAttribute()	Creates an attribute node
createElement()	Creates an element node
createTextNode()	Create a text node
getElementById(id)	Returns the element node whose id attribute matches the passed id parameter.
getElementsByTagName(name)	Returns a nodeList of elements whose tag name matches the passed name parameter.

Changing the document

The document.write() method is used to create output to the bottom (append) of the HTML page from JavaScript.

Usually used for testing

```
document.write("<h1>Hello World!</h1><p>Have a nice day!</p>");
```

Modifying a DOM Node

To modify a particular element there is a specific **innerHTML** property (will parse the text as html code)

```
var latest = document.getElementById("latestComment");
var oldMessage = latest.innerHTML;
latest.innerHTML = oldMessage + "<p>Updated this div with JS</p>";
```

LISTING 6.8 Changing the HTML using innerHTML

```
var text = latest.textContent; //access the text

latest.textContent = “new text content”;

//textContent uses straight text!
```

Modifying a DOM element

More verbosely, and validated

DOM functions sush as `createElement()`, `createTextNode()`, `removeChild()`, and `appendChild()` allow us to modify an element in a more rigorous way

```
var latest = document.getElementById("latestComment");
var oldMessage = latest.innerHTML;
var newMessage = oldMessage + "<p>Updated this div with JS</p>";
latest.removeChild(latest.firstChild);
latest.appendChild(document. createTextNode(newMessage));
```

LISTING 6.9 Changing the HTML using createTextNode( ) and appendChild( )

Modifying a DOM element

Element nodes have a number of methods that can be used to change their content.

- `removeChild` `appendChild` `insertBefore`
- `replaceChild` `cloneNode`

Remember: A node can exist in only one place

```
<p>One</p>
<p>Two</p>
<p>Three</p>

<script>
  var paragraphs = document.body.getElementsByTagName("p");
  document.body.insertBefore(paragraphs[2], paragraphs[0]);
</script>
```

Element node Object

Essential Element Node Properties

Property	Description
className	The current value for the class attribute of this HTML element.
id	The current value for the id of this element.
innerHTML	Represents all the things inside of the tags. This can be read or written to and is the primary way which we update particular div's using JS.
style	The style attribute of an element. We can read and modify this property.
tagName	The tag name for the element.

Attributes

Attributes are only available on element nodes, and can be accessed by methods such as:

`getAttribute(name)` `removeAttribute(name)`

`setAttribute(name,val)`

```
<a href='foo.html' class='test one' name='fooAnchor' id='fooAnchor'>Hi</a>
```

Attributes are defined by HTML and contained within the `attributes` property of that html object.

Attributes and Properties

```
<a href='foo.html' class='test one' name='fooAnchor' id='fooAnchor'>Hi</a>
```

Some HTML attributes have oneZtoZone matching onto a property, like `id`

```
+-----------------------------------------+
| a                                       |
+-----------------------------------------+
| href:       "http://example.com/foo.html" |
| name:       "fooAnchor"                 |
| id:         "fooAnchor"                 |
| className:  "test one"                  |
| attributes:                             |
|    href:  "foo.html"                    |
|    name:  "fooAnchor"                   |
|    id:    "fooAnchor"                   |
|    class: "test one"                    |
+-----------------------------------------+
```

Changing an element's style

We can add or remove any style using the **style** or **className** property of the Element node.

```
var commentTag = document.getElementById("specificTag");

commentTag.style.backgroundColour = "#FFFF00";

commentTag.style.borderWidth="3px";
```

Element.style get access to CSS2Properties object

borderZtopZcolor → borderTopColor

Changing an element's style

With class

The className property is normally a better choice, because it allows the styles to be created outside the code, and thus be better accessible to designers.

```
var commentTag = document.getElementById("specificTag");
```

```
commentTag.className = "someClassName";
```

HTML5 introduces the classList element, which allows you to add, remove, or toggle a CSS class on an element.

```
label.classList.addClass("someClassName");
```

Browser Object Model

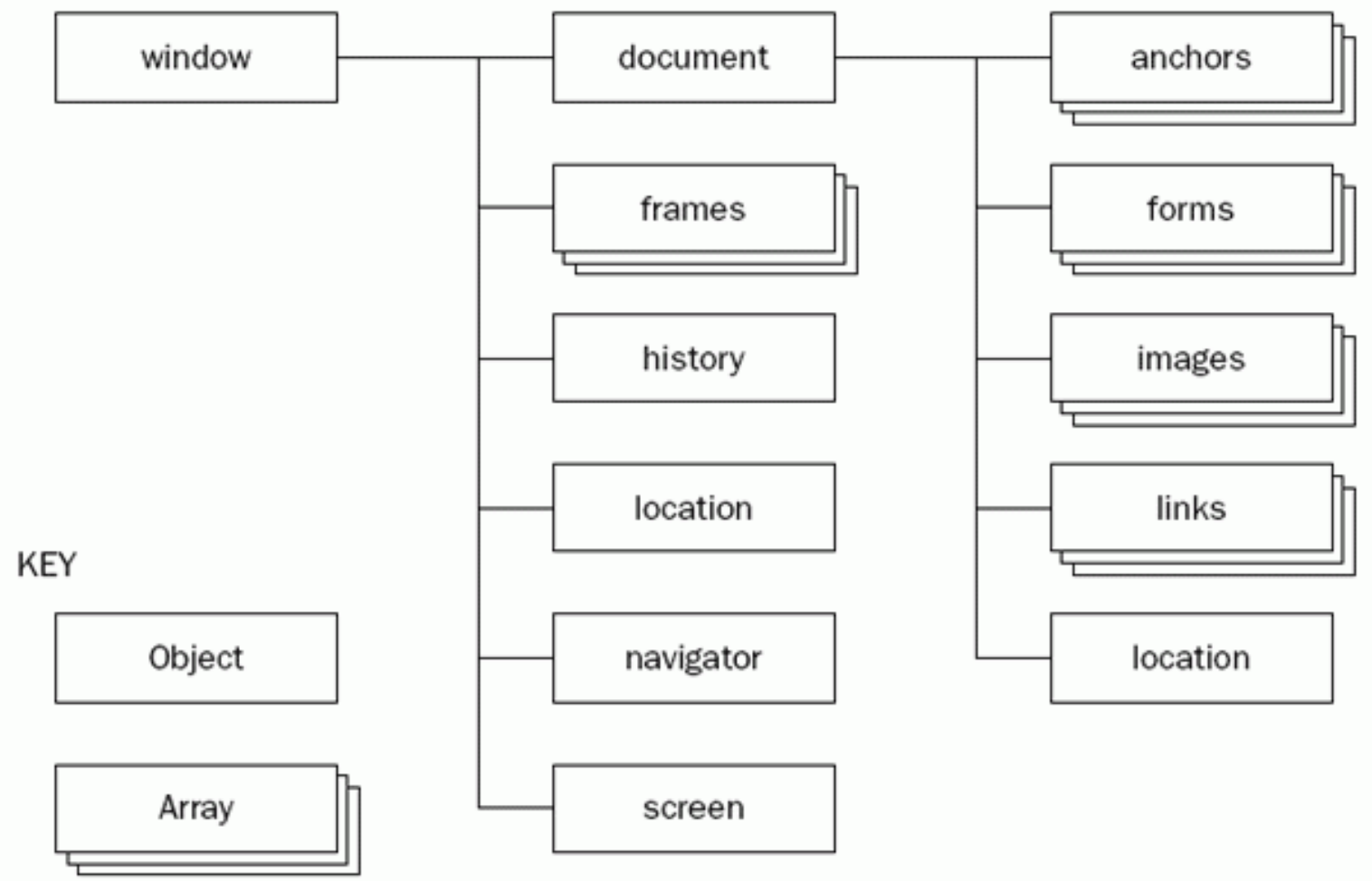


Figure 5-3

Browser Object Model

Frames: Array of the subframes of the current window

History: Manipulate browser history

Location: current url

Navigator: information such as userAgent, platform, language,..

Screen: to inspect the properties of the screen on which the current window is being rendered

Section 7 of 8

JAVASCRIPT EVENTS

JavaScript Events

A JavaScript **event** is an action that can be detected by JavaScript.

We say then that an event is *triggered* and then it can be *caught* by JavaScript functions, which then do something in response.

JavaScript Events

In the original JavaScript world, events could be specified right in the HTML markup with *hooks* to the JavaScript code (and still can).

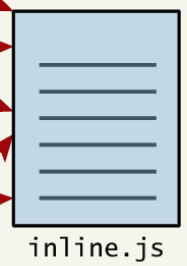
As more powerful frameworks were developed, and website design and best practices were refined, this original mechanism was supplanted by the **listener** approach.

JavaScript Events

Two approaches

Old, Inline technique

```
...
 <script type="text/javascript"  src="inline.js"></script>
...

<form name='mainForm' 'onsubmit="validate(this);">
      <input name="name" type="text" onhover="hover(this);" onfocus="focus(this);">
      <input name="email" type="text" onhover="hover(this);" onfocus="focus(this);">
      <input type="submit" onclick="validate(this);">
...
```



New, Layered Listener technique

```
...
 <script type="text/javascript"  src="listener.js"></script>
...

<form name='mainForm'>
      <input name="name" type="text">
      <input name="email" type="text">
      <input type="submit">
...
```

Inline Event Handler Approach

For example, if you wanted an alert to popZup when clicking a you might program:

<div id="example1" **onclick**="alert('hello')">Click for popZup

The problem with this type of programming is that the HTML markup and the corresponding JavaScript logic are woven together. It does not make use of layers; that is, it does not separate content from behavior.

Listener Approach

Using functions

What if we wanted to do something more elaborate when an event is triggered? In such a case, the behavior would have to be encapsulated within a function, as shown in Listing 6.12.

```
function displayTheDate() {
   var d = new Date();
   alert ("You clicked this on "+ d.toString());
}
var element = document.getElementById('example1');
element.onclick = displayTheDate;

// or using the other approach
element.addEventListener('click',displayTheDate);
```

LISTING 6.12 Listening to an event with a function

Listener Approach

Anonymous functions

An alternative to that shown in Listing 6.12 is to use an anonymous function (that is, one without a name), as shown in Listing 6.13.

```
var element = document.getElementById('example1');
element.onclick = function() {
   var d = new Date();
   alert ("You clicked this on " + d.toString());
};
```

LISTING 6.13 Listening to an event with an anonymous function

Event Object

No matter which type of event we encounter, they are all **DOM event objects** and the event handlers associated with them can access and manipulate them. Typically we see the events passed to the function handler as a parameter named *e*.

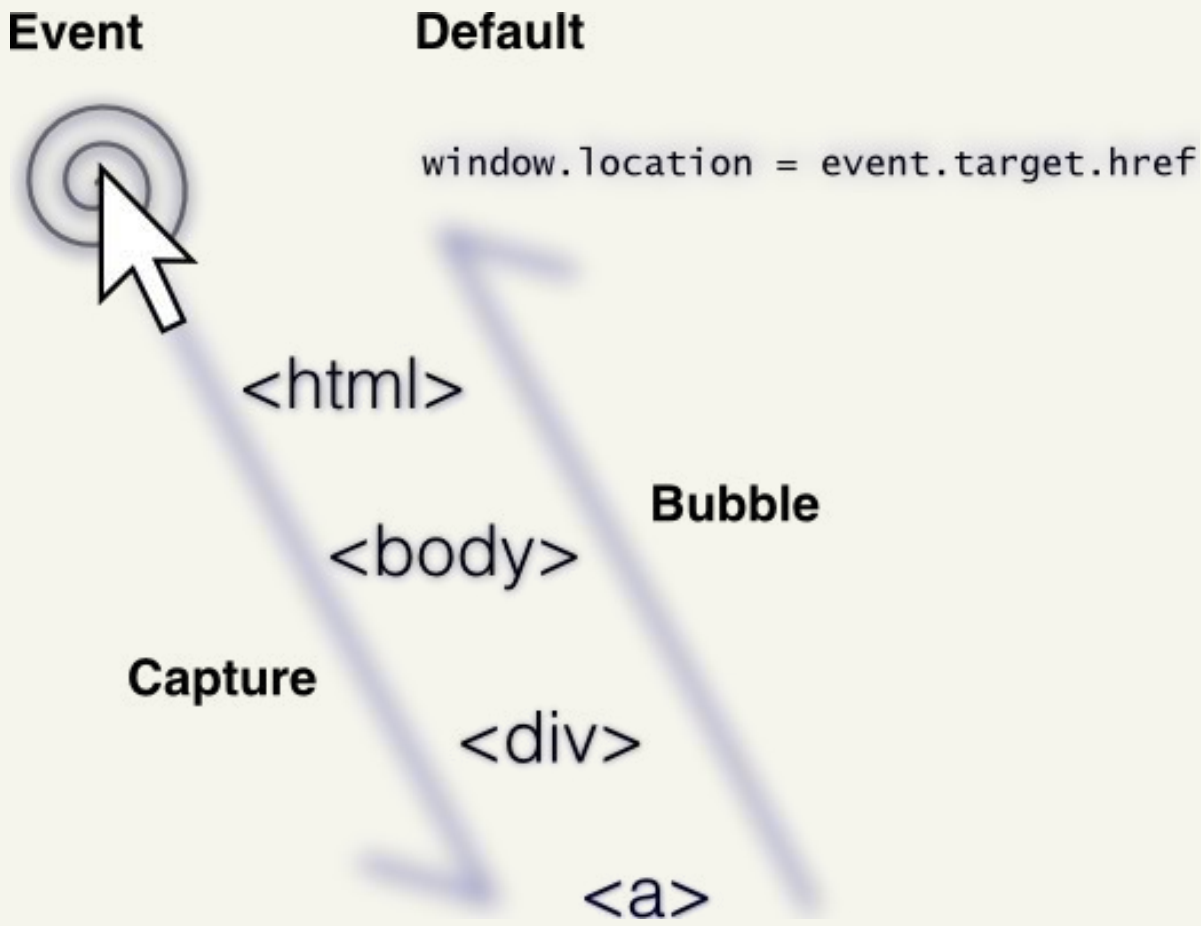
```
function doSomething(e) {
// e is the event that triggered this handler.
if (!e) var e = window.event;
// this refers to the HTML element which currently
handles the event
// e.target or e. srcElement
//refer to the HTML element the event originally took
place on
}
```

Event Object

```
<button>Click me any way you want</button>
<script>
  var button = document.querySelector("button");
  button.addEventListener("mousedown", function(event) {
    if (event.which == 1)
      console.log("Left button");
    else if (event.which == 2)
      console.log("Middle button");
    else if (event.which == 3)
      console.log("Right button");
  });
</script>
```

Each event type provides different information.

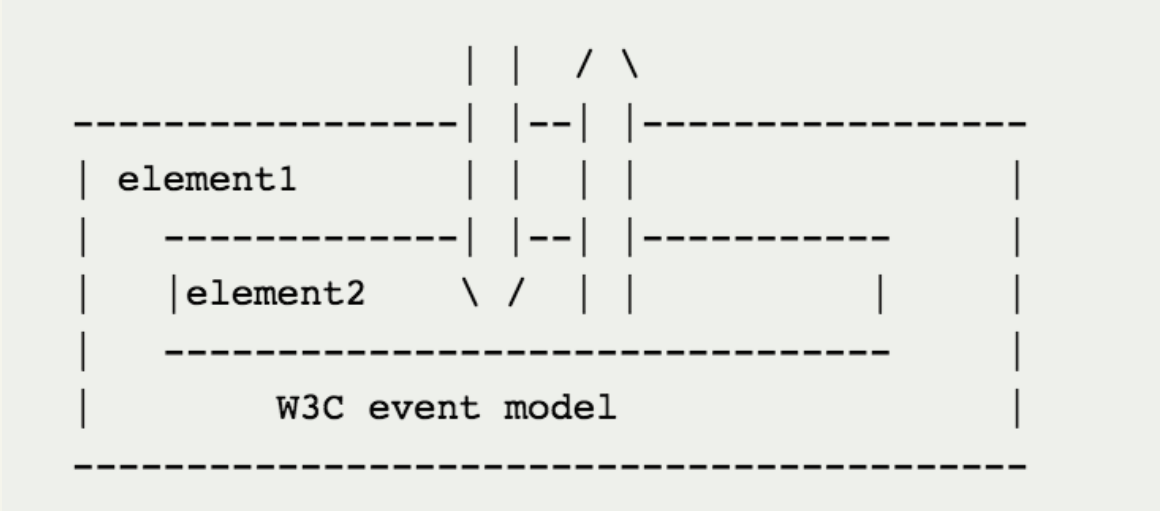
Capturing vs Bubbling



Capturing vs Bubbling

Event handlers can be registered for either the capturing phase or the bubbling phase.

```
//true → capturing
element1.addEventListener('click',doSomething2,true);
//false → bubbling (default)
element2.addEventListener('click',doSomething,false);
```



Capturing vs Bubbling

When the user clicks on element2:

- the click event starts in the capturing phase.
- the event looks if any ancestor element of element2 has a onclick event handler for the capturing phase.
- The event finds one on element1 so doSomething2() is executed.
- The event travels down to the target itself, no more event handlers for the capturing phase are found.
- The event moves to its bubbling phase and executes doSomething(), which is registered to element2 for the bubbling phase.
- The event travels upwards again and checks if any ancestor element of the target has an event handler for the bubbling phase. This is not the case, so nothing happens.

Capturing vs Bubbling

```
element1.addEventListener('click',doSomething2,false);

element2.addEventListener('click',doSomething,false);
```

What will happen clicking on element2?

Use of event bubbling:

- In Web pages as they are made today, it is simply not necessary to let a bubbling event be handled by several different event handlers.
- Users get confused
- Mainly used for default actions

Stop Propagation

Each event can stop the bubbling propagation using

```
e.stopPropagation();
```

```
function doSomething(e){
      if (!e) var e = window.event;
      //do your work here
      if (e.stopPropagation)
            e.stopPropagation();
}
```

Prevent Default

Many events have a default action associated with them.

Clicking a link, press the down arrow, rightZclick, and so on.

Default actions are executed after the registered handlers. Use event.preventDefault() to prevent the default action to be executed

```
<a href="https://developer.mozilla.org/">MDN</a>
<script>
  var link = document.querySelector("a");
  link.addEventListener("click", function(event) {
    console.log("Nope.");
    event.preventDefault();
  });
</script>
```

Event Types

There are several classes of event, with several types of event within each class specified by the W3C:

- mouse events
- keyboard events
- form events
- frame events

Mouse events

Event	Description
onclick	The mouse was clicked on an element
ondblclick	The mouse was double clicked on an element
onmousedown	The mouse was pressed down over an element
onmouseup	The mouse was released over an element
onmouseover	The mouse was moved (not clicked) over an element
onmouseout	The mouse was moved off of an element
onmousemove	The mouse was moved while over an element

Keyboard events

Event	Description
onkeydown	The user is pressing a key (this happens first)
onkeypress	The user presses a key (this happens after onkeydown)
onkeyup	The user releases a key that was down (this happens last)

Keyboard events

Example

```
<p>This page turns violet when you hold the V key.</p>
<script>
  addEventListener("keydown", function(event) {
    if (event.keyCode == 86)
      document.body.style.background = "violet";
  });
  addEventListener("keyup", function(event) {
    if (event.keyCode == 86)
      document.body.style.background = "";
  });
</script>
```

Frame Events

Frame events are the events related to the browser frame that contains your web page.

The most important event is the **onload** event, which tells us an object is loaded and therefore ready to work with. If the code attempts to set up a listener on this notZyetZloaded , then an error will be triggered.

```
window.onload= function(){

        //all JavaScript initialization here.

}
```

Frame Events

Table of frame events

Event	Description
onabort	An object was stopped from loading
onerror	An object or image did not properly load
onload	When a document or object has been loaded
onresize	The document view was resized
onscroll	The document view was scrolled
onunload	The document has unloaded

Form Events

Event	Description
onblur	A form element has lost focus (that is, control has moved to a different element, perhaps due to a click or Tab key press.
onchange	Some <input>, <textarea> or <select> field had their value change. This could mean the user typed something, or selected a new choice.
onfocus	Complementing the onblur event, this is triggered when an element gets focus (the user clicks in the field or tabs to it)
onreset	HTML forms have the ability to be reset. This event is triggered when that happens.
onselect	When the users selects some text. This is often used to try and prevent copy/paste.
onsubmit	When the form is submitted this event is triggered. We can do some preZvalidation when the user submits the form in JavaScript before sending the data on to the server.

Validating Forms

Empty field

```
document.getElementById("loginForm").onsubmit = function(e){
   var fieldValue=document.getElementByID("username").value;
   if(fieldValue==null || fieldValue== ""){
      // the field was empty. Stop form submission
      e.preventDefault();
      // Now tell the user something went wrong
      alert("you must enter a username");
   }
}
```

LISTING 6.18 A simple validation script to check for empty fields