

JavaScript: DOM and Events (Continued)

Computer Science and Engineering ■ College of Engineering ■ The Ohio State University

Lecture 28

Interactive Documents

- To make a document interactive, you need:
 - Widgets (ie HTML elements)
 - Buttons, windows, menus, etc.
 - Events
 - Mouse clicked, window closed, button clicked, etc.
 - Event listeners
 - Listen (ie wait) for events to be triggered, and then perform actions to handle them

Events Drive the Flow of Control

- This style is *event driven* programming
- Event handling occurs as a loop:
 - Program is idle
 - User performs an action
 - Eg moves the mouse, clicks a button, types in a text box, selects an item from menu, ...
 - This action generates an event (object)
 - That event is sent to the program, which responds
 - Code executes, could update document
 - Program returns to being idle

Handling Events Mechanism

- Three parts of the event-handling mechanism
 - *Event source*: the widget with which the user interacts
 - *Event object*: encapsulated information about the occurred event
 - *Event listener*: a function that is called when an event occurs, and responds to the event



Simple Example: Color Swaps

```
<p>This page illustrates changing colors</p>
<form>
  <p>
    <label> background:
      <input type="text" name="back" size="10"
        onchange="modifyColor('bg', this.value)" />
    </label> <br />
    <label> foreground:
      <input type="text" name="fore" size="10"
        onchange="modifyColor('fg', this.value)" />
    </label>
  </p>
</form>
```

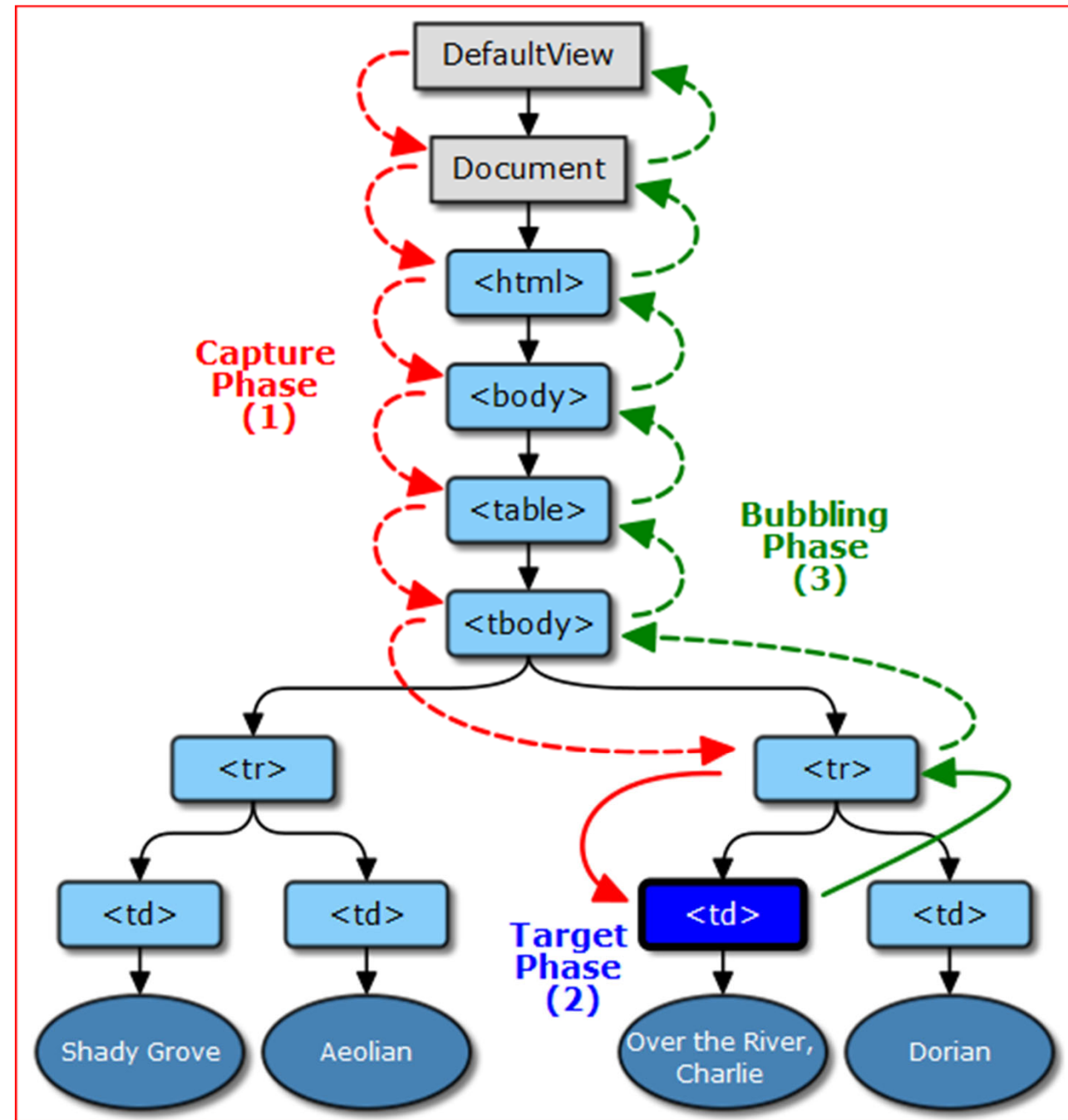
Color Swaps (JavaScript)

```
function modifyColor(place, color) {  
    if (place === "bg")  
        document.body.style.backgroundColor = color;  
    else  
        document.body.style.color = color;  
}
```

Event Propagation

- ❑ Elements are nested in tree
- ❑ When an event occurs, which element's handler(s) is(are) notified?
- ❑ First, *propagation path* is calculated: from root to smallest element
- ❑ Then event dispatch occurs in 3 phases
 1. Capture (going *down* the path)
 2. Target (smallest element)
 3. Bubble (going *up* the path, reverse of 1)

<http://www.w3.org/TR/DOM-Level-3-Events/>



Bubbling Up

- Handling is usually done in phase 2 and 3
- Example: mouse click on hyperlink
 - Handler for `<a>` element displays a pop-up ("Are you sure you want to leave?")
 - Once that is dismissed, event flows up to enclosing `<p>` element, then `<div>` then... *etc.* until it arrives at root element of DOM
 - This root element (*i.e.* **window**) has a handler that loads the new document in the current window

Programmer Tasks

- Define a handler
 - Easy, any function will do
- Register handler
 - Link (HTML) tree element with (JavaScript) function(s)
- Invoke the handler when event occurs
 - Ha! Not our job
- Get information about triggering event
 - Handler is invoked with a parameter: an event object

Registering an Event Handler

□ Three techniques, ordered from:

- Oldest (most brittle, simplest) to
- Newest (most general)

1. Inline (set in the HTML itself)

```
<a href="page.html" onclick="foo()">...
```

2. Direct property (set in JavaScript)

```
let e = ... // find source element in tree
e.onclick = foo;
```

3. Chained (set in JavaScript)

```
let e = ... // find source element in tree
e.addEventListener("click", foo, false);
```

Inline Registration (pre DOM)

- Use HTML *attributes* (vary by element type)
 - For window: `onload`, `onresize`, `onunload`,...
 - Forms & elements: `onchange`, `onblur`, `onfocus`, `onsubmit`,...
 - Mouse events: `onclick`, `onmouseover`, `onmouseout`,...
 - Keyboard events: `onkeypress`, `onkeyup`,...
- The *value* of these attributes is JavaScript code to be executed
 - Normally just a function invocation
- Example
 - `<a href="page.html" onclick="foo()">...`
- Advantage: Quick, easy, universal
- Disadvantage: mixes code with content

Direct Registration (DOM 1)

- Use *properties* of DOM element objects

- `onchange, onblur, onfocus, ...`
- `onclick, onmouseover, onmouseout, ...`
- `onkeypress, onkeyup, ...`

- Set this property to appropriate handler

```
let e = ... // find source element in tree
e.onclick = foo;
```

- Note: no parentheses!

```
e.onclick() = foo; // what does this do?
e.onclick = foo(); // what does this do?
```

- Disadvantage? No arguments to handler

- Not a problem, handler gets event object

- Real disadvantage: 1 handler/element

Example: Mouse Events

```
let divs = document.querySelectorAll("div");
for (let d of divs) {
  d.onmouseover = function() {
    this.style.backgroundColor = "red"
  }
  d.onmouseout = function() {
    this.style.backgroundColor = "blue"
  } // *this* will be the element (div)
    // that listener is registered with
}
```

Handler Registration in DOM

- ❑ Each element has a *collection* of handlers
- ❑ Add/remove handler to this collection

```
let e = ... // find source element in tree
e.addEventListener("click", foo);
```
- ❑ First parameter: event name
 - Note: no "on" in event names, ie just "click"
- ❑ Second parameter: handler function
 - This function takes an argument: event
- ❑ Third parameter: handling phase
 - Default is `false` (target or bubbling phase)
 - For capture phase (unusual) use `true`

Example: Clicks

```
let divs = document.querySelectorAll("div");
for (let d of divs) {
  d.addEventListener("click",
    function(event) {
      this.activated = this.activated || false;
      this.activated = !this.activated;
      this.style.backgroundColor =
        (this.activated ? "red" : "gray");
    });
}
```

Pitfall: Wrong this with =>

```
let divs = document.querySelectorAll("div");
for (let d of divs) {
  d.addEventListener("click",
    (event) => { // wrong this
      this.activated = this.activated || false;
      this.actitvate = !this.activated;
      this.style.backgroundColor =
        (this.activated ? "red" : "gray");
    });
}
```

Better: Use event Argument

```
let divs = document.querySelectorAll("div");
for (let d of divs) {
  d.addEventListener("click",
    (event) => { // use parameter, not this
      let t = event.currentTarget;
      t.activated = t.activated || false;
      t.activated = !t.activated;
      t.style.backgroundColor =
        (t.activated ? "red" : "gray");
    });
}
```

Summary: DOM, Events

- DOM: Document Object Model
 - Programmatic way to use document tree
 - Get, create, delete, and modify nodes
- Event-driven programming
 - Source: element in HTML (a node in DOM)
 - Handler: JavaScript function
 - Registration: in-line, direct, chained
 - Event is available to handler for inspection