Events

Sridhar Alagar

Event

An *event* is a signal that something has happened

All DOM nodes generate such signals

List of DOM Events:

- Mouse events: click, contextmenu, mouseover/mouseout, mousedown/mouseup, mouse move

Lists of DOM Events

Mouse events:

click, contextmenu, mouseover/mouseout, mousedown/mouseup, mouse move

Keyboard events:

keydown, keyup

Form element events:

submit, focus

Document events:

DOMContentLoaded - when the HTML is loaded and processed, DOM is fully built

CSS events:

transitioned – animation finished

There are many more events…

Event Handlers

To react on an event, we can assign a handler

- a function that runs when an event occurs

Handlers are a way to run JavaScript code in case of user actions

There are several ways to assign a handler

- Setting HTML event attribute
- Setting DOM event property
- Using addEventListener

Setting HTML event attribute

A handler can be set in HTML with an attribute named on<event>

For instance, to assign a click handler for an input, we can use onclick

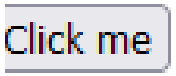
```
1  <input value="Click me" onclick="alert('Click!')" type="button">
```

Old style…bad practice…don’t use

Setting DOM property

A handler can be assigned to a DOM property named on<event>

```
1  <input id="elem" type="button" value="Click me">
2  <script>
3    elem.onclick = function() {
4      alert('Thank you');
5    };
6  </script>
```



There's only one onclick property, so can't assign more than one event handler

Note: DOM properties are case sensitive. It is 'onclick' not 'onClick'

Accessing the element: this

The value of ‘this’ inside a handler is the element. The one which has the handler, not the one in which event happened

addEventListener

Can assign multiple handlers to an event

Syntax:

```
element.addEventListener(event, handler, [options]);
```

Options:

- once: if true, handler removed automatically after it triggers
- capture: covered later
- passive: used to prevent calling preventDefault()

addEventlistener

```
<input id="elem" type="button" value="Click me"/>

<script>
  function handler1() {
    alert('Thanks!');
  };

  function handler2() {
    alert('Thanks again!');
  }

  elem.onclick = () => alert("Hello");
  elem.addEventListener("click", handler1); // Thanks!
  elem.addEventListener("click", handler2); // Thanks again!
</script>
```

addEventListener

Handlers for some event can be set only with addEventListener

- For example, the ‘DOMContentLoaded’ event

Using addEventListener() is the ***preferred*** way

removeEventListener

To remove a handler, must pass the same function as was assigned

This wont work

```
elem.addEventListener( "click" , () => alert('Thanks!'));
// ....
elem.removeEventListener( "click", () => alert('Thanks!'));
```

Same code, but two different functions

removeEventListener

To remove a handler, must pass the same function as was assigned

This is the right way

```
function handler() {
  alert( 'Thanks!' );
}

input.addEventListener("click", handler);
// ....
input.removeEventListener("click", handler);
```

Event object

When an event occurs:

1. Browser creates an event object
2. Fills in the details about the event
3. Passes it to the handler as an argument

```
<input type="button" value="Click me" id="elem">

<script>
  elem.onclick = function(event) {
    // show event type, element and coordinates of the click
    alert(event.type + " at " + event.currentTarget);
    alert("Coordinates: " + event.clientX + ":" + event.clientY);
  };
</script>
```

Quiz

There’s a button in the variable. There are no handlers on it.

Which handlers run on click after the following code? Which alerts show up?

```
1  button.addEventListener("click", () => alert("1"));
2
3  button.removeEventListener("click", () => alert("1"));
4
5  button.onclick = () => alert(2);
```

Event Bubbling

When an event happens on an element,

- it first runs the handlers on it,
- then on its parent,
- then all the way up on other ancestors till the document object

Almost all events bubble

- focus event doesn't bubble, and few others too

event.target

The most deeply nested element that triggered the event is called a target element, accessible as event.target

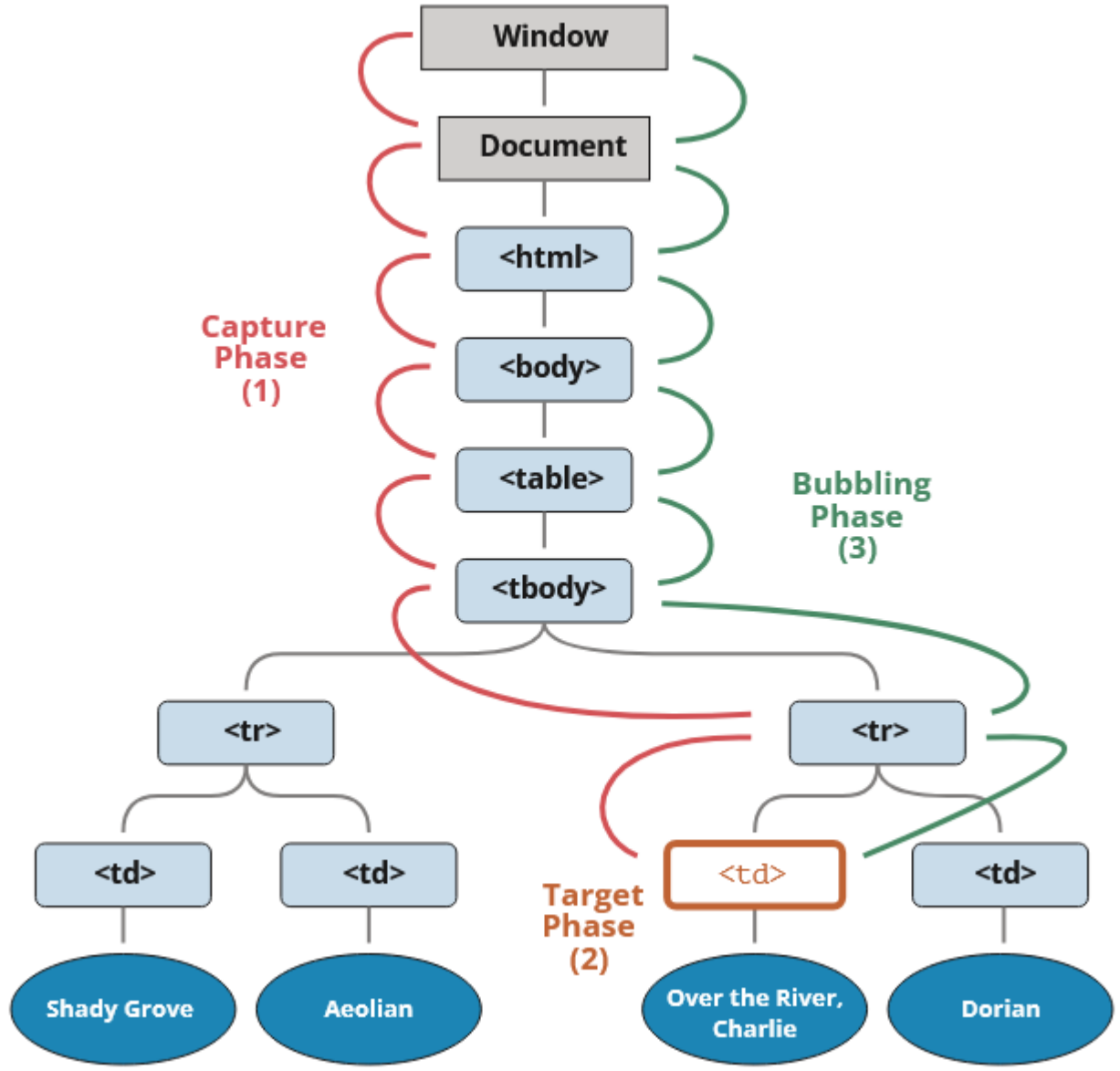
Note the differences from **`this`(= `event.currentTarget`)**:

- **`event.target`** – is the "target" element that initiated the event, it doesn't change through the bubbling process
- **`this`** – is the "current" element, the one that has a currently running handler on it

Event Capturing

Inverse of bubbling – event propagates from least to most nested element

Very rarely used



quiz

Given a html that has table with many rows (<tr>) and columns (<td>)

If a <td> is clicked, it’s background color should be set to red.

Where will be the handler set?

Some more on events to be aware of…

- Can stop event propagation using event.stopPropogation()
- Browser may have default actions on some events. Can prevent the browser action using event.preventDefault()
- Events can be generated from JavaScript. Use Event objects.
  - Useful for automated testing
  - Software generated events can be custom events too

Event concurrency

- Events are serialized and processed one-at-a-time
- Event handling does not interleave with other JavaScript execution.
  - Handlers run to completion
  - No multi-threading
- Make reasoning about concurrency easier
  - Rarely need locks
- Background processing is harder than with threads

Sources

1. https://javascript.info/events
2. MDN - Introduction to events