Browser Event Handling

Junghoo Cho

cho@cs.ucla.edu

Event Object

- Event object is passed as the (only) argument to the event ha function
  - *Event object*: detailed information on the triggered event
    - `event.target`: the target to which the event was originally triggered
    - `event.type`: event type
    - ...

```
<script>
    document.body.addEventListener("click", function (event) {
        console.log(event.target.id);
    });
</script>
<body id="body_id">
    ...
</body>
```

Event Handler

- We can set our own event handler to catch any DOM event
- The original event handler is also invoked after our custom e handler
- Example

```
<body>
    <a href="http://google.com" onclick="console.log('Clicked!
</body>
```

  - To prevent the original handler, call `event.preventDefault()`
- If event handler is set using `onX="stmt;"` inside HTML tag:
  - `stmt;` is wrapped into a function the the single input parameter `eve`

Event Bubbling

- Most DOM events “bubble up” through the DOM tree
  - Target’s ancestors get the event all the way through the `document` (a
    sometimes even `window`) object
  - Exceptions: focus, scroll, …
  - To stop event propagation, call `event.stopPropagation()` inside ev
- Example

```
<section onclick="console.log('section!');">
    <h1 onclick="console.log('h1!');">Please click
        <em onclick="console.log('em!');">here</em>
    </h1>
</section>
```

Single-Thread Execution

- JavaScript code in a browser is executed in a ***single thread***
- ***No*** two event handlers will *ever* run at the same time
- Document contents are never updated by two threads simul
  - No worries about locks, deadlock, or critical region
- But web browser “stops” responding to user input while scrip running

JavaScript Execution Timeline in Browser

1. `document` object is created and `document.readyState` is set to `loading`
2. Browser downloads and parses the page. Scripts are downloaded and exec synchronously in the order they appear in the page (if no `async`)
    - `async` script starts to be downloaded asynchronously in the background and gets ex as they are available
3. Once the page is completely parsed, `document.readyState` is set to `intera`
4. Browser fires `DOMContentLoaded` event and calls `document.onload` callback
5. `document.readyState` is set to `complete`
6. Browser waits for events and calls appropriate event handlers

Execution Timeline Example

```
<html>
<head><title>JavaScript Example</title></head>
<body>Click on this document!</body>
<script>
    let colors = [ "yellow", "blue", "red" ];
    let i=0;
    function ChangeColor(event) {
        document.body.style.backgroundColor = colors[i++%3];
    }
    document.body.addEventListener("click", ChangeColor);
</script>
</html>
```

- Q: What will happen if we move the `<script>...</script>` `<body>...</body>`?

Notes on JavaScript Execution

- HTML DOM object manipulation can be done only after the
  been parsed and loaded, not before
- To run initialization code, set the `onload` handler with the ini
  code
- To run final cleanup code, set the `onunload` handler

`window`: Global Object (1)

- `window` object is the “global object” within a browser
  - All global variables and functions become properties and methods
    - e.g., `document` is in fact `window.document`
- `window.location`: the URL of the current page
  - By setting this property, we can load a different page
- `window.history`: browsing history
  - `window.history.back()`, `window.history.forward()`

`window`: Global Object (2)

- `window.alert()`, `confirm()`, `prompt()`: open a dialog box

```
alert("hello, world!");            // returns nothing
response = confirm("Click OK to proceed, Cancel to return");
                                   // returns boolean
name = prompt("Type your name");   // returns string
```

References

- DOM Technical Reports: https://www.w3.org/DOM/DOMTR
- DOM Level 3 Events: https://www.w3.org/TR/DOM-Level-3-E
- Reference for common CSS property names in JavaScript: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference