Client/Server Programming for Internet Applications

TCSS460

Summer 2020

Lecture 5 JavaScript (Part II) - Events

TACOMA

Events

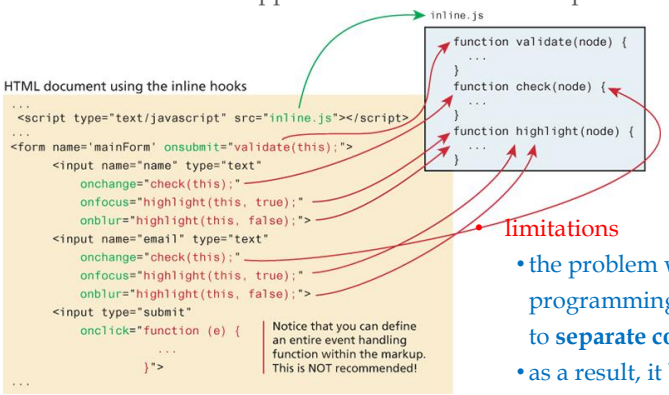
- a JavaScript **event** is an **action** that can be detected by JavaScript
  - this action is handled by code we write
- three approaches

1. **inline event-handling approach** (embedded within markup)
2. attaching **callbacks** to event properties
3. using **event listeners**

2

Inline Event-Handling Approach

this approach leads to a mess of dependencies…



- limitations
  - the problem with this type of programming is that it is difficult to **separate content** from **behavior**
  - as a result, it becomes extremely difficult to **maintain** code
  - difficulty for designers to work **separately** from programmers

sometimes referred to as **spaghetti coding**

Randy Connolly, Ricardo Hoar, Fundamentals of Web Development (2nd Edition), 2017 3

Event Property Approach

- this approach allows us to separate specification of an event handler from markup
  - supported by all browsers

```
var myButton = document.getElementById('example');
myButton.onclick = alert('some message');
```

  - **first line**: creates a temporary variable for the HTML element that will trigger the event
  - **second line**: attaches the button element's `onclick` event to the event handler (which invokes `alert()` function)
- **advantage**: code can be written anywhere (internal & external)
- **limitation**: one handler can respond to any given element event

4

Event Listener Approach

- all modern browsers support the event listener approach
- is considered to be the **preferred approach**

```
var myButton = document.getElementById('example');
myButton.addEventListener('click', alert('some message'));
myButton.addEventListener('mouseout', alert('another message'));
```

- `addEventListener()` function is used to **register** a handler for the event specified in the **first** parameter
- **second** parameter is the **handler** for the event

5

Event Listener Approach (cont'd)

Common Properties and Methods of the Event Object

Event	Description
`bubbles`	Indicates whether the event bubbles up through the DOM
`cancelable`	Indicates whether the event can be cancelled
`target`	The object that generated (or dispatched) the event
`type`	The type of the event

Mouse Events in JavaScript

Event	Description
`click`	The mouse was clicked on an element
`dblclick`	The mouse was double clicked on an element
`mousedown`	The mouse was pressed down over an element
`mouseup`	The mouse was released over an element
`mouseover`	The mouse was moved (not clicked) over an element
`mouseout`	The mouse was moved off of an element
`mousemove`	The mouse was moved while over an element

Keyboard Events in JavaScript

Event	Description
`keydown`	The user is pressing a key (this happens first)
`keypress`	The user presses a key (this happens after keydown)
`keyup`	The user releases a key that was down (this happens last)

6

Event Listener Approach (cont'd)

Form Events in JavaScript

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

- has all the advantages as event property approach
- one more key **advantage**…
  - **multiple handlers can be assigned to a single object's event**

7

Event Object

ex6

- when an **event** is triggered, the browser will construct an **event object** that contains information about the event
- **event handlers** can access this event object simply by including it as a parameter to the **callback function**
  - this object parameter is often named **e**

```
var div = document.querySelector('div#example');
div.addEventListener('click', function(e) {
// find out where the user clicked
var x = e.clientX;
var y = e.clientY;
// output the information for debugging purposes
console.log(e.type + ' event triggered by ' + e.target);
console.log(' at location ' + x + ' ' + y);
// …
});
```

8

Event Types
→ mouse events

- **mouse events** are defined to capture a range of interactions driven by the mouse

Event	Description
`click`	The mouse was clicked on an element
`dblclick`	The mouse was double clicked on an element
`mousedown`	The mouse was pressed down over an element
`mouseup`	The mouse was released over an element
`mouseover`	The mouse was moved (not clicked) over an element
`mouseout`	The mouse was moved off of an element
`mousemove`	The mouse was moved while over an element

- many mouse events can be sent at a time
  - user could be moving the mouse off of one `<div>` and onto another in the same moment, triggering `mouseon` and `mouseout` events as well as the `mousemove` event

9

Event Types
→ mouse events → example

ex8

```
<script type="text/javascript">
      var main = document.querySelector("main");
      main.addEventListener("mousedown", function (e) {
            if (e.target && e.target.id.toLowerCase() == "par1") {
              e.target.style.color = "SlateBlue";
            }
      });
      main.addEventListener("mouseup", function (e) {
            if (e.target && e.target.id.toLowerCase() == "par1") {
              e.target.style.color = "DodgerBlue";
            }
      });
      main.addEventListener("mousemove", function (e) {
            if (e.target && e.target.id.toLowerCase() == "par1") {
              e.target.style.color = "Tomato";
            }
      });

      main.addEventListener("mouseout", function (e) {
            if (e.target && e.target.id.toLowerCase() == "par1") {
              e.target.style.color = "Black";
            }
      });
      main.addEventListener("dblclick", function (e) {
            if (e.target && e.target.id.toLowerCase() == "par1") {
              selectedText = window.getSelection().toString();
              document.execCommand("copy"); // copy to clipboard
              //console.log(selectedText);
      }
      });
</script>
```

10

Event Types
→ keyboard events

ex9

- **keyboard events** are useful within input fields
- validate an email address or send anonymous request for a dropdown list of suggestions with each key press

Event	Description
keydown	The user is pressing a key (this happens first)
keypress	The user presses a key (this happens after keydown)
keyup	The user releases a key that was down (this happens last)

```
<body>
     <input type="text" id="key">
     <p style='font-family:Arial; color:DodgerBlue;' id="text"></p>
</body>
</html>
<script type="text/javascript">
     document.getElementById("key").addEventListener("keydown",
     function (e) {
     var keyPressed = e.keyCode;
     // get the raw key code
     var character = String.fromCharCode(keyPressed);
     // update paragraph tag as keys are pressed
     document.getElementById("text").innerHTML += character;
     });
</script>
```

11

Goals of This Lecture

- You should be comfortable with using the DOM
- You should be able to know how to manipulate HTML elements using JavaScript
- You should be comfortable with using getElementById, getElementByClassName, getElementByTagName, querySelector, querySelectorAll
- You should be able to know how to create event listeners
- You should be familiar with mouse and keyboard events

TCSS 460 - Summer 2020

12

Module Topics

- JavaScript (Part I)
- JavaScript (Part II)

TCSS 460 - Summer 2020

13