More DOM and events

Michael Chang

Spring 2020

Announcements

assign2 out

Due next Wed

Project info up

Proposal due next Mon

Plan for today

More about JS events

- Default behavior, propagation

More DOM techniques

- Adding/removing elements
- Data attributes
- Traversing the tree

If time, introduction to JS classes

JS syntax aside

Object destructuring

```
let a = 42, b = "193X";
/* Variable names become object keys */
let obj = { a, b };

let obj2 = { c: 193, d: null };
/* Object keys become variables */
let { c, d } = obj2;
```

DOM events aside

Older-style events

```
<button onclick="/* js code */">...</button>
button.onclick = /* js function */;
```

Best practice: don't use these

These are older, less flexible

#1 breaks separation of concerns

Use `addEventListener` instead

More HTML elements

`<label>`: associate labels to inputs

- Can contain the input or use `for` attribute with id
- Important for accessibility + easier navigation (e.g. click label to focus input)

`<form>`: group a interactors

- Typically used to send data to server
- But gives us some nice features, e.g. Enter key on input

Event default behavior

Some actions have default behaviors

- Clicking a link or submit button
- Typing text in an input

Event handler called before default

- Default action will happen afterward

To prevent default action

- Call `event.preventDefault()`

Modifying the DOM so far

Approach

- Write all the elements into the HTML up front
- Set `display: none` (e.g. use a "`hidden`" class) on unused elements
- Add/remove display from JS

To be clear, this is a good approach

But what if we need more control?

- Many similar elements, don't want to copy/paste
- Variable number of elements

Creating elements

`document.createElement(tag)`

Create new element with tag (e.g. "img")

`node.cloneNode(deep)`

Shallow or deep copy of node

Should always pass arg (default changed)

Not added to tree

Set attributes, add children

Then add to tree in desired location

Traversing and changing the tree

`node.parentNode`

`node.childNodes`

Traverse the DOM tree

`node.appendChild(childNode)`

Add childNode to the end of node

`node.remove()`

Remove node from the tree (still valid object)

Aside: Node vs. Element

An Element is a type of Node

Represents an HTML tag

Another type is a Text node

Represents the text inside elements

Often most useful to traverse elements

`elem.children`: return the **elements** under elem

StackOverflow: childNodes vs. children

I'll probably use them interchangeably

Event bubbling

When an event fires

- Start at `event.target`
- For each event handler, in order they were added
  - Call event handler
- Repeat on `parentNode`
  - `event.currentTarget` updated, `event.target` doesn't

This is called bubbling

- To prevent, call `event.stopPropagation()`

Aside: event capturing

When an event fires

- Start at root, with `event.target` set
- Call each capturing event handler
- Repeat on child along path to `event.target`

- Call each normal handler
- Repeat on `parentNode`

Adding capturing handlers

- `elem.addEventListener(type, fn, true)`

Recommendation: use capturing handlers sparingly

- Sometimes useful, but gets messy (e.g. body handler fired on every event)

Data attributes

data-* HTML attribute

- Stores information with the element
- Can be accessed in CSS or JS

JS: `elem.dataset`

- Object whose keys are the attr name (after `data-`)
- Values are strings

CSS: `[data-key=value]`

- Can be used with any attribute
- Quote value if special chars
- Can be combined with other selectors
  - E.g. `p[data-myattr="some value"]`

Data attributes tradeoffs

Pros

- Associate information directly with elements
- No additional meaning (c.f. CSS classes, ids)
- Can read/write in JS, find nodes matching value

Cons

- Puts data into your HTML, when it might belong in JS
  - E.g. use an array or object
- Worse performance if you need to query a lot
- Can only store strings, with some length limits

Summary

So far

- JavaScript language essentials
- DOM and events

Next time

- Classes and modules
- JSON, data, and APIs