Advanced DOM & Events

> 🎓 **CS Perspective:** This section deepens DOM understanding with **event propagation**—specifically the capture and bubble phases, which implement a two-phase **event dispatch algorithm**. Event delegation leverages **event bubbling** to handle events efficiently (O(1) listeners instead of O(n)), demonstrating the **proxy pattern**. The Intersection Observer API implements an efficient **lazy evaluation** strategy for viewport-based triggers, far more performant than polling scroll positions. Smooth scrolling and lazy loading relate to **perceived performance optimization**—improving user experience through progressive rendering. The distinction between live (HTMLCollection) and static (NodeList) collections shows different **iteration semantics** with implications for concurrent modification.

📌 Selecting Elements

```
// Special elements
document.documentElement;  // <html>
document.head;
document.body;

// Query methods
document.querySelector('.class');          // First match
document.querySelectorAll('.class');       // NodeList (static)
document.getElementById('id');             // Single element
document.getElementsByTagName('button');   // HTMLCollection (live)
document.getElementsByClassName('btn');    // HTMLCollection (live)
```

NodeList vs HTMLCollection

NodeList	HTMLCollection
Static (snapshot)	Live (updates)
`querySelectorAll`	`getElementsBy*`

📌 Creating & Inserting Elements

```
// Create element
const message = document.createElement('div');
message.classList.add('cookie-message');
message.innerHTML = 'We use cookies <button>Got it!</button>';

// Insert element
header.prepend(message);     // First child
header.append(message);      // Last child (moves element!)
header.before(message);      // Before header
```

```
header.after(message);       // After header

// Clone to insert multiple times
header.append(message.cloneNode(true));

// Insert HTML string
container.insertAdjacentHTML('beforeend', htmlString);
// Positions: 'beforebegin', 'afterbegin', 'beforeend', 'afterend'
```

📌 Deleting Elements

```
// Modern
message.remove();

// Old way
message.parentElement.removeChild(message);
```

📌 Styles

Inline Styles

```
message.style.backgroundColor = '#37383d';
message.style.width = '120%';

// Read inline styles only
console.log(message.style.backgroundColor); // Works
console.log(message.style.color);           // '' (not inline)
```

Computed Styles

```
getComputedStyle(message).color;   // Actual applied style
getComputedStyle(message).height;  // Returns string '50px'

// Modify using computed
message.style.height =
  Number.parseFloat(getComputedStyle(message).height) + 30 + 'px';
```

CSS Custom Properties (Variables)

```
document.documentElement.style.setProperty('--color-primary',
'orangered');
```

📌 Attributes

```
const logo = document.querySelector('.nav__logo');

// Standard attributes
logo.alt;                    // Read
logo.alt = 'New alt text';   // Write
logo.src;                    // Absolute URL
logo.getAttribute('src');    // Relative URL

// Non-standard attributes
logo.getAttribute('designer');
logo.setAttribute('company', 'Bankist');

// Data attributes (data-*)
// <img data-version-number="3.0">
logo.dataset.versionNumber;  // '3.0' (camelCase!)
```

📌 Classes

```
logo.classList.add('c', 'j');
logo.classList.remove('c');
logo.classList.toggle('c');
logo.classList.contains('c');  // Not 'includes'!

// ❌ Don't use (overwrites all classes)
logo.className = 'new-class';
```

📌 Smooth Scrolling

```
// Modern (best)
section1.scrollIntoView({ behavior: 'smooth' });

// Manual calculation
const s1coords = section1.getBoundingClientRect();
window.scrollTo({
  left: s1coords.left + window.pageXOffset,
  top: s1coords.top + window.pageYOffset,
  behavior: 'smooth'
});
```

getBoundingClientRect()

Returns position relative to **viewport**:

```
element.getBoundingClientRect();
// { x, y, width, height, top, right, bottom, left }

// Current scroll position
window.pageXOffset;  // or window.scrollX
window.pageYOffset;  // or window.scrollY

// Viewport dimensions
document.documentElement.clientHeight;
document.documentElement.clientWidth;
```

📌 Event Types

Mouse Events

Event	Description
`click`	Click
`dblclick`	Double click
`mouseenter`	Mouse enters (no bubble)
`mouseleave`	Mouse leaves (no bubble)
`mouseover`	Mouse enters (bubbles)
`mouseout`	Mouse leaves (bubbles)

Keyboard Events

Event	Description
`keydown`	Key pressed
`keyup`	Key released
`keypress`	Character typed (deprecated)

Form Events

Event	Description
`submit`	Form submitted
`focus`	Element focused
`blur`	Element lost focus
`input`	Input value changed

Event	Description
change	Value changed (on blur)

📌 Event Handling

```
// Add listener
const alertH1 = function(e) {
  alert('Mouse entered!');
};
h1.addEventListener('mouseenter', alertH1);

// Remove listener (must use same function reference)
h1.removeEventListener('mouseenter', alertH1);

// Remove after first trigger
h1.addEventListener('mouseenter', function(e) {
  alert('Once!');
  h1.removeEventListener('mouseenter', this);
});
```

📌 Event Propagation (Bubbling & Capturing)

Events travel in 3 phases:

1. **Capturing** - Root → Target
2. **Target** - At the element
3. **Bubbling** - Target → Root (default)

```
// e.target = Element that triggered event
// e.currentTarget = Element with listener attached (= this)

document.querySelector('.nav__link').addEventListener('click', function(e)
{
  console.log('LINK', e.target, e.currentTarget);

  // Stop bubbling
  e.stopPropagation();
});

// Listen during capture phase
element.addEventListener('click', handler, true);
```

📌 Event Delegation

Add listener to **parent** instead of each child:

```
// ❌ Inefficient
document.querySelectorAll('.nav__link').forEach(el => {
  el.addEventListener('click', function(e) {
    e.preventDefault();
    // ...
  });
});

// ✅ Event delegation
document.querySelector('.nav__links').addEventListener('click',
function(e) {
  e.preventDefault();

  // Match target
  if (e.target.classList.contains('nav__link')) {
    const id = e.target.getAttribute('href');
    document.querySelector(id).scrollIntoView({ behavior: 'smooth' });
  }
});
```

📌 DOM Traversing

```
const h1 = document.querySelector('h1');

// Downward (children)
h1.querySelectorAll('.highlight');  // Deep search
h1.childNodes;                      // All nodes (incl. text)
h1.children;                        // Direct element children
h1.firstElementChild;
h1.lastElementChild;

// Upward (parents)
h1.parentNode;
h1.parentElement;
h1.closest('.header');              // Nearest ancestor matching selector

// Sideways (siblings)
h1.previousElementSibling;
h1.nextElementSibling;
h1.parentElement.children;          // All siblings
```

closest() vs querySelector()

- `querySelector()` finds **children**
- `closest()` finds **parents**

📌 Intersection Observer API

Observe when element enters/exits viewport:

```
const obsCallback = function(entries, observer) {
  entries.forEach(entry => {
    if (entry.isIntersecting) {
      // Element is visible
      entry.target.classList.remove('hidden');
      observer.unobserve(entry.target);  // Stop observing
    }
  });
};

const obsOptions = {
  root: null,           // null = viewport
  threshold: 0.15,      // 15% visible triggers callback
  rootMargin: '-90px'   // Offset from root
};

const observer = new IntersectionObserver(obsCallback, obsOptions);
observer.observe(section1);
```

Sticky Navigation Example

```
const stickyNav = function(entries) {
  const [entry] = entries;
  if (!entry.isIntersecting) {
    nav.classList.add('sticky');
  } else {
    nav.classList.remove('sticky');
  }
};

const headerObserver = new IntersectionObserver(stickyNav, {
  root: null,
  threshold: 0,
  rootMargin: `-${navHeight}px`
});

headerObserver.observe(header);
```

📌 Lazy Loading Images

```
<img src="lazy-img.jpg" data-src="real-img.jpg" class="lazy-img">
```

```
const loadImg = function(entries, observer) {
  const [entry] = entries;
  if (!entry.isIntersecting) return;

  // Replace src
  entry.target.src = entry.target.dataset.src;

  // Remove blur after load
  entry.target.addEventListener('load', function() {
    entry.target.classList.remove('lazy-img');
  });

  observer.unobserve(entry.target);
};

const imgObserver = new IntersectionObserver(loadImg, {
  root: null,
  threshold: 0,
  rootMargin: '200px'  // Start loading before visible
});

imgTargets.forEach(img => imgObserver.observe(img));
```

📌 Lifecycle DOM Events

```
// HTML parsed, DOM ready
document.addEventListener('DOMContentLoaded', function(e) {
  console.log('DOM ready');
});

// Page fully loaded (images, etc.)
window.addEventListener('load', function(e) {
  console.log('Page loaded');
});

// Before leaving page
window.addEventListener('beforeunload', function(e) {
  e.preventDefault();
  e.returnValue = '';  // Shows confirmation dialog
});
```

📌 Passing Arguments to Event Handlers

```
// Using bind (sets this)
const handleHover = function(e) {
  // this = opacity value
```

```
  link.style.opacity = this;
};

nav.addEventListener('mouseover', handleHover.bind(0.5));
nav.addEventListener('mouseout', handleHover.bind(1));
```

🧪 Quick Reference Cheatsheet

```
// Select
document.querySelector('.class');
document.querySelectorAll('.class');

// Create & Insert
document.createElement('div');
parent.append(element);
parent.prepend(element);
element.insertAdjacentHTML('beforeend', html);

// Delete
element.remove();

// Classes
el.classList.add/remove/toggle/contains('class');

// Traverse
el.closest('.class');     // Up
el.children;              // Down
el.nextElementSibling;    // Sideways

// Scroll
el.scrollIntoView({ behavior: 'smooth' });

// Event delegation
parent.addEventListener('click', e => {
  if (e.target.classList.contains('child')) { }
});

// Intersection Observer
new IntersectionObserver(callback, { root, threshold, rootMargin });
```

✅ Exam Tips

1. `querySelectorAll` returns **static** NodeList, `getElementsBy*` returns **live** HTMLCollection
2. `append/prepend` **moves** element; use `cloneNode(true)` to duplicate
3. `e.target` = clicked element, `e.currentTarget` = element with listener
4. Use **event delegation** for better performance
5. `closest()` is like `querySelector` but goes **up** the DOM

6. `dataset` converts `data-version-number` to `versionNumber` (camelCase)
7. Intersection Observer `threshold: 0` means any pixel visible
8. `rootMargin` accepts negative values for offset
9. Remove lazy-img blur **after** image loads, not immediately
10. `classList.contains()` not `includes()`
11. `DOMContentLoaded` fires before images load
12. `e.stopPropagation()` prevents bubbling