Event delegation and event bubbling/capturing

Let's break down **Event Delegation**, **Event Bubbling**, and **Event Capturing** — three essential concepts for efficient DOM event handling in JavaScript.

🔁 Event Flow in the DOM

Whenever an event occurs (e.g., a click), it goes through **three phases**:

1. **Capturing Phase** – Event starts from the `window` and travels **down** to the target element.
2. **Target Phase** – The event reaches the **target element**.
3. **Bubbling Phase** – Event bubbles **up** from the target to the root (`document`).

```
window ↓        ← Capturing
element        ← Target
window ↑        ← Bubbling
```

📍 Event Bubbling

- Default behavior in JavaScript.
- Events propagate **from the target element upward** through its ancestors.

Example:

```
<ul id="list">
  <li>Item 1</li>
</ul>

<script>
  document.getElementById("list").addEventListener("click", function () {
    console.log("UL clicked");
  });

  document.querySelector("li").addEventListener("click", function () {
    console.log("LI clicked");
  });
</script>
```

Click Output (on `<li>`):

```
LI clicked
UL clicked
```

🔁 The `li` click handler fires first (target), then the `ul` handler (bubble).

🔽 Event Capturing (a.k.a. Trickling)

- The opposite of bubbling: the event flows **from top to bottom**.
- You must **enable it manually** using the third argument in `addEventListener`.

Syntax:

```
element.addEventListener("click", handler, true); // true = capture phase
```

Example:

```
document.getElementById("list").addEventListener(
  "click",
  function () {
    console.log("UL clicked (capturing)");
  },
  true
);
```

🔁 This handler will fire **before** the child's click handler.

⚡ Event Delegation

What is it?

Event delegation is a technique where a **single event listener** on a **parent element** handles events for **its children**, using **bubbling**.

Why use it?

- ✅ Better **performance**: one listener vs. many.
- ✅ Useful for **dynamic elements** (e.g., elements added after page load).
- ✅ Cleaner, DRY code.

Example:

```
<ul id="menu">
  <li>Home</li>
  <li>About</li>
  <li>Contact</li>
</ul>

<script>
  document.getElementById("menu").addEventListener("click", function (e) {
    if (e.target.tagName === "LI") {
      console.log("Clicked:", e.target.textContent);
    }
  });
</script>
```

🟢 No need to add listeners to each `<li>` — `ul` handles everything.

🔐 How It Works

- The parent listens for events on itself.

- During bubbling, the event reaches the parent.
- Inside the handler, `e.target` tells you **which child** was clicked.
- Use `e.stopPropagation()` to **stop bubbling** if needed.

🔍 Summary Table

Concept	Description	Trigger Order
Capturing	Top → down phase (optional)	Outer → Inner
Target	Event hits the actual target element	
Bubbling	Bottom → up phase (default in JS)	Inner → Outer
Delegation	One parent handles events for many children	Uses bubbling

---

🚀 Best Practices

- Use **event delegation** for dynamic elements (e.g., todo lists, menus).
- Prefer bubbling (default) unless capturing is needed.
- Use `event.currentTarget` to refer to the element the listener is attached to.
- Avoid attaching many listeners to many child elements — use delegation.

Would you like a visual diagram of event flow, or a hands-on example with dynamic element addition next?