JavaScript learning path

🔥 Perfect — let's now dive into **Advanced Tricks & Pro-Use Cases of `...spread` and `...rest` in JavaScript**.
This is where most developers just "use" them without thinking, but *internals* + *hidden traps* will make you stand out in interviews & real projects.

🚀 Advanced Tricks & Pro-Use Cases of Spread / Rest

---

1. Shallow vs. Deep Copy — Trap

```js
const obj = { a: 1, b: { c: 2 } };
const copy = { ...obj };

copy.b.c = 99;

console.log(obj.b.c); // 99 ❌ (unexpected)
```

👉 **Why?**
Spread only performs a **shallow copy**.
Nested objects/arrays are copied **by reference**, not cloned.

✅ Solution (deep copy ways):

```js
// JSON hack (fast but fails on Dates/Functions/undefined)
const deep1 = JSON.parse(JSON.stringify(obj));

// Structured clone (modern, supports Dates/Maps/Sets)
const deep2 = structuredClone(obj);

// Libraries (Lodash: cloneDeep)
const deep3 = _.cloneDeep(obj);
```

Interview Brain-Bender:

👉 Why does `{ ...obj }` *not deep copy*?

Because internally, `[[GetOwnProperty]]` only copies **references**, not actual nested memory.

---

2. Merging Objects — Property Overwrites

```js
const a = { x: 1, y: 2 };
const b = { y: 10, z: 3 };

const merged = { ...a, ...b };
console.log(merged); // { x: 1, y: 10, z: 3 }
```

- Order matters: later spreads overwrite earlier ones.
- Not obvious for interview questions.

👉 Trick: Use spread **reversal** if you want “priority to earlier”:

```js
const mergedSafe = { ...b, ...a };
```

---

3. Spread in Arrays — Flattening & Insert

```js
const arr = [1, 2, 3];
const newArr = [0, ...arr, 4];
console.log(newArr); // [0,1,2,3,4]
```

Pro Use Case — Immutable Updates (React/Redux style):

```js
const users = [{ id: 1 }, { id: 2 }];
const updated = users.map(u =>
  u.id === 2 ? { ...u, active: true } : u
);
```

👉 Spread avoids mutation, ensures **referential integrity** → React knows what changed.

---

4. Rest Parameters — Flexible APIs

```js
function logAll(first, ...rest) {
  console.log("First:", first);
  console.log("Rest:", rest);
}
logAll("A", "B", "C");
// First: A
// Rest: [ 'B', 'C' ]
```

Internals:

- `...rest` collects args into a **real Array** (unlike `arguments` object).
- This means `.map()`, `.filter()`, etc., directly work.

---

5. Destructuring + Rest

```js
const { a, ...rest } = { a: 1, b: 2, c: 3 };
console.log(rest); // { b: 2, c: 3 }

const [head, ...tail] = [10, 20, 30];
console.log(tail); // [20,30]
```

👉 **Hidden Gotcha**: order matters → `...rest` must be last in destructuring.

---

6. Handling Non-Iterables (Corner Case)

```js
const obj = { a: 1, b: 2 };
console.log([...obj]); // ❌ TypeError: obj is not iterable
```

👉 Why? Spread requires **iterables** (`Array`, `String`, `Map`, `Set`).
Objects are not inherently iterable.

✅ Trick: convert before spreading:

```js
console.log([...Object.entries(obj)]);
// [ ['a',1], ['b',2] ]
```

---

7. Performance Trade-offs

- Spread **copies all enumerable properties** each time → heavy on large objects.
- Mutation (direct assignment) is faster, but dangerous for state management.
- Optimization tip: Use spread **only when immutability is critical** (React, Redux).

📌 Example benchmark insight:

```js
let arr = Array(1e5).fill(0);

// Spread (slow)
console.time("spread");
let newArr1 = [...arr, 1];
console.timeEnd("spread");

// Push (fast, but mutates)
console.time("push");
arr.push(1);
console.timeEnd("push");
```

👉 Spread is **2–10x slower** than direct mutation for large arrays.

---

8. Advanced Trick — Conditional Spreading

```js
const condition = true;
const obj = {
  a: 1,
  ...(condition ? { b: 2 } : {}),
};
console.log(obj); // { a: 1, b: 2 }
```

👉 Clean way to include props **conditionally**.

---

9. Merging Defaults with Rest

```js
function setup(options) {
  const defaults = { debug: false, verbose: false };
  return { ...defaults, ...options };
}

console.log(setup({ debug: true }));
// { debug: true, verbose: false }
```

👉 Pro trick: Overriding configs without mutating defaults.

🎯 Interview Brain-Benders

1. What happens if you spread a `Set` into an Array?
2. How does spread handle symbol properties?
3. Why is spreading arrays slower than `concat()`?
4. Can you use `rest` in object destructuring with getters?

✅ Now tell me:

Do you want me to **answer these brain-benders right away**, or do you want to **try them first and I'll reveal solutions after** (like our earlier style)?