JavaScript Spread Operator (...)

From Basics to Advanced – Practical Notes with Examples & Tables

Prepared: August 13, 2025

The **spread operator** (**...**) expands iterable values (like arrays, strings, maps, sets) and copies enumerable own properties from objects into new containers. It is essential for writing clean, immutable JavaScript in real-world projects—especially in frameworks like React and in modern Node/Browser code. This guide goes from basic to advanced with examples and common pitfalls.

Topic	Array Spread	Object Spread
What it does	Expands an iterable into elements	Copies own enumerable props into a new obje
Common uses	Clone, merge, pass args	Clone, merge, add/override fields
Mutates original?	No	No
Keeps prototype?	No (creates plain array)	No (plain object without original prototype)
Shallow or deep?	Shallow copy	Shallow copy

BASICS

1) Array cloning & merging

```
const a = [1, 2, 3];
const clone = [...a];          // [1, 2, 3]
const merged = [...a, 4, 5];   // [1, 2, 3, 4, 5]

// Does NOT mutate 'a'
console.log(a);     // [1, 2, 3]
```

2) Passing arguments to functions

```
const nums = [10, 20, 30];
Math.max(...nums);  // 30

// Works with constructors
const d = new Date(...[2025, 7, 13]); // August 13, 2025 (month is 0-based)
```

3) Object cloning & merging

```
const user = { id: 1, name: "Zohaib" };
const cloned = { ...user };               // { id: 1, name: "Zohaib" }
const withRole = { ...user, role: "admin" }; // later keys win (override)

// Merge multiple objects
const a = { x: 1, y: 2 };
const b = { y: 20, z: 3 };
const merged = { ...a, ...b };            // { x:1, y:20, z:3 }
```

4) Spread vs. Rest (don’t confuse!)

Operator	Syntax position	Purpose	Example
Spread	In literals/calls	Expand/Copy	[...arr], {...obj}, fn(...args)
Rest	In params/left side	Collect	function f(...args) {}, const [a, ...rest] = arr

5) Converting iterables/array-like to arrays

```
// NodeList to Array
const nodes = document.querySelectorAll("div");
const arr = [...nodes]; // easy iteration, map/filter

// String to Array of chars
[..."hello"]; // ["h","e","l","l","o"]

// Alternative: Array.from(iterable)
Array.from(nodes);
```

INTERMEDIATE

6) Immutability patterns (no mutation)

```
// Add item (arrays)
const items = [1, 2, 3];
const add4 = [...items, 4]; // [1,2,3,4]

// Remove item (by index)
const idx = 1;
const removed = [...items.slice(0, idx), ...items.slice(idx + 1)]; // [1,3]

// Update item (by index)
const updated = items.map((v, i) => i === idx ? v * 10 : v);

// Update object field
const user = { id: 1, name: "Sara", role: "user" };
const promoted = { ...user, role: "admin" };
```

7) Conditional & short-circuit spreads

```
// Only add property when value exists
const role = null;
const profile = {
  id: 1,
  name: "Sara",
  ...(role && { role }), // if role is truthy, spread { role: role }
};

// Add array parts conditionally
const isProd = true;
const middlewares = [
  baseMw,
  ...(isProd ? [prodMw] : [devMw])
];
```

8) Dedupe arrays with Set + spread

```
const nums = [2, 3, 3, 2, 5];
const unique = [...new Set(nums)]; // [2,3,5]
```

9) Shallow copy caveat

```
const a = [{ x: 1 }, { x: 2 }];
const b = [...a];        // shallow copy: inner objects are the same references
b[0].x = 100;
console.log(a[0].x);     // 100 (changed!)

// For deep structures consider structuredClone(a) or a library.
```

> The spread operator performs a **shallow** copy. Nested objects/arrays are not cloned—only the top-level container is new.

10) Spread with Maps/Sets and non-iterables

```
// Maps/Sets are iterable -> spread to arrays of entries/values
const m = new Map([["a", 1], ["b", 2]]);
const entries = [...m]; // [["a",1],["b",2]]

// Non-iterables with array spread -> TypeError
const notIterable = 123;
// [...notIterable]; // TypeError
```

```
// Object spread works with any non-null object-like value
const fromString = { ..."abc" }; // { "0":"a", "1":"b", "2":"c" }
// But null/undefined throw with object spread
// { ...null } // TypeError
```

ADVANCED

11) Object spread semantics (spec nuances)

Own enumerable properties are copied (including symbol keys). Non-enumerable props are skipped; property descriptors (getters/setters) become plain values at copy time.
Prototype is not preserved: result is a plain object. Methods depending on prototype chain may break after copy.
Assignment order: right-most properties win on key conflicts. This is often used to override defaults.
Property order: preserves creation/insertion order of keys (strings before symbols). Avoid relying on it for logic.

12) Performance considerations

```
// Spread is O(n). For very large arrays/objects, consider alternatives:
const big = new Array(1e6).fill(0);

// Sometimes faster:
const clone1 = big.slice();      // array clone (historically optimized)
const clone2 = Array.from(big);  // array clone
// For objects:
const objClone = Object.assign({}, someObj);
// Measure in your runtime with performance.now() or console.time().
```

13) React/Redux-style state updates

```
// Arrays
setTodos(todos => todos.map(t => t.id === id ? { ...t, done: true } : t));

// Objects
setUser(user => ({ ...user, name: "New Name" }));

// Nested updates (shallow each level)
setPost(p => ({
  ...p,
  author: { ...p.author, name: "Ali" },
  tags: [...p.tags, "javascript"]
}));
```

14) Deep cloning options & when not to use spread

```
// Spread is shallow. For deep clone:
const deep1 = structuredClone(obj); // modern, handles cycles, Dates, Maps, Sets, etc.
const deep2 = JSON.parse(JSON.stringify(obj)); // loses functions, Dates, undefined, symbols

// When you must preserve prototype/descriptor behavior, use libraries like lodash.cloneDeep
// or write custom logic.
```

15) Common mistakes & fixes

Mistake	Why	Fix
Using spread thinking it’s deep	Only top level cloned	Use structuredClone / cloneDeep for deep copy
Spreading null/undefined (object)	Throws TypeError	Guard: {...(obj || {})}
Spreading non-iterable in array	Throws TypeError	Wrap in array or convert first
Expecting prototype methods to remain	Prototype is lost	Use Object.create(proto) + assign
Overriding order confusion	Right-most wins	{...base, ...override}

16) Handy patterns (cheat sheet)

```
// Defaults + overrides
const options = { retries: 3, timeout: 5000 };
const userOpts = { timeout: 10000 };
const finalOpts = { ...options, ...userOpts }; // timeout=10000

// Remove object key immutably
const { password, ...safeUser } = user; // rest removes key

// Array insert at index
function insertAt(arr, idx, ...items) {
  return [...arr.slice(0, idx), ...items, ...arr.slice(idx)];
}

// Function args from array
const args = [1, 2, 3];
fn(...args);

// Dedupe + sort
const sortedUnique = [...new Set(arr)].sort((a,b) => a - b);
```

PRACTICE EXERCISES (with expected outputs)

```
// 1) Merge two arrays, append 99 at the end.
const a = [1,2], b = [3,4];
const out1 = [...a, ...b, 99]; // [1,2,3,4,99]

// 2) Remove index 2 from an array immutably.
const arr = [10,20,30,40];
const out2 = [...arr.slice(0,2), ...arr.slice(3)]; // [10,20,40]

// 3) Update user.role to 'admin' without mutation.
const user = { id:1, name:'Ali', role:'user' };
const out3 = { ...user, role:'admin' };

// 4) Deep clone with structuredClone; change nested value.
const post = { meta:{ likes:1 }, tags:['js'] };
const deep = structuredClone(post);
deep.meta.likes = 10; // post.meta.likes still 1

// 5) Convert a NodeList to array and map textContent.
const nodes = document.querySelectorAll('li');
const texts = [...nodes].map(n => n.textContent);

// 6) Guard against null/undefined in object spread:
const cfg = null;
const safe = { ...{ a:1 }, ...(cfg || {}) }; // { a:1 }
```

You now have a complete mental model of the spread operator suitable for real-world projects.