Optimize Object Creation: Boost JavaScript Performance in React & Next.js

SUMANTH M

Frontend developer

1

Why Optimize Object Creation?

- Definition: Object creation is the process of instantiating new objects in JavaScript, often using object literals, constructors, or classes.
- Importance: While creating objects is essential, excessive or inefficient object creation can lead to performance bottlenecks, especially in React and Next.js applications where performance is critical

SUMANTH M
Frontend developer

Performance Impact of Object Creation

- Memory Usage: Each time an object is created, memory is allocated. If objects are created unnecessarily or too frequently, it can lead to high memory consumption.
- Garbage Collection: Objects that are created and discarded quickly can trigger frequent garbage collection cycles, which can slow down your app.
- Re-rendering in React: Creating new objects in every render (e.g., passing new objects as props) can cause unnecessary re-renders, affecting app performance.

SUMANTH M
Frontend developer

1. Avoid Creating New Objects in Every Render

- What To Do: Avoid creating new objects inside your component's render method or JSX. This can trigger
- unnecessary re-renders in React.

  Why It Matters: React compares old and new props to
- determine whether to re-render components. If you pass a new object every time, React will always think the props have changed, leading to frequent re-renders.

  How to Optimize: Use useMemo to memoize objects or arrays that don't change often.

```
const memoizedObject = useMemo(() => ({ key: 'value' }), []);
return <Component obj={memoizedObject} />;
```

SUMANTH M

Frontend developer

2. Reuse Objects Instead of Creating New Ones

- What It Is: Reuse existing objects whenever possible instead of creating new ones, especially in loops or functions that run frequently.
- Why It Matters: Reusing objects avoids memory overhead and reduces the frequency of garbage collection.
- How to Optimize: Use object references or store reusable objects in a shared variable rather than creating a new instance each time.

```
const user = { name: 'Alice', age: 30 };

// Reuse the same object instead of creating a new one
function updateUser(user) {
  user.age += 1;
  return user;
}
```

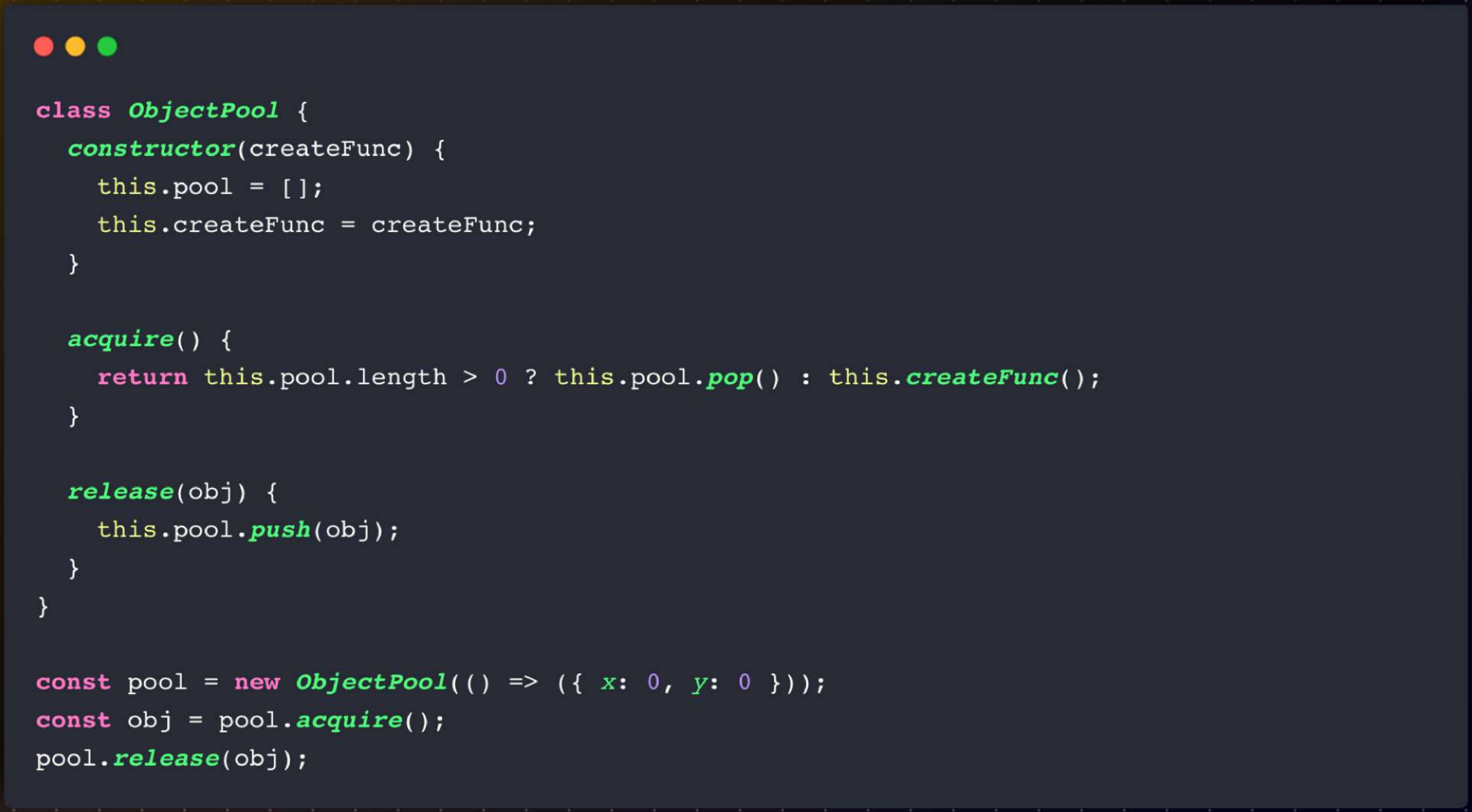
SUMANTH M
Frontend developer

3. Use Object Pooling for Frequent Object Creation

- What It Is: Object pooling involves maintaining a set of reusable objects, reducing the need to frequently create and destroy objects.
- Why It Matters: For apps that require frequent object creation (like game engines or UI updates), pooling helps reduce memory churn and optimize performance.
- How to Optimize: Implement a pool of objects that can be reused instead of creating new ones each time.

SUMANTH M
Frontend developer

6



```
class ObjectPool {
  constructor(createFunc) {
    this.pool = [];
    this.createFunc = createFunc;
  }

  acquire() {
    return this.pool.length > 0 ? this.pool.pop() : this.createFunc();
  }

  release(obj) {
    this.pool.push(obj);
  }
}

const pool = new ObjectPool(() => ({ x: 0, y: 0 }));
const obj = pool.acquire();
pool.release(obj);
```

SUMANTH M
Frontend developer

4. Use Object.create() for Prototype-based Objects

What It Is: Object.create() creates a new object with a specified prototype object and properties.
Why It Matters: This can be more efficient than using a constructor function or class, as it avoids unnecessary object wrapping.
How to Optimize: Use Object.create() to create objects with a specific prototype without going through the overhead of classes or constructors

```
const prototype = { greet() { console.log('Hello'); } };
const obj = Object.create(prototype);
obj.greet(); // 'Hello'

const pool = new ObjectPool(() => ({ x: 0, y: 0 }));
const obj = pool.acquire();
pool.release(obj);
```

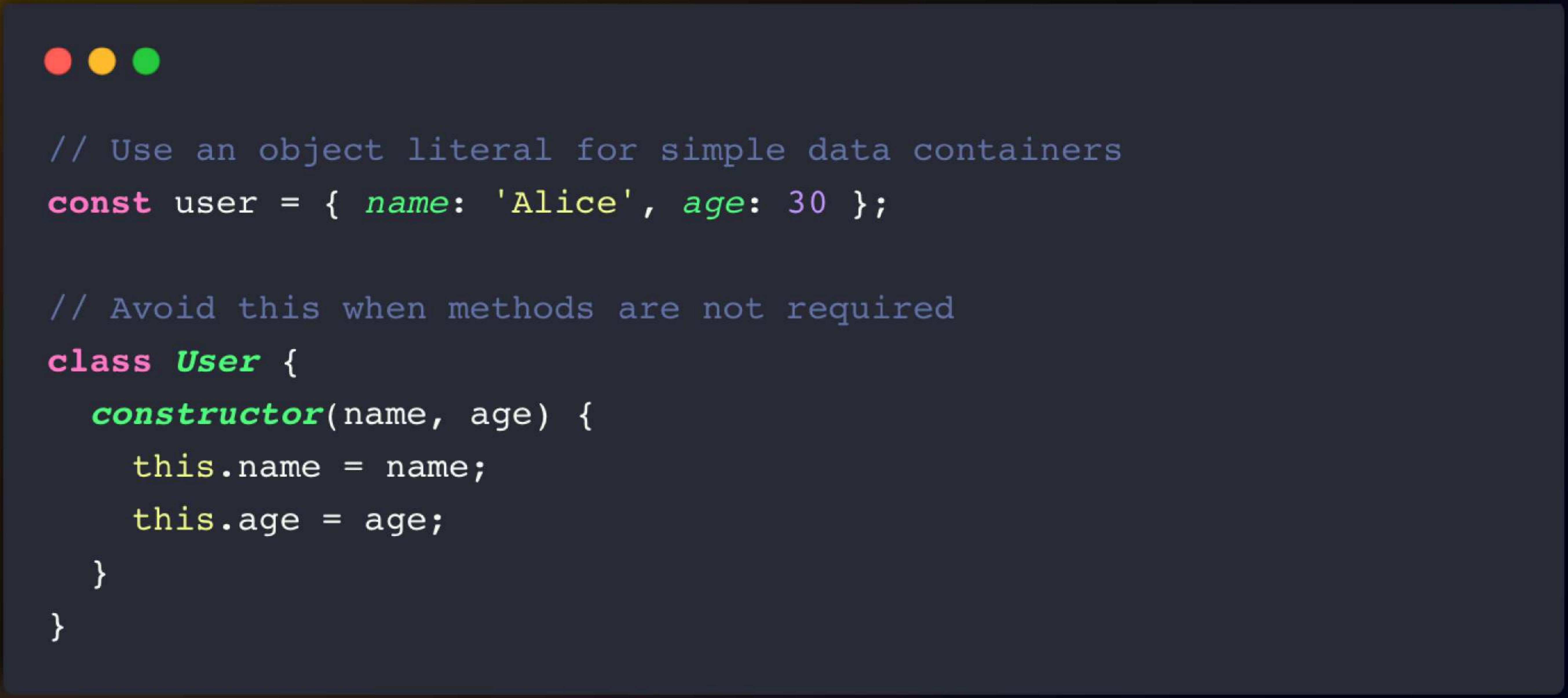
5. Avoid Using Classes for Lightweight Objects

What It Is: Classes in JavaScript introduce a lot of boilerplate code, which can lead to unnecessary overhead for lightweight objects.
Why It Matters: For simple objects, using plain object literals is more efficient than using a class with constructors, methods, and prototypes.
How to Optimize: Use object literals instead of classes when you only need a lightweight object without methods.

SUMANTH M
Frontend developer

9

React Js



```
// Use an object literal for simple data containers
const user = { name: 'Alice', age: 30 };

// Avoid this when methods are not required
class User {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}
```

SUMANTH M

Frontend developer

6. Use Immutable Objects for State Management

What It Is: Immutable objects cannot be modified after they are created. This makes state management easier and more predictable in React.

Why It Matters: Immutability ensures that objects aren’t mutated directly, reducing bugs and improving performance by eliminating unnecessary re-renders.

How to Optimize: Use libraries like Immer for immutability or create new objects when updating state instead of mutating existing ones.

```
import produce from 'immer';

const initialState = { name: 'Alice', age: 30 };
const newState = produce(initialState, draft => {
  draft.age += 1;
});
```

11 Disadvantages of setInterval

- Requires Manual Clearing: Needs clearInterval to stop running, or it keeps going indefinitely.
- Memory Leaks: Can cause memory issues if not cleared properly.
- Delayed Start: Starts executing only after the first interval elapses, not immediately.

SUMANTH M
Frontend developer

Conclusion

Optimizing object creation is crucial for enhancing performance in your React and Next.js applications. By following these strategies, you can reduce memory consumption, improve responsiveness, and ensure that your app runs efficiently.

SUMANTH M
Frontend developer