Understanding the Prototype Chain in JavaScript

Last Updated : 18 Jan, 2025

The prototype chain is a core JavaScript concept enabling the inheritance of properties and methods between objects. It facilitates code reuse, efficient property lookup, and object hierarchy creation.

- Every JavaScript object has an internal link to another object, called its prototype.
- The prototype chain forms when objects inherit properties and methods from their prototypes.
- Property or method access starts from the object itself and traverses up the chain if not found.
- The chain ends at null, the prototype of Object.prototype.

```
const parent = { greet: () => "Hello!" };
const child = Object.create(parent);

↔
```

In this example

- parent is the prototype of child.
- child inherits the greet method from parent.
- The chain consists of child → parent → Object.prototype → null.

Syntax

```
function ConstructorName(params) {
  // Initialization code
}

ConstructorName.prototype.methodName = function () {
```

```
  // Method code
};
```

Real-World Use Cases

Extending Built-in Objects

```
Array.prototype.sum = function () {
    return this.reduce((acc, val) =>
        acc + val, 0);
};
console.log([1, 2, 3].sum());
```

Output

```
6
```

Custom Object Hierarchies

```
const vehicle = {
    start() {
        console.log("Engine started.");
    },
};
const car = Object.create(vehicle);
car.drive = function () {
    console.log("Car is driving.");
};

car.start();
car.drive();
```

Output

```
Engine started.
Car is driving.
```

Checking Property Existence

```
const car = {
    wheels: 4
};
const myCar = Object.create(car);
myCar.color = "red";

↔
```

Output

```
true
true
false
```

Prototype Chain Traversal

```
function fun(obj) {
    let current = obj;
    while (current) {
        console.log(current);
        current = Object.getPrototypeOf(current);
    }
}
const animal = { eats: true };
const mammal = Object.create(animal);
mammal.hasFur = true;
const dog = Object.create(mammal);
dog.barks = true;
fun(dog);
```

Output

```
{ barks: true }
{ hasFur: true }
{ eats: true }
[Object: null prototype] {}
```

Prototype Method Overriding

```
function Vehicle() { }
Vehicle.prototype.start = function () {
    return "Vehicle is starting";
};
function Car() { }
Car.prototype = Object.create(Vehicle.prototype);
Car.prototype.constructor = Car;
Car.prototype.start = function () {
    return "Car is starting";
};
const myCar = new Car();
console.log(myCar.start());
```

Output

```
Car is starting
```

Advantages of the Prototype Chain

- **Code Reusability:** Shared properties and methods reduce redundancy.
- **Efficient Memory Usage:** Shared prototypes lower memory consumption.
- **Dynamic Behavior:** Prototypes can be extended at runtime.
- **Scalable Object Hierarchies:** Simplifies creation and management of relationships between objects.

Comment More info

Campus Training Program

Next Article

Nesting For Loops in JavaScript

Similar Reads

JavaScript Prototype