Metaprogramming with Proxy

🧠 Metaprogramming with Proxy in JavaScript — Explained in Detail

Metaprogramming means writing code that can manipulate or modify other code at runtime. In JavaScript, one of the most powerful tools for metaprogramming is the `Proxy` object.

🔧 What is a `Proxy`?

A `Proxy` is an object that wraps another object and **intercepts operations** performed on it (like reading or writing properties), allowing you to customize or override behavior.

```
const target = { message: "Hello" };

const handler = {
  get: function (obj, prop) {
    return prop in obj ? obj[prop] : "Property does not exist";
  }
};

const proxy = new Proxy(target, handler);

console.log(proxy.message); // Hello
console.log(proxy.nonExistent); // Property does not exist
```

🧩 Proxy Syntax

```
const proxy = new Proxy(target, handler);
```

- `target`: The original object.
- `handler`: An object containing **traps** (functions that intercept operations).

🔄 Common Proxy Traps

Trap	Description
`get`	Intercepts property access (`proxy.prop`)
`set`	Intercepts property assignment (`proxy.prop = value`)
`has`	Intercepts the `in` operator (`'prop' in proxy`)
`deleteProperty`	Intercepts deletion (`delete proxy.prop`)
`apply`	Intercepts function calls
`construct`	Intercepts `new` operator
`ownKeys`	Intercepts methods that list object keys (`Object.keys` , `for...in`)

🧪 Example: Validation Using `set`

```
const user = {
  name: "Abhi",
  age: 21
};

const handler = {
  set(obj, prop, value) {
    if (prop === "age" && typeof value !== "number") {
      throw new TypeError("Age must be a number");
    }
    obj[prop] = value;
    return true;
  }
};

const proxyUser = new Proxy(user, handler);
proxyUser.age = 25;      // ✅ Works
// proxyUser.age = "old"; // ❌ Throws TypeError
```

📦 Real-World Use Cases

1. **Validation**: Prevent invalid assignments (as shown above).
2. **Access Control**: Control who can read or write certain properties.
3. **Auto-Tracking**: Track reads/writes (used in frameworks like Vue.js).
4. **Default Values**: Provide fallback values.
5. **Revocable Proxies**: Temporarily grant access to objects.

🚫 Caveats

- **Performance overhead** due to traps.
- Can't proxy certain internal JavaScript operations (e.g., object equality).
- Not supported in some older browsers (e.g., IE11).

🔄 Revocable Proxy

You can create a proxy that can be disabled later:

```
const { proxy, revoke } = Proxy.revocable({}, {
  get: () => "Intercepted"
});

console.log(proxy.test); // Intercepted
revoke();
console.log(proxy.test); // ❌ TypeError: Cannot perform 'get' on a proxy that has been revoked
```

✅ Summary

Feature	Description
Flexible Interception	Modify default behaviors of objects
Cleaner Code	Ideal for logging, validation, reactive programming
Powerful Tool	Powers modern frameworks (e.g., Vue 3 uses Proxy)

Let me know if you want code samples for a specific use case like **access logging**, **Vue-style reactivity**, or **deep property validation**.