JavaScript Proxy

Summary: in this tutorial, you will learn about the JavaScript Proxy object in ES6.

What is a JavaScript Proxy object

A JavaScript Proxy is an object that wraps another object (target) and intercepts the fundamental operations of the target object.

The fundamental operations can be the property lookup, assignment, enumeration, function invocations, etc.

Creating a proxy object

To create a new proxy object, you use the following syntax:

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

In this syntax:

- `target` – is an object to wrap.
- `handler` – is an object that contains methods to control the behaviors of the `target`. The methods inside the `handler` object are called traps.

A simple proxy example

First, define an object called `user`:

```
const user = {
    firstName: 'John',
    lastName: 'Doe',
```

```
    email: 'john.doe@example.com',
}
```

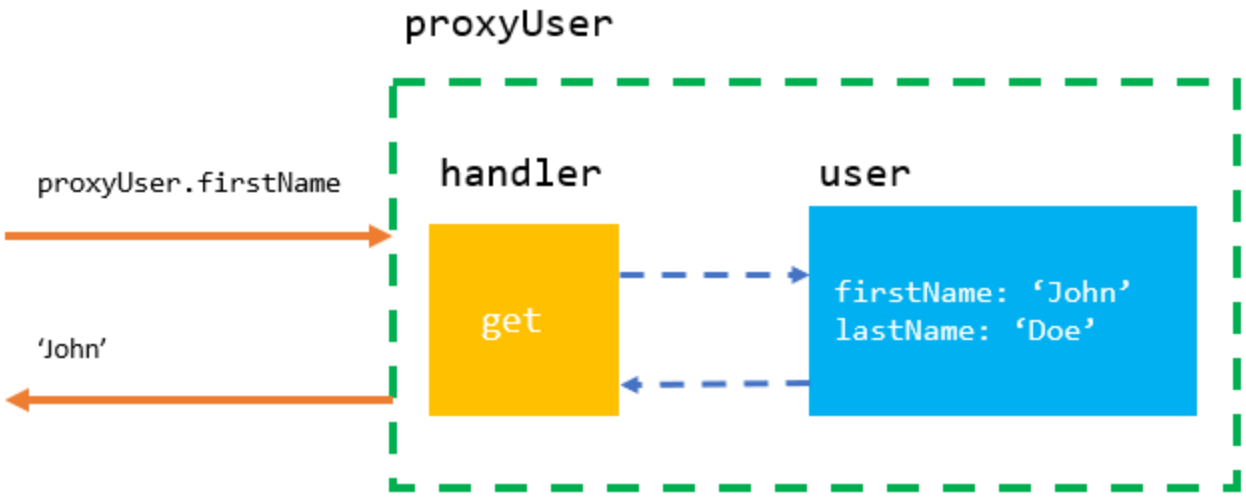
Second, define a `handler` object:

```
const handler = {
    get(target, property) {
        console.log(`Property ${property} has been read.`);
        return target[property];
    }
}
```

Third, create a `proxy` object:

```
const proxyUser = new Proxy(user, handler);
```

The `proxyUser` object uses the `user` object to store data. The `proxyUser` can access all properties of the `user` object.



Fourth, access the `firstName` and `lastName` properties of the `user` object via the `proxyUser` object:

```
console.log(proxyUser.firstName);
console.log(proxyUser.lastName);
```

Output:

```
Property firstName has been read.
John
Property lastName has been read.
Doe
```

When you access a property of the `user` object via the `proxyUser` object, the `get()` method in the `handler` object is called.

Fifth, if you modify the original object `user`, the change is reflected in the `proxyUser`:

```
user.firstName = 'Jane';
console.log(proxyUser.firstName);
```

Output:

```
Property firstName has been read.
Jane
```

Similarly, a change in the `proxyUser` object will be reflected in the original object (`user`):

```
proxyUser.lastName = 'William';
console.log(user.lastName);
```

Output:

```
William
```

Proxy Traps

The get() trap

The `get()` trap is fired when a property of the `target` object is accessed via the proxy object.

In the previous example, a message is printed out when a property of the `user` object is accessed by the `proxyUser` object.

Generally, you can develop a custom logic in the `get()` trap when a property is accessed.

For example, you can use the `get()` trap to define computed properties for the target object. The computed properties are properties whose values are calculated based on values of existing properties.

The `user` object does not have a property `fullName`, you can use the `get()` trap to create the `fullName` property based on the `firstName` and `lastName` properties:

```
const user = {
    firstName: 'John',
    lastName: 'Doe'
}

const handler = {
    get(target, property) {
        return property === 'fullName' ?
            `${target.firstName} ${target.lastName}` :
            target[property];
    }
};

const proxyUser = new Proxy(user, handler);

console.log(proxyUser.fullName);
```

Output:

```
John Doe
```

The set() trap

The `set()` trap controls behavior when a property of the `target` object is set.

Suppose that the `age` of the user must be greater than 18. To enforce this constraint, you develop a `set()` trap as follows:

```
const user = {
    firstName: 'John',
    lastName: 'Doe',
    age: 20
}

const handler = {
    set(target, property, value) {
        if (property === 'age') {
            if (typeof value !== 'number') {
                throw new Error('Age must be a number.');
            }
            if (value < 18) {
                throw new Error('The user must be 18 or older.')
            }
        }
        target[property] = value;
    }
};

const proxyUser = new Proxy(user, handler);
```

First, set the `age` of user to a string:

```
proxyUser.age = 'foo';
```

Output:

```
Error: Age must be a number.
```

Second, set the age of the user to 16:

```
proxyUser.age = '16';
```

Output:

```
The user must be 18 or older.
```

Third, set the age of the user to 21:

```
proxyUser.age = 21;
```

No error occurred.

The apply() trap

The `handler.apply()` method is a trap for a function call. Here is the syntax:

```
let proxy = new Proxy(target, {
    apply: function(target, thisArg, args) {
        //...
    }
});
```

See the following example:

```
const user = {
    firstName: 'John',
    lastName: 'Doe'
}

const getFullName = function (user) {
    return `${user.firstName} ${user.lastName}`;
}


const getFullNameProxy = new Proxy(getFullName, {
    apply(target, thisArg, args) {
        return target(...args).toUpperCase();
    }
```

```
});

console.log(getFullNameProxy(user)); //
```

Output

```
JOHN DOE
```

More traps

The following are more traps:

- `construct` – traps usage of the `new` operator
- `getPrototypeOf` – traps an internal call to `[[GetPrototypeOf]]`
- `setPrototypeOf` – traps a call to `Object.setPrototypeOf`
- `isExtensible` – traps a call to `Object.isExtensible`
- `preventExtensions` – traps a call to `Object.preventExtensions`
- `getOwnPropertyDescriptor` – traps a call to `Object.getOwnPropertyDescriptor`

In this tutorial, you have learned about the JavaScript Proxy object used to wrap another object to change the fundamental behaviors of that object.