JavaScript Object Properties

Summary: in this tutorial, you will learn about the JavaScript object's properties and attributes such as configurable, enumerable, writable, get, set, and value.

Object Property types

JavaScript specifies the characteristics of properties of objects via internal attributes surrounded by the two pairs of square brackets, e.g., `[[Enumerable]]` .

Objects have two types of properties: data and accessor properties.

1) Data properties

A data property contains a single location for a data value. A data property has four attributes:

- `[[Configurarable]]` – determines whether a property can be redefined or removed via `delete` operator.
- `[[Enumerable]]` – indicates if a property can be returned in the `for...in` loop.
- `[[Writable]]` – specifies that the value of a property can be changed.
- `[[Value]]` – contains the actual value of a property.

By default, the `[[Configurable]]` , `[[Enumerable]]` And `[[Writable]]` attributes set to `true` for all properties defined directly on an object. The default value of the `[[Value]]` attribute is `undefined` .

For example, the following creates a `person` object with two properties `firstName` and `lastName` with the configurable, enumerable, and writable attributes set to `true` and their values are set to `'John'` and `'Doe'` respectively:

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};
```

To change any attribute of a property, you use the `Object.defineProperty()` method.

The `Object.defineProperty()` method accepts three arguments:

- An object.
- A property name of the object.
- A property descriptor object that has four properties: `configurable` , `enumerable` , `writable` , and `value` .

If you use the `Object.defineProperty()` method to define a property of the object, the default values of `[[Configurable]]` , `[[Enumerable]]` , and `[[Writable]]` are set to `false` unless otherwise specified.

The following example creates a `person` object with the `age` property:

```
let person = {};
person.age = 25;
```

Since the default value of the `[[Configurable]]` attribute is set to `true` , you can remove it via the `delete` operator:

```
delete person.age;
console.log(person.age);
```

Output:

```
undefined
```

The following example creates a `person` object and adds the `ssn` property to it using the `Object.defineProperty()` method:

```
'use strict';

let person = {};

Object.defineProperty(person, 'ssn', {
    configurable: false,
    value: '012-38-9119'
});

delete person.ssn;
```

Output:

```
TypeError: Cannot delete property 'ssn' of #<Object>
```

In this example, the `configurable` attribute is set to `false`. Therefore, deleting the `ssn` property causes an error.

Also, once you define a property as non-configurable, you cannot change it to configurable.

If you use the `Object.defineProperty()` method to change any attribute other than the writable, you'll get an error. or example:

```
'use strict';

let person = {};

Object.defineProperty(person, 'ssn', {
    configurable: false,
    value: '012-38-9119'
});


Object.defineProperty(person, 'ssn', {
    configurable: true
});
```

Output:

```
TypeError: Cannot redefine property: ssn
```

By default, the `enumerable` attribute of all the properties defined on an object is `true`. It means that you can iterate over all object properties using the `for...in` loop like this:

```
let person = {};
person.age = 25;
person.ssn = '012-38-9119';

for (let property in person) {
    console.log(property);
}
```

Output:

```
age
ssn
```

The following makes the `ssn` property non-enumerable by setting the `enumerable` attribute to `false`.

```
let person = {};
person.age = 25;
person.ssn = '012-38-9119';
```

```
Object.defineProperty(person, 'ssn', {
    enumerable: false
});

for (let prop in person) {
    console.log(prop);
}
```

Output

```
age
```

2) Accessor properties

Similar to data properties, accessor properties also have `[[Configurable]]` and `[[Enumerable]]` attributes.

But the accessor properties have the `[[Get]]` and `[[Set]]` attributes instead of `[[Value]]` and `[[Writable]]` .

When you read data from an accessor property, the `[[Get]]` function is called automatically to return a value. The default return value of the `[[Get]]` function is `undefined` .

If you assign a value to an accessor property, the `[[Set]]` function is called automatically.

To define an accessor property, you must use the `Object.defineProperty()` method. For example:

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
}

Object.defineProperty(person, 'fullName', {
    get: function () {
        return this.firstName + ' ' + this.lastName;
    },
    set: function (value) {
        let parts = value.split(' ');
        if (parts.length == 2) {
            this.firstName = parts[0];
            this.lastName = parts[1];
        } else {
            throw 'Invalid name format';
        }
    }
});

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

Output:

```
'John Doe'
```

In this example:

- First, define the `person` object that contains two properties: `firstName` and `lastName` .
- Then, add the `fullName` property to the `person` object as an accessor property.

In the `fullname` accessor property:

- The `[[Get]]` returns the full name that is the result of concatenating of `firstName` , `space` , and `lastName` .
- The `[[Set]]` method splits the argument by the space and assigns the `firstName` and `lastName` properties of the corresponding parts of the name.
- If the full name is not in the correct format i.e., first name, space, and last name, it will throw an error.

Define multiple properties: Object.defineProperties()

In ES5, you can define multiple properties in a single statement using the `Object.defineProperties()` method. or example:

```
var product = {};

Object.defineProperties(product, {
    name: {
        value: 'Smartphone'
    },
    price: {
        value: 799
    },
    tax: {
        value: 0.1
    },
    netPrice: {
        get: function () {
            return this.price * (1 + this.tax);
        }
    }
});

console.log('The net price of a ' + product.name + ' is ' + product.netPrice.toFixed(2) + ' USD');
```

Output:

```
The net price of a Smartphone is 878.90 USD
```

In this example, we defined three data properties: `name`, `price`, and `tax`, and one accessor property `netPrice` for the `product` object.

JavaScript object property descriptor

The `Object.getOwnPropertyDescriptor()` method allows you to get the descriptor object of a property. The `Object.getOwnPropertyDescriptor()` method takes two arguments:

1. An object
2. A property of the object

It returns a descriptor object that describes a property. The descriptor object has four properties: configurable, enumerable, writable, and value.

The following example gets the descriptor object of the `name` property of the `product` object in the prior example.

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};


let descriptor = Object.getOwnPropertyDescriptor(person, 'firstName');

console.log(descriptor);
```

Output:

```
{ value: 'John',
  writable: true,
  enumerable: true,
  configurable: true }
```

Summary

- JavaScript objects have two types of properties: data properties and accessor properties.

- JavaScript uses internal attributes denoted `[[...]]` to describe the characteristics of properties such as `[[Configurable]]`, `[[Enumerable]]`, `[[Writable]]`, and `[[Value]]`, `[[Get]]`, and `[[Set]]`.
- The method `Object.getOwnPropertyDescriptor()` return a property descriptor of a property in an object.
- A property can be defined directly on an object or indirectly via the `Object.defineProperty()` or `Object.defineProperties()` methods. These methods can be used to change the attributes of a property.