JavaScript
TUTORIAL

JavaScript Getters and Setters

Summary: in this tutorial, you will learn about JavaScript getters and setters and how to use them effectively.

Introduction to the JavaScript getters and setters

The following example defines a class called `Person`:

```
class Person {
    constructor(name) {
        this.name = name;
    }
}

let person = new Person("John");
console.log(person.name); // John
```

The `Person` class has a property `name` and a constructor. The constructor initializes the `name` property to a string.

Sometimes, you don't want the `name` property to be accessed directly like this:

```
person.name
```

To do that, you may come up with a pair of methods that manipulate the `name` property. For example:

```
class Person {
    constructor(name) {
        this.setName(name);
    }
    getName() {
        return this.name;
    }
    setName(newName) {
        newName = newName.trim();
        if (newName === '') {
            throw 'The name cannot be empty';
        }
        this.name = newName;
    }
}

let person = new Person('Jane Doe');
console.log(person); // Jane Doe

person.setName('Jane Smith');
console.log(person.getName()); // Jane Smith
```

In this example, the `Person` class has the `name` property. Also, it has two additional methods `getName()` and `setName()`.

The `getName()` method returns the value of the `name` property.

The `setName()` method assigns an argument to the `name` property. The `setName()` removes the whitespaces from both ends of the `newName` argument and throws an exception if the `newName` is empty.

The `constructor()` calls the `setName()` method to initialize the `name` property:

```
constructor(name) {
    this.setName(name);
}
```

The `getName()` and `setName()` methods are known as getter and setter in other programming languages such as Java and C++.

ES6 provides a specific syntax for defining the getter and setter using the get and set keywords. For example:

```
class Person {
    constructor(name) {
        this._name = name;
    }
    get name() {
        return this._name;
    }
    set name(newName) {
        newName = newName.trim();
        if (newName === '') {
            throw 'The name cannot be empty';
        }
        this._name = newName;
    }
}
```

How it works.

First, the `name` property is changed to `_name` to avoid the name collision with the getter and setter.

Second, the getter uses the `get` keyword followed by the method name:

```
get name() {
    return this._name;
}
```

To call the getter, you use the following syntax:

```
let name = person.name;
```

When JavaScript sees the access to `name` property of the `Person` class, it checks if the `Person` class has any `name` property.

If not, JavaScript checks if the Person class has any method that binds to the `name` property. In this example, the `name()` method binds to the `name` property via the `get` keyword. Once JavaScript finds the getter method, it executes the getter method and returns a value.

Third, the setter uses the `set` keyword followed by the method name:

```
set name(newName) {
    newName = newName.trim();
    if (newName === '') {
        throw 'The name cannot be empty';
    }
    this._name = newName;
}
```

JavaScript will call the `name()` setter when you assign a value to the `name` property like this:

```
person.name = 'Jane Smith';
```

If a class has only a getter but not a setter and you attempt to use the setter, the change won't take any effect. See the following example:

```
class Person {
    constructor(name) {
        this._name = name;
    }
    get name() {
        return this._name;
    }
}

let person = new Person("Jane Doe");
console.log(person.name);

// attempt to change the name, but cannot
person.name = 'Jane Smith';
console.log(person.name); // Jane Doe
```

In this example, the `Person` class has the `name` getter but not the `name` setter. It attempts to call the setter. However, the change doesn't take effect since the Person class doesn't have the name setter.

Using getter in an object literal

The following example defines a getter called `latest` to return the latest attendee of the `meeting` object:

```
let meeting = {
    attendees: [],
    add(attendee) {
        console.log(`${attendee} joined the meeting.`);
        this.attendees.push(attendee);
        return this;
    },
    get latest() {
        let count = this.attendees.length;
        return count == 0 ? undefined : this.attendees[count - 1];
    }
};

meeting.add('John').add('Jane').add('Peter');
console.log(`The latest attendee is ${meeting.latest}.`);
```

Output:

```
John joined a meeting.
Jane joined a meeting.
Peter joined a meeting.
The latest attendee is Peter.
```

Summary

- Use the `get` and `set` keywords to define the JavaScript getters and setters for a class or an object.
- The `get` keyword binds an object property to a method that will be invoked when that property is looked up.

- The `set` keyword binds an object property to a method that will be invoked when that property is assigned.