Theory: Object methods and keyword "this"

790 users solved this topic. Latest completion was about 16 hours ago.

13 minutes 0 / 5 problems solved

Skip this topic Start practicing

When writing scripts, we usually need to create objects to present something from the real world: users, products, maps, and more. In real life, it is possible to consider the products in detail or put something in a shopping cart, so in programming it is also an option. Such actions in JavaScript are available thanks to **object methods**, meaning the functions you can use as object properties.

§1. Object method creation

Let's try to understand how object methods are created. To do this, consider the following example:

```
let product = {
  name: "Microwave",
  description: "With oven mode",
  price: 398
};

product.giveDiscount = function() {
  console.log("You have a 10% discount!");
};



product.giveDiscount(); // You have a 10% discount!
```

In this code, we have allowed the object to report a discount on the product by writing the method `giveDiscount`.

§2. Shortened recording

Setting functions as properties can be written shorter. For example:

```
let person = {
  greetings: function() {
    console.log("Hello");
  }
};
```

We can also skip the keyword `function`:

```
let person = {
  greetings() {
    console.log("Hello");
  }
};
```

Shortened syntax not only saves you time, but also makes the code more readable.

§3. *"this"* keyword

In natural language, we can point to a specific person, animal, or object using pronouns. In JavaScript you may just as well refer to a specific object using the keyword `this`:

Current topic:

Object methods and keyword "this"

Topic depends on:

✕ Arrow functions

✕ Type conversion

Topic is required for:

ForEach method

Table of contents:

```
let user = {
  firstName: "Elliot",
  lastName: "Alderson",
  fullName() {
    return this.firstName + " " + this.lastName;
  }
};

console.log(user.fullName()); // Elliot Alderson
```

In the above example, with the help of `this`, the `fullName` function has accessed the `firstName` and `lastName` information stored in the `user` object because it is declared inside it.

Applying `this` to nested objects can create some confusion. In such situations, you should keep it in mind that the keyword `this` refers to the object in whose method it is used.

§4. *"this"* and arrow functions

Arrow functions *cannot* bind `this`. In other words, the arrow functions do not have their own `this`.

If we use `this` inside an arrow function, its value is taken from an external function declared the usual way:

```
let movie = {
  name: "The Thirteenth Floor",
  age: 1999,
  getInfo() {
    let arrow = () => console.log("The movie " + this.name + " was shot in " + this.age);
    arrow();
  }
};

movie.getInfo(); // The movie The Thirteenth Floor was shot in 1999
```

In fact, the specifics of using `this` in JavaScript are not limited to the examples described above. Here we covered only the basics of working with this keyword, so there's a lot more to learn.

Report a typo

77 users liked this theory. 1 didn't like it. What about you?

😍 🙂 😐 🙁 😡

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion