Theory: setTimeout and setInterval

644 users solved this topic. Latest completion was **about 9 hours ago**.

16 minutes 0 / 5 problems solved

Skip this topic Start practicing

When writing scripts with two lines, we assume that the browser will execute the second line after the first. However, some situations ask for a different approach: for example, what if you want to show an ad only after the user has spent 5 seconds on your website? Let's take a closer look at how to control the functions in situations like that.

§1. setTimeout

Imagine you have a shopping list, but you don't want to go to the store immediately. You plan to do it in an hour, after you have watched a movie. If we put it in programming terms, you have a function called "buy food" that you want to execute after 60 minutes.

In the JavaScript world, we would say that you are **"scheduling a call"**: you have already written the code and want to launch it at a specific time.

One of the methods to help with scheduling a call is `setTimeout`. It doesn't run instantly: `setTimeout` is executed only after a specific period of time. Look at the example for an advertising message that is showed after 5 seconds:

```
setTimeout(function() {
console.log("Welcome to our website! You have a special discount today!");
}, 5000)
```

Here is another way to write `setTimeout`. As you see, we call a function named *welcome* instead of an anonymous function:

```
function welcome(name) {
console.log(name + ", welcome to our website! You have a special discount today!
");
}
let userWelcomeMessage = setTimeout(welcome, 5000, "Mary");
```

Let's look at the variables of the `setTimeout` method:

- `function() {console.log(`…`)}`: the function to call after the delay;
- `5000`: the delay period in milliseconds (1000 ms = 1 second), that is, the time after which the function is executed.
- `Mary`: an argument for the *welcome* function;

The text "Mary" goes to the *welcome* function as the *name* variable, and we will see the message "Mary, welcome to our website! You have a special discount today!" in 5 seconds as a result. You can pass more than one argument to a function by adding additional variables to the `setTimeout` method.

§2. Zero delays setTimeout

Let's consider another case with the `setTimeout` method. Here's our example:

```
setTimeout(function() {
  console.log("Welcome to our website!");
}, 5000)
```

Now, what happens if we change the delay variable to zero or delete it?

```
function welcome() {
  console.log("Welcome to our website!");
}

setTimeout(welcome, 0)
```

Current topic:

setTimeout and setInterval

Topic depends on:

× Functions

Topic is required for:

Anonymous function

Table of contents:

The function will be executed immediately, without delay. However, you should remember that if another function is written after it, it will be executed first. Let's consider an example to better understand it:

```
function welcome() {
  console.log("Welcome to our website!");
}

setTimeout(welcome, 0)

console.log("2nd of May");
```

The browser calls `setTimeout` only after the function `console.log` is executed. The `welcome()` function is placed in the queue and launched after the current code is complete. As a result, the user will see "2nd of May" first and only then "Welcome to our website!".

§3. setInterval

Imagine you want to wake up at 8 a.m. every day. To do it, you add an alarm for a specific time and set it to repeat every 24 hours. In programming terms, you have a function called "alarm" and you want to execute it again in a certain interval.

The method `setInterval` does just that: it calls the function again and again after a specific time period.

Here is an example:

```
function alarm(time) {
  console.log("Wake up! It’s " + time + " o'clock!");
}

setInterval(alarm, 3000, 8);
```

The user will see "Wake up! It's 8 o'clock!" in three seconds and then every three seconds.

Like the `setTimeout` method, `setInterval` has similar variables:

- the function to be executed;
- an interval (in ms) in which function will be run every time;
- function arguments.

§4. Clearing intervals

As you know, it's possible to run a function with a delay. Jack the programmer has done it 1000 times on the webpage to show different ads for a user, and the user's browser started to work very slowly. How can we help Jack?

When you launch the timer, the browser keeps running the task forever and hence spends resources. To avoid it, we should stop timers that aren't needed. You run the timer with `setTimeout`, and to break it, you should use `clearTimeout`.

In this case, you should know the id of the timer you want to stop. `setTimeout` helps with that: it returns the identifier of the timer that had been created. Look at the following example:

```
let timerId = setTimeout(...);

clearTimeout(timerId);
```

If you want to break the timer made by `setInterval`, you have to use `clearInterval`:

```
const intervalId = setInterval(alarm, 2000);

clearInterval(intervalId);
```

Again: don't forget to stop all unnecessary timers, because otherwise, running all processes will take up significant computer resources.

§5. Conclusion

In this topic, we have considered two methods for executing code with delay: `setTimeout` and `setInterval`. The `setTimeout` method is used to launch code once, while `setInterval` runs every time with a specific interval. You also know how to stop timers using `clearTimeout` and `clearInterval` methods so that the browser is not overwhelmed.

Report a typo

66 users liked this theory. 0 didn't like it. What about you?

Start practicing

Comments (2) Hints (0) Useful links (0) Show discussion