JS JavaScript
TUTORIAL

JavaScript Recursive Function

Summary: in this tutorial, you will learn how to use the recursion technique to develop a JavaScript recursive function, which is a function that calls itself.

Introduction to the JavaScript recursive functions

A recursive function is a function that calls itself until it doesn't. This technique is called recursion.

Suppose that you have a function called `recurse()` . The `recurse()` is a recursive function if it calls itself inside its body, like this:

```
function recurse() {
    // ...
    recurse();
    // ...
}
```

A recursive function always has a condition to stop calling itself. Otherwise, it will call itself indefinitely. So a recursive function typically looks like the following:

```
function recurse() {
    if(condition) {
        // stop calling itself
        //...
    } else {
        recurse();
    }
}
```

Generally, you use recursive functions to break down a big problem into smaller ones. Typically, you will find the recursive functions in data structures like binary trees and graphs and algorithms such as binary search and quicksort.

JavaScript recursive function examples

Let's take some examples of using recursive functions.

1) A simple JavaScript recursive function example

Suppose that you need to develop a function that counts down from a specified number to 1. For example, to count down from 3 to 1:

```
3
2
1
```

The following shows the `countDown()` function:

```
function countDown(fromNumber) {
    console.log(fromNumber);
}

countDown(3);
```

This `countDown(3)` shows only the number 3.

To count down from the number 3 to 1, you can:

1. show the number 3.
2. and call the `countDown(2)` that shows the number 2.
3. and call the `countDown(1)` that shows the number 1.

The following changes the `countDown()` to a recursive function:

```
function countDown(fromNumber) {
    console.log(fromNumber);
    countDown(fromNumber-1);
}

countDown(3);
```

This `countDown(3)` will run until the call stack size is exceeded, like this:

```
Uncaught RangeError: Maximum call stack size exceeded.
```

… because it doesn't have the condition to stop calling itself.

The countdown will stop when the next number is zero. Therefore, you add an if condition as follows:

```
function countDown(fromNumber) {
    console.log(fromNumber);

    let nextNumber = fromNumber - 1;

    if (nextNumber > 0) {
        countDown(nextNumber);
    }
}
countDown(3);
```

Output:

```
3
2
1
```

The `countDown()` seems to work as expected.

However, as mentioned in the Function type tutorial, the function's name is a reference to the actual function object.

If the function name is set to `null` somewhere in the code, the recursive function will stop working.

For example, the following code will result in an error:

```
let newYearCountDown = countDown;
// somewhere in the code
countDown = null;
// the following function call will cause an error
newYearCountDown(10);
```

Error:

```
Uncaught TypeError: countDown is not a function
```

How the script works:

- First, assign the `countDown` function name to the variable `newYearCountDown`.
- Second, set the `countDown` function reference to `null`.
- Third, call the `newYearCountDown` function.

The code causes an error because the body of the `countDown()` function references the `countDown` function name, which was set to `null` at the time of calling the function.

To fix it, you can use a named function expression as follows:

```
let countDown = function f(fromNumber) {
    console.log(fromNumber);

    let nextNumber = fromNumber - 1;

    if (nextNumber > 0) {
        f(nextNumber);
    }
}

let newYearCountDown = countDown;
countDown = null;
newYearCountDown(10);
```

2) Calculate the sum of n natural numbers example

Suppose you need to calculate the sum of natural numbers from 1 to n using the recursion technique. To do that, you need to define the `sum()` recursively as follows:

```
sum(n) = n + sum(n-1)
sum(n-1) = n - 1 + sum(n-2)
...
sum(1) = 1
```

The following illustrates the `sum()` recursive function:

```
function sum(n) {
  if (n <= 1) {
    return n;
  }
  return n + sum(n - 1);
}
```

Base Case:

- The function starts with an `if` statement that checks if `n` is less than or equal to 1.
- If `n` is 1 or less, the function returns `n`. This is the base case, which serves as the stopping condition for the recursion.

Recursive Case:

- When n is greater than 1, the base case is not met; the function enters the block after the `if` statement.
- The function returns the sum of `n` and the result of calling itself with the argument `(n - 1)`. This is where the recursion happens.

How it Works:

- For example, if you call `sum(3)`, the function first checks if 3 is less than or equal to 1 (base case not met).
- Since it's not the base case, it calculates `3 + sum(2)`. Now, it calls itself with the argument 2.
- In the next recursive call with `sum(2)`, it calculates `2 + sum(1)`.
- Again, in the next recursive call with `sum(1)`, it reaches the base case and returns 1.
- Now, the previous calls are resolved: `2 + 1` gives 3, and `3 + 3` gives the final result of 6.

Termination:

- The recursion keeps happening, reducing the problem to smaller subproblems until it reaches the base case.
- Once the base case is reached, the function starts to unwind, combining the results from each level of recursion until the final result is obtained.

Summary

- A recursive function is a function that calls itself until it doesn't
- A recursive function always has a condition that stops the function from calling itself.

Quiz