JavaScript Rest Parameters

Summary: in this tutorial, you will learn how to use the JavaScript rest parameters to gather parameters and put them all in an array.

Introduction to JavaScript rest parameters

ES6 provides a new kind of parameter so-called rest parameter that has a prefix of three dots `(...)`. A rest parameter allows you to represent an indefinite number of arguments as an array. See the following syntax:

```
function fn(a,b,...args) {
   //...
}
```

The last parameter (`args`) is prefixed with the three dots (`...`). It's called a rest parameter (`...args`).

All the arguments you pass to the function will map to the parameter list. In the syntax above, the first argument maps to `a`, the second one maps to `b`, and the third, the fourth, etc., will be stored in the rest parameter `args` as an array. For example:

```
fn(1, 2, 3, "A", "B", "C");
```

The `args` array stores the following values:

```
[3,'A','B','C']
```

If you pass only the first two parameters, the rest parameter will be an empty array:

```
fn(1,2);
```

The `args` will be:

```
[]
```

Notice that the rest parameters must appear at the end of the argument list. The following code will result in an error:

```
function fn(a,...rest, b) {
 // error
}
```

Error:

```
SyntaxError: Rest parameter must be last formal parameter
```

More JavaScript rest parameters examples

See the following example:

```
function sum(...args) {
    let total = 0;
    for (const a of args) {
        total += a;
    }
    return total;
}

sum(1, 2, 3);
```

The output of the script is:

```
6
```

In this example, `args` in an array. Therefore, you could use the `for..of` loop to iterate over its elements and sum them up.

Assuming that the caller of the `sum()` function may pass arguments with various kinds of data types such as number, string, and boolean, and you want to calculate the total of numbers only:

```
function sum(...args) {
  return args
    .filter(function (e) {
      return typeof e === 'number';
    })
    .reduce(function (prev, curr) {
      return prev + curr;
    });
}
```

The following script uses the new `sum()` function to sum only numeric arguments:

```
let result = sum(10,'Hi',null,undefined,20);
console.log(result);
```

Output:

```
30
```

Note that without the rest parameters, you have to use the `arguments` object of the function.

However, the `arguments` object itself is not an instance of the `Array` type. Therefore, you cannot use the `filter()` method directly. In ES5, you have to use `Array.prototype.filter.call()` as follows:

```
function sum() {
  return Array.prototype.filter
```

```
    .call(arguments, function (e) {
      return typeof e === 'number';
    })
    .reduce(function (prev, curr) {
      return prev + curr;
    });
}
```

As you see, the rest parameter makes the code more elegant. Suppose you need to filter the arguments based on a specific type such as numbers, strings, boolean, and null. The following function helps you to do it:

```
function filterBy(type, ...args) {
  return args.filter(function (e) {
    return typeof e === type;
  });
}
```

JavaScript rest parameters and arrow function

An arrow function does not have the `arguments` object. Therefore, if you want to pass some arguments to the arrow function, you must use the rest parameters. See the following example:

```
const combine = (...args) => {
  return args.reduce(function (prev, curr) {
    return prev + ' ' + curr;
  });
};

let message = combine('JavaScript', 'Rest', 'Parameters'); // =>
console.log(message); // JavaScript Rest Parameters
```

Output:

```
JavaScript Rest Parameters
```

The `combine()` function is an arrow that takes an indefinite number of arguments and concatenates these arguments.

JavaScript rest parameter in a dynamic function

JavaScript allows you to create dynamic functions through the Function constructor. And it is possible to use the rest parameter in a dynamic function. Here is an example:

```
var showNumbers = new Function('...numbers', 'console.log(numbers)');
showNumbers(1, 2, 3);
```

Output:

```
[ 1, 2, 3 ]
```

In this tutorial, you have learned how to use the JavaScript rest parameter to represent an indefinite number of arguments as an array.