Spread Operator

an introduction to spread operator, limitations of rest parameter and their solution using the spread parameter

In ES5, we often used the `apply` method to call a function with a variable number of arguments. The spread operator makes it possible to achieve the exact same thing in a more compact way.

Suppose you would like to write a method that returns the sum of its arguments. Let's write this function in ES5:

```
function sumArgs() {
    var result = 0;
    for( var i = 0; i < arguments.length; ++i ) {
        result += arguments[i];
    }
    return result;
}

console.log(sumArgs( 1, 2, 3, 4, 5 ));
```

When we know the parameters passed to a function, we have an easy job calling `sumArgs`. However, sometimes it makes little to no sense to write down 100 parameters. In other cases, the number of parameters is not known. This was when the `apply` method of JavaScript was used in ES5.

```
var arr = [];
for( var i = 0; i < 100; ++i ) arr[i] = Math.random();
console.log("Sum:\t"+sumArgs.apply( null, arr ));
```

In ES2015, our job is a lot easier. We can simply use the *spread operator* to call `sumArgs` in the same way as above. The spread operator spreads the elements of an array, transforming them into a parameter list.

```
sumArgs( ...arr );
```

As opposed to rest parameters, there are no restrictions on the location of the *Spread operator* in the parameter list. Therefore, the following call is also valid:

```
sumArgs( ...arr, ...arr, 100 );
```

Strings are spread as arrays of characters

If you would like to process a string character by character, use the spread operator to create an array of one character long strings in the following way:

```
let spreadingStrings = 'Spreading Strings';
let charArray = [ ...spreadingStrings ];
```

In the next lesson, let’s move on to destructuring using the spread operator.