The 80/20 Guide to ES2015 Generators

Valeri Karpov

Table of Contents

Chapter 1: Getting Started

What is a Generator?

Generators are a powerful new feature in ES2015. Generators are far from a new programming construct - they first appeared in 1975 and Python has had them since Python 2.2 in 2001. However, as you'll see, generators are even more powerful in an event-driven language like JavaScript. In JavaScript (assuming Node.js >= 4.0.0), a **generator function** is defined as shown below.

```
const generatorFunction = function*() {
  console.log('Hello, World!');
};
```

However, if you run `generatorFunction`, you'll notice that the return value is an object.

```
$ node
> var generatorFunction = function*() { console.log('Hello, World!'); };
undefined
> generatorFunction()
{}
```

That's because a generator function creates and returns a **generator object**. Typically, the term **generator** refers to a generator object rather than a generator function. A generator object has a single function, `next()`. If you execute the generator object's `next()` function, you'll notice that Node.js printed 'Hello, World!' to the screen.

```
$ node
> var generatorFunction = function*() { console.log('Hello, World!'); };
undefined
> generatorFunction()
{}
> generatorFunction().next()
Hello, World!
{ value: undefined, done: true }
>
```

Notice that `next()` returned an object, `{ value: undefined, done: true }`. The meaning of this object is tied to the `yield` keyword. To introduce you to the `yield` keyword, consider the following generator function.

```
const generatorFunction = function*() {
  yield 'Hello, World!';
};
```

Let's see what happens when you call `next()` on the resulting generator.

```
$ node
> var generatorFunction = function*() { yield 'Hello, World!'; };
undefined
> var generator = generatorFunction();
undefined
> generator.next();
{ value: 'Hello, World!', done: false }
> generator.next();
{ value: undefined, done: true }
>
```

Notice that, the first time you call `generator.next()`, the `value` property is equal to the string your generator function yielded. You can think of `yield` as the generator-specific equivalent of the `return` statement.

You might be wondering why the return value of `generator.next()` has a `done` property. The reason is tied to why `yield` is different from `return`.

`yield` vs `return`

The `yield` keyword can be thought of as a `return` that allows **re-entry**. In other words, once `return` executes, the currently executing function is done forever. However, when you call `generator.next()`, the JavaScript interpreter executes the generator function until the first `yield` statement. When you call `generator.next()` again, the generator function picks up where it left off. You can think of a generator as a function that can "return" multiple values.

```
const generatorFunction = function*() {
  let message = 'Hello';
  yield message;
  message += ', World!';
  yield message;
}();

const generator = generatorFunction();
// { value: 'Hello', done: false };
const v1 = generator.next();
// { value: 'Hello, World!', done: false }
const v2 = generator.next();
// { value: undefined, done: true }
const v3 = generator.next();
```