Introduction

ES6 introduces iteration as a way to traverse over JavaScript data structures. Let's take a look at how this is made possible.

- » Iterable : An object which has enumerable properties and can perform iteration operations. All iterables implement a method `Symbol.iterator` , a special symbol which performs the iteration. This concept allows us to make objects useable in a `for...of` loop which isn't normally possible.
- » Iterator : An iterator traverses over the elements of an iterable, this iterator is returned by `Symbol.iterator` . The iterator returns a method `next` that returns an object with keys : `value` — indicating the current item and `done` — indicating `true` if the traversion is done or `false` if it isn't done.

Table of Contents

Symbol.iterator

The best way to understand how the `Symbol.iterator` method works is by implementing a *makeshift*. For instance, we want to make an object that's not an array to work in a `for...of` loop. First, we create an iterator via a method whose key is `Symbol.iterator` .

- » When the `for...loop` starts it checks if a method `Symbol.iterator` is available and it calls it, else it goes up the prototype chain, if it's still not available it throws a `TypeError` .
- » As stated earlier, this `Symbol.iterator` method returns an iterator object which contains a method `next` .
- » `next` also returns an object with two values : `done` which can either be true or false and `value` which represents the current item.
- » `next` is called repeatedly until it returns an object with the value of `done` to be true.

```js
let obj = {
    start : 1,
    end : 5
};

//for..of initially calls this method
obj[Symbol.iterator] = function(){
    ....
    //iterator object that is returned
    return {
        start : this.start,
        end : this.end,

        //next is called on each iteration
        next : function(){
            if( this.start <= this.end){
                return { done : false, value : this.
            }else{
                return { done : true };
            }
        }
    }
}
```

Note : It is important to know that the object being iterated over(in this case, obj) doesn't have the `next` method , but

rather it is the `Symbol.iterator` method, which when invoked returns an iterator object that contains the `next` method that performs the iteration.

Also note, this is a *makeshift*, the real built-in `Symbol.iterator` method might be more complex and also have some performance optimizations,etc...

Array-likes and Iterables

These are two common terms that are somewhat different but easily misunderstood

- » Array-likes : are objects that have indexes(0,1,2,etc..) and a `length` property just like normal arrays.
- » Iterables : are objects that have the `Symbol.iterator` method implemented.

Array-likes and iterables are generally not arrays and therefore don't have array methods such as shift, unshift, map, etc...ES6 comes with a handy method called `Array.from` that actually converts them into real arrays, we'll talk about it in detail later on.

It is also possible for an iterable not to be an array-like and vice versa. Take a look at the examples below :

```js
//Array-like but not iterable

    let obj = {
        0 : 'dev',
        1 : 'gson',
        length : 2
    };
    //Uncaught TypeError : obj[Symbol.iterator] is r
    for(let arrLike of obj){ }

//Iterable but not array-like

    // No index or length property like array-likes
    let obj = {
        name : 'devgson',
        job : 'devman'
    };

    obj[Symbol.iterator] = function (){
        ...
    }
```

Iterable Sources

I'll use the `for..of` loop to iterate over the following iterable sources. The `Symbol.iterator` is built-in and therefore doesn't need to be implemented.

Arrays

Arrays are iterable(obviously) over their elements and they are the most commonly used iterables.

```js
let arr = [1,2,3];
for(let value of arr){
    console.log(value)
}
//Output :
//1
//2
//3
```

Strings

Strings are both array-like and iterable, but strings are iterable over unicode points.

```js
let str = ' a\uD83D\uDC0A' ;
for (const value of str) {
    console.log(x);
}
// Output:
// 'a'
// '\uD83D\uDC0A' (crocodile emoji)
```

Arguments

Arguments(although less popular now with the advent of rest parameters) are also iterable.

```js
function dev(arguments){
    for(let value of arguments){
        console.log(value);
    }
}
```

```
dev('dev','gson');
//Output :
//'dev'
//'gson'
```

Other iterable sources include Maps, Sets, DOM data structures, etc...

Array.from

```
Array.from(obj,[mapFn, thisArg])
```

ES6 introduced a method that helps us convert array-likes and iterables into real arrays thereby enabling us to be able to use built-in array methods such as shift, unshift,etc.... Let's take a look at how this works on array-likes :

```js
let obj = {
    0 : 'dev',
    1 : 'gson',
    length : 2
};
let arr = Array.from(obj);

for(let value of arr){
    console.log(value);
}
//Output :
//'dev'
//'gson'

console.log(arr.length); //2
```

The exact same thing could be done on the first example where implemented our makeshift `Symbol.iterator` , to make it a real array :

```js
//Example taken from the Symbol.iterator impleme
let arr = Array.from(obj);
```

```
arr.pop();
arr.length; // 1
```

`Array.from` also has a useful second parameter which is a function, this function is executed on each element before adding it to the array, lets take a look at how this works :

```js
let obj = {
    0 : 1,
    1 : 2,
    2 : 3,
    3 : 4,
    length : 4
}
let arr = Array.from( obj , num => num * 2 );
consol.log(arr) // 2,4,6,8
```

Conclusion

All iterables must have the `Symbol.iterator` method implemented.

- » `for...of` loops depend on the `Symbol.iterator` method.
- » Built-in iterables like array and strings have this method built-in.

`Array.from` converts array-likes and iterables into real arrays.

Thanks to

- » Axel Rauschmayer for Exploring ES6—Iteration
- » Ilyar Kantor for Iterable
- » MDN for Iteration protocols (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols)

Like this article?

Follow @Devgson_ on Twitter (https://twitter.com/Devgson_)

A new conversation is coming to
Scotch.IO

Email Address

Conversation

Have a Disqus Account? **Log In**

Be the first to comment...

Terms · Privacy

Add Spot.IM to your site