ГЕНЕРАТОРЫ В JAVASCRIPT. ЧАСТЬ 1

@aqrln

ИТЕРАБЕЛЬНЫЕ ОБЪЕКТЫ И ИТЕРАТОРЫ В ES6

Среди нововведений в ES6, помимо новых синтаксических конструкций и типов данных, есть и два новых интерфейса: протокол итерабельных объектов и протокол итераторов.

MDN:

- Iterable protocol
- Iterator protocol

ITERABLE PROTOCOL

```
let myIterable = {
  ...

  [Symbol.iterator]() {
    let iterator = ...;
    return iterator;
  }

  ...
}
```

ITERATOR PROTOCOL

```
let iterator = {
  ...

  next() {
    // Если ещё остались значения
    return {
      value: currentValue,
      done: false
    };

    // Если значения закончились
    return {
      value: undefined,
      done: true
    }
  }
}
```

ПРИМЕРЫ ИТЕРАБЕЛЬНЫХ ОБЪЕКТОВ

- Array
- TypedArray
- String
- Buffer
- Map
- Set

МЕТОДЫ КОЛЛЕКЦИЙ, КОТОРЫЕ ВОЗВРАЩАЮТ ИТЕРАТОРЫ

- keys()
- values()
- entries()

НЕКОТОРЫЕ ФУНКЦИИ, КОТОРЫЕ ПРИНИМАЮТ ИТЕРАБЕЛЬНЫЕ ОБЪЕКТЫ

- Конструкторы новых коллекций в ES6 (Map, Set...)
- Array.from()
- Promise.all(), Promise.race()

СИНТАКСИЧЕСКИЕ КОНСТРУКЦИИ, КОТОРЫЕ РАБОТАЮТ С ИТЕРАБЕЛЬНЫМИ ОБЪЕКТАМИ

- Цикл for-of
- Spread
- Destructuring assignment
- yield*

ГЕНЕРАТОРЫ

Generator function — это функция, возвращающая специальный итератор (generator object), управляющий её выполнением. Оператор `yield` в теле функции приостанавливает выполнение, метод `next()` итератора — продолжает.

Объекты-генераторы одновременно являются и итерабельными объектами, и итераторами:

```
> function* generatorFunction() { }
undefined
> let generatorObject = generatorFunction()
undefined
> generatorObject.next
[Function: next]
> generatorObject[Symbol.iterator]() === generatorObject
true
```

YIELD

Оператор `yield` служит для двунаправленного обмена данными с генератором. Значение, стоящее после ключевого слова `yield`, становится текущим значением итератора, а в метод `next()` опционально можно передать значение, которое примет yield-выражение.

YIELD*

```
yield* from iterable;
```

≈

```
for (let value of iterable) {
  yield value;
}
```

YIELD* EXPRESSION

```
function* g1() {
  yield* [1, 2, 3];
  return 'result';
}

let result = null;

function* g2() {
  result = yield* g1();
}

console.log([...g2()]); // [1, 2, 3]
console.log(result);    // 'result'
```

Q&A