Modern JS

ITERATORS & GENERATORS & ASYNC/AWAIT

Overview

- В ES6 (он же ES2015) очень много нового
- Итераторы и Iterator Protocol
- Генераторы и зачем они нужны
- Зачем тут промисы
- async/await

for … of … cycle

В современном JS есть конструкция for … of ... – еще один цикл.

```
let arr = [ 1, 2, 3, 5, 8, 13, 21, 35 ];
for (let i in arr) { console.log(i); } // всем знакомо? Индексы
for (let i of arr) { console.log(i); } // выведет значения

obj = { a: 1, b: 2, c: 3 }
for (let i in obj) { console.log(i); } // названия свойств
for (let i of obj) { console.log(i); } // не взлетит
```

for … of … cycle + iterators

Чтобы цикл «взлетел» на произвольно взятом объекте, объект должен поддерживать **Iterable protocol**, т.е. иметь свойство с именем **Symbol.iterator** – в котором находится функция, возвращающая итератор (напрямую или у одного из прототипов) – объект с методом next().

```
a = { a: 1, b: 2, c: 3, [Symbol.iterator]: myIteratorFunction }

function myIteratorFunction() {
   let i = 0;
   let keys = Object.keys(this);
  return { next: () => (!!keys[i] ? { value: keys[i++], done: false } : { value: null, done: true}) }
}


for (let i of a) { console.log(i); } // 1, 2, 3
```

Iterators are everywhere

- String
- Array
- TypedArray

  (Uint8Array, etc)
- Map
- Set
- arguments

Iterables are accepted

- new Map(iterable)
- new WeakMap(iterable)
- new Set(iterable)
- new WeakSet(iterable)
- Array.from(iterable)
- Promise.all(iterable)
- Promise.race(iterable)

Array spread & destructuring

Есть такая фича – раскрытие массива на список элементов:

a = [1, 2, 3]

b = [4, 5]

c = [a, b] // массив из двух массивов

d = [...a, ...b] // [1 ... 5]

И деструктуризация:

let [e, f] = a // внутри e будет 1, в f – 2

[e, ...f] = a // e = 1, f = [2, 3]

Все эти прелести работают и если вместо массива справа будет итерируемый объект.

Generator functions

```
function* testGen() {
  yield 1;
  yield 2;
  yield 3;
  return 100;
}
```

Вот это и есть генератор. Зачем надо?

1) Автоматически формирует итератор
2) Не теряет контекст между вызовами yield (**самое главное**)

Generator execution

Как используется – первый вызов функции-генератора возвращает итератор (объект с функцией ***next()***).

function* testGen() { yield 1; let b = Date.now(); yield b; }

Итератор имеет прототип (свойство __proto__) Generator, и (в случае Хрома), нативную реализацию next(). Для Babel всё страшно.

Первый вызов next() запускает код генератора.

Порядок выполнения внутри генератора – прерывистый. Он приостанавливается, когда достигает оператора yield. В этой точке данные отправляются наружу, а сам генератор встает.

Последующие вызовы next() восстанавливают исполнение.

Generator example

```
a = { a: 1, b: 2, c: 3, [Symbol.iterator]: function* () {
  let keys = Object.keys(this);
  for (let i = 0; i < keys.length; i++) {
    yield this[keys[i]];
  }
}

console.log(...a);
for (let i of a) { console.log(i); }
```

Generator calls generator

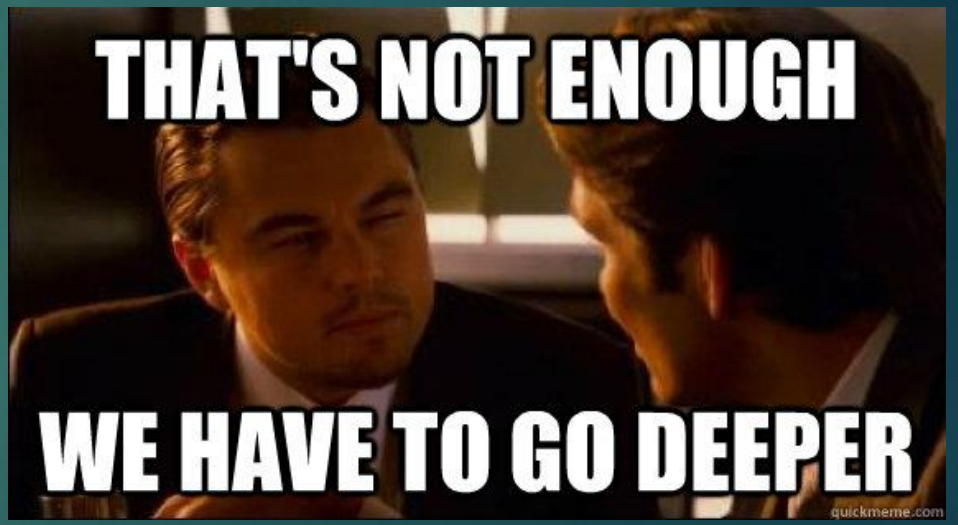
Генератор может делегировать свое исполнение другому

```
function* xrange(from, to, step = 1) {
   for (let i = from; i <= to; i += step) { yield i; }
}

function* shifts() {
  yield* xrange(1, 9);
  yield 0;
}

for (let shiftNr of shifts()) { console.log(shiftNr); } // 1 … 9, 0
```

Recursive generator

```
let a = { name: 'abc', children: [{name:'bcd'}, {name:'cde', children:
[{name: 'def'}, {name:'efg'}]}]};
```

```
function* traverse(node) {
    yield node.name;
    if (node.children && node.children.length > 0)
        for (let i = 0; i < node.children.length; i++)
            yield* traverse(node.children[i]);
}
```




```
for(let x of traverse(a)) { console.log(x); }
```

```
[...traverse(a)] // массив из всех имен элементов
```

Don’t panic

Бывает надо выполнить большую задачу, не убивая браузер:

```
let sum = 0;
for (let i = 1; i < 5000000; i++) { sum += Math.random(); }
```

Планировщик в помощь:

```
function scheduler(task) {
  setTimeout(() => { if (!task.next().done) { scheduler(task); } }, 0);
}
```

Задача – в генераторе, отдает иногда управление через yield.

Generator receives data

Генератор может получать данные из вызова next()

```
function* getData() {
  console.log(“value: ” + (yield));
}

let gd = getData();
for (let i = 0; i < 5 ; i++) {
   console.log(“i: “ + i); gd.next(i);
}
```

Synchronous async functions

Генераторы позволяют провернуть фокус – сделать асинхронный код «синхронным»

```
function* main() {
    let result = yield new Promise(
            resolve => { setTimeout( () => { resolve(5); }, 2000); }
            );
    result += 10;
    console.log(result);
}

let m = main();
m.next().then(m.next);
```

Co

Последний фокус – легко обобщается, когда генератор выдает промисы всегда, и внешний «исполнитель» отдает разрешенные промисы генератору на вход через next().

Есть библиотека **Co** (https://github.com/tj/co), которая делает именно это:

```
co(function* () {
  // resolve multiple promises in parallel
  var a = Promise.resolve(1);
  var b = Promise.resolve(2);
  var c = Promise.resolve(3);
  var res = yield [a, b, c];
  console.log(res); // => [1, 2, 3]
})
```

Async/Await

ECMAScript 2017 предлагает сахар для промисов – async функции и оператор await.

Уже поддерживаются – Babel (основное, использует генераторы внутри), текущие Firefox, Chrome, Edge 15, Safari 10.1, Node.js > 7.x

Реализовано сначала в C# в 2012 году. Синтаксис – один в один из C#. Автор предложения в TC#39 – сотрудник Microsoft.

Принят в начале 2017, внесён в начале 2014

Async/Await

```
function getVal(x ,delay = 2000) {
  return new Promise(
    resolve => {
      setTimeout( () => { resolve(x); }, 2000);
    });
}


async function test() {
  let a = getVal(100);
  let b = getVal(500, 3000);
  return 1 + await a + await b;
}


test().then(console.log);
```

Links

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function*
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function
- https://learn.javascript.ru/generator
- https://github.com/tj/co
- http://2ality.com/2013/06/iterators-generators.html