Новое в JavaScript ES.Next, ECMAScript 2020

github.com/HowProgrammingWorks

Тимур Шемсединов

Chief Software Architect at Metarhia
Lecturer at Kiev Polytechnic Institute

github.com/tshemsedinov

ECMA Script versions

ES1	Jun 1997
ES2	Jun 1998
ES3	Dec 1999
ES5	Dec 2009
ES5.1	Jun 2011
ES6	ES2015
ES7	ES2016
ES8	ES2017
ES9	ES2018
ES10	ES2019
ES11	ES2020
ES.Next	

Links

ES2020	https://tc39.es/ecma262/
Node Green	https://node.green/
Proposals	https://github.com/tc39/proposals
Can I use	https://caniuse.com/

Array

Array.prototype.includes(value[, fromIndex])

Array.prototype.flat([depth])
Array.prototype.flatMap(callback[, thisArg])

Array.prototype.sort([compareFunction])
    QuickSort to TimSort

Object

Object.values(object)
Object.keys(object)
Object.entries(object)
Object.fromEntries(object)

String

String.prototype.padStart(targetLength [, padString])
String.prototype.padEnd(targetLength [, padString])
String.prototype.trimStart()
String.prototype.trimEnd()

Operators

Rest

```
const f = (a, b, ...array) => {};
const g = ({ a, b, ...array }) => {};
const { name, ...rest } = obj;
```

Spread

```
f(a, b, ...array);
const obj2 = { name, ...obj1 };
const clone = { ...obj };
```

Operators

Exponentiation

```
Math.pow(x, y)    x ** y    x **= y    x = x ** y
```

Optional chaining

```
const spqr = {
  emperor: { name: 'Marcus' }
};
console.log(spqr.emperor?.name);
console.log(spqr.president?.name);
```

Asynchronous function: async/await

```
const fn = async (a, b, c) => {
  // do something
  await callSomething();
  // do something
  return aValue;
};
```

Trailing commas

```
const fn = (arg1, arg2, arg3,) => {
  console.log({
    arg1,
    arg2,
    arg3,
  });
};
fn(...['val1', 'val2', ], 'val3',);
```

Asynchronous iterable contract

Symbol.iterator

```
iterable[Symbol.iterator]()
```

Symbol.asyncIterator

```
asyncIterable[Symbol.asyncIterator]()
```

Try...catch

```
try {
  throw new Error('message');
} catch {
  console.log('no arguments catched');
}
```

Function

```
((a, b) => {
    const c = a + b; // hello there
    return c;
}).toString()
"(a, b) => {
    const c = a + b; // hello there
    return c;
}"
```

Symbol

```
const sym = Symbol('description');

console.log(sym);
// Symbol(description);

console.log(sym.description);
// description
```

Promise.finally

```
new Promise(executor)
  .then(onFulfilled[, onRejected])
  .catch(onRejected)
  .finally(onFinally);
```

Promise.allSettled

```
const p1 = Promise.resolve('p1');
const p2 = new Promise((resolve, reject) => {
  setTimeout(resolve, 1000, 'p2');
});
const p3 = new Promise((resolve, reject) => {
 setTimeout(reject, 100, 'p3');
});

Promise.all([p1, p2, p3]).then(values => {
 console.log(values);
});
```

Promise.allSettled

```
Promise.all([p1, p2, p3]).then(values => {
 console.log(values);
});
```

```
node:26549) UnhandledPromiseRejectionWarning: p3
(node:26549) UnhandledPromiseRejectionWarning: Unhandled promise
rejection. This error originated either by throwing inside of an
async function without a catch block, or by rejecting a promise
which was not handled with .catch(). (rejection id: 1)
(node:26549) [DEP0018] DeprecationWarning: Unhandled promise
rejections are deprecated. In the future, promise rejections that
are not handled will terminate the Node.js process with
a non-zero exit code.
```

Promise.allSettled

```
const p1 = Promise.resolve('p1');
const p2 = new Promise((resolve, reject) => {
  setTimeout(resolve, 1000, 'p2');
});
const p3 = new Promise((resolve, reject) => {
 setTimeout(reject, 100, 'p3');
});

Promise.allSettled([p1, p2, p3]).then(values => {
 console.log(values);
});
```

Promise.allSettled

```
Promise.allSettled([p1, p2, p3]).then(values => {
 console.log(values);
});

[
  { status: 'fulfilled', value: 'p1' },
  { status: 'fulfilled', value: 'p2' },
  { status: 'rejected', reason: 'p3' }
]
```

More features

Atomics
SheredArrayBuffer
Set, Map, WeakSet, WeakMap
globalThis
Private fields
Static fields