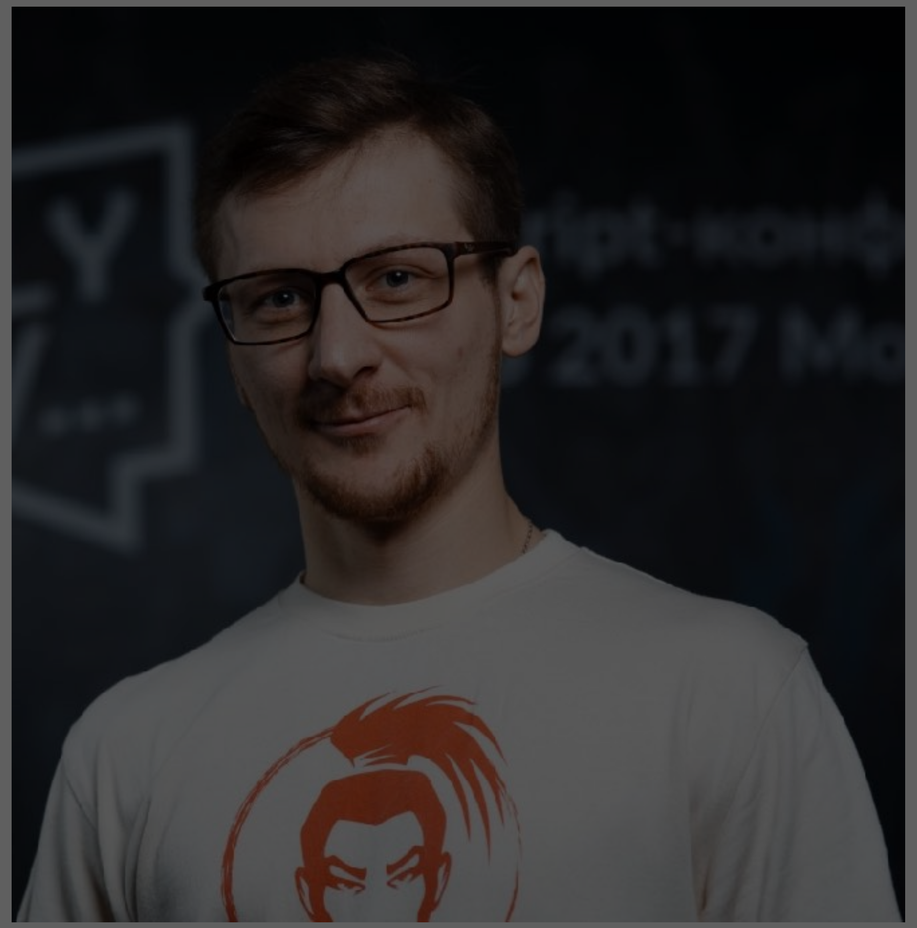
JS

Mikhail Poluboyarinov

@mike1pol

Mikhail Poluboyarinov



@mike1pol

HEALTH SAMURAi

JS

<piter/>
<united/>

HOLY JS/...

NODESCHOOL

EcmaScript

EcmaScript

TC39

TC39
meeting
23-29 January 2018

tc39/agendas

TC39 Process

stage-0 - Strawman

stage-1 - Proposal

stage-2 - Draft

stage-3 - Candidate

stage-4 - Finished

TC39 Process

stage-4 - Finished

Что нового?

4 новых метода

4 новых регулярных выражения

Регулярные выражения

RegExp named capture groups
(Gorkem Yakin, Daniel Ehrenberg)

Motivation

Чтобы группе регулярного выражения можно было присвоить имя, пример: ***(?<name>…)***.

Examples

Example #1

```
const RE_DATE =
  /(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/u;
```

Example #1

```
const RE_DATE =
  /(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/u;
```

Backreferences

```
let duplicate = /^(?<half>.-).\k<half>$/u;
```

Replacement targets

```
let re =
  /(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/u;
let result = '2018-05-20'
    .replace(re, '$<day>.$<month>.$<year>');
console.log(result); // 20.05.2018
```

Replacement targets

```
let re =
  /(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/u;
let result = '2018-05-20'.replace(re, ( ... args) => {
  let {day, month, year} = args[args.length - 1];
  return `${day}.${month}.${year}`;
});
console.log(result); // '20.05.2018'
```

Implementations

- V8, shipping in Chrome 64
- Transpiler (Babel plugin)
- Safari beginning in Safari Technology Preview 40

RegExp Unicode Property Escapes
(Mathias Bynens)

Motivation

Solution

```
1    const regexGreekSymbol = /\p{Script=Greek}/u;
2    regexGreekSymbol.test('π'); // → true
```

Hi-level API

```
\p{UnicodePropertyName=UnicodePropertyValue}
```

`\p{General_Category=Letter}`.

```
\p{LoneUnicodePropertyNameOrValue}
```

`\p{Letter}`

Hi-level API

```
\P{…}
```

How it work now?

Библиотека XRegExp:

```
1    const regexGreekSymbol = XRegExp('\\p{Greek}', 'A');
2    regexGreekSymbol.test('π'); // true
```

Недостатки:

- Внешняя зависимость
- Работает медленней
- Размер в сжатом виде более 35кб
- Обновление Unicode стандарта

Библиотека Regenerate

```
1  const regenerate = require('regenerate');
2  const codePoints = require('unicode-9.0.0/Script/Greek/code-points.js');
3  const set = regenerate(codePoints);
4  set.toString();
5  // → '[\u0370-\u0373\u0375-\u0377\u037A-\u037D\u037F\u0384\u0386\u0388
     -\u038A\u038C\u038E-\u03A1\u03A3-\u03E1\u03F0-\u03FF\u1D26-\u1D2A\u1D5D
     -\u1D61\u1D66-\u1D6A\u1DBF\u1F00-\u1F15\u1F18-\u1F1D\u1F20-\u1F45\u1F48
     -\u1F4D\u1F50-\u1F57\u1F59\u1F5B\u1F5D\u1F5F-\u1F7D\u1F80-\u1FB4\u1FB6
     -\u1FC4\u1FC6-\u1FD3\u1FD6-\u1FDB\u1FDD-\u1FEF\u1FF2-\u1FF4\u1FF6-\u1FFE
     \u2126\uAB65]|\uD800[\uDD40-\uDD8E\uDDA0]|\uD834[\uDE00-\uDE45]'
6  // Imagine there's more code here to save this pattern to a file.
```

Библиотека Regenerate

- Оптимальная производительность

Недостатки:

- Внешняя зависимость
- Регулярные выражения имеют большой размер
- Необходима сборка
- Обновление Unicode стандарта

Examples

Unicode-aware version of `\d`

```
1  const regex = /^\p{Decimal_Number}+$/u;
2  regex.test('1234567890123456'); // true
```

Unicode-aware version of `\D`

```
1  const regex = /^\P{Decimal_Number}+$/u;
2  regex.test('Իմ օդաթիռը լի է օձաձկերով'); // true
```

Unicode-aware version of \w \W

```
const regex =
  /([\p{Alphabetic}\p{Mark}\p{Decimal_Number}\p{Connector_Punctuation}\p{Join_Control}]+)/gu;
const text = `
Amharic: የኔ ማንዣበቢያ መኪና በዓሣዎች ተሞልቷል
Bengali: আমার হভারক্রাফ্ট কুঁচে মাছ-এ ভরা হয়ে গেছে
Georgian: ჩემი ხომალდი საჰაერო ბალიშზე სავსეა გველთევზებით
Macedonian: Моето летачко возило е полно со јагули
Vietnamese: Tàu cánh ngầm của tôi đầy lươn
`;
let match;
while (match = regex.exec(text)) {
  const word = match[1];
  console.log(`Matched word with length ${ word.length }: ${ word }`);
}
```

Matching emoji

Поиск эмоджи происходит в соответствии с спецификацией UTR51

```
const regex =
  /\p{Emoji_Modifier_Base}\p{Emoji_Modifier}?|\p
  {Emoji_Presentation}|\p{Emoji}\uFE0F/gu;
```

1. стандартные модификаторы эмоджи
   (\p{Emoji_Modifier_Base\p{Emoji_Modifier}?}); **\u{231A}**

2. любые оставшиеся символы, которые отображаются
   как эмоджи, не текст **(\p{Emoji_Presentation});**

Matching emoji

```
const regex =
  /\p{Emoji_M                                                    n}|\p{Emoji}
\uFE0F/gu;
const text =
\u{231A}: ⌚ d                                                   ion)
\u{2194}\u{FE                                                   as emoji

\u{1F469}: 👩
\u{1F469}\u{1
`;
let match;
while (match =
  const emoji
  console
    .log(`Mat                                                   ngth}`);
}
```

```
Matched sequence ⌚ — code points: 1
Matched sequence ⌚ — code points: 1
Matched sequence ↔️ — code points: 2
Matched sequence ↔️ — code points: 2
Matched sequence 👩 — code points: 1
Matched sequence 👩 — code points: 1
Matched sequence 👩🏿 — code points: 2
Matched sequence 👩🏿 — code points: 2
```

Implementations

- V8, shipping in Chrome 64
- Safari/JavaScriptCore beginning in Safari Technology Preview 42

- regexpu (transpiler) with the { unicodePropertyEscape: true } option enabled
  - online demo
  - exhaustive list of supported properties
  - Babel plugin

s (dotAll) flag for regular expressions (Mathias Bynens)

Motivation

В регулярных выражения, точка `.` соответствует одиночному символу, независимо от того какой символ. В ECMAScript, есть 2 исключения:

- не работает для unicode. Установка флага `u` (`unicode`) исправляет это.
- не работает для переноса строк.

Что-бы это обойти мы используем различные хаки `[\s\S]` или `[^]`:

```
/foo[^]bar/.test('foo\nbar');
// → true
```

Solution

Добавить новый флан `s` в спецификацию **ECMAScript regular expressions** что-бы `.` находила все символы включая переносы строк.

```
/foo.bar/s.test('foo\nbar');
// → true
```

High-level API

```
const re = /foo.bar/s;
// Or, const re = new RegExp('foo.bar', 's');
re.test('foo\nbar'); // → true
re.dotAll // → true
re.flags // → 's'
```

Implementations

- V8, shipping in Chrome 62
- JavaScriptCore, shipping in Safari Technology Preview 39a
- regexpu (transpiler) with the `{ dotAllFlag: true }` option enabled
  - online demo
  - Babel plugin
- Compat-transpiler of RegExp Tree
  - Babel plugin

RegExp Lookbehind Assertions

(Gorkem Yakin, Nozomu Katō, Daniel Ehrenberg)

Lookahead assertions

A positive lookahead assertion

```
const RE_AS_BS = /HolyJS(?=\sSPB)/;
const match1 = RE_AS_BS.exec('HolyJS SPB 2018');
console.log(match1); // ["HolyJS"]

const match2 = RE_AS_BS.exec('HolyJS MSK 2018');
console.log(match2); // null
```

A negative lookahead assertion means that what comes next must *not* match the assertion.

```
const RE_AS_NO_BS = /HolyJS(?!\sNSK)/;
RE_AS_NO_BS.test('HolyJS NSK 2018') // false
RE_AS_NO_BS.test('HolyJS SPB 2018') // true
RE_AS_NO_BS.test('HolyJS MSK 2018') // true
```

Lookbehind assertions

Positive lookbehind assertions

```
const RE_DOLLAR_PREFIX = /(?<=Holy)js/g;
'PiterJS MoscowJS Holyjs'.replace(RE_DOLLAR_PREFIX, 'JS');
// 'PiterJS MoscowJS HolyJS'
```

Negative lookbehind assertions

```
const RE_NO_DOLLAR_PREFIX = /(?<!Holy)js/g;
'Piterjs Moscowjs HolyJS'.replace(RE_NO_DOLLAR_PREFIX, 'JS');
// 'PiterJS MoscowJS HolyJS'
```

Conclusions

Выражения **Lookahead** имеют наибольший смысл в конце регулярных выражений.
Выражения **Lookbehind** имеют наибольший смысл в начале регулярных выражений.

Где можно использовать:

- **`replace()`**
- **`match()`** - особенно если регулярное выражение с флагом **`/g`**
- **`split()`** **-** например так:

```
'a, b,c'.split(/,(?= )/)
// [ 'a', ' b,c' ]
```

Implementations

- V8, shipping in Chrome 62

Новые методы

Promise.prototype.finally()
(Jordan Harband)

JavaScript

```
function getChats() {
  const loader = document.querySelector('#loader');
  loader.style.display = 'block';
  fetch('/chats')
    .then(data => {
      // handling success
      loader.style.display = 'none';
    })
    .catch(err => {
      // handling error
      loader.style.display = 'none';
    })
}
getChats();
```

JavaScript

```
function getChats() {
  const loader = document.querySelector('#loader');
  loader.style.display = 'block';
  fetch('/chats')
    .then(data => {
      // handling success
    })
    .catch(err => {
      // handling error
    })
    .finally(() => loader.style.display = 'none')
}
getChats();
```

Node.js

```
function getChats() {
  let connection;
  db.open()
    .then(c => {
      connection = c;
      // work with db
    })
    .catch(() => {
      // handling error
    })
    .finally(() => db.close())
}
```

Asynchronous Iteration
(Domenic Denicola, Kevin Smith)

Synchronous Iteration

Synchronous Iteration

```
1    const iterable = ["a", "b"];
2    const iterator = iterable[Symbol.iterator]();
3    iterator.next(); // { value: "a", done: false }
4    iterator.next(); // { value: "b", done: false }
5    iterator.next(); // { value: undefined, done: true }
```

Asynchronous Iteration
(Domenic Denicola, Kevin Smith)

TypeScript notation

TypeScript notation

```typescript
interface IteratorResult {
    value: any;
    done: boolean;
}
interface AsyncIterator {
    next(): Promise<IteratorResult>;
}
interface AsyncIterable {
    [Symbol.asyncIterator](): AsyncIterator;
}
```

Example

```
const asyncIterable = createAsyncIterable(["a", "b"]);
const asyncIterotor = asyncIterable[Symbol.asyncIterator]();
asyncIterotor
  .next()
  .then(res => {
    console.log(res); // { value: "a", done: false }
    return asyncIterotor.next();
  })
  .then(res => {
    console.log(res); // { value: "b", done: false }
    return asyncIterotor.next();
  })
  .then(res => {
    console.log(res); // { value: undefined, done: true }
  });
```

Example

```
const asyncIterable = createAsyncIterable(["a", "b"]);
const asyncIterotor = asyncIterable[Symbol.asyncIterator]();
asyncIterotor
  .next()
  .then(res => {
    console.log(res); // { value: "a", done: false }
    return asyncIterotor.next();
  })
  .then(res => {
    console.log(res); // { value: "b", done: false }
    return asyncIterotor.next();
  })
  .then(res => {
    console.log(res); // { value: undefined, done: true }
  });
```

Example

```
1   const asyncIterable = createAsyncIterable(["a", "b"]);
2   const asyncIterotor = asyncIterable[Symbol.asyncIterator]();
3   asyncIterotor
4     .next()
5     .then(res => {
6       console.log(res); // { value: "a", done: false }
7       return asyncIterotor.next();
8     })
9     .then(res => {
10      console.log(res); // { value: "b", done: false }
11      return asyncIterotor.next();
12    })
13    .then(res => {
14      console.log(res); // { value: undefined, done: true }
15    });
```

Example

```
1   const asyncIterable = createAsyncIterable(["a", "b"]);
2   const asyncIterotor = asyncIterable[Symbol.asyncIterator]();
3   asyncIterotor
4     .next()
5     .then(res ⇒ {
6       console.log(res); // { value: "a", done: false }
7       return asyncIterotor.next();
8     })
9     .then(res ⇒ {
10      console.log(res); // { value: "b", done: false }
11      return asyncIterotor.next();
12    })
13    .then(res ⇒ {
14      console.log(res); // { value: undefined, done: true }
15    });
```

Example

```
const asyncIterable = createAsyncIterable(["a", "b"]);
const asyncIterotor = asyncIterable[Symbol.asyncIterator]();
asyncIterotor
  .next()
  .then(res => {
    console.log(res); // { value: "a", done: false }
    return asyncIterotor.next();
  })
  .then(res => {
    console.log(res); // { value: "b", done: false }
    return asyncIterotor.next();
  })
  .then(res => {
    console.log(res); // { value: undefined, done: true }
  });
```

Example

```
const asyncIterable = createAsyncIterable(["a", "b"]);
const asyncIterotor = asyncIterable[Symbol.asyncIterator]();
asyncIterotor
  .next()
  .then(res => {
    console.log(res); // { value: "a", done: false }
    return asyncIterotor.next();
  })
  .then(res => {
    console.log(res); // { value: "b", done: false }
    return asyncIterotor.next();
  })
  .then(res => {
    console.log(res); // { value: undefined, done: true }
  });
```

Example

```
const asyncIterable = createAsyncIterable(["a", "b"]);
const asyncIterotor = asyncIterable[Symbol.asyncIterator]();
asyncIterotor
  .next()
  .then(res => {
    console.log(res); // { value: "a", done: false }
    return asyncIterotor.next();
  })
  .then(res => {
    console.log(res); // { value: "b", done: false }
    return asyncIterotor.next();
  })
  .then(res => {
    console.log(res); // { value: undefined, done: true }
  });
```

for-await-of

```
async function f() {
  for await (const x of createAsyncIterable(["a", "b"])){
    console.log(x);
  }
}

f(); // Output:
     // "a"
     // "b"
```

for-await-of and rejections

```
1   function createRI () {
2     return {
3       [Symbol.asyncIterator] () { return this; },
4       next() { return Promise.reject("Problem"); }
5     }
6   }
7
8   (async function() {
9     try {
10      for await (const x of createRI()) {
11        console.log(x);
12      }
13    } catch (e) {
14      console.error(e) // Error: Problem
15    }
16  })();
```

for-await-of and rejections

```
function createRI () {
  return {
    [Symbol.asyncIterator] () { return this; },
    next() { return Promise.reject("Problem"); }
  }
}

(async function() {
  try {
    for await (const x of createRI()) {
      console.log(x);
    }
  } catch (e) {
    console.error(e) // Error: Problem
  }
})();
```

for-await-of and rejections

```
function createRI () {
  return {
    [Symbol.asyncIterator] () { return this; },
    next() { return Promise.reject("Problem"); }
  }
}

(async function() {
  try {
    for await (const x of createRI()) {
      console.log(x);
    }
  } catch (e) {
    console.error(e) // Error: Problem
  }
})();
```

Rest/Spread properties
(Sebastian Markbåge)

The rest operator (`...`) in object destructuring

The spread operator (`...`) in object literals

The rest operator (...) in object destructuring

```
const talk = {
  speaker: "Mikhail Poluboyarinov",
  conference: "HolyJS",
  theme: "Чего ждать от JavaScript в 2018 году?",
  description: "Some text..."
};
const {speaker, conference, ...rest} = obj;
console.log(speaker); // Mikhail Poluboyarinov
console.log(conference); // HolyJS
console.log(rest);
// { theme: "...", description: "..." }
```

The rest operator (...) in object destructuring

```
const talk = {
  speaker: "Mikhail Poluboyarinov",
  conference: "HolyJS",
  theme: "Чего ждать от JavaScript в 2018 году?",
  description: "Some text..."
};
const {speaker, conference, ...rest} = obj;
console.log(speaker); // Mikhail Poluboyarinov
console.log(conference); // HolyJS
console.log(rest);
// { theme: "...", description: "..." }
```

The rest operator (...) in object destructuring

```
const talk = {
  speaker: "Mikhail Poluboyarinov",
  conference: "HolyJS",
  theme: "Чего ждать от JavaScript в 2018 году?",
  description: "Some text..."
};
const {speaker, conference, ...rest} = obj;
console.log(speaker); // Mikhail Poluboyarinov
console.log(conference); // HolyJS
console.log(rest);
// { theme: "...", description: "..." }
```

The rest operator (...) in object destructuring

```
function f({price, count, ...rest}) {
  console.log('rest params:', rest);
  return price * count;
}
const sum = f({ count: 5, price: 10, name: "Ботинки" });
// rest params: { name: "Ботинки" }
console.log(sum); //  50
```

The rest operator (...) in object destructuring

```
function f({price, count, ...rest}) {
  console.log('rest params:', rest);
  return price * count;
}
const sum = f({ count: 5, price: 10, name: "Ботинки" });
// rest params: { name: "Ботинки" }
console.log(sum); //  50
```

The rest operator (...) in object destructuring

```
function f({price, count, ...rest}) {
  console.log('rest params:', rest);
  return price * count;
}
const sum = f({ count: 5, price: 10, name: "Ботинки" });
// rest params: { name: "Ботинки" }
console.log(sum); //  50
```

The rest operator (...) in object destructuring

```
const { ...rest, obj} = obj;
// Syntax error
const {obj, ...rest, ...rest} = obj;
// Syntax error
```

The rest operator (...) in object destructuring

```
const shoes = {
  name: "Ботинки",
  price: {
    main: 100,
    discount: 90,
    wholesale: 80
  },
  description: "Some text...",
  photos: ["url", "url"]
};
const {
  name,
  price: {main, ...priceRest},
  ...shoesRest
} = shoes;
console.log(name); // Ботинки
console.log(main); // 100
console.log(priceRest); // { discount: 90, wholesale: 80 }
console.log(shoesRest); // { description: "...", phoros: [...] }
```

The rest operator (...) in object destructuring

```
const shoes = {
  name: "Ботинки",
  price: {
    main: 100,
    discount: 90,
    wholesale: 80
  },
  description: "Some text...",
  photos: ["url", "url"]
};
const {
  name,
  price: {main, ...priceRest},
  ...shoesRest
} = shoes;
console.log(name); // Ботинки
console.log(main); // 100
console.log(priceRest); // { discount: 90, wholesale: 80 }
console.log(shoesRest); // { description: "...", phoros: [...] }
```

The rest operator (...) in object destructuring

```
const shoes = {
  name: "Ботинки",
  price: {
    main: 100,
    discount: 90,
    wholesale: 80
  },
  description: "Some text...",
  photos: ["url", "url"]
};
const {
  name,
  price: {main, ...priceRest},
  ...shoesRest
} = shoes;
console.log(name); // Ботинки
console.log(main); // 100
console.log(priceRest); // { discount: 90, wholesale: 80 }
console.log(shoesRest); // { description: "...", phoros: [...] }
```

The spread operator (...) in object literals

```
const shoes = {
  name: "Ботинки",
  description: "Some text..."
};
const sw1 = {...shoes, price: 100};
console.log(sw1);
// { name: "Ботинки", description: "Some text...", price: 100 }
const sw2 = {price: 100, ...shoes};
console.log(sw2)
// { price: 100, name: "Ботинки", description: "Some text..." }
```

The spread operator (...) in object literals

```
const shoes = {
  name: "Ботинки",
  description: "Some text..."
};
const sw1 = {...shoes, price: 100};
console.log(sw1);
// { name: "Ботинки", description: "Some text...", price: 100 }
const sw2 = {price: 100, ...shoes};
console.log(sw2)
// { price: 100, name: "Ботинки", description: "Some text..." }
```

The spread operator (...) in object literals

```
const shoes = {
  name: "Ботинки",
  description: "Some text..."
};
const sw1 = {...shoes, price: 100};
console.log(sw1);
// { name: "Ботинки", description: "Some text...", price: 100 }
const sw2 = {price: 100, ...shoes};
console.log(sw2)
// { price: 100, name: "Ботинки", description: "Some text..." }
```

The spread operator (...) in object literals

```
const shoes = {
  name: "Ботинки",
  description: "Some text..."
};
const sw1 = {...shoes, price: 100};
console.log(sw1);
// { name: "Ботинки", description: "Some text...", price: 100 }
const sw2 = {price: 100, ...shoes};
console.log(sw2)
// { price: 100, name: "Ботинки", description: "Some text..." }
```

The spread operator (...) in object literals

```
const clone = {...object};
// Some as:
const clone = Object.assign({}, object);

const merge = {...object1, ...object2};
// Some as:
const merge = Object.assign({}, object1, object2);
```

Template Literal Revision
(Tim Disney)

String.raw

```
console.log(`\u{4B}`); // K
console.log(String.raw`\u{4B}`)
```

How?

```
function testTag(strings) {
  return {
    cookied: strings,
    raw: strings.raw
  };
}
console.log(testTag`\u{4B}`);
// { cookied: "K", raw: "\\u{4B}"}
```

How?

```
function testTag(strings) {
  return {
    cookied: strings,
    raw: strings.raw
  };
}
console.log(testTag`\u{4B}`);
// { cookied: "K", raw: "\\u{4B}"}
```

GraphQL tag

```
const gql = require("graphql-tag")
const q = gql`
query {
  users {
    id
    name
  }
}
`;
console.log(q);
```

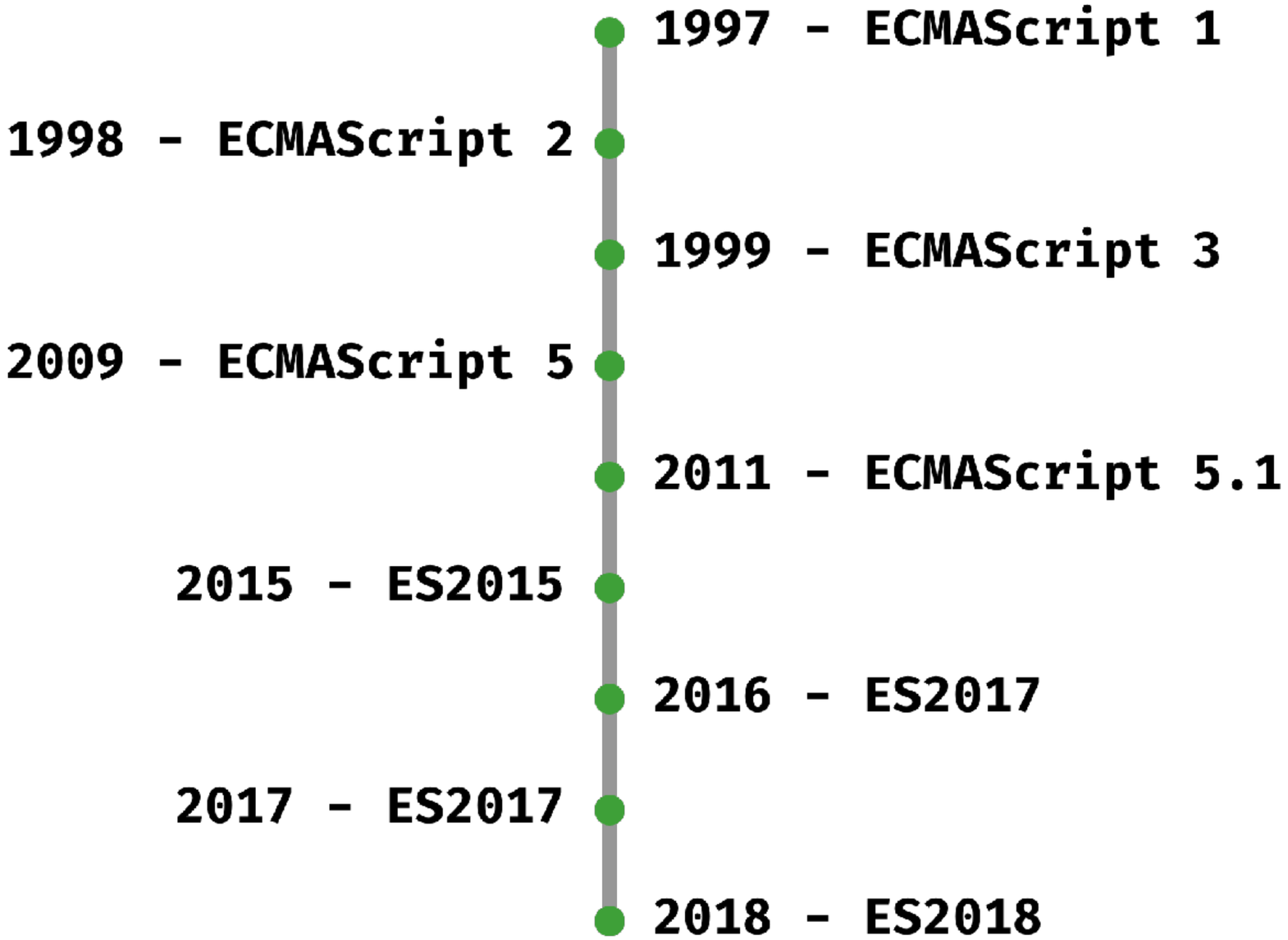
GraphQL tag


```
{
  "kind": "Document",
  "definitions": [{
    "kind": "OperationDefinition",
    "operation": "query",
    "variableDefinitions": [],
    "directives": [],
    "selectionSet": {
      "kind": "SelectionSet",
      "selections": [{
        "kind": "Field",
        "name": {"kind": "Name", "value": "users"},
        "arguments": [],
        "directives": [],
        "selectionSet": {
          "kind": "SelectionSet",
          "selections": [{
            "kind": "Field",
            "name": {"kind": "Name", "value": "id"},
            "arguments": [],
            "directives": []
          }, {"kind": "Field", "name": {"kind": "Name", "value": "name"},
"arguments": [], "directives": []}]
        }
      }]
    }
  }],
  "loc": {"start": 0, "end": 41}
}
```

Выводы

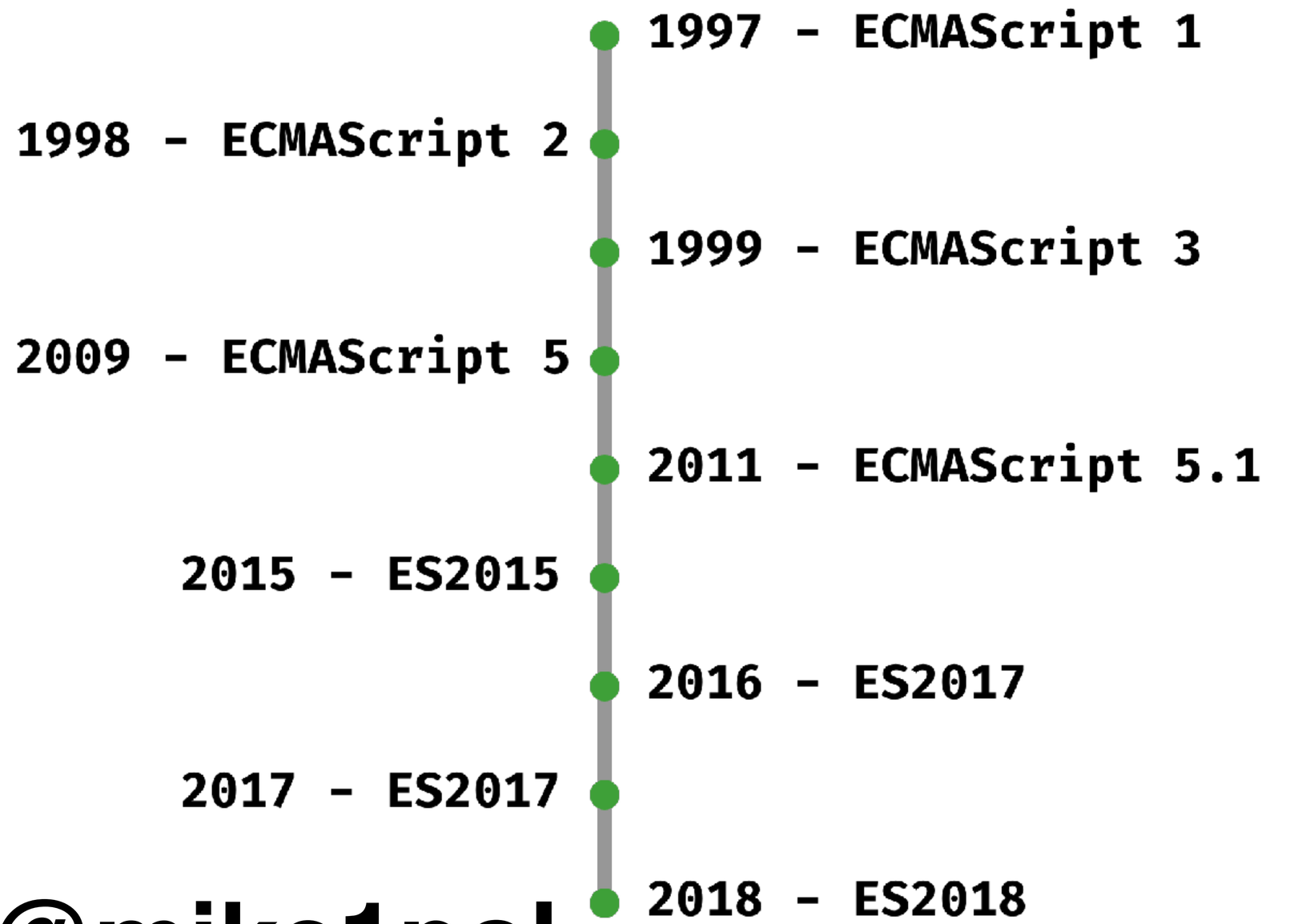
- Курс на стабилизацию
- Лучшие практики из других языков
- Стабильные годичные релизы

EcmaScript 4



Вопросы?

@mike1pol



EcmaScript 4