はじめに

Ecma International で定められている ES2020 について2020年6月にリリースされたので改めてまとめたいと思います。
仕様についてはこちらに記載されています。

Optional Chaining

Optional Chaining は、 `?` を使って、 `null` や `undefined` の要素にも安全にアクセスできる仕組みです。

```
const obj = {
  a: 1,
  b: {
    ba: 21
    bb: 22
  }
}
console.log(obj.b?.ba)
// => 21
console.log(obj.b?.bc)
// => undefined
console.log(obj.c?.ca)
// => undefined
console.log(obj.d?.da?.daa?.daaa)
// => undefined
```

このように要素がない場合は、エラーになることなく、 `undefined` が返されます。

Nullish coalescing Operator

Nullish coalescing Operator は、 `??` を使って、変数が `null` の場合の値を指定することができます。

```
const getValue = (val) => val || 'default'
const getValue2 = (val) => val ?? 'default'

console.log(getValue(''))
// => 'default'
console.log(getValue(0))
// => 'default'

console.log(getValue2(''))
// => ''
console.log(getValue2(0))
// => 0

console.log(getValue2('hoge'))
// => 'hoge'
console.log(getValue2(null))
// => 'default'
```

このように、 `val` が `false` 判定される場合には、 `'default'` が返されます。

Dynamic import

Dynamic import は、

module.js

```
export const hoge = "hoge!!"
```

```
import("./module.js").then(module => {
  console.log(module.hoge)
 // => hoge!!
})

setTimeout(async () => {
  const { hoge } = await import("./module.js")
```

```
  console.log(hoge)
  // => hoge!!
}, 1000)
```

このように Dynamic import では、 `Promise` の形でモジュールを読み込むことができます。
なので、使いたい時だけモジュール非同期で `import` して使用するということも可能です。

Promise.allSettled

Promise.allSettled は、複数の `Promise` を扱うことができます。 `Promise.all` と違い、複数のうちどれか1つが `reject` されても他の `Promise` は問題なく実行されます。

```
const promiseList = [
  Promise.resolve("ok"),
  Promise.resolve("ok"),
  Promise.reject("ng"),
  Promise.resolve("ok")
]

Promise.all(promiseList).then(
  resolve => console.log(`resolve: ${resolve}`),
  reject => console.log(`reject: ${reject}`)
)
// => reject: ng

Promise.allSettled(promiseList).then(
  resolveList => {
    console.log("resolve")
    for (const resolve of resolveList) {
      console.log(resolve)
    }
  },
  reject => {
    console.log("reject")
    console.log(reject)
  }
)
// => resolve
// => { status: 'fulfilled', value: 'ok' }
```

```
// => { status: 'fulfilled', value: 'ok' }
// => { status: 'rejected', reason: 'ng' }
// => { status: 'fulfilled', value: 'ok' }
```

String#matchAll

String#matchAll は、対象文字列について、正規表現で一致したものをイテレータで返します。

```
const text = "Test String";
const regex = /t/g;
for (const match of text.matchAll(regex)) {
  console.log(match)
}
// => [ 't', index: 3, input: 'Test String', groups: undefined ]
// => [ 't', index: 6, input: 'Test String', groups: undefined ]
```

このように、 `regex` にマッチしたものをイテレータで回すことができるので便利です。

globalThis

globalThis は、ウェブブラウザでもNode.jsもグローバルオブジェクトを参照できるオブジェクトです。

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title></title>
  </head>
  <body>
    <script>
      console.log(globalThis) // Window が出力されます
    </script>
```

```
  </body>
</html>
```

```
console.log(globalThis); // Global が出力されます
```

このようにウェブブラウザ・Node.jsで共通のグローバルオブジェクトを参照できます。

BigInt

BigInt は、 Number より大きな整数 2^53 以上の整数を扱えるオブジェクトです。

number を使うと

```
console.log(Number.MAX_SAFE_INTEGER)
// => 9007199254740991
console.log(Number.MAX_SAFE_INTEGER + 1)
// => 9007199254740992
console.log(Number.MAX_SAFE_INTEGER + 2)
// => 9007199254740992
//    （9007199254740993ではない）
```

このように誤差が生じてしまいます。

BigInt を使うことでこのような値も正しく扱うことができます。 BigInt は、数値に n を追加することで使用することができます。

```
console.log(BigInt(Number.MAX_SAFE_INTEGER) + 2n)
// => 9007199254740993
```

Well defined for-in order

従来の ECMAScript の使用では、 `for-in` の順序は保証されていませんでしたが、順序が固定されるようになりました。

```
const data = { name: "hoge", value: 100, text: "hoge" }

for (const key in data) {
  console.log(`${key}: ${data[key]}`)
}
// => name: hoge
// => value: 100
// => text: hoge
```

Module Namespace Exports

Module Namespace Exports は、 `import` してきたものをそのまま `export` することができます。

```
import * as utils from './utils.js'
export { utils }
```

これと同じことが以下のように書くことができます。

```
export * as utils from './utils.js'
```

import.meta

import.meta を使うことで、 import したモジュールのメタ情報にアクセスできます。

```
<script type="module" src="module.js"></script>
```

```
console.log(import.meta)
// => { url: "file:///home/user/module.js" }
```

さいごに

いかがだったでしょうか？
いろんな便利な機能が追加されたように思います。
それぞれ駆使して、より良いソースコードをかけるようになりましょう！

参考

https://ics.media/entry/200128/
https://shisama.hatenablog.com/entry/2019/12/01/080000#ES2020
https://www.freecodecamp.org/news/javascript-new-features-es2020/