ES12 (ECMAScript 2021) new features

2021 (ES12)

JS

1. Logical assignment operators

`&&=` operator

```
// "And And Equals"
a &&= b;

a && (a = b);
```

eqivalent expression:

```
if (a) {
    a = b;
}
```

`||=` operator

```
// "Or Or Equals"
a ||= b;

a || (a = b);
```

eqivalent expression:

```
if (!a) {
    a = b;
}
```

`??=` operator

```
// "QQ Equals"
a ??= b;

a ?? (a = b);
```

eqivalent expression:

```
a !== null && a !== void 0 ? a : (a = b);

// see transpiled js
```

Falsy values

- `false`
- `0`
- `null`
- `undefined`
- `""`
- `''`
- `NaN`

```
let username;

const foo = (name) => name ||= "Anonimous User";

foo(username);

console.log(username);


username = foo(username);

console.log(username);
```

Playground Link

2. String.prototype.replaceAll

```javascript
let sentence = "the quick brown fox jumps over the fence of the yard"


console.log(sentence.replace('the', 'a'));

console.log(sentence.replaceAll('the', 'a'));

console.log(sentence);
```

3. Numeric separators

```
const z = 1000000000;
console.log(z);

const z2 = 10_000_000_000;
console.log(z);

const z3 = 0.000_001; // 1 millionth
console.log(z3);
```

4. Promise.any + AggregateError

Promise combinators

- `Promise.all`
- `Promise.race`
- `Promise.AllSettled` (ES2020)
- `Promise.any` (ES2021)

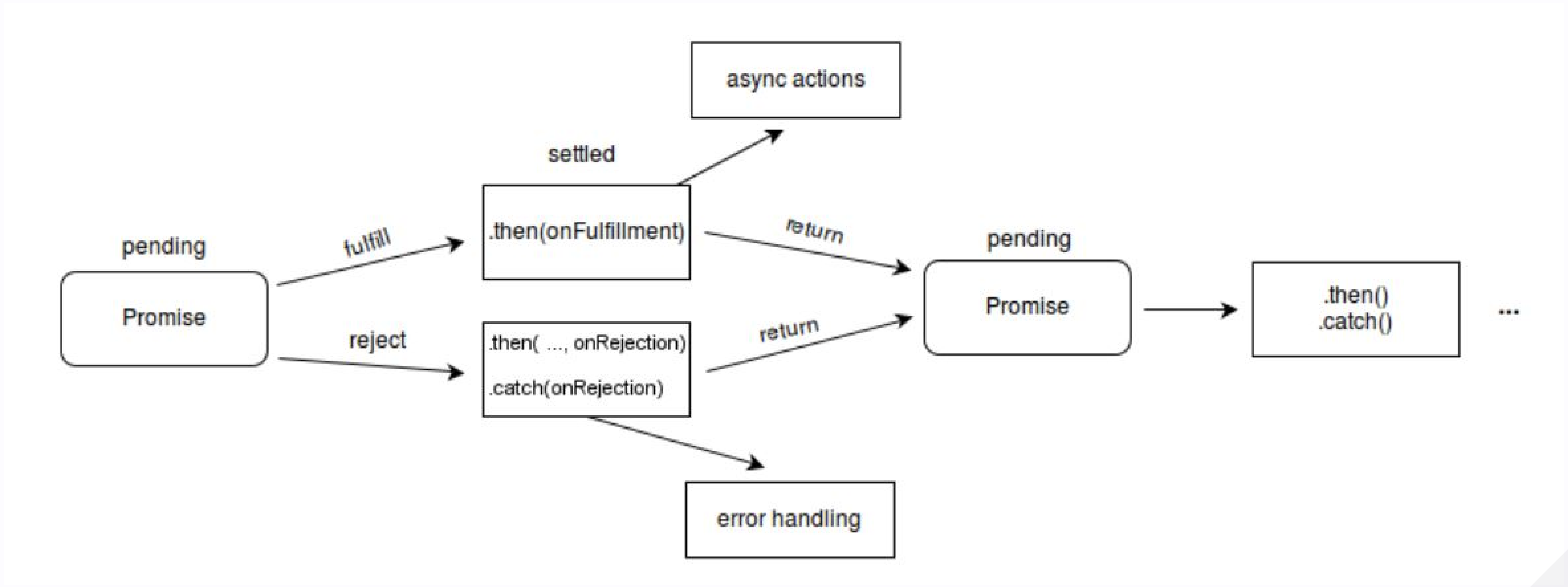
What is a promise?

“ The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value. ”

source: MDN

```
{
    state: pending | fulfilled | rejected,
    value: undefined | value | error object
}
```

Promise flow



Example

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(response => response.json())
  .then(json => console.log(json))
```

Promise.all

```
const p1 = Promise.resolve(42);
const p2 = Promise.reject(new Error("rejection!"));
const p3 = new Promise(resolve => setTimeout(resolve, 10000, "foo"));

const promises = [p1, p2, p3];

Promise.all(promises)
    .then(() => console.log("all resolved! (succeeded)"))
    .catch(error =>console.log(error));

// rejects as soon as one of the promises rejects
// (potentially not waiting for others to settle);
```

Promise.allSettled

```
const p1 = Promise.resolve(42);
const p2 = Promise.reject(new Error("rejection!"));
const p3 = new Promise(resolve => setTimeout(resolve, 10000, "foo"));

const promises = [p1, p2, p3];



Promise.allSettled(promises)
    .then(() => console.log("all promises have settled"))
    .catch(error => console.log(error)); // will never get here

// had to wait 10 seconds, even though one rejected immediately
```

Promise.race

```
const p5 = new Promise(resolve => setTimeout(resolve, 1000, "one"));
const p6 = new Promise(reject => setTimeout(reject, 2000, new Error("p6 was re
const p7 = new Promise(reject => setTimeout(reject, 500, new Error("p7 was rej


Promise.race([p5, p6])
    .then(val => console.log(val))
    .catch(error => console.log(error));


Promise.race([p5, p7])
    .then(val => console.log(val))
    .catch(error => console.log(error));
```

Promise.any

```
// Promise any does not behave properly yet:

Promise.any([p5, p6, p7])
    .then(val => console.log(val))
    .catch(error => console.log(error.errors)); // expects: "one"


Promise.any([p6, p7])
.then(val => console.log(val))
.catch(error => console.log(error.errors)); // expects: aggregate error
```

AggregateError

```
try {
  throw new AggregateError([
    new Error("some error"),
  ], 'Hello');
} catch (e) {
  console.log(e instanceof AggregateError); // true
  console.log(e.message);                   // "Hello"
  console.log(e.name);                      // "AggregateError"
  console.log(e.errors);                    // [ Error: "some error" ]
}
```

Promise.any throws AggregateError

```
Promise.any([
  Promise.reject(new Error("some error")),
]).catch(e => {
  console.log(e instanceof AggregateError); // true
  console.log(e.message);                   // "All Promises rejected"
  console.log(e.name);                      // "AggregateError"
  console.log(e.errors);                    // [ Error: "some error" ]
});
```

Promise Combinators - summary

name	description	added
Promise.allSettled	does not short-circuit	ES2020
Promise.all	short-circuits when an input value is rejected	ES2015
Promise.race	short-circuits when an input value is settled	ES2015
Promise.any	short-circuits when an input value is fulfilled	ES2021

5. Optional chaining (ES2020)

```
let person = {
    fullName: "Carl Johnson",
    address: {
        street: "101 Grove st.",
        city: ":Los Santos",
        zipcode: 123456
    }
}

let street = person.defaultAddress.street;
console.log(street);

let zip = person.defaultAddress.zipcode; // error
```

```javascript
zip = person.defaultAddress && person.defaultAddress.zipcode;
console.log(zip); // undefined

// vs:
zip = person.defaultAddress?.zipcode;
console.log(zip);

// if any of the properties is undefined,
// the variable will resolve to undefined
```

Appendix - Event Loop

```
(function() {

  console.log('this is the start');

  setTimeout(function cb1() {
    console.log('Callback 1: this is a msg from call back');
  }); // has a default time value of 0

  console.log('this is just a message');

  setTimeout(function cb2() {
    console.log('Callback 2: this is a msg from call back');
  }, 0);

  console.log('this is the end');

})();
```

Thank You

Links

- https://github.com/tc39/proposals/blob/master/finished-proposals.md
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Prom
retiredLocale=he
- https://v8.dev/features/promise-combinators
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop