Await operations

tc39 93rd meeting

The Promise way and the async way

Consume a promise:

```
// Promise
task.then(data => {
    data // ...
})

// async-await
const data = await task
```

The Promise way and the async way

Consume many promise at once:

```
// Promise
Promise.all([a, b]).then(([a, b]) => {
    a // ...
    b // ...
})

// async-await (today)
const [a, b] = await Promise.all([a, b])

// async-await (this proposal)
const [a, b] = await.all [a, b]
```

The Promise way and the async way

Ignoring error:

```
// Promise
task.catch(noop).finally(() => {
    rest_task // ...
})

// async: oh no
try {
    const value = await task
} catch {}
rest_task // ...

// async: mixed is better?
await task.catch(noop)
rest_task // ...
```

The Promise way and the async way

Create a new promise that can be written using async-await:

proposal mentioned here: https://github.com/tc39/proposal-do-expressions

```
// async-await: IIFE
(async () => {
    await prepare()
    return doSomething(await value)
})()

// Promise, but WRONG (rejections are not caught)
new Promise(async (resolve, reject) => {
    await prepare()
    resolve(doSomething(await value))
})

// Promise
prepare()
      .then(() => value)
      .then(doSomething)

// async-do expression proposal
async do {
    await prepare()
    doSomething(await value)
}
```

What has changed since stage 1?

Improved motivation statement:

Old:

~~Simplify the usage of Promise~~

New:

Developers have to know about Promise if they want to handle multiple tasks concurrently (Promise.all, eg), this is some kind of "abstraction leak" under the async-await mental model. Fix this problem by introducing concurrent Promise utils in the async-await style.

What has changed since stage 1?

Syntax:

No change

Semantics:

No change

Syntax

AwaitExpression[Yield] :

await *UnaryExpression*[?Yield, +Await]

await . **all** *UnaryExpression*[?Yield, +Await]

await . **allSettled** *UnaryExpression*[?Yield, +Await]

await . **any** *UnaryExpression*[?Yield, +Await]

await . **race** *UnaryExpression*[?Yield, +Await]

15.8.5 Runtime Semantics: Evaluation

AwaitExpression : **`await . all`** *UnaryExpression*

1. Let *exprRef* be ? Evaluation of *UnaryExpression*.
2. Let *value* be ? GetValue(*exprRef*).
3. Let *pendingPromise* be ? Call(%Promise.all%, %Promise%, [*value*]).
4. Return ? Await(*pendingPromise*).

AwaitExpression : **`await . allSettled`** *UnaryExpression*

1. Let *exprRef* be ? Evaluation of *UnaryExpression*.
2. Let *value* be ? GetValue(*exprRef*).
3. Let *pendingPromise* be ? Call(%Promise.allSettled%, %Promise%, [*value*]).
4. Return ? Await(*pendingPromise*).

AwaitExpression : **`await . any`** *UnaryExpression*

1. Let *exprRef* be ? Evaluation of *UnaryExpression*.
2. Let *value* be ? GetValue(*exprRef*).
3. Let *pendingPromise* be ? Call(%Promise.any%, %Promise%, [*value*]).
4. Return ? Await(*pendingPromise*).

AwaitExpression : **`await . race`** *UnaryExpression*

1. Let *exprRef* be ? Evaluation of *UnaryExpression*.
2. Let *value* be ? GetValue(*exprRef*).
3. Let *pendingPromise* be ? Call(%Promise.race%, %Promise%, [*value*]).
4. Return ? Await(*pendingPromise*).

Question?