Javascript
Promise, Async/Await

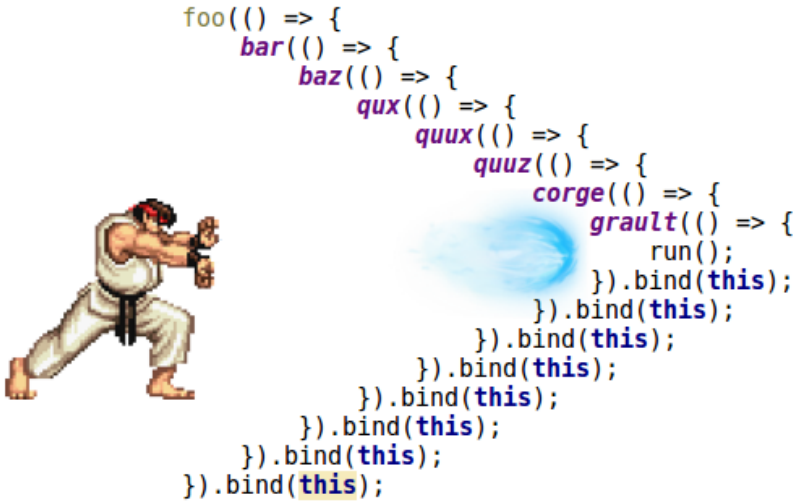
J075 박상신

목차

Javascript 비동기 동시성 프로그래밍

Javascript 에서 비동기 동시성프로그래밍을 하는 방법은 3가지

1. callback 사용(옛날부터 사용되어옴)
2. Promise (Promise method chain 을 통해 함수를 합성)
3. Async, Await 함께 사용하는 방법



```
foo(() => {
    bar(() => {
        baz(() => {
            qux(() => {
                quux(() => {
                    quuz(() => {
                        corge(() => {
                            grault(() => {
                                run();
                            }).bind(this);
                        }).bind(this);
                    }).bind(this);
                }).bind(this);
            }).bind(this);
        }).bind(this);
    }).bind(this);
}).bind(this);
```

Promise

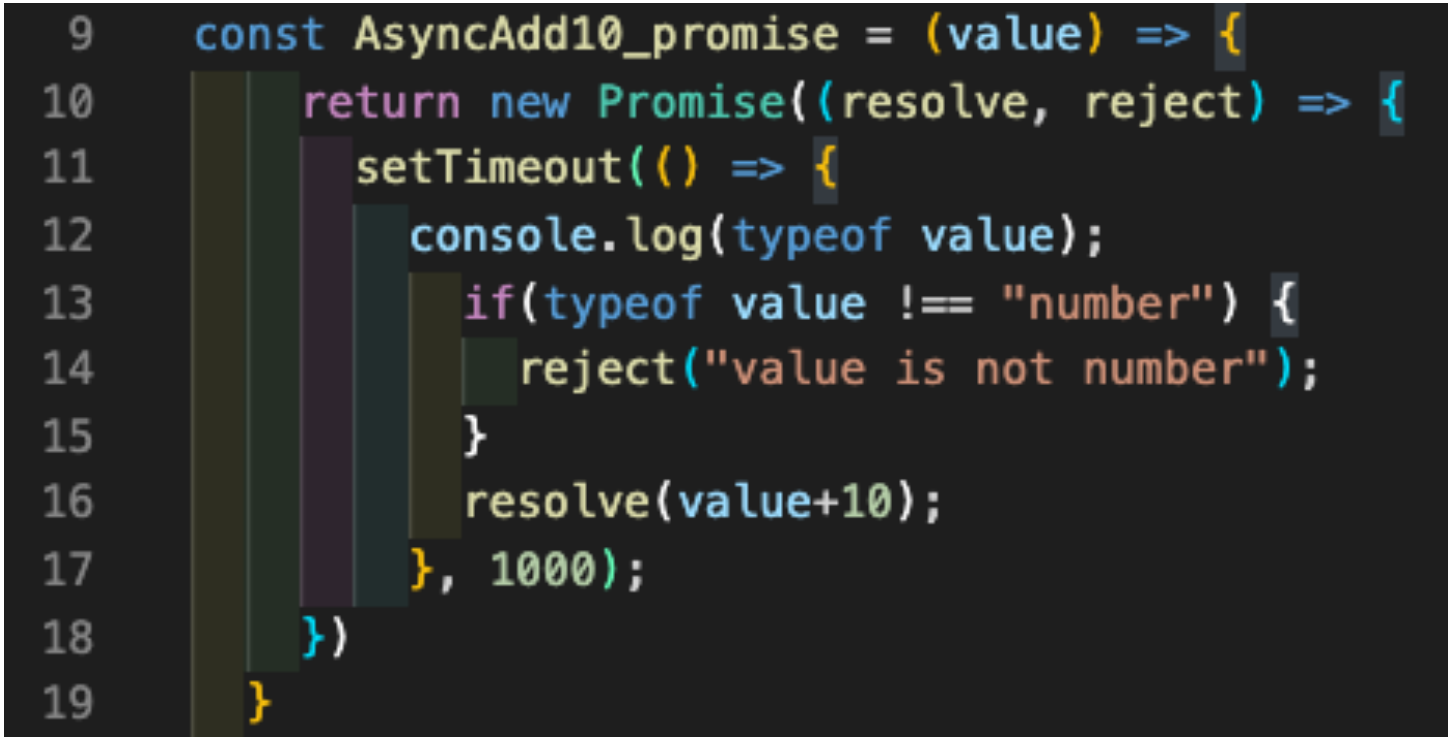
Callback hell을 해결하기 위한 새로운 도구

비동기 상황을 값으로 다룰 수 있게 해주는 도구

```
19  const main = () => {
20    AsyncAdd10(1000, res => {
21      AsyncAdd10(res, res => {
22        AsyncAdd10(res, res => {
23          console.log("Async Add 10 result : ", res);
24        })
25      })
26    });
27
28    AsyncAdd10_promise(1000)
29    .then(AsyncAdd10_promise)
30    .then(AsyncAdd10_promise)
31    .then(res => {console.log("Async Add 10 result : ", res)});
32  }
33
34  main();
```

Promise

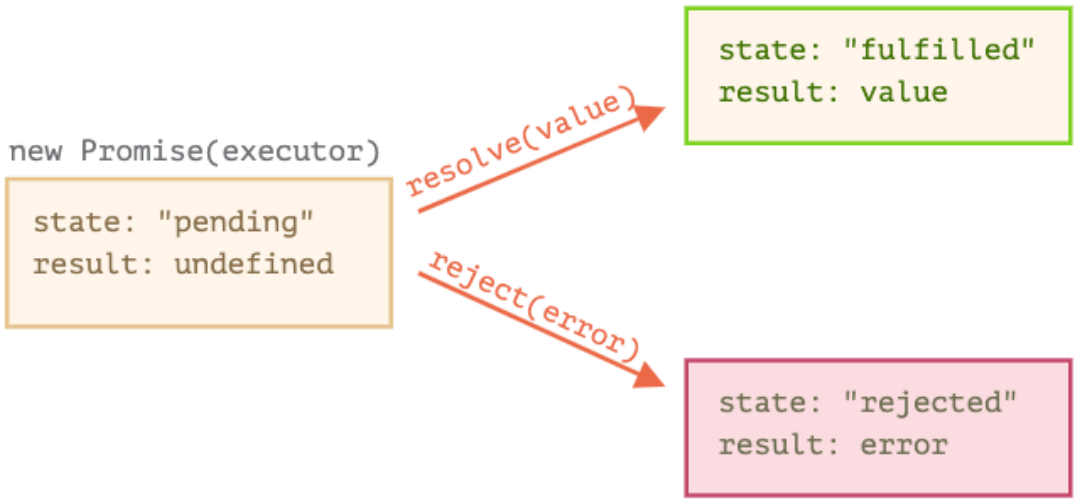
Promise를 사용한 비동기 함수 예



```
9   const AsyncAdd10_promise = (value) => {
10      return new Promise((resolve, reject) => {
11          setTimeout(() => {
12              console.log(typeof value);
13              if(typeof value !== "number") {
14                  reject("value is not number");
15              }
16              resolve(value+10);
17          }, 1000);
18      })
19  }
```

Promise

내부 동작 – (js promise polyfill)



Promise 객체 : state, result, 함수를 쌓아둘 무언가.

Promise

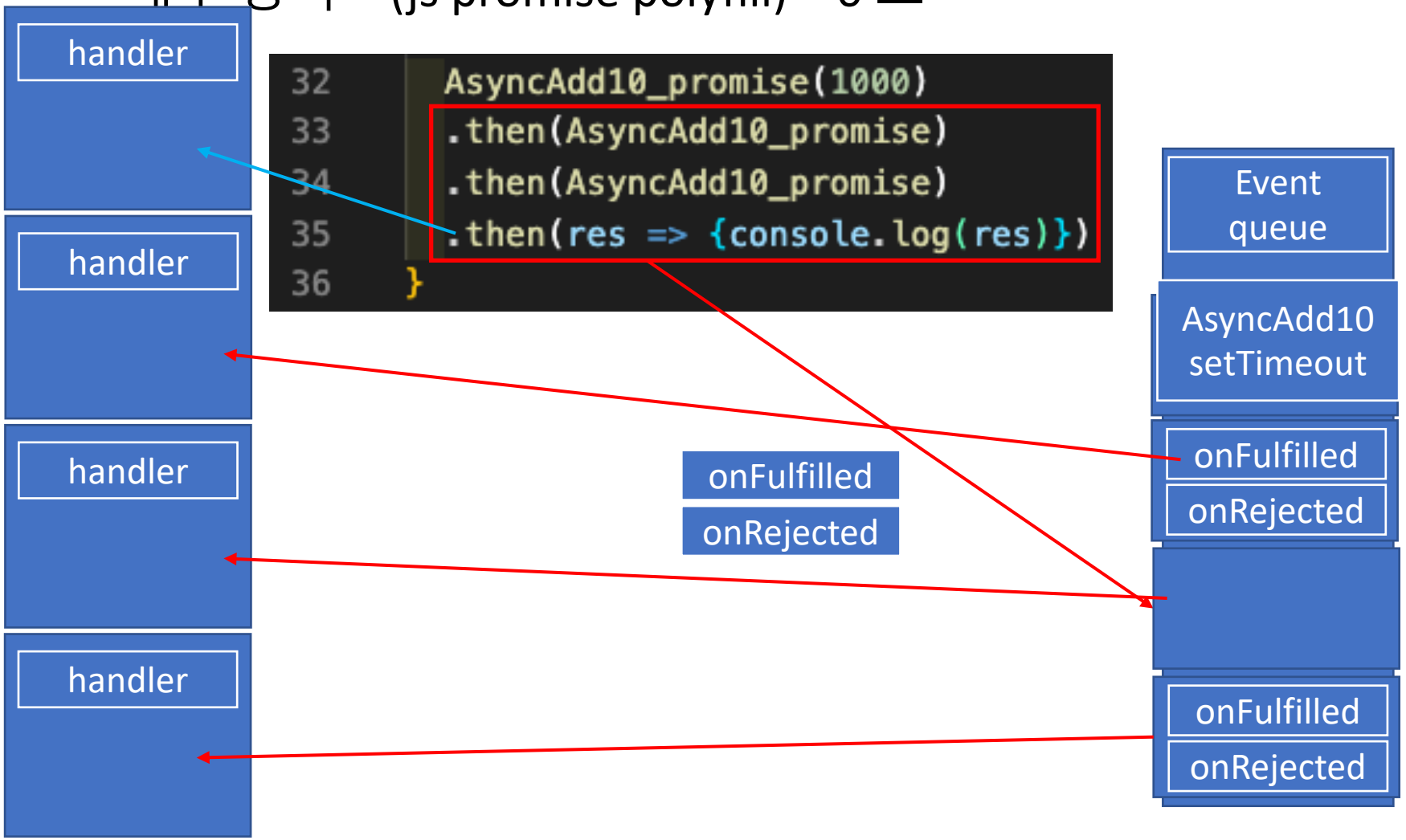
내부 동작 – (js promise polyfill)

```
39  function MyPromise(fn) {
40    // store state which can be PENDING, FULFILLED or REJECTED
41    var state = PENDING;
42
43    // store value once FULFILLED or REJECTED
44    var value = null;
45
46    // store sucess & failure handlers
47    var handlers = [];
```

Promise 객체 : state, result, 함수를 쌓아둘 무언가.

Promise

내부 동작 – (js promise polyfill) – 0 초



Promise

내부 동작 – (js promise polyfill) – 0 초


```
32    AsyncAdd10_promise(1000)
33    .then(AsyncAdd10_promise)
34    .then(AsyncAdd10_promise)
35    .then(res => {console.log(res)})
36  }
```

```
115   this.then = function (onFulfilled, onRejected) {
116     var self = this;
117     return new MyPromise(function (resolve, reject) {
118       return self.done(function (value) {
```

```
10  const AsyncAdd10_promise = (value) => {
11    return new MyPromise((resolve, reject) => {
12      console.log("promise layer");
13      setTimeout(() => {
14        console.log("async : 1 min");
15        if(typeof value !== "number") {
16          reject("value is not number");
17        }
18        resolve(value+10);
19      }, 100);
```

Promise

내부 동작 – (js promise polyfill) – 이후 동작


```
10  const AsyncAdd10_promise = (value) => {
11      return new MyPromise((resolve, reject) => {
12          console.log("promise layer");
13          setTimeout(() => {
14              console.log("async : 1 min");
15              if(typeof value !== "number") {
16                  reject("value is not number");
17              }
18              resolve(value+10);
19          }, 100);
```

Promise

Callback hell을 해결하기 위한 새로운 도구

비동기 상황을 값으로 다룰 수 있게 해주는 도구

```
31    const result = AsyncAdd10_promise(1000)
32    .then(AsyncAdd10_promise)
33    .then(AsyncAdd10_promise)
34    .then(res => {console.log(res)})
35    console.log("result : ", result);
36  }
```

```
result :  Promise { <pending> }
```

Async로 감싸고 Await 을 붙이면 값 확인할 수 있는 것 모두가 알고 있다.

Generator

```javascript
// 제너레이터 생성 : 일반함수 앞에 *을 붙여서
function *gen() {
    yield 1;
    yield 2;
    yield 3;
    return 100;
}
let iter = gen();
console.log(iter.next());
console.log(iter.next());
console.log(iter.next());
console.log(iter.next());

for(const a of gen()) log(a);
```

JavaScript

```
▶ {value: 1, done: false}
▶ {value: 2, done: false}
▶ {value: 3, done: false}
▶ {value: 100, done: true}
1
2
3
```

원하는 타이밍에 원하는 만큼 꺼낼 수 있다.

Async / Await

```
11  const asyncFunc = async () => {
12    await A();
13    await B();
14    await C();
15  }
16
17  asyncFunc();
```

```
11  function* asyncFunc () {
12    yield A();
13    yield B();
14    yield C();
15  }
```

Async / Await

```
return new Promise(function (resolve, reject) {
  function step(key, arg) {
    try {
      var info = gen[key](arg);
      //console.log(info)
      var value = info.value;
    }
    catch (error) {
      reject(error); return;
    }
    if (info.done) {
      resolve(value);
    }
    else {
      return Promise.resolve(value)
        .then(
          function (value) { step("next", value); },
          function (err) { step("throw", err); }
        );
    }
  }
  return step("next");
});
```

출처

https://medium.com/sjk5766/async-await-%EC%9B%90%EB%A6%AC-cc643f18526d

https://meetup.toast.com/posts/73

https://medium.com/chequer/js-promise-%ED%8C%A8%ED%84%B4-%EA%B5%AC%ED%98%84%ED%95%B4%EB%B3%B4%EA%B8%B0-a79cba99c9a5

https://ko.javascript.info/promise-basics