Async programming APIs

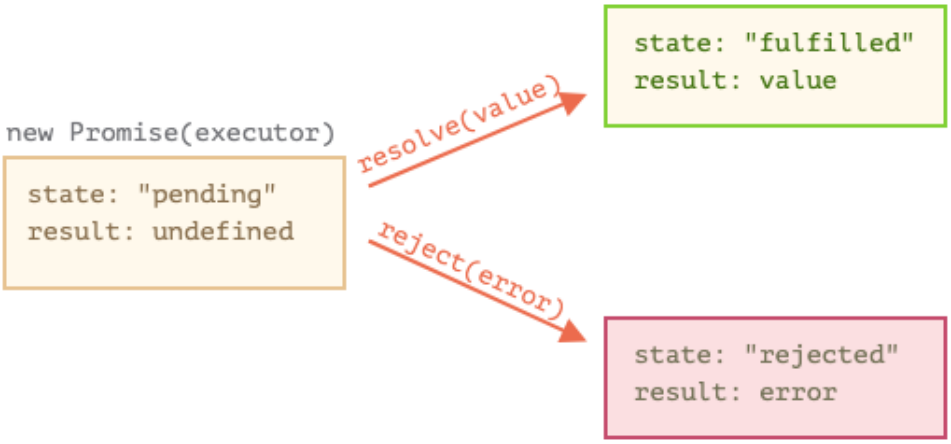
1. Promise Api
2. Reactive Programming Library - RxJS

Promise

> 📌 A promise is a special JavaScript object that links the “producing code” and the “consuming code” together

The constructor syntax for a promise object is:

```
let promise = new Promise(function(resolve, reject) {
  // executor (the producing code)
});
```



Promises, async/await

We want to make this open-source project available for people all around the world.

https://javascript.info/async

JAVASCRIPT.INFO
The Modern JavaScript Tutorial

---

async & await (from es6)

Reactive programming

```
const Rx = require('rxjs')
const Subject = Rx.Subject;


// reactive programming ==> using reactive extensions ==> using rxjs

//----------------------------------------
// producer module
//----------------------------------------

const stream = new Subject()  // observable stream

const subjects = [
    "javascript",
    "UI",
    "react.js",
    "react native",
    "database",
    "Node.js + Express.js",
    "devops & aws cloud",
    "microservices & servlerless appln"
]

let trainer = {
    getSubjects() {
        let i = 0
        let interval = setInterval(() => {
            let subject = subjects[i]
            i++
            if (i !== subjects.length) {
                stream.next(subject)
            } else {
                stream.complete()
                clearInterval(interval)
            }
            if (Math.random() < 0.2) {
                stream.error("oops")
                clearInterval(interval)
            }

        }, 2000)
        return stream;
    }
}

//----------------------------------------
// consumer module
//----------------------------------------

let student = {
    learn() {
        let stream = trainer.getSubjects()
        stream
            .subscribe(
```

```
                result => { console.log("student learning : " + result) },
                error => { console.log("student : " + error) },
                () => console.log("thanks for all subjects")
            )
    }
}

student.learn();
```

Reactive Programming with RxJS

```
const Rx = require('rxjs')
const { Subject } = Rx


const doorStream = new Subject()

//-----------------------------------
// Light
//-----------------------------------

doorStream.subscribe(event => {
    if (event.type === "open")
        light.on()
    if (event.type === "close")
        light.off()
})

const light = {
    on() {
        console.log("light on")
    },
    off() {
        console.log("light off")
    }
}


//-----------------------------------
// Ac
//-----------------------------------


doorStream.subscribe(event => {
    if (event.type === "open")
        ac.on()
    if (event.type === "close")
        ac.off()
})
```

```
const ac = {
    on() {
        console.log("ac on")
    },
    off() {
        console.log("ac off")
    }
}


//-----------------------------------
// Fan
//-----------------------------------


doorStream.subscribe(event => {
    if (event.type === "open")
        fan.on()
    if (event.type === "close")
        fan.off()
})

const fan = {
    on() {
        console.log("fan on")
    },
    off() {
        console.log("fan off")
    }
}


//-----------------------------------
// door
//-----------------------------------

class Door {
    // constructor() {
    //     this.listeners = []
    // }
    // addDoorListener(listener) {
    //     this.listeners.push(listener)
    // }
    // removeDoorListener(listener) {
    //     //..
    // }
    open() {
        // this.listeners.forEach(listener => {
        //     listener.on()
        // })
        doorStream.next({ type: 'open' })
        console.log("door opened")
    }
    close() {
        // this.listeners.forEach(listener => {
        //     listener.off()
        // })
```

```
        doorStream.next({ type: 'close' })
        console.log("door closed")
    }
}

const door = new Door()

// door.addDoorListener(light)
// door.addDoorListener(ac)
// door.addDoorListener(fan)


setTimeout(() => {
    door.open()
    setTimeout(() => {
        door.close()
    }, 2000)
}, 2000)
```

```
const Rx = require('rxjs')
const { share } = require('rxjs/operators')

const observable = Rx.Observable.create((observer) => {
    let i = 0
    let interval = setInterval(() => {
        console.log("emitting a number")
        observer.next(Math.random()); // emitting
        i++
        if (i === 10) {
            clearInterval(interval)
        }
    }, 2000)
}).pipe(share())


//-----------------------------
// module-1
//-----------------------------

// // subscription 1
const unsubscribe=observable.subscribe((data) => {
    console.log("subscriber-1 :" + data);
});
```

```
//-----------------------------
// module-2
//-----------------------------


setTimeout(() => {
    // // subscription 2
    observable.subscribe((data) => {
        console.log("subscriber-2 :" + data);
    });
}, 5000)
```

---

promise vs reactive programming api

- a *Promise* is eager, whereas an *Observable* is lazy,
- a *Promise* can provide a single value, whereas an *Observable* is a stream of values (from 0 to multiple values),

---