JSPromise

JS는 거대한 thread개념이어서 사용이 가능

선언

```
setTimeout(function() { // Code here }, delay);
```

위 함수는 보통 두 개의 인자를 설정하여 사용합니다.

- i. 호출될 콜백함수
- ii. 지연시간(delay time)

Basic function style

```
const promise = new Promise (function (resolve, reject) {
    console.log("do something...")
    setTimeout(function () {
        console.log("주어진 시간 만료!")
        resolve();
    }, 2000)
})

promise.then(function () {
    console.log("전송완료!")
})
```

arrow function style

```
    const promise =new Promise((resolve,reject)=>{
        console.log("do something...")
        setTimeout(()=>{
            console.log("주어진 시간 만료");
        resolve("전송완료");
    }, 2000);
});

promise.then((result) => {
    console.log("응답: "+result)
})
```

설명:new promise를 새롭게 객체화 했으며, 익명 함수를 만듦

**주의점: 외부에서 호출을 할 때 promise를 맨 위로 올려버리면 자동으로 실행 시켜버리기에 맨 위로 올리면 안 된다.

promise객체가 만들어 지는 시점이 처음이면 안 됨

Resolve,Rejiect,Finally

resolve는 .then이 받는다

reject는 .catch가 받는다

```
promise.then((response)=>console.log("응답: " + response))
.catch((error)=> console.log("에러: " + error))
```

와 같이 한 코드에 정리해서 쓸 수 있다.

.finally(()=>{})는 무조건 실행이됨 .then.catch의 젤 후미에 접속

```
function minus (num1, num2) {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            res = num1 - num2
            console.log(num1 + " - " + num2 + " = ")
            resolve(res)
        }, 1500)
    })
}
```

```
function plus (num1, num2) {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            res = num1 + num2
            console.log(num1 + " + " + num2 + " = ")
            resolve(res)
        }, 2000)
    })
}
```

```
function divide (num1, num2) {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            res = num1 / num2
            console.log(num1 + " / " + num2 + " = ")
            resolve(res)
        }, 1000)
    })
}
```

```
function multiply (num1, num2) {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            res = num1 * num2
            console.log(num1 + " * " + num2 + " = ")
            resolve(res)
        }, 500)
    })
}
```

.then().then()

두 번 붙여서 써 주게 되면 두 번째 꺼는 undefince값이 나오게 됨

```
const promise = new Promise(function (resolve, reject) {
    setTimeout(function () {
        console.log("가즈아!!!")
        resolve("우디를 ?")
    }, 2000)
})

promise.then(
    (response) => console.log("응답: " + response)
).then(
    (response) => console.log("응답: " + response)
)
```

.then().then() 적용되게 하기

첫 번째 .then()내부에서 새롭게 promise를 객체화를 추가로 시켜준다

```
promise.then(function (response) {
    console.log("응답: " + response)

    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            console.log("레츠고")
            resolve("간드아!!!!!!!")
        }, 1000)
    })
}).then(
    (response) => console.log("응답: " + response)
)
```

parallel병렬

.then()을 활용하면 여러 가지의 작업을 한 작업으로 돌릴 수 있다.

```
plus(
    100, 200
).then(function (res) {
    console.log(res)
    return minus(100, 200)
}).then(function (res) {
    console.log(res)
    return multiply(100, 200)
}).then(function (res) {
    console.log(res)
    return divide(100, 200)
}).then(function (res) {
    console.log(res)
})
```

.all(): 동시에 구동 시킴

수식들 간에 종속된 관계가 없을 경우 아래와 같이 동시에 돌릴 수 있다.

```
Promise.all([
    plus(100, 200),
    minus(100, 200),
    multiply(100, 200),
    divide(100, 200)
]).then(function (res) {
    console.log(res)
})
```

.race(): 가장 빠른 녀석의 값만 읽어냄

주식이나 코인등 시간싸움이 요구 되는 상황에서 여러 알고리즘을 두고 주로 활용

↑ ↑ ↑ ↑ ↑ ↑

는 실시간적으로 빠른 구동이 요구될 때의 예제들

빠른 놈을 채택하는 케이스

```
Promise.race([
    plus(100, 200),
    minus(100, 200),
    multiply(100, 200),
    divide(100, 200)
]).then(function (res) {
    console.log(res)
})
```

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

는 의도적으로 응답을 기다리게 해야 하는 예제들

TeamProject때 활용可

```
function minus (num1, num2) {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            res = num1 - num2
            console.log(num1 + " - " + num2 + " = ")
            resolve(res)
        }, 1500)
    })
}
```

```
function plus (num1, num2) {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            res = num1 + num2
            console.log(num1 + " + " + num2 + " = ")
            resolve(res)
        }, 2000)
    })
}
```

```
function divide (num1, num2) {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            res = num1 / num2
            console.log(num1 + " / " + num2 + " = ")
            resolve(res)
        }, 1000)
    })
}
```

```
function multiply (num1, num2) {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            res = num1 * num2
            console.log(num1 + " * " + num2 + " = ")
            resolve(res)
        }, 500)
    })
}
```

async function 함수명+await: java의 start(),join()과 같은 역할 다른 항목들의 값이 나올 때 까지 기다려줌

(https://developer.mozilla.org/ko/docs/Learn/JavaScript/Asynchronous/Async_await)

각각 입력해둔 대기 시간이 끝난 후 입력된 순서대로 출력

```
async function asyncProcess () {
    const res1 = await plus(100, 200)
    console.log(res1)
    const res2 = await minus(100, 200)
    console.log(res2)
    const res3 = await multiply(100, 200)
    console.log(res3)
    const res4 = await divide(100, 200)
    console.log(res4)
}
```

```
100 + 200 =
300
100 - 200 =
-100
100 * 200 =
20000
100 / 200 =
0.5
```

async function 함수명+await(改)

```
async function asyncProcess () {
    const p1 = plus(100, 200)
    const p2 = minus(100, 200)
    const p3 = multiply(100, 200)
    const p4 = divide(100, 200)

    const res1 = await p1
    const res2 = await p2
    const res3 = await p3
    const res4 = await p4

    console.log(res1)
    console.log(res2)
    console.log(res3)
    console.log(res4)
}
```

```
JavaScript Promise Pa
자바 스크립트는 스레드
100 * 200 =
100 / 200 =
100 - 200 =
100 + 200 =
300
-100
20000
0.5
>
```

값을 받아놨다가 모든 연산이 끝나면 한번에 출력

async function 함수명+await(改) promise.all()과 연계

```
async function asyncProcess () {
    const [res1, res2, res3, res4] = await Promise.all([
        plus(100, 200),
        minus(100, 200),
        multiply(100, 200),
        divide(100, 200)
    ])

    console.log(res1)
    console.log(res2)
    console.log(res3)
    console.log(res4)
}
```