디지털컨버전스 기반 UXUI Front 전문 개발자 양성과정

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - JungHyun LEE(이정현)
akdl911215@naver.com

promiseTest7

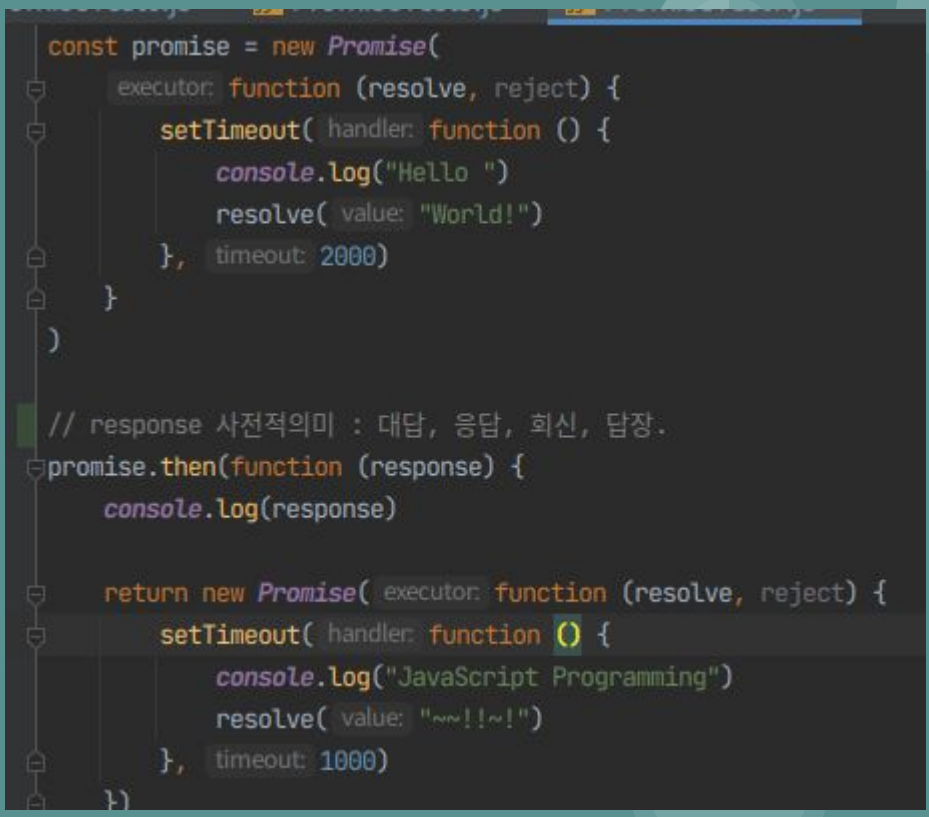
```
// response 사전적의미 : 대답, 응답, 회신, 답장.
promise.then(function (response) {
    console.log(response)

    return new Promise( executor: function (resolve, reject) {
        setTimeout( handler: function () {
            console.log("JavaScript Programming")
            resolve( value: "~~!!~!")
        }, timeout: 1000)
    })
}).then(response =>
    console.log("Success")
).catch(function (error) {
    console.log(error)
}).finally( onFinally: () => {
    console.log("PromiseTest7: 나는 무조건 실행된다!!!")
})
```

```
// 출력 :
// PromiseTest7 Satrt
// PromiseTest7 Fin
// Hello
// World!
// JavaScript Programing
// Success
// PromiseTest7: 나는 무조건 실행된다!!!
```

출력이 **World**부분까지 이해가 됐습니다. 하지만 **JavaScropt Programing** 출력은 **return** 으로 인해서 무조건 출력이고, **resove(“~~!!~!”)** 은 **reject**가 되서 출력이 안된건가요? **resject** 가 되서 출력이 안된거면 이유를 알 수 있을까요?

promiseTest7



```
const promise = new Promise(
    executor: function (resolve, reject) {
        setTimeout( handler: function () {
            console.log("Hello ")
            resolve( value: "World!")
        }, timeout: 2000)
    }
)

// response 사전적의미 : 대답, 응답, 회신, 답장.
promise.then(function (response) {
    console.log(response)

    return new Promise( executor: function (resolve, reject) {
        setTimeout( handler: function () {
            console.log("JavaScript Programming")
            resolve( value: "~~!!~!")
        }, timeout: 1000)
    })
```

const promise = new promise 는 timeout이 2초

return new Promise는 timeout 1초 인데 const promise = new promise 먼저 출력되는 이유가 return new Promise는 리턴값이기 때문에 시간초는 짧아도 후에 출력되는게 맞나요?

비동기 처리

자바스크립트의 비동기 처리란 코드의 연산이 끝날때까지 코드의 실행을 멈추지 않고 다음 코드를 먼저 실행하는 자바스크립트의 특성을 의미합니다.

비유로 이해하는 비동기적 방식

카페에서 주문을 하게되면 주문을 받고 제조되는 순서대로 커피를 받게 됩니다. 은행 업무와 달리 먼저 주문한 사람의 커피가 다 제조될 때까지 다음 사람이 기다릴 필요가 없기 때문에 카페에 '들어온 순서'보다 먼저 '제조된 순서'가 중요하게 됩니다.

이처럼 연속적으로 발생하는 이벤트를 담은 후 완료되는 순서대로 일을 처리하는 실행 순서가 아닌 확실하지 않은 방식을 비동기적 방식이라고 합니다.

콜백함수

제어권을 대상에게 넘겨줍니다. 함수 a(getData)의 매개변수로 콜백함수 b(callback)를 전달하면, a(getData)가 b(callback)의 제어권을 갖게 됩니다.
특별한 요청(bind)가 없는 한 a(getData)에 미리 정해진 방식에 따라 b(callback)를 호출한다.

비유로 이해하는 콜백 함수 동작 방식

콜백 함수의 동작 방식은 일종의 식당자리 예약과 같습니다. 일반적으로 맛집을 가면 사람이 많아 자리가 없습니다. 그래서 대기자 명단에 이름을 쓴 다음에 자리가 날 때까지 주변 식당을 돌아다니죠. 만약 식당에서 자리가 생기면 전화로 자리가 났다고 연락이 옵니다. 그 전화를 받는 시점이 여기서의 콜백함수가 호출되는 시점과 같습니다. 손님 입장에서는 자리가 날때까지 식당에서 기다리지 않고 근처 가게에서 잠깐 쇼핑을 할 수도 있고 아니면 다른 식당 자리를 알아볼수도 있습니다.

자리가 났을때만 연락이 오기때문에 미리 가서 기다릴 필요도 없고, 직접 식당안에 들어가서 자리가 비어있는지 확인할 필요도 없습니다. 자리가 준비된 시점, 즉 데이터가 준비된 시점에서만 저희가 원하는 동작(자리에 앉는다, 특정 값을 출력한다 등)ㅇㅡㄹ 수행할 수 있습니다.

Async/Await

자바스크립트 비동기처리패턴의 최신문법입니다.

Promise와 **callback**에서 주는 단점들을 해결

(에러처리, 비동기적 사고 방식에서 벗어나

코드의 절차적 작성)을 도와줍니다. 좀 더

동기적인 코드처럼 작성할 수 있게 만들어줍니다.

Promise를 사용하지만 **thn, catch** 메소드를

사용하여 컨트롤 하는 것이 아닌 동기적 코드처럼

반환 값을 변수에 할당하여 작성할 수 있게끔

도와주는 문법입니다.

```
async function asyncProcess() {
    // await는 async 함수 안에서만 동작합니다.
    // 자바스크립트는 await 키워드를 만나면 프라미스가 처리(settled)될
    // 때까지 기다립니다. 결과는 그 이후 반환됩니다.

    // await('기다리다'라는 뜻)는 말 그대로 프라미스가 처리될 때까지 함수
    // 실행을 기다리게 만듭니다. 프라미스가 처리되면 그 결과와 함께 실행이
    // 재개됩니다. 프라미스가 처리되길 기다리는 동안엔 엔진이 다른 일
    // (다른 스크립트를 실행, 이벤트 처리 등)을 할 수 있기 때문에,
    // CPU 리소스가 낭비되지 않습니다.

    // await 는 promise.then 보다 좀 더 세련된게 프라미스의 result 값을
    // 얻을 수 있도록 해주는 문법입니다. promise.then 보다 가독성 좋고
    // 쓰기도 쉽습니다. 즉, then 대신 사용하면 됩니다.
    const res1 = await plus( num1: 100,  num2: 200)
    console.log(res1)
   const res2 = await minus( num1: 100,  num2: 200)
    console.log(res2)

    const res3 = await mult( num1: 100,  num2: 200)
    console.log(res3)

    const res4 = await divide( num1: 100,  num2: 200)
    console.log(res4)
}
```

Promise

Promise객체를 사용하여 비동기 작업이 (성공 혹은 실패) 완료된 후의 결과값을 받을 수 있습니다. 그래서 이후 처리를 쉽게 컨트롤 가능합니다. 그래서 이후 처리를 쉽게 컨트롤 가능합니다.

Promise 는 함수를 인자로 받으며 인자로 들어온 함수는 다시 resolve(비동기 처리 성공)와 reject(비동기 처리 실패) 2개의 함수를 인자로 받게 됩니다.

resolve시 then 메소드, reject시 catch 메소드의 인자로 넘어갑니다. Promose는 프로미스가 생성될 때 꼭 알 수 있지는 않은 값을 위한 대리자로, 비동기 연산이 종료된 이후의 결과값이나 실패 이유를 처리하기 위한 처리기를 연결할 수 있도록 합니다. 프로미스를 사용하면 비동기 메서드에서 마치 동기 메서드처럼 값을 반환할 수 있습니다. 다만 최종 결과를 반환하지는 않고, 대신 프로미스를 반환해서 미래의 어떤 시점에 결과를 제공합니다.

Promise는 다음 중 하나의 상태를 가집니다.

-대기(pending): 이행하거나 거부되지 않은 초기 상태

-이행(fulfilld): 연산이 성공적으로 완료됨

-거부(rejected): 연산이 실패함

대기중인 프로미스는 값과 함께 이행할 수도, 어떤 이유(오류)로 인해 거부될 수 있습니다. 이행이나 거부될 때, 프로미스에 연결한 처리기는 그 프로미스의 then 메서드에 의해 대기열에 오릅니다. 이미 이행했거나 거부된 프로미스에 연결한 처리기도 호출하므로, 비동기 연산과 처리기 연결 사이에 경합조건(race condition)은 없습니다.\

Promise.prototype.the() 및 Promise.prototype.catch()메서드의 반환 값은 다른 프로미스]'이므로, 서로 연결할 수 있습니다.

프로퍼티란

'Property'는 속성이란 뜻으로, JS에서는 객체 내부의 속성을 의미합니다. 객체 안에 프로퍼티가 속해있다는 이미지로 생각하면 됩니다.

객체 생성시 프로퍼티 할당

// 객체 생성

var ob = {a: 1};

객체의 프로퍼티에 접근하기

property에 접근하는 방법 프로퍼티에 접근하는 방식은 2가지 입니다.

1.대괄호([ ]) 로 접근 // 브라캣 연산자라고도 부릅니다

2.점 표기법을 이용한 접근

예제1)

var text = “purple haze”;

test[“length”]; // 11

test.length; // 11

예제 2)

// property 접근하기

ob.a; => 1

ob[“a] => 1

객체의 프로퍼티 수정하기

// property 수정하기

ob.a = 0;

객체의 프로퍼티 추가하기
// property 추가하기
ob.b = 2
pb.b; // => 2

객체의 프로퍼티 삭제하기
// property
delete ob.b;
ob.bl // => undefined

Property는 총 6가지의 속성을 가지고 있습니다.
1.Value
2.get
3.set
4.enumerable
5.writable
6.configurable