05

05-1. JavaScript 비동기

Web APIs

Web APIs

- API(Application Programming Interface) 를 한마디로 정리하면 프로그래밍 언어로 만들어진 데이터와 기능의 모음입니다.
- 어떤 기능을 구현할 때 "유튜브 API 를 이용해서" 혹은 "카카오 API 를 이용해서" 구현했다는 용어를 많이 씁니다.
- 이 의미는 유튜브 혹은 카카오와 애플리케이션에서 연동할 때 연동방법을 추상화 시켜서 제공하는 API 를 사용한다는 의미입니다.
- API 는 변수, 함수 혹은 클래스로 제공됩니다.

- Browser APIs : 브라우저에 내장되어 있는 API
- Third-Party APIs : 다른 벤더 혹은 플랫폼에서 제공하는 API
- Browser API 는 정말 많은 것들이 있으며 자세한 것은 https://developer.mozilla.org/en-US/docs/Web/API 에서 확인할 수 있습니다.

setTimeout(), setInterval()

- 함수 호출을 예약해야 하는 경우가 있습니다.
- 1초 후에 함수를 호출해야 하거나 1초마다 반복적으로 함수를 호출해야 하는 경우가 있는데 이를 "호출 스케줄링" 이라고 하며 호출 스케줄링을 위해 setTimeout() 과 setInterval() 을 제공합니다.

- setTimeout() : 일정 시간이 지난 후 함수 실행
- setInterval() : 일정 시간 간격으로 반복적 함수 실행

setTimeout(), setInterval()

setTimeout()

- setTimeout() 은 시간을 지정하고 그 시간이 지난 후 함수를 실행시키기 위해 사용됩니다.

```
setTimeout(functionRef, delay, param1, param2, /* ..., */ paramN)
```

```
function sayHello(){
  console.timeEnd()
  console.log('Hello')
}
console.time()
setTimeout(sayHello, 1000)
```

setTimeout(), setInterval()

setTimeout()

- setTimeout()으로 함수 호출을 예약하기는 했지만 상황이 바뀌어 함수 호출 예약을 취소해야 하는 경우가 있습니다.
- 함수 호출 예약 취소는 clearTimeout() 을 이용합니다.

```
let id = setTimeout(sayHello, 1000)
clearTimeout(id)
```

setTimeout(), setInterval()

setInterval()

- setInterval()은 반복적으로 함수 호출을 예약하는 것입니다.
- 그럼으로 setInterval()에 설정되는 시간은 반복 주기에 해당되는 시간입니다.

```
setInterval(func, delay, arg1, arg2, /* ..., */ argN)
```

```
function sayHello(name){
  console.log(`Hello ${name}`)
}
setInterval(sayHello, 1000, '홍길동')
```

setTimeout(), setInterval()

setInterval()

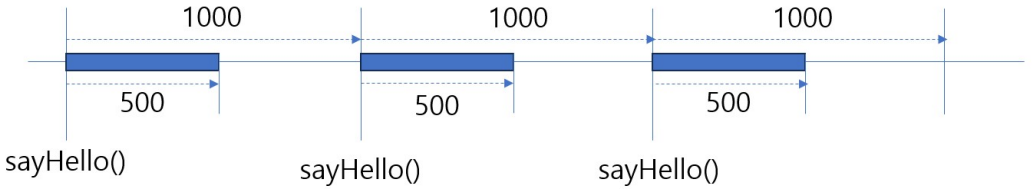
- setInterval() 으로 예약된 함수 호출을 취소하고 싶다면 clearInterval() 을 이용하면 됩니다.

```
let id = setInterval(sayHello, 1000, '홍길동')
setTimeout(() => clearInterval(id), 3000)
```

setTimeout(), setInterval()

setInterval()

- setInterval() 에서 반복 주기 시간은 함수가 호출되는 주기를 의미합니다.
- 만약 1초 주기로 설정되었고 함수가 실행되는데 0.5초가 걸린다고 하면 다음 번 함수가 호출되는 지연시간이 1초가 아니라 0.5초 후에 다시 함수가 호출되게 됩니다.

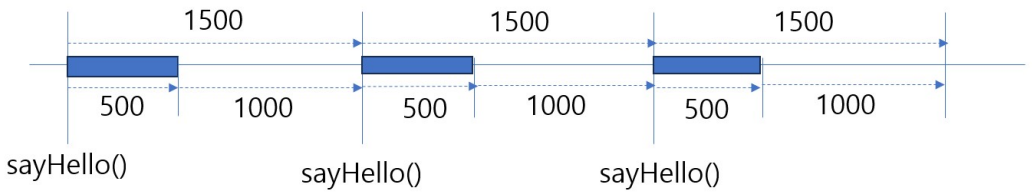


setTimeout(), setInterval()

setInterval()

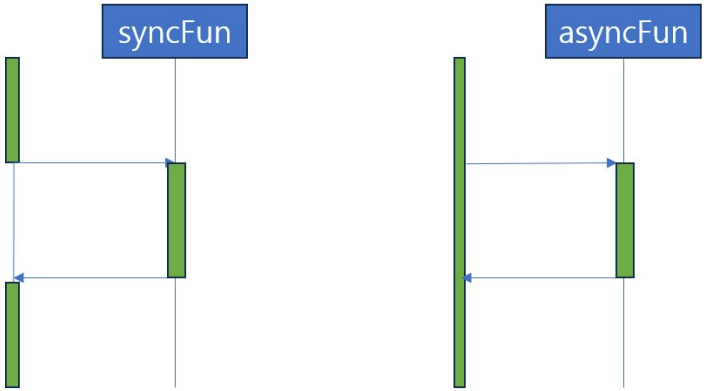
- 만약 함수가 주기적으로 실행되는데 한번 호출한 함수가 실행이 끝난 다음 지연시간을 지정하고 이 지연시간이 지나면 다시 호출하게 하고자 한다면 setTimeout() 을 중첩으로 구현할 수 있습니다.

```
let sayHello = async () => {
  console.log('Hello')
  setTimeout(sayHello, 1000)
}
setTimeout(sayHello, 1000)
```



비동기

- 프로그래밍 코드가 실행되는 형태를 보고 동기 프로그래밍(Synchronous Programming)과 비동기 프로그래밍(Asynchronous Programming) 으로 구분됩니다.
- 동기적으로 함수를 실행시키면 그 함수가 끝날 때까지 함수를 호출한 곳은 아무것도 실행시키지 못하고 대기하게 됩니다.
- 하지만 비동기적으로 함수가 실행되면 함수 호출한 곳이 그 함수가 끝날 때까지 대기하지 않습니다. 함수 호출한 후 바로 자신의 다음줄을 실행시키게 됩니다.



Promise

- 프라미스(Promise) 는 자바스크립트에서 비동기 프로그램을 작성하기 위한 가장 기본적인 기법입니다.
- 프라미스는 영어 단어 뜻으로 약속이며 비동기 업무를 진행하는 곳과 그 업무의 실행 결과를 받아야 하는 곳이 상호 어떻게 연동하는지의 약속을 정의한 객체입니다.

- 시간이 오래 걸리는 작업이 있어 비동기적으로 실행시켜야 한다면 호출한 곳에서 함수가 실행이 완료될 때까지 대기하지 않고 그 다음 줄을 실행시킬 수 있기는 하지만 비동기 업무가 끝난건지? 아니면 비동기 업무가 실행된 결과 무엇인지를 알수가 없게 됩니다.
- 그럼으로 비동기 업무가 진행된 곳에서 나의 업무가 끝났음을, 업무에 대한 결과가 무엇인지를 알려줄 규칙이 필요한데 이 규칙(약속)을 정의한 것이 프라미스입니다.

Promise

- 비동기적으로 실행되는 함수에서 Promise 를 생성해 반환합니다. 이렇게 되면 함수를 호출한 곳은 대기하지 않고 다음 라인을 실행시킬 수 있습니다.
- Promise 생성자의 매개변수에 함수를 등록했는데 이 함수는 Promise 객체가 만들어지자 마자 호출됩니다. 그리고 이 함수가 시간이 오래 걸리는 비동기적으로 실행시켜야 하는 업무를 가지는 함수입니다.
- Promise 생성자에 지정한 함수의 첫번째 매개변수를 이용해야 합니다. 첫번째 매개변수가 함수인데 이 함수를 호출하면서 결과값을 전달합니다.

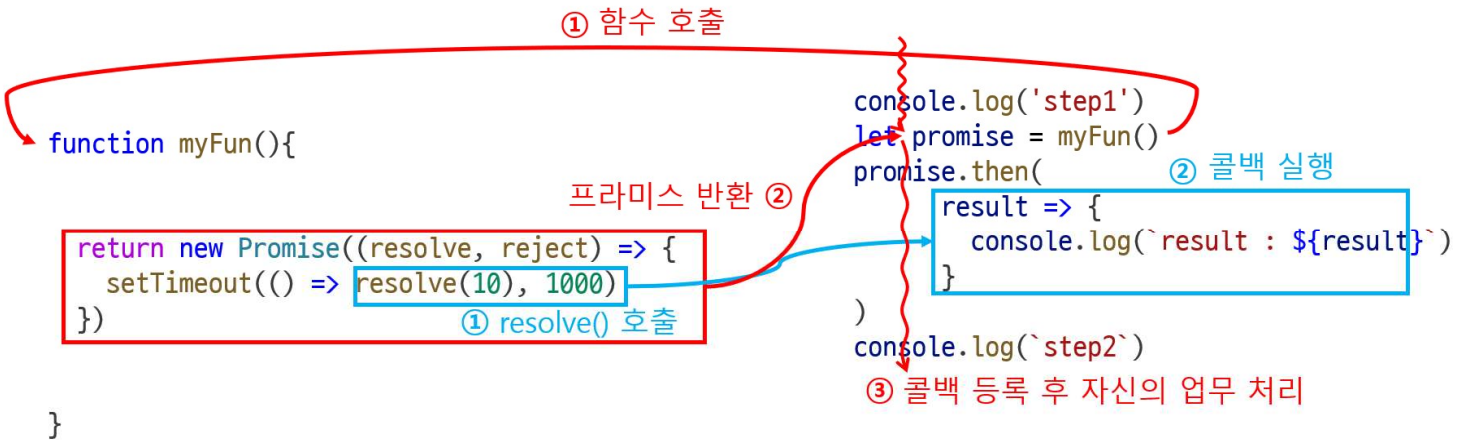
```
function myFun(){
  return new Promise((resolve, reject) => {
    setTimeout(() => resolve(10), 1000)
  })
}
```

Promise

- 비동기 업무의 함수를 호출하는 곳에서는 호출되자 마자 Promise 객체를 반환 했기 때문에 대기하지 않고 그 즉시 다음을 실행시킬 수 있게 됩니다.
- 비동기 업무의 결과를 받기 위해서는 리턴받은 Promise 객체의 then() 을 호출하고 이 then() 에 결과가 반환될 때 호출할 함수를 등록해 놓아야 합니다.
- 이를 흔히 콜백(무언가 상황이 되었을 때 자동으로 호출된다는 의미에서)함수라고 하는데 then() 에 콜백함수를 등록해 놓으면 Promise 에 의해 데이터가 발생하는 순간 호출되어 결과 데이터를 이용하게 됩니다.

```
let promise = myFun()
promise.then(
  result => {
    console.log(`result : ${result}`)
  }
)
```

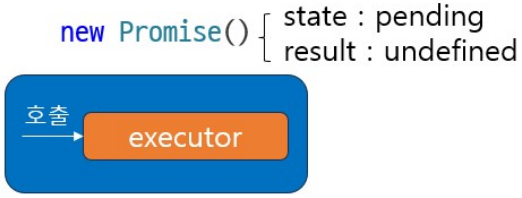
Promise


① 함수 호출
function myFun(){
return new Promise((resolve, reject) => {
setTimeout(() => resolve(10), 1000)
})
① resolve() 호출
}
프라미스 반환 ②
console.log('step1')
let promise = myFun()
promise.then(
② 콜백 실행
result => {
console.log(`result : ${result}`)
}
)
console.log(`step2`)
③ 콜백 등록 후 자신의 업무 처리

Promise 작성 규칙

프라미스 생성 – executor

- 비동기적으로 코드가 실행되어야 하는 부분에서는 자신을 실행시킨 곳이 대기상태가 되지 않도록 Promise 객체를 생성해서 반환해야 합니다.
- Promise 객체 생성할 때 생성자 매개변수에 함수를 지정해야 하는데 이 함수를 executor 라고 합니다. 프라미스에 executor 가 지정이 되면 프라미스는 이 executor 함수를 비동기적으로 즉시 실행시켜 줍니다.
- executor 함수가 실행되는 동안 다른곳이 대기상태가 되지 않습니다.
- 그럼으로 프라미스에 의해 비동기적으로 처리되어야 할 업무는 executor 함수 내에 구현해야 합니다.
- 프라미스 객체 내부에 state 와 result 프로퍼티가 유지되며 state 는 프라미스의 상태값을, result 는 프라미스에 의해 발행된 결과값을 가집니다.
- Promise 객체를 생성하게 되면 state 값은 pending 이며 pending 은 아직 실행이 끝나지 않았음을 의미합니다.



Promise 작성 규칙

프라미스 결과 발생 – resolve

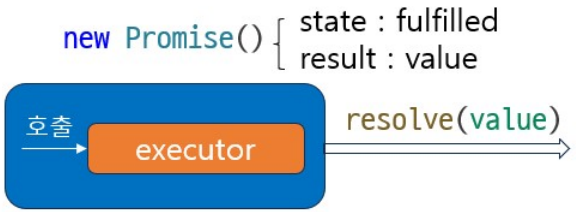
- executor 함수의 실행이 끝났음을 혹은 비동기 업무에 의한 결과 데이터를 외부에서 꼭 알아야 하는 것은 아닙니다.
- 프라미스에 의해 비동기적으로 업무 처리가 진행되고 이 비동기 업무가 외부 코드에 어떠한 영향도 못미칠 수도 있습니다.

- 하지만 많은 경우 executor의 비동기 업무가 끝난 순간 혹은 업무 처리에 의한 결과를 외부에서 알아야 합니다.
- 이 경우 executor 함수의 매개변수를 이용해 외부에 비동기 업무가 종료되거나 발생한 결과 데이터를 전달할 수 있습니다.

Promise 작성 규칙

프라미스 결과 발생 – resolve

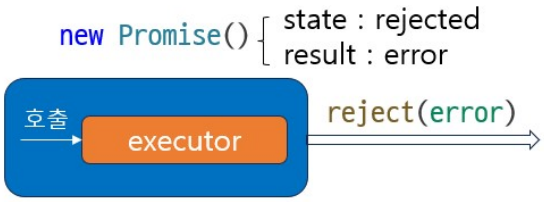
- executor 의 매개변수에 전달된 것은 함수이며 resolve 함수라고 부릅니다.
- 이 resolve 함수를 호출하게 되면 executor 의 업무 처리가 종료된 것이며 resolve 함수 호출시 매개변수에 지정한 값이 프라미스에 의해 발행되는 결과 값입니다.
- executor 의 resolve 함수를 호출하게 되면 프라미스의 state 는 fulfilled 상태가 되며 result 는 발생한 값이 됩니다.



Promise 작성 규칙

프라미스 결과 발생 – reject

- 프라미스의 executor 함수에서 업무를 처리하다가 정상적으로 결과 데이터가 발생한 경우도 있지만 어떤 경우에는 에러가 발생할 수도 있습니다.
- 이 경우 executor 함수에서 에러가 발생했음을, 발생한 에러 정보를 외부에 발행할 필요가 있을 수도 있습니다.
- 이 경우 executor 의 매개변수로 전달되는 reject 함수를 이용하면 됩니다.
- reject 함수가 호출되게 되면 프라미스의 state 는 rejected 상태가 되며 result 는 발행한 에러 내용이 됩니다.

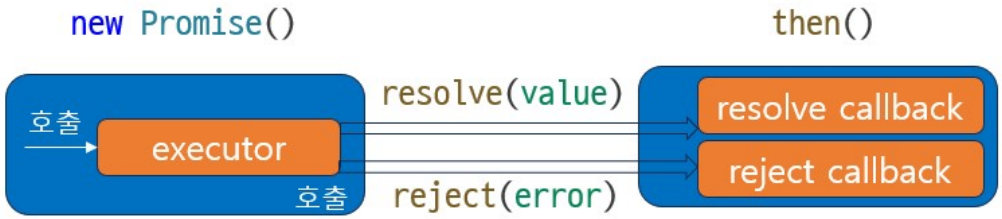



```
function myFun(){
  return new Promise((resolve, reject) => {
    setTimeout(() => reject(new Error('에러 발생')), 1000)
  })
}
```

Promise 작성 규칙

프라미스 콜백

- 프라미스에서 발행한 값을 프라미스를 이용하는 외부에서 전달받아야 합니다.
- 이를 위해서는 프라미스 객체의 then() 함수를 이용합니다.
- then() 의 매개변수에 콜백함수를 등록해 놓으면 프라미스에 의해 resolve 가 호출되거나 reject 이 호출되는 순간 콜백함수가 실행되게 됩니다.
- then() 함수에는 콜백 두개를 등록할 수 있는데 첫번째 매개변수로 지정한 콜백함수가 resolve 에 의해 실행되는 함수이며 두번째 매개변수로 지정한 콜백함수가 reject 에 의해 실행될 함수입니다.



Promise 작성 규칙

catch, finally

- 프라미스 객체가 발행하는 결과를 받기 위한 함수는 then() 만 있는 것이 아니라 catch() 와 finally() 도 있습니다.
- then() 함수만 이용해서 resolve 에 의해 실행될 콜백과 reject 에 의해 실행될 콜백을 모두 등록할 수 있습니다.
- 그런데 만약 reject 에 의해 실행될 결과를 원한다면 then() 이 아닌 catch() 로 등록 할 수 있습니다.
- 또한 finally()를 이용해 콜백을 등록하면 resolve, reject 모든 경우에 실행할 코드를 등록할 수도 있습니다.

async, await

- 프라미스를 이용하다 보면 "then() 을 이용해 프라미스의 결과를 받아야 하는데 then() 에 콜백 함수를 등록해 결과를 받는데 이 부분의 코드를 조금 간결하게 작성할 수 있게 제공되는 것이 async await 입니다.
- async await 는 단지 코드 작성 기법의 차이인 것 뿐이지 내부적으로 프라미스를 이용합니다.

```
function myFun(){
  getData(1)
  .then((value) => {
    console.log(value)
    return getData(2)
  })
  .then((value) => {
    console.log(value)
    return getData(3)
  })
  .then((value) => {
    console.log(value)
  })
}
```

```
async function myFun(){
  console.log(await getData(1))
  console.log(await getData(2))
  console.log(await getData(3))
}
```

async, await

async

- async 는 함수 선언 부분에 사용되는 예약어입니다.
- async 로 선언된 함수는 내부적으로 Promise 를 반환합니다.
- async 로 선언된 함수에서 데이터를 발행하려면 일반 함수가 결과를 반환하듯 return 구문을 사용 합니다.

```
async function myFun1(){
  console.log('myFun1 call..')
}
let myFun2 = async () => {
  console.log('myFun2 call..')
}
```

```
async function myFun2(){
  return 2
}
myFun2().then((value) => console.log(value))//2
```

async, await

await

- async 와 더불어 await 를 같이 사용할 수도 있습니다.
- await 는 함수 내에 사용되는 예약어입니다.
- 그런데 일반 함수에서는 사용할 수 없으며 async 함수내에서만 사용이 가능합니다.

- await 는 '기다리다' 라는 뜻을 가진 단어처럼 async 함수의 실행이 await 로 선언된 구문이 실행이 완료 될 때까지 기다리게 하는 역할을 합니다.

```
async function myFun2() {
  let data = await getData(2)
  console.log(data)
}
```

async, await

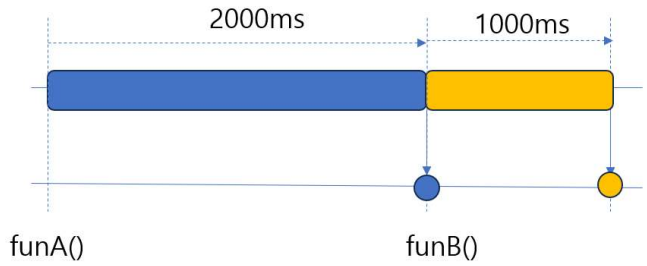
await 를 이용한 순차적 실행

- await 를 이용해 어떤 비동기 업무들이 순차적으로 진행되게 할 수 있습니다.
- 예로 들어 비동기 업무를 구현한 funA, funB 함수 두개가 있다고 가정합시다.
- 이 함수들의 실행이 항상 funA 가 실행이 완료되고 funB 가 실행되어야 한다면 await 를 이용해 아래처럼 작성할 수 있습니다.

```
async function myFun(){
  console.time()

  let aData = await funA()
  console.log(aData)
  let bData = await funB()
  console.log(bData)

  console.timeEnd()
}
```

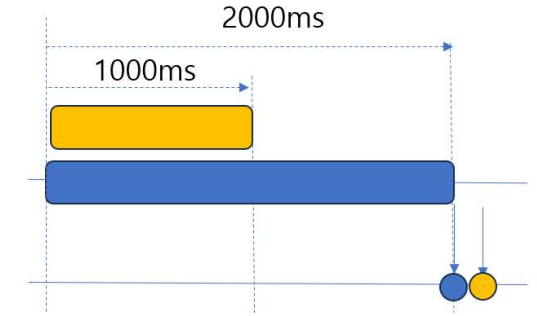


async, await

동시 실행

- funA(), funB() 함수가 동시에 실행되어도 되는 경우를 생각해봅시다.
- 동시에 실행해도 되는 경우 위의 예처럼 await 로 순차적으로 실행시키면 다른 함수가 실행이 완료 될때까지 대기해야 함으로 불필요하게 전체 실행 시간이 지체될 수 있습니다.
- await 의 사용 위치가 함수를 호출하는 위치가 아닌 함수의 결과 데이터 부분에 사용되었습니다. 이렇게 되면 함수 호출은 연달아서 실행됨으로 두 함수가 동시에 실행되게 됩니다.

```
let aData = funA()
let bData = funB()
console.log(await aData)
console.log(await bData)
```



async, await

동시 실행

- Promise.all() 은 배열 정보로 프라미스를 등록하고 모든 프라미스의 데이터 발행이 완료되게 되면 then() 의 콜백을 호출해 줍니다.
- 프라미스에서 모두 데이터가 발행되게 되면 then() 의 콜백함수가 호출이 되며 콜백함수의 매개변수는 배열입니다.

```
async function myFun(){
  Promise.all([funA(), funB()]).then((values) => {
    console.log(values)
  })
}
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare