비동기 호출

비동기 쉽게 이해하기

"동기적(synchronous)"

커피숍에서 커피를 주문하려고 줄을 서는 모습을 상상해보자, 커피숍 사정 상 커피를 주문한 먼저 온 김코딩이 주문한 커피를 받을 때까지, 줄 서 있는 박해커가 주문조차 할 수 없다. 즉, 요청이 온 순 대로 응답을 보내고 다음 요청은 응답이 끝난 후에 요청을 받고, 응답을 보낼 수 있다.

```
const javascript = () => {
    console.log('hello')
    console.log('world')
    console.log('2021.10.28')
}

// hello
// world
// 2021.10.28
```

"비동기(asynchronous)"

커피 주문이 blocking되지 않고, 언제든지 주문을 받을 수 있고, 커피가 완성되는 즉시 커피를 제공하면, 김코딩의 주문 완료 시점과 박해커의 주문 시작 시점이 같을 필요가 없다.

즉, 요청이 온 순 대로가 아닌 요청 완료 처리된 순서대로 응답을 처리한다.

비동기 호출에는 callback, promise, async/await 중 하나의 문법을 이용하여 구현한다.

```
//callback.js
const delay = (wait, callback) => {
  setTimeout(callback, wait);
}

//promiseConstructor
const sleep = (wait) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("hello")
    }, wait);
  });
}

//callback button => 함수를 호출하여 비동기 호출을 한다.
function runCallback() {

  delay(1000, () => {
    pauseVideo();
    displayTitle();

    delay(500, () => {
         highlightTitle();

        delay(2000, resetTitle);
    });
  });
}

//promise button
function runPromise() {

  sleep(1000).then((params) => {
      pauseVideo();
      displayTitle();
      return "world"'
  })
  .then((params) => {
```

```
      sleep(500);
  })
  .then(sleep.bind(null, 500))
  .then(highlightTitle)
  .then(sleep.bind(null, 2000))
  .then(resetTitle)

}

const module = {
  x: 42,
  getX: function() {
    return this.x;
  }
};

const unboundGetX = module.getX;
console.log(unboundGetX()); // The function gets invoked at the global scope
// expected output: undefined

const boundGetX = unboundGetX.bind(module);
console.log(boundGetX());
// expected output: 42
```

공식문서(MDN)

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Function/bind

```
async function runAsync() {

  let returnValue = await sleep(100)

  await sleep(1000)
  pauseVideo();
  displayTitle();
```

비동기 호출하는 함수를 쓸 때는 return값이 항상 Promise형태이다

만일 Promise형태가 아닐 시에는 async/await 함수, promise객체는 사용할 필요가 없는 것이다.

Callback function

```
function greeting(name) {
  alert('hello' + name);
}

function processUserInput(callback) {
  var name = prompt('Please enter your name.')'
  callback(name);
}

processUserInput(greeting);
// 'hello please enter your name'
```

비동기 호출

```
const fs = require('fs')

const getDataFromFile = function(filePath, callback) {

  fs.readFile(filePath, (err, data) => {
     if(data === undefined) {
       callback(err, null)
    } else {
       callback(null, data.toString())
    }
  })
};
```

공식문서(MDN)

https://developer.mozilla.org/en-US/docs/Glossary/Callback_function

Promise 클래스

```
const fs = require('fs')

const getDataFromFilePromise = filePath => {

   return new Promise((resolve, reject) => {
     fs.readFile(filePath, (err, data) => {
        if(err) {
          reject(err)
        } eles {
          resolve(data.toString())
        }
      })
    })
}
```

Promise는 비동기 작업의 최종 완료 또는 실패를 나타내는 객체이다.

Promise는 아래와 같은 특징을 보장한다.

- 콜백은 자바스크립트 Event Loop이 현재 실행중인 콜 스택을 완료하기 이전에는 절대 호출되지 않는다.
- 비동기 작업이 성공하거나 실패한 뒤에 then()을 이용하여 추가한 콜백의 경우에도 위와 같다.
- then()을 여러번 사용하여 여러개의 콜백을 추가 할 수 있다. 그리고 각각의 콜백은 주어진 순서대로 하나 하나 실행되게 된다.

```
//callback 비동기호출
function successCallback(result) {
   console.log("audio file ready at URL" + result);
}
function failureCallback(error) {
   console.log("Error generating audio file:" + error);
}
createAudioFileAsync(audioSettings, successCallback, failureCallback)

//promise 비동기호출
createAudioFileAsync(audioSettings).then(successCallback, failueCallback)

const promise = createAudioFileAsync(audioSettings);
promise.then(successCallback, failureCallback)
```

공식문서MDN

https://developer.mozilla.org/ko/docs/Web/JavaScript/Guide/Using_promises

async function

AsyncFunction객체를 반환하는 하나의 비동기 함수를 정의한다. 비동기 함수는 이벤트 루프를 통해 비동기적으로 작동하는 함수로, 암시적으로 Promise를 사용하여 결과를 반환한다.

```
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('resolve');
```

```
      }, 2000);
    })
}

async function asyncCall() {
  console.log('calling');
  const result = await resolveAfter2Seconds();
  console.log(result)
}

asyncCall();
// > "calling"
// > "resolved"
```

```
const path = require('path')
const { getDataFromFilePromise } = require('./02_promiseConstructor');

const user1Path = path.join(__dirname, 'files/user1.json');
const user2Path = path.join(__dirname, 'files/user2.json');

const readAllUsersAsyncAwait = async () => {

  const user1 = await getDataFromFilePromise(user1Path)
  const user2 = await getDataFromFIlePromise(user2Path)

  return [JSON.parse(user1), JSON.parse(user2)]
}

//getDataFromFilePromise(user1Path)이 함수의 return값이 Promise형태로 들어온다. 그렇기 때문에 await를 써줘야한다. (promise의 then과 같은 역할을 하는 것이다)

//user1의 값은 json형태이며 JSON.parse(user1)을 하게되면 객체의 key, value값으로 parse해준다.
```

공식문서(MDN)

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Statements/async_function