

프로젝트 명세서

비 동기 프로그래밍 이해

목차

1. 개요	3
2. 환경 설정	3
3. 비 동기 함수.....	5
4. Chaining 처리, Hard Code	5
5. Chaining 처리, Soft Code	7
6. All 처리, 비 순차 결과.....	8
7. All 처리, 순차 결과.....	9
8. 과제	10
9. 심화 과제	12
10. 산출물 제출.....	12









1. 개요

보통 프로그램 함수는 순차적으로 실행된다. 즉 동기식으로 진행된다. 이러한 동기식 작업은 많은 시간이 걸리는 작업을 진행 할 경우 그것이 완료 될 때까지 오랫동안 다음 작업을 진행하지 못하는 단점이 있다. 비 동기식은 이러한 문제를 피 할 수 있다. 본 과제는 비 동기식 프로그래밍을 이해하고, 능숙하게 그것을 구현 하는데 목적이 있다.

2. 환경 설정

1. 해당 url 에서 8 개 코드 파일들을 다운 받자. 각 파일은 node.js 에서 단독으로 실행 가능하다.

<https://lab.ssafy.com/crazygun22/shared/-/tree/master/async/test>

Name	Last commit
..	
 await_all_non_sequence.js	gg
 await_all_sequence.js	gg
 await_hard_code.js	gg
 await_soft_code.js	gg
 promise_all_non_sequence.js	gg
 promise_all_sequence.js	gg
 promise_hard_code.js	test
 promise_soft_code.js	test

2. 다음으로 node.js 를 설치하자. (참고 - <https://javapro.tistory.com/62>)
설치하지 않고 온라인에서 실행도 가능하다. (<https://www.jdoodle.com/execute-nodejs-online>), 하지만 결과값이 어떤 시간 경과에 따라 출력되는지 파악하기 어렵다. 가급적 개인 PC 에 설치 후 실행 권한다.

3. 아래는 명령어(파란색 테두리)와 결과(빨간색 테두리)이다.

```
(cnn) D:\#aa>node promise_hard_code.js
SAMSUNG
SW
ACADEMY
FOR
YOUTH
```

3. 비 동기 함수

아래는 비 동기 함수이다. delay 만큼 기다린 후 입력 값 word 그대로 return 하는 간단한 동작을 한다. 제공되는 각각의 코드 파일에서 이 비 동기 함수를 처리하는 방식은 다 다르다.

```
function delay_word(word, delay) {  
  return new Promise(resolve => {  
    setTimeout(function () {  
      resolve(word)  
    }, delay)  
  })  
}
```

4. Chaining 처리, Hard Code

보통 하나 또는 두 개 이상의 비 동기 작업을 순차적으로 실행하는 상황이 있다. 이전 비 동기 작업이 완료 후 다음 작업을 실행해야 하는 경우이다.

```

delay_word('SAMSUNG', 500).then((resolve) => {
  console.log(resolve)
  delay_word('SW', 490).then((resolve) => {
    console.log(resolve)
    delay_word('ACADEMY', 480).then((resolve) => {
      console.log(resolve)
      delay_word('FOR', 470).then((resolve) => {
        console.log(resolve)
        delay_word('YOUTH', 460).then((resolve) => {
          console.log(resolve)
        })
      })
    })
  })
})

```

promise_hard_code.js

위의 promise 코드를 async/await 코드로 변환할 수 있다. 코드만 다를 뿐 동작은 완전히 동일하다.

```

async function test(){
  const resolve_0 = await delay_word('SAMSUNG', 500)
  console.log(resolve_0)
  const resolve_1 = await delay_word('SW', 490)
  console.log(resolve_1)
  const resolve_2 = await delay_word('ACADEMY', 480)
  console.log(resolve_2)
  const resolve_3 = await delay_word('FOR', 470)
  console.log(resolve_3)
  const resolve_4 = await delay_word('YOUTH', 460)
  console.log(resolve_4)
}

```

await_hard_code.js

5. Chaining 처리, Soft Code

위의 예제 4 는 Hard Code 로 구현하였다. 만약 비 동기 함수 호출 횟수나 입력 값이 가변적이라면, 아래와 같이 Soft Code 로 구현해야 한다.

```
const array = [{word:'SAMSUNG', delay:500}, {word:'SW', delay:490},
               {word:'ACADEMY', delay:480}, {word:'FOR', delay:470},
               {word:'YOUTH', delay:460}]

array.reduce((prev, item) => {

  return prev.then(() =>
    delay_word(item.word, item.delay).then((promise) => {console.log(promise)}))

}, Promise.resolve())
```

promise_soft_code.js

위의 promise 코드를 async/await 코드로 변환할 수 있다. 코드만 다를 뿐 동작은 완전히 동일하다.

```
const array = [{word:'SAMSUNG', delay:500}, {word:'SW', delay:490},
               {word:'ACADEMY', delay:480}, {word:'FOR', delay:470},
               {word:'YOUTH', delay:460}]

async function test(){

  for(const item of array) {
    const resolve = await delay_word(item.word, item.delay)

    console.log(resolve)
  }
}
```

await_soft_code.js

6. A11 처리, 비 순차 결과

이전 비 동기 작업이 다음 비 동기 작업에 영향을 주지 않을 경우, 이전 비 동기 함수 작업이 끝나기 전에, 새 작업을 실행해도 무방하다. 그리고 함수 호출 순서와 상관없이 작업이 먼저 끝나는 순으로 결과를 받는다.

```
const array = [{word:'SAMSUNG', delay:500}, {word:'SW', delay:490},
               {word:'ACADEMY', delay:480}, {word:'FOR', delay:470},
               {word:'YOUTH', delay:460}]

array.forEach(async (item) => {

    delay_word(item.word, item.delay).then((resolve) => {console.log(resolve)})
})
```

promise_all_non_sequence.js

위의 promise 코드를 async/await 코드로 변환할 수 있다. 코드만 다를 뿐 동작은 완전히 동일하다.

```
const array = [{word:'SAMSUNG', delay:500}, {word:'SW', delay:490},
               {word:'ACADEMY', delay:480}, {word:'FOR', delay:470},
               {word:'YOUTH', delay:460}]

array.forEach(async (item) => {

    const resolve = await delay_word(item.word, item.delay)

    console.log(resolve)

})
```

await_all_non_sequence.js

7. A11 처리, 순차 결과

위의 6 번(A11 처리, 비 순차적으로 결과)에서 결과는 순차적으로, 즉 함수 시작 순으로 결과를 받을 수 있다.

```
const array = [{word:'SAMSUNG', delay:500}, {word:'SW', delay:490},  
               {word:'ACADEMY', delay:480}, {word:'FOR', delay:470},  
               {word:'YOUTH', delay:460}]  
  
const promise_list = []  
  
array.forEach((item) => {  
    const promise = delay_word(item.word, item.delay)  
    promise_list.push(promise)  
})  
  
Promise.all(promise_list).then((values) => {  
    values.forEach((resolve) => {console.log(resolve)})  
})
```

promise_all_sequence.js

위의 promise 코드를 async/await 코드로 변환할 수 있다. 코드만 다를 뿐 동작은 완전히 동일하다.

```
const array = [{word:'SAMSUNG', delay:500}, {word:'SW', delay:490},  
               {word:'ACADEMY', delay:480}, {word:'FOR', delay:470},  
               {word:'YOUTH', delay:460}]  
  
async function test(){  
  const async_fun_list = []  
  
  for(item of array){  
    const async_fun = delay_word(item.word, item.delay)  
  
    async_fun_list.push(async_fun)  
  }  
  
  for(async_fun of async_fun_list){  
    const resolve = await async_fun  
  
    console.log(resolve)  
  }  
}
```

await_all_sequence.js

8. 과제

아래 url 에서 promise_hard_code.js 파일을 다운 받을 수 있다. 제공되는 코드를 보면 axios 를 사용하는 비동기 함수가 있다.

<https://lab.ssafy.com/crazygun22/shared/-/tree/master/async/test/>

```
const axios = require('axios');

async function request(sub_path){

    const url = 'http://13.124.193.201:8844/' + sub_path

    try{

        const response = await axios.get(url);

        return response.data
    }
    catch(e){

        console.log(e)
    }
}
```

axios 비동기 함수

위의 비 동기 함수로 총 8 개의 비동기 프로그램을 개발할 수 있다. 아래와 같이 모두 4 개의 처리 방식이 있고, 각 방식 마다 2 개의 코드(promise____.js, await____.js) 가 있다. 이 중 promise_hard_code.js 는 제공되므로 그것을 제외한 나머지 7 개에 대한 코드 파일을 작성해서 제출한다.

Chaining 처리, Hard Code	promise_hard_code.js(제공됨) await_hard_code.js
Chaining 처리, Soft Code	promise_soft_code.js await_soft_code.js
All 처리, 비 순차 결과	promise_all_non_sequence.js await_all_non_sequence.js
All 처리, 순차 결과	promise_all_sequence.js await_all_sequence.js

9. 심화 과제

file read/write 같은 다른 비 동기 함수를 찾아서 위의 방식들을 개발해보자.

10. 산출물 제출

<https://lab.ssafy.com/s10-study/self-project/> 의 “산출물 제출 가이드”.docx
참조