inno SOPT Server

3차 세미나

Contents

22기 SOPT Server
3차 세미나

SHOUT OUR PASSION TOGETHER
SOPT

01

Callback Hell

01
Callback Hell


```
crypto.randomBytes(32, function(err, buffer) {
  if (err) {
    console.log(err);
  } else {
    console.log('buffer : ' + buffer.toString('base64'));
    crypto.pbkdf2(str, buffer.toString('base64'), 100000, 64, 'sha512', function(err, hashed) {
      if (err) {
        console.log(err);
      } else {
        console.log('hashed : ' + hashed.toString('base64'));
      }
    });
  }
});
```

Node.js 에서는 순차적으로 작업을 수행하기 위해 callback 함수 안에 callback 함수가 있는 형태로 코드를 작성해야 한다.

01
Callback Hell

가독성이 떨어지고, 유지보수가 어려움

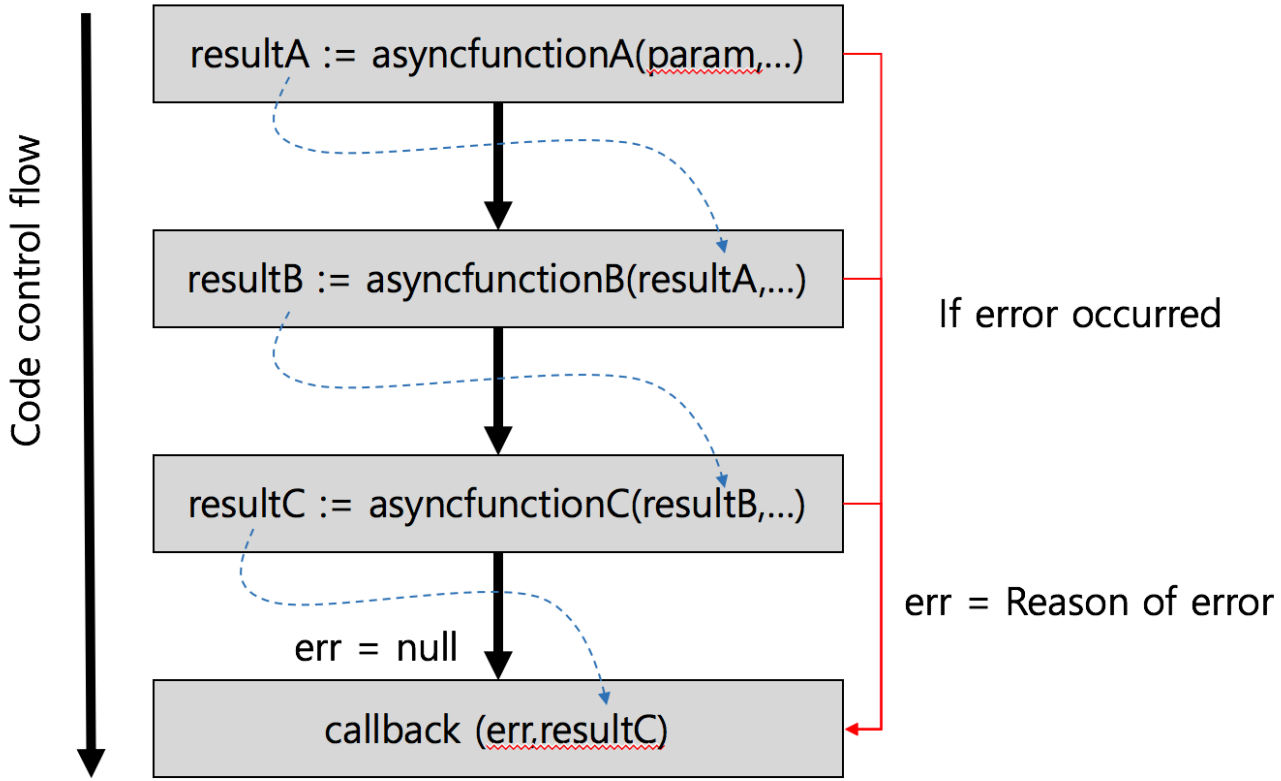
=> **Callback Hell!!**

```
                                                                              let star
                                                                              if (role
                                                                              else if (
                                                                              connectio
                                                                                  if (e
                                                                                      c
                                                                                      r
                                                                                      c
                                                                                  } els
                                                                                      c
                                                                                      r
                                                                                      c
                                                                                  }
                                                                              });
                                                                          }
                                                                      });
                                                                  }
                                                              });
                                                          }
                                                      });
                                                  }
                                              });
                                          }
                                      });
                                  }
                              });
                          }
                      });
                  }

              }
          });
      } else {
```


02
Async

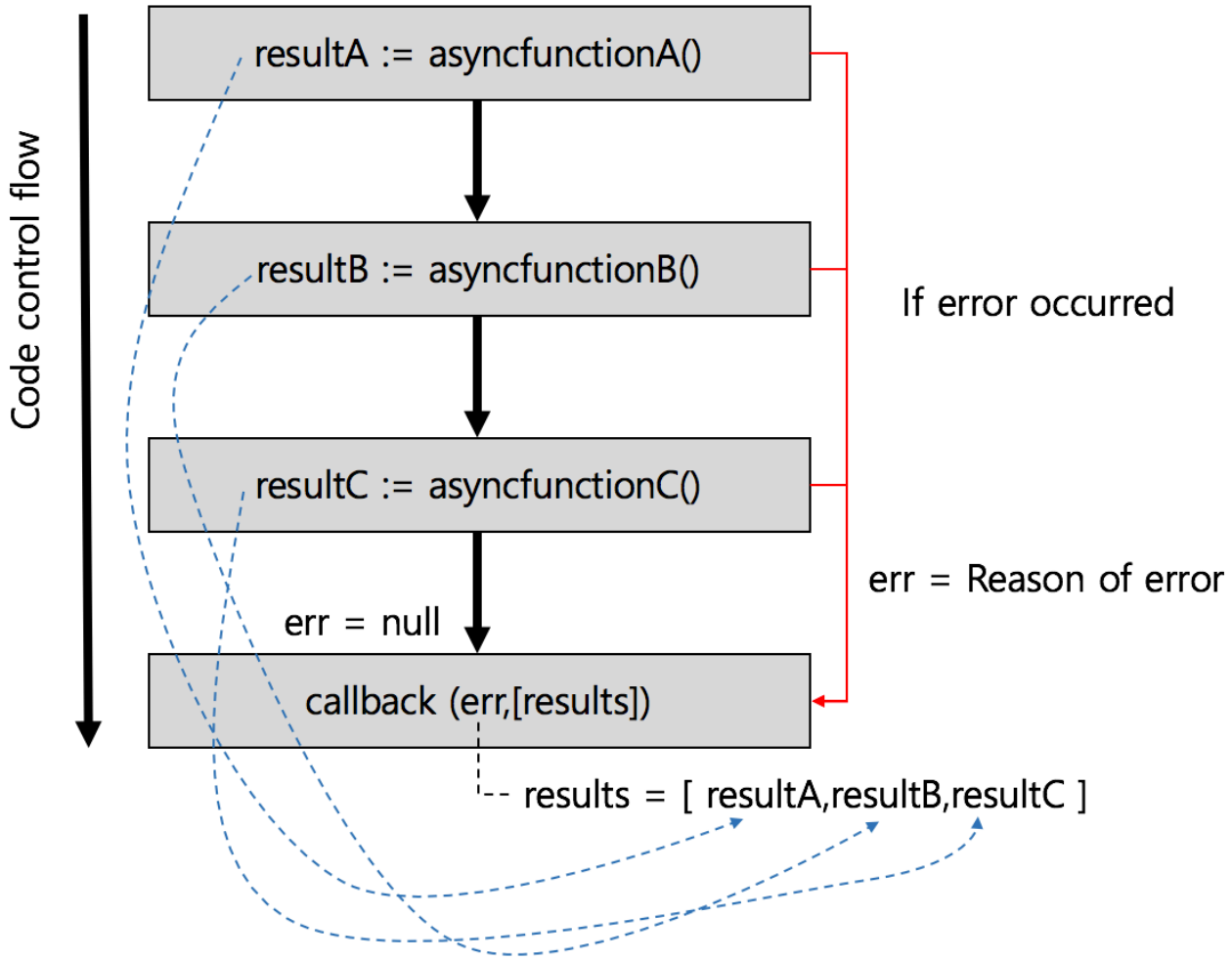
- 비동기 동작의 흐름제어를 도와주는 모듈
- 설치 : npm install async
- 주요 메소드
  - series
  - waterfall
  - parallel
- 그 외의 메소드 : https://caolan.github.io/async/



Waterfall

- 여러개의 비동기 함수를 순차적으로 실행하되 앞의 비동기 함수에서 나온 값(혹은 다음 함수로 넘기고 싶은 값) 을 뒤의 비동기 함수에 인자로 전달하는 waterfall 흐름
- callback 함수 첫 번째 인자

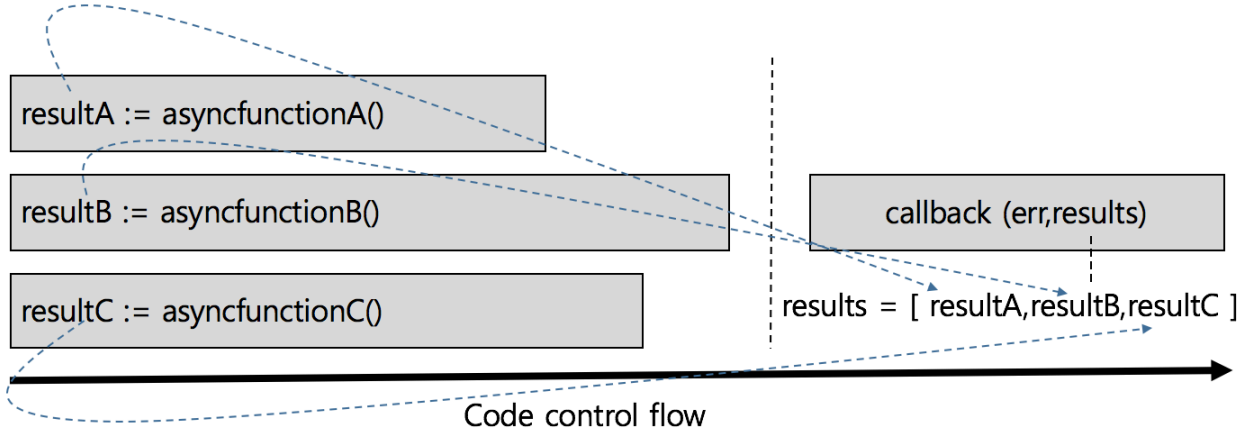
  null -> 정상 수행 / not null -> 에러 발생
  - task 정상적으로 모두 수행 -> 마지막에 정의된 callback 함수 수행
  - task 수행 도중 에러 발생 시 -> task 수행 멈추고 main callback 호출 -> error(첫 번째 인자)에 에러 내용을 넘김



Series

- 여러개의 비동기 함수를 순차적으로 실행하되 각 task 에서 나온 값(혹은 다음 함수로 넘기고 싶은 값)을 취합하여, 최종 callback에 배열 형태로 넘김
- callback 함수 첫 번째 인자

  null -> 정상 수행 / not null -> 에러 발생
  - task 정상적으로 모두 수행 -> 마지막에 정의된 callback 함수 수행
  - task 수행 도중 에러 발생 시 -> task 수행 멈추고 main callback 호출 -> error(첫 번째 인자)에 에러 내용을 넘김



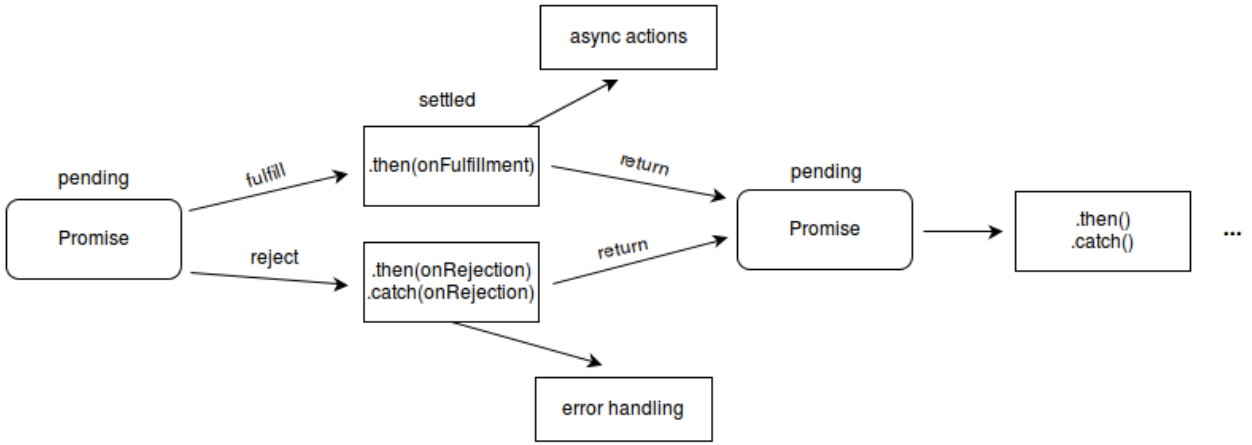
Parallel

- 여러개의 비동기 함수를 동시에 실행하는 방법. 마치 멀티 쓰레드**같은** 효과를 낼 수 있음.
- 각 task 에서 나온 값(혹은 다음 함수로 넘기고 싶은 값)을 취합하여, 최종 callback에 배열 형태로 넘김
- task 정상적으로 모두 수행 -> 마지막에 정의된 callback 함수 수행
- task 수행 도중 에러 발생 시 -> 바로 최종 callback에 에러를 넘김. 에러가 발생하지 않은 task는 계속 수행


03
Promise

Promise

- Javascript ES6에 추가
- Promise의 상태
  - pending : 동작 완료 전(성공과 실패가 되기 전) 수행중인 상태
  - fulfill : 비동기 동작 성공
  - reject : 동작 실패
- return new Promise(function(fulfill, reject))
- .then().catch()
- parameter는 최대 한 개(여러 개는 배열 형태로)
- 참조 문서 : https://www.promisejs.org/api/



```
function create() {
  return new Promise(function(resolve, reject) {
    resolve();
    console.log("1. resolve");
  });
};
```

Promise – pending 상태

1. new Promise() 로 인스턴스 생성
2. executer 실행 -> 성공과 실패에 따라 호출할 핸들러 함수 바인딩

```
create().then(function() {
  console.log("3. 성공");
}, function() {
  console.log("3. 실패");
});
```

Promise – settled 상태

1. pending 상태 종료 -> settled 상태로 변환
2. 이 때 처리의 성공과 실패 알 수 있음
3. fulfill (성공) / reject (실패) 상태
4. 상태에 따라 pending 단계에서 바인딩한 핸들러 함수 호출

04

Async/await

Arrow Function

- ES6에 추가된 함수 작성을 위한 새로운 구문
- function 키워드 대신에 간단하게 => 형태로 함수를 선언
- (params) => { code }
- (...args) => { code }
    - args[0], args[1], args[2]

```
var create = async (req, res, next) => {}

var create = async (...args) => {}
```

사용자 정의 모듈

- exports는 속성을 추가할 수 있어, 변수나 메소드를 **여러 개** 정의 할 수 있음
- module.exports는 **하나**의 변수나 메소드, 객체를 직접 할당할 수 있음. 파일 자체를 속성이나 메소드로 사용
- require() 을 통해 변수로 불러와 모듈로 사용할 수 있음

- 참조 문서 : https://nodejs.org/dist/latest-v8.x/docs/api/modules.html

await

- await 연산자는 Promise 를 기다리기 위해 사용
- async function 내부에서만 사용
- await 문은 async 함수의 실행을 중단시키고, Promise 가 fulfill 되거나 reject 되기를 기다림

  다시 async 함수를 실행시킴

  이 때, await 문의 값은 Promise에서 fulfill되거나, reject 된 값
- non-async function 에서 await 를 사용하면 syntax error 발생
- crypto-promise, promise-mysql 등 async/await 를 제공하는 모듈

Homework

SHOUT OUR PASSION TOGETHER SOPT

과제

1. 임의의 문자열을 해싱합니다.

   * algorithm : SHA512, salting, key stretching을 모두 사용해주세요.

2. 해싱이 성공하면 Text File에 해싱된 문자열을 저장합니다.

   또한, JSON 데이터로 Response합니다. *JSON객체의 프로퍼티 키에는 msg, hashed가 있습니다.

   msg에는 성공 혹은 실패 메세지를, hashed에는 해싱된 문자열을 넣어주세요.

3. 이 때 받은 JSON Response 를 웹페이지에 띄웁니다. Content-Type은 자유입니다.

4. 3000번 포트로 해당 서버가 동작하도록 열어주세요.

5. 위의 과제를 async(waterfall, series 중 선택), promise를 사용하여 작성해주세요.

코드는 **async : 이름_homework3-1.js**, **promise : 이름_homework3-2**.js 로 저장하세요.

두 파일을 압축하여 이름_homework3 으로 저장하여 드라이브에 업로드해주세요!

inno SOPT

3차 The End :)