

MEAN 스택 활용 웹 개발

Node.js 콜백 관리



2016년도 2학기 2학년 방과후학교

Node.js 콜백 함수 사용

- 비동기 Task들이 연달아 사용되는 경우를 생각해보자.
- Task1이 종료되면, 그 결과를 이용하여 Task2가 실행되어야 하고, 다시 Task2의 결과를 이용해서 Task3가 실행되어야 한다면?
- 콜백 함수 안에 또 콜백 함수를 연속해서 작성해야 한다!!

```
function task1(arg1, function (err, result) {  
  ...  
  function task2(arg2, function (err, result2) {  
    ...  
    function task3(arg3, function (err, result3) {  
      ...  
    }); // task3  
  }); // task2  
}); // task1
```

콜백 Hell 탈출 시도

- 콜백 함수를 인라인으로 작성하지 않고 밖으로 빼면 중첩되는 것은 막을 수 있겠다.
- 하지만, 코드 연관성이 떨어지고, 콜백 함수를 찾기 위한 과정이 필요 TT

```
task1Call('a', task1);  
function task1(result) {  
    task2Call('b', task2);  
}  
function task2(result) {  
    task3Call('c', task3);  
}  
function task3(result) {  
    ...  
}
```

비동기 동작 흐름 제어 - Async

- `async` 모듈은 비동기 동작들의 흐름을 제어하는 기능을 제공하는 모듈이다.
- `npm install async`
- `async` 모듈이 제공하는 주요 기능으로는 비동기 동작들의 순서 제어, 콜렉션과 비동기 동작의 제어 등이 있다.

비동기 동작 흐름 제어 - Async

- `async` 모듈의 `series`는 비동기 동작을 순차적으로 실행하는 대표적인 함수다.
- `series` 함수의 첫 번째 파라미터에는 비동기 태스크에 해당하는 함수를 배열로 작성한다.

```
async.series( [  
    task1,  // 첫번째 함수  
    task2,  // 두번째 함수  
    task3,  // 세번째 함수  
], function (err, result) {  
    // 완료 콜백  
} );
```

비동기 동작 흐름 제어 - Async

- async 모듈의 waterfall은 비동기 태스크 간에 데이터를 전달할 수 있다는 특징이 있다.

```
async.waterfall( [  
    task1, // 첫번째 함수  
    task2, // 두번째 함수  
    task3, // 세번째 함수  
], function (err, result) {  
    // 완료 콜백  
} );
```

```
function task1(callback) {  
    ...  
    callback(null, 'result');  
}  
function task2(arg, callback) {  
    ...  
    callback(null, 'result1', 'result2');  
}  
function task3(arg1, arg2, callback) {  
    ...  
    callback(null, 'result');  
}
```

비동기 동작 흐름 제어 - Async

- async 모듈에서 series나 waterfall은 순차적으로 실행해야 하는 경우에 사용하고 비동기 태스크를 동시에 동작시키려면 parallel을 사용한다.
- 비동기 동작이 모두 끝나고 나서의 동작을 작성하는 경우 사용한다.

```
async.parallel( [  
    task1,  
    task2,  
    task3,  
], function (err, result) {  
    // 완료 콜백  
} );
```

```
function task1(callback) {  
    ...  
    callback(null, 'result1');  
}  
function task2(callback) {  
    ...  
    callback(null, 'result2');  
}  
function task3(callback) {  
    ...  
    callback(null, 'result3');  
}
```

비동기 동작 흐름 제어 - Promise

- Promise은 체인 방식으로 비동기 동작의 흐름을 제어하는 방법이다.
- <http://www.promisejs.org>
- Node.js 4 이후부터는 별도의 모듈을 설치하지 않고 사용할 수 있다.
- Promise의 상태
 - ✓ pending : Promise 객체가 생성되고 아직 실행되지 않은 상태
 - ✓ fulfilled : 성공적으로 실행된 상태
 - ✓ rejected : 동작이 실패한 상태

비동기 동작 흐름 제어 - Promise

- 비동기 태스크가 성공하면, fulfill 함수를 호출해서 fulfilled 상태로 변경하고 실패하면 reject 함수를 호출해서 프라미스가 rejected 된 상태로 변경한다.

```
new Promise(function (fulfill, reject) {  
  // 비동기 동작  
  if ( err ) {  
    reject(err);    // 실패인 경우  
  } else {  
    fulfill(result); // 성공인 경우  
  }  
});
```

비동기 동작 흐름 제어 - Promise

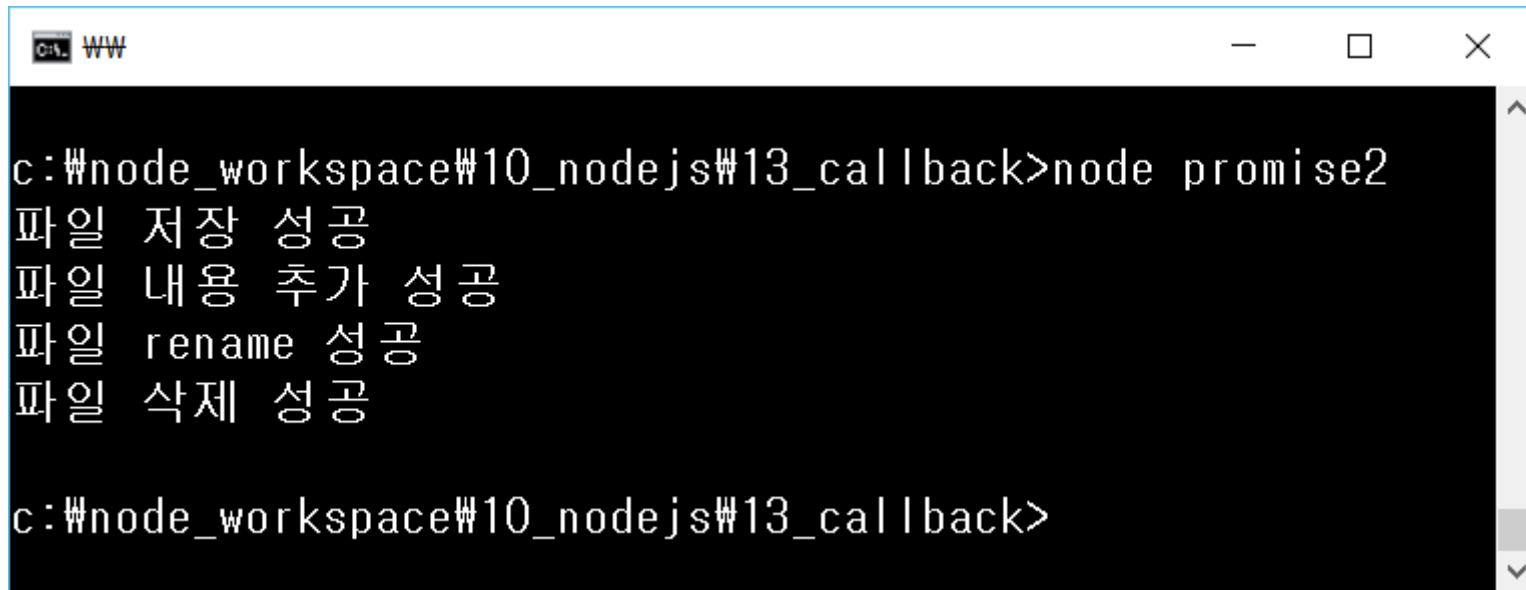
- Promise 이후의 동작 : then
- 비동기 태스크가 성공적으로 끝나면 then함수의 첫 번째 콜백(fulfilled)이, 비동기 태스크에 문제가 생기면 두 번째 콜백(rejected)이 실행된다.

```
new Promise( task ).then(fulfilled, rejected);
```

```
function fulfilled(result) {  
    // fulfilled 상태일 때 동작  
}  
function rejected(result) {  
    // rejected 상태일 때 동작  
}
```

[실습] Promise

- Async 모듈을 이용해서 처리했던 async.js 파일을 Promise Chaining을 사용하여 변경해보자!!
- 파일명 : 13_callback→promise2.js



```
c:\node_workspace\10_nodejs\13_callback>node promise2
파일 저장 성공
파일 내용 추가 성공
파일 rename 성공
파일 삭제 성공

c:\node_workspace\10_nodejs\13_callback>
```