T아카데미 온라인 강의

Node.js 프로그래밍

06. 흐름 제어

CONTENTS

학습 목표

1. 콜백과 콜백으로 발생하는 복잡한 상황을 확인할 수 있습니다.
2. Async 모듈과 Promise를 이용해서 복잡도를 낮추면서 작업 흐름 제어를 할 수 있습니다.

1. 콜백과 콜백 헬

1. 콜백과 콜백 헬

- 콜백 함수 다시 보기

비동기 함수 – 콜백 함수 사용

```
function asyncTask(path, function(result) {
   // 결과 처리
});
```

1. 콜백과 콜백 헬

비동기 동작과 콜백

비동기 동작의 연속

- task1 실행 이후에 task2 실행
- task1 실행 결과를 이용해서 task2 실행

콜백의 연속된 호출

1. 콜백과 콜백 헬

- 연속된 비동기 동작의 예

  - 이미지 업로드 후 데이터 베이스에 저장
  - 다수의 이미지에서 썸네일 생성 후 업로드

1. 콜백과 콜백 헬

연속된 비동기 동작 코드

비동기 동작 연속 실행

```
function task1(args, function(result) {
  // 결과 처리
});
function task2(args, function(result) {
  // 결과 처리
});
```

실행

```
task1()
task2()
```

1. 콜백과 콜백 헬

- 연속된 비동기 동작 코드

  - task1 실행 이후에 task2 실행

```
task1(arg1, function(result) {
    task2(arg2, function(result) {
    });
);
```

1. 콜백과 콜백 헬

- 연속된 비동기 동작 코드
  - task1 실행 결과를 task2에서 사용

```
task1(arg1, function(result) {
    var arg2 = result.value;
    task2(arg2, function(result) {
    });
);
```

1. 콜백과 콜백 헬

- 콜백의 연속으로 발생한 현상

그러다 콜백 지옥에 도착

```
task1(a, b, function(err, result1) {
    task2(c, function(err, result2) {
        task3(d, e, f, function(result3) {
            task4(h, i, function(result4) {
                // 비동기 동작
            }); task4
        }); // task3
    }); // task2
}); // task1
```

1. 콜백과 콜백 헬

콜백 지옥 탈출 시도

콜백 지옥 탈출하기

```
task1('a', 'b', task1Callback);

function task1Callback(result) {
  task2('c', task2Callback);
}

function task2Callback(result) {
  // task3 호출
}
```

2. Async

2. Async

- 비동기 동작의 흐름 제어

Async 모듈

- https://github.com/caolan/async

모듈 설치

- npm install async

2. Async

- Async의 대표적인 기능
  - 행위 순서 제어
    - series, seriesEach
    - parallels
    - waterfall
  - 콜렉션(배열, 객체)
    - each
    - forEachOf
    - map, filter

2. Async

Async의 순차 실행

series(tasks, [callback])

```
async.series(
    [
        태스크1,
        태스크2,
        태스크3
    ],
    function(err, results) {
        완료 콜백
    }
);
```

2. Async

- 순차 실행
  - callback 호출 : 다음 태스크로 진행
  - 태스크 완료 : 다음 태스크 실행

```
function(callback) {
    // 태스크 성공
    callback(null, result);
}
```

  - 완료 콜백으로 동작 결과 전달

2. Async

- 순차 실행
  - 태스크 에러 발생 : 에러 전달

```
function(callback) {
    // 에러 발생
    callback(err, null);
}
```

  - 다음 태스크 실행 안함.
  - 마무리 콜백으로 에러 전달

2. Async

- 연속 동작 마무리

serial 마무리 콜백

```
async.series([ 태스크1, 태스크2, 태스크3],
    function(err, results) {
        if ( err ) {
            // 태스크 진행 중 에러 : callback(err, null)
            return;
        }
        // 마무리 동작
    }
);
```

results 에는 각 태스크의 결과가 배열 형태로 전달

2. Async

async.series 예제 코드

async.series

```
async.series([
  function task1(callback) {
    callback(null, ‘result1’);
  },
  function task2(callback) {
    callback(null, ‘result2’);
  },
  function task3(callback) {
    callback(null, ‘result3’);
  }
  ],
  function (err, results) {
    // results : [‘result1’, ‘result2’, 'result3']
  }
);
```

2. Async

- 순차 실행

태스크로 정보를 전달하려고 할 때 : async.waterfall

- 다음 태스크로 전달할 값을 콜백의 파라미터로
- 태스크 함수의 파라미터로 전달 이전 태스크의 값 전달

```
function task1(callback) {
  callback(null, 'value');
},
function task2(arg, callback) {
  callback(null, 'hello', 'world');
}
```

2. Async

async.waterfall 예제 코드

async.waterfall 샘플 코드

```
async.waterfall([
  function task1(callback) {
    callback(null, 'value');
  },
  function task2(arg, callback) {
    callback(null, 'value1', 'value2');
  },
  function task3(arg1, arg2, callback) {
    callback(null, 'result');
  }
  ],
  function (err, results) {
  }
);
```

2. Async

동시 실행

여러 태스크를 동시에 실행

모든 태스크를 마치면 완료 콜백

- parallel(tasks, [callback])

사용 방식

```
async.parallel( [ task1, task2, task3 ],
  function(err, results) {
    // ['태스크 1 결과', '태스크 2 결과', '태스크 3 결과']
  }
);
```

2. Async

동시 실행 예제 코드

async.parallel 예제 코드

```
async.parallel(
  [
    function(callback) {
      callback(null, '태스크 1 결과');
    },
    function(callback) {
      callback(null, '태스크 2 결과');
    },
    function(callback) {
      callback(null, '태스크 3 결과');
    }
  ],
  function(err, results) {
    console.log('모든 태스크 종료, 결과 : ', results); // ['태스크 1 결과', '태스크 2 결과', '태스크 3 결과']
  }
);
```

2. Async

- 콜렉션과 비동기 동작
  - 콜렉션 내 각 항목을 사용하는 비동기 동작
    - 다수의 파일(배열)을 비동기 API로 읽기
    - 다수의 파일을 비동기 API로 존재하는지 확인하기

2. Async

- 콜렉션과 비동기 동작
  - 비동기 순회 동작
    - each, eachSeries, eachLimit
    - map, filter
    - reject, reduct
    - …

2. Async

- 콜렉션과 비동기 동작

콜렉션과 사용하기 : each

- each(arr, iterator, [callback])

예제 코드

```
async.each(array, function (item, callback) {
  // 배열 내 항목 item을 사용하는 비동기 동작
  callback(null);
}, function (err) {
  // async.each 완료
});
```

3. Promise

3. Promise

- Promise

  - 비동기 동작의 흐름 제어
  - 사이트 : www.promisejs.org
  - JavaScript ES6에 추가
    - ✓Node.js 4.x 이후 모듈 설치 필요 없음

3. Promise

Promise 생성

Promise 객체 생성

```
new Promise(function() {
  // 비동기 동작
});
```

3. Promise

Promise 상태

Promise의 상태

- pending : 동작 완료 전
- fulfilled : 비동기 동작 성공
- rejected : 동작 실패

3. Promise

Promise 상태 반영

Promise 생성, 상태 반영

- 성공적으로 완료 : fullfill 호출
- 에러 상황 : reject 호출

Promise 생성, 상태 반영

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

3. Promise

Promise 이후 동작

Promise 이후의 동작 : then

- fullfilled 상태일 때의 콜백
- rejected 상태일 때의 콜백

코드 형태

```
new Promise(task).then(fullfilled, rejected);
function fullfilled(result) {
    // fullfilled 상태일 때의 동작
}
function rejected(err) {
    // rejected 상태일 때의 동작
}
```

3. Promise

Promise를 사용하는 태스크

Promise를 사용하는 태스크

```
function task() {
  return new Promise(function(fullfill, reject) {
    if ( success )
      fullfill('Success');
    else
      reject('Error');
  });
}
```

태스크 사용 코드

```
task(arg).then(fullfilled, rejected);
```

3. Promise

- 흐름 제어
  - Async, Promise
    - Async : Flow Control
    - Promise : Chain

학습정리

학습정리

- 지금까지 '흐름 제어'에 대해 살펴보았습니다.

콜백과 콜백 지옥

비동기 동작을 연속적으로 수행하는 경우 콜백의 중첩으로 복잡도가 증가하는 현상을 살펴봤습니다.

Async

Async 모듈을 이용해서 비동기 동작의 흐름을 제어했습니다.

Promise

자바 스크립의 표준에 추가된 Promise를 살펴봤습니다. 체인 방식으로 비동기 동작을 제어합니다.