

비동기 이벤트와 데이터를 관리

이 장의 내용

- 비동기 코드 개발의 어려움
- 함수형 기법으로 중첩된 콜백 사용을 막음
- 프라미스를 응용하여 비동기 코드를 능률화
- 함수 제너레이터로 데이터를 느긋하게 만듦
- 리액티브 프로그래밍 입문
- 리액티브 프로그래밍을 적용한 이벤트 중심 코드

덩어리가 큰 클라이언트 코드가 출현 - 높은 수준의 무결성을 지켜야
하는 시스템의 이상적인 해결책: 함수형 프로그래밍

8.1 골칫덩이 비동기 코드


- 함수 간에 일시적 의존 관계가 형성
- 어쩔 수 없이 콜백 피라미드의 늪에 빠짐
- 동기/비동기 코드의 호환되지 않는 조합

함수 간에 일시적 의존 관계가 형성

```
var student = null;
getJSON('/students', function(studentObjs) {
    students = studentObjs;
},
function (errorObj) {
    console.log(errorObj.message);
}
);

showStudents(students); //students가 null 임...
```

콜백 피라미드



```
pan.pourWater(function() {  
  range.bringToBoil(function() {  
    range.lowerHeat(function() {  
      pan.addRice(function() {  
        setTimeout(function() {  
          range.turnOff();  
          serve();  
        }, 15 * 60 * 1000);  
      });  
    });  
  });  
});
```

pyramid of doom

mozilla

연속체 전달 스타일

```
const showStudentGrades = R.curry(function(student, grades) {
  appendData(student, average(grades));
});

const handleError = error => console.log(error.message);

const processStudent = function (student) {
  if (student.address.coutry === 'US') {
    getJSON(`/students/${student.ssn}/grades`,
            showStudentGrades(student), handleError);
  }
};

for (let i = 0; i < student.length; i++) {
  processStudent(students[i]);
}
```

함수형 프로그래밍의 특성

- 합성과 무인수 프로그래밍을 이용합니다.
- 중첩된 구조를 보다 선형적으로 흐르게 눌러 펍니다.
- 일시적 결합은 추상하기 때문에 개발자는 더 이상 신경 쓸 필요가 없습니다.
- 여러 콜백 대신, 단일 함수로 여러 처리 로직을 통합하여 코드 흐름을 원활하게 합니다

비동기 로직을 프라미스로 일급화

프라미스: 오래 걸리는 계산을 모나드로 감싸는 개념

```
Promise.of(<오래 걸리는 작업>).map(fun1).map(fun2); //->  
Promise(결과)
```

모나드와는 달리 오래 걸리는 계산이 끝날 때까지 기다렸다가 미리 매핑한 함수를 실행합니다.

상태는 항상 보류, 이룸, 버림, 귀결 중 하나.


```
var Scheduler = (function() {
  let delayedFn = _.bind(setTimeout, undefined, _, _);
  return {
    delay5: _.partial(delayedFn, _, 5000),
    delay10: _.partial(delayedFn, _, 10000),
    delay: _partial(delayedFn, _, _)
  };
})();

var promiseDemo = new Promise(function(resolve, reject) {
  Scheduler.delay5(function() {
    resolve('완료!');
  });
});

promiseDemo.then(function(status) {
  console.log('5초 후 상태: ' + status);
});
```