2.1.7 프로미스

자바스크립트와 노드에서는 주로 비동기를 접한다. 특히 이벤트 리스너를 사용할 때 콜백 함수를 자주 사용한다. ES6 부터는 자바스크립트와 노드의 API들이 콜백 대신 프로미스기반으로 재구성되며, 악명 높은 콜백지옥현상을 극복했다는 평가를 받고 있다. 프로미스는 반드시 알아두자.

프로미스는 다음과 같은 규칙이 있다.

1. 먼저 프로미스 객체를 생성해야 한다.

```
const condition = true;

const promise = new Promise((resolve, reject) => {
  if(condition){
    resolve('success');
  } else {
    reject('failed');
  }
});

promise
  .then((msg) => {
    // 성공 시
    console.log(msg)
  })
  .catch((error) => {
    // 실패 시
    console.log(error);
  })
  .finally(() => {
    console.log('무조건');
  });
```

`new Promise` 로 프로미스를 생성할 수 있으며, 그 내부에 resolve와 reject를 매개변수로 갖는 콜백 함수를 넣는다. 이렇게 만든 promise 변수에 then과 catch 메서드를 붙일 수 있다. 프로미스 내부에서 resolve가 호출되면 then이 실행되고, reject가 호출되면 catch가 실행된다. finally는 성공 / 실패에 관계없이 실행된다.

resolve와 reject에 넣어준 인수는 각각 then과 catch의 매개변수에서 받을 수 있다. 즉, 위 코드의 경우resolve('success')이라했으니, then의 매개변수 msg는 'success'가 된다 반대로 reject의 경우는 catch의 매개변수가 된다. error는 'failed'가 될 것이다.

프로미스는 쉽게 설명하여, 실행은 바로 하되 결과값은 나중에 받는 객체이다. 결과값은 then을 붙였을 때 받게 된다.

then이나 catch에서 다시 다른 then이나 catch를 붙일 수 있다. 이전 then의 return 값을 다음 then의 매개변수로 넘긴다. 단 여기서 다음 then에서 받으려면 new Promise를 리턴해야 한다는 것을 잊지말자.

```
promise
  .then((msg1) => {
    return new Promise((resolve, reject) => resolve(msg1));
  })
  .then((msg2) => {
    return new Promise((resolve, reject) => resolve(msg2));
  })
  .then((msg3) => {
    return new Promise((resolve, reject) => resolve(msg3));
  })
  ...
```

이것을 활용해서 콜백을 프로미스로 바꿀 수 있다.

다음은 콜백을 쓰는 패턴 중 하나이다.

```
const findAndSaveUser = (Users) => {
  Users.findOne({}, (err, user) => { // first callback
    if(err) return console.error(err);
    user.name = 'you';
    user.save((err) => { // second callback
      if(err) return console.error(err);
      Users.findOne({ gender: 'm' }, (err, user) => { // third callback
        ...
      })
    });
  })
}
```

콜백 함수가 세 번 중첩되있다. 콜백 함수가 나올 때 마다 코드의 depth는 깊어진다. 각 콜백 함수마다 에러도 따로 처리해줘야 한다. promise를 이용하여 위의 코드를 변경해 보자

```
const findAndSaveUser = (Users) => {
  Users.findOne({})
    .then((user) => {
      user.name = 'you';
      return user.save(); // 이는 프로미스 객체를 리턴해야 한다.
    })
```

```
    .then((user) => {
      user.name = 'you';
      return User.findOne({ gender: 'm' }) // 마찬가지로 프로미스 객체를 리턴하는 함수여야 한다.
    })
    .then((user) => {
      ...
    })
    .catch((err) => {
      console.log(err);
    })
}
```

코드의 깊이가 세 단계 이상 깊어지지 않는다. 위 코드엣 ㅓthen 메서드들은 순차적으로 실행된다. 콜백에 매번 따로 처리해야 햇던 에러도 마지막 catch에서 한 번에 처리할 수 있다. 하지만 모든 콜백 함수를 위와 같이 바꿀 수 있는 것은 아니다. 메서드가 프로미스 방식을 지원해야한다. 다시 말해, 위의 예시에서 user.save와 User.findOne이 promise객체를 리턴해야 한다는 의미이다.

프로미스 여러 개를 한 번에 실행할 수 있는 방법이 있다. 기존의 콜백 팬턴이었다면 콜백을 여러 번 중첩해서 사용해야 할 것이나, 하지만 `Promise.all` 을 활용하면 간단히 할 수 있다.

```
const promise1 = Promise.resolve('success 1');
const promise2 = Promise.resolve('success 2');

Promise.all([promise1, promsie2])
  .then((result) => {
    console.log(result);
  })
  .catch((error) => {
    console.log(error);
  });
```

Promise.resolve는 즉시 resolve하는 프로미스를 만드는 방법이다. 비슷한 것으로 즉시 reject를 하는 Promise.reject도 있다. 프로미스가 여러 개 있을 때 `Promise.all에 넣으면 모두 resolve될 때 까지 기다렸다가` then으로 넘어간다. result 매개변수에 각각의 프로미스 결괏값이 배열로 들어 있습니다. Promise 중 하나라도 reject가 된다면 바로 catch로 넘어간다.