# 2.1.8 async / await

promise가 콜백 지옥을 해결해줬지만 여전히 코드는 장황하다. then과 catch가 계속 반복되기 때문이다.

async / await 문법은 프로미스를 사용한 코드를 한 번 더 깔끔하게 줄여줄 수 있다.

```
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);
    })
}
```

앞서 프로미스 장에서 봤던 예제이다. 콜백과 다르게 코드의 깊이가 깊어지진 않지만, 여전히 코드가 길다.

async / await 문법을 사용하면 다음과 같이 바꿀 수 있다.

```
const findAndSaveUser = async (User) => {
  let user = await User.findOne({});
  user.name = 'you';
  user = await user.save();
  user = await User.findOne({ gender: 'm' });
}
```

함수 키워드 앞에 async를 붙이고, promise 객체 앞에 await를 붙인다. 이제 함수는 해당 프로미스가 resolve될 때 까지 기다린 뒤 다음 로직으로 넘어 간다.

해당 문법과 예외처리를 함께하는 패턴은 `try - catch` 와 함께 쓰는 것이다.

```
const findAndSaveUser = async (User) => {
  try{
    let user = await User.findOne({});
    user.name = 'you';
```

```
    user = await user.save();
    user = await User.findOne({ gender: 'm' });
  } catch(err) {
    console.error(err);
  }
}
```

for문과 async / await을 같이 써서 프로미스를 순차적으로 실행할 수 있다. for문과 함께 쓰는 것은 노드 10 버전부터 지원하는 es2018부터이다.

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

(async () => {
  for await (promise of [promise1, promise2]) {
    console.log(promise);
  }
})();
```

위 코드는 for await of 문을 사용해서 프로미스 배열을 순회하는 코드이다. async 함수의 반환값은 항상 Promise로 감싸진다. 따라서 실행 후 then을 붙이거나 또 다른 async 함수 안에서 await을 붙여서 처리할 수 있다.

혼용해서 사용하면 아래와 같이 할 수도 있겠다.

```
const findAndSaveUser = async(User) {
  /* define */
}

// 다른 곳에서 호출

// method 1
findAndUser(Users).then((res) => {
  /* do something */
})

//method 2
async function other() {
  const result = await findAndSaveUser(User);
}
```