Chap46 제너레이터와 async/awit

▼ 제너레이터란?

제너레이터 : 코드 블록의 실행을 일시 중지 했다가 필요한 시점에 재개할 수 있는 특수한 함수

▼ 제너레이터와 일반 함수의 차이

1. **제너레이터 함수는 함수 호출자에게 함수 실행의 제어권을 양도할 수 있다.**

   일반 함수를 호출하면 제어권이 함수에게 넘어가고 함수 코드를 일괄 실행한다.

2. **제너레이터 함수는 함수 호출자와 함수의 상태를 주고받을 수 있다.**

   일반 함수를 호출하면 매개변수를 통해 함수 외부에서 값을 주입받고 함수 코드를 일괄 실행하여 결과값을 외부로 반환한다. **즉, 함수가 실행되고 있는 동안에는 함수 외부에서 함수 내부로 값을 전달하여 함수의 상태를 변경할 수 없다.**

3. **제너레이터 함수를 호출하면 제너레이터 객체를 반환한다.**

   일반함수를 호출하면 함수 코드를 일괄 실행하고 값을 반환하지만 제너레이터는 함수를 호출하면 함수 코드를 실행하는 것이 아니라 이터러블이면서 동시에 이터레이터인 제너레이터 객체를 반환한다.

▼ 제너레이터 함수의 정의

- **function* 키워드로 선언.** 하나 이상의 yield 표현식을 포함

```
//제너레이터 함수 선언문
function* genDecFunc(){
 yield 1;
}

//제너레이터 함수 표현식
const genExpFunc=function*(){
 yield 1;
```

```
}

//제너레이터 메서드
const obj={
 * genObjMethod(){
  yield 1;
 }
};

//제너레이터 클래스 메서드
class MyClass{
 *genClsMehod(){
  yield 1;
 }
}
```

- 애스터리스크(*)의 위치는 function 키워드와 함수 이름 사이라면 어디든지 상관없다.
- **화살표 함수로 정의할 수 없다.**
- 제너레이터 함수는 **new 연산자와 함께 생성자 함수로 호출할 수 없다.**

▼ 제너레이터 객체

제너레이터 함수를 호출하면 일반 함수처럼 함수 코드 블록을 실행하는 것이 아니라 제너레이터 객체를 생성해 반환한다. 제너레이터 함수가 반환한 **제너레이터 객체는 이터러블이면서 동시에 이터레이터다.**

→ Symbol.iterator 메서드 상속받음. 따라서 next메서드 소유

→ 추가로 return, throw 메서드를 갖는다.

- next 메서드를 호출하면 제너레이터 함수의 yield 표현식까지 블록을 실행하고 yield된 값을 value 프로퍼티 값으로, flase를 done프로퍼티 값으로 갖는 이터레이터 리절트 객체를 반환
- return 메서드를 호출하면 인수로 전달받은 값을 value프로퍼티 값으로, true를 done 프로퍼티 값으로 갖는 이터레이터 리절트 객체를 반환
- throw메서드를 호출하면 인수로 전달받은 에러를 발생시키고 undefined를 value 프로퍼티 값으로, true를 done프로퍼티 값으로 갖는 이터레이터 리절트 객체를 반환한다.

```
// 제너레이터 함수
function* getFunc(){
 try{
    yield 1;
    yield 2;
    yield 3;
}catch(e){
 console.error(e);
}
}


// 제너레이터 함수를 호출하면 제너레이터 객체를 반환한다.
const generator = getFunc();

// 제너레이터 객체는 이터러블이면서 동시에 이터레이터다.
// 이터러블은 Symbol.iterator 메서드를 직접 구형하거나 프로토타입 체
console.log(Symbol.iterator in generator); // true

// 이터레이터는 next 메서드를 갖는다.
console.log('next' in generator); //true

console.log(generator.next()); //{value:1,done:false}
console.log(generator.return('End')); // {value:"End!", dor
console.log(generator.throw('Error!')); //{value:undefined,
```

▼ 제너레이터의 일시 중지와 재개

제너레이터는 yield 키워드와 next 메서드를 통해 실행을 일시 중지했다가 필요한 시점에 다시 재개할 수 있다.

즉 next 메서드와 yield 표현식을 통해 함수 호출자와 함수의 상태를 주고받을 수 있다.

함수 호출자는 next 메서드를 통해 yield 표현식까지 함수를 실행시켜 제너레이터 객체가 관리하는 상태(yield된 값)를 꺼내올 수 있고, next메서드에 인수를 전달해서 제너레이터 객체에 상태(yield표현식을 할당받는 변수)를 밀어넣을 수 있다.

이처럼 제너레이터의 특성을 활용하면 비동기 처리를 동기 처리처럼 구현할 수 있다.

▼ 제너레이터의 활용

▼ 이터러블의 구현

제너레이터 함수를 사용하면 이터레이션 프로토콜을 준수해 이터러블을 생성하는 방식보다 간단히 이터러블을 구현할 수 있다.

▼ 비동기 처리

제너레이터 함수는 next메서드와 yield표현식을 통해 프로미스를 사용한 비동기 처리처럼 구현할 수 있다.

즉, then/catch/finally 없이 비동기 처리 결과를 반환하도록 구현할 수 있다.

▼ async/await

제너레이터보다 가독성 좋게 비동기 처리를 동기처리처럼 동작하도록 구현할 수 있다.

프로미스 기반으로 동작한다.

▼ async 함수

- 언제나 프로미스 반환
- 프로미스 반환자히자 않더라도 암묵적으로 반환값을 resolve하는 프로미스를 반환
- 클래스의 constructor메서드는 async 메서드가 될 수 없다. (클래스의 constructor메서드는 인스턴스를 반환해야하지만 async 함수는 언제나 프로미스를 반환해야 한다.)

▼ await키워드

- 프로미스가 settled 상태가 될 때까지 대기하다가 settled 상태가 되면 프로미스가 resolve한 처리 결과를 반환한다.

- async/ await에서 에러처리는 try...catch문을 사용할 수 있다.
- async함수 내에서 catch문을 사용해서 에러 처리를 하지 않으면 asycn함수는 발생한 에러를 reject하는 프로미스를 반환한다.