Chap45 프로미스

▼ 비동기 처리를 위한 콜백 패턴의 단점

▼ 콜백 헬

비동기 함수는 비동기 처리 결과를 외부에 반환할 수 없고, 상위 스코프의 변수에 할당할 수도 없다. 따라서 비동기 함수의 처리 결과(서버의 응답 등)에 대한 후속 처리는 비동기 함수 내부에서 수행해야 한다.

이때 비동기 함수를 범용적으로 사용하기 위해 비동기 함수에 비동기 처리 결과에 대한 **후속처리를 수행하는 콜백함수를 전달하는 것이 일반적**이다.

필요에 따라 비동기 처리가 성공하면 호출될 콜백 함수와 비동기 처리가 실패하면 호출될 콜백함수를 전달할 수 있다.

이처럼 콜백함수를 통해 비동기 처리 결과에 대한 후속 처리를 수행하는 비동기 함수가 비동기 처리 결과를 가지고 또다시 비동기 함수를 호출해야 한다면 **콜백 함수 호출이 중첩되어 복잡도가 높아지는 현상인 콜백 헬이 발생**이할 것이다.

콜백 헬은 가독성을 나쁘게 하여 실수를 유발하는 원인이 된다.

▼ 에러 처리의 한계

비동기 처리를 위한 콜백 패턴의 문제점 중에서 가장 심각한 것은 에러 처리이다.

```
try{
 setTimeout(()=>{throw new Error('Error!');},1000);
 }catch (e){
  console.log('캐치한 에러',e);
 }
```

▼ tr...catch...finally문

일단 try코드블록 실행된다. 만약 에러가 발생하면 catch문의 error변수에 전달되고 catch문 실행. finally문은 에러 발새 유무와 상관없이 무조건 한번 실행

setTimeout함수 콜백함수가 1초 후에 실행되도록 타이머 설정. 이후 콜백 함수는 에러를 발생.

하지만 catch문에는 캐치x

why?

setTimeout 호출되면 실행 컨텍스트가 생성되어 콜 스택에 푸시되어 함수 실행됨. 근데 setTimeout은 비동기 함수니깐 콜백 함수가 호출되는 것 기다리지 않고 즉시 종료되어 콜 스택에서 제거됨. 이후 타이머 만료되면 setTimeout함수의 콜백함수는 태스크 큐로 푸시되고 콜 스택이 비어졌을 때 이벤트 루프에 의해 콜 스택으로 푸시되어 실행.

즉, setTimeout함수의 콜백 함수가 실행될 때 이미 setTimeout 함수는 콜스택에서 제거된다.**이것은 setTimeout함수의 콜백 함수를 호출한 것이 setTimeout함수가 아니라는 것을 의미. 따라서 에러는 catch블록에서 캐치되지 않는다.**

이러한 예시로 인해 에러처리가 곤란하다는 것을 알 수 있다.

▼ 프로미스의 생성

```
const promise = new Promise((resolve, reject) => {
    // Promise 함수의 콜백 함수 내부에서 비동기 처리를 수행한다.
    if (/* 비동기 처리 성공 */) {
        resolve('result');
    } else { /* 비동기 처리 실패 */
        reject('failure reason');
    }
});
```

- Promise생성자 함수를 new연산자와 함께 호출하면 프로피스 객체가 생성됨
- 비동기 처리를 수행할 콜백함수를 인수로 전달받는데 이 콜백 함수는 resolve와 reject 함수를 인수로 받는다.
    - 성공시 resolve 함수 호출
    - 실패시 reject 함수 호출
- 프로미스는 현재 비동기 처리가 어떻게 진행되고 있는지 나타내는 상태 정보를 갖는다.

- pending : 비동기 처리가 아직 수행되지 않은 상태를 의미하고 프로미스가 생성된 직후 기본 상태 일때가 상태 변경 조건이다.
- fulfilled : 비동기 처리가 수행된 상태(성공)를 의미. resolve함수 호출하면 상태를 변경한다.
- rejected : 비동기 처리가 수행된 상태(실패)르 의미. reject함수 호출하면 상태를 변경

→ 즉, 비동기 처리가 실패하면 프로미스는 pending에서 rejected로 상태를 변화합니다. 그리고 비동기 처리 결과인 Error객체를 값으로 갖게 됩니다.

- **프로미스는 비동기 처리 상태와 처리 결과를 관리하는 객체이다.**

▼ 프로미스의 후속 처리 메서드

프로미스의 비동기 처리 상태가 변화하면 이에 따른 후속 처리를 해야한다. 따라서 프로미스는 후속 처리 메서드 then, catch, finally를 제공

모든 후속 처리 메서드는 프로미스를 반환하며, 비동기로 동작한다.

▼ Promise.prototype.then

- 두 개의 콜백 함수를 인수로 전달받는다.
  - 첫 번째 콜백 함수는 프로미스가 fulfilled상태가 되면 호출된다. 이때 콜백 함수는 프로미스의 비동기 처리 결과를 인수로 전달받는다.
  - 두 번째 콜백 함수는 프로미스가 rejected 상태가 되면 호출된다. 이때 콜백 함수는 프로미스의 에러를 인수로 전달받는다.
- then은 언제나 프로미스를 반환한다. 만약 프로미스가 안니 값을 반환하면 그 값을 암묵적으로 resolve 또는 reject하여 프로미스를 생성해 반환한다.

▼ Promise.prototype.catch

- 한 개의 콜백 함수를 인수로 전달받는다.
  - catch메서드의 콜백 함수는 프로미스가 rejected상태인 경우만 호출된다.
- catch는 then가 마찬가지로 언제나 프로미스를 반환한다.

▼ Promise.prototype.finally

- 한 개의 콜백 함수를 인수로 전달받는다.
  - 프로미스의 성공 또는 실패와 상관없이 무조건 한 번 호출된다.
  - 프로미스의 상태와 상관없이 공통적으로 수행해야 할 처리 내용이 있을 때 유용

- 언제나 프로미스를 반환한다.

▼ 프로미스의 에러 처리

catch 메서드를 모든 then 메서드를 호출한 이후에 호출하면 then 메서드 내부에서 발생한 에러까지 모두 캐치할 수 있다.

▼ 프로미스 체이닝

then, catch, finally 후속 처리 메서드는 언제나 **프로미스를 반환하므로 연속적으로 호출** 할 수 있다. (=**프로미스 체이닝**)

즉 프로미스는 프로미스 체이닝의 통해 비동기 처리 결과를 전달받아 후속 처리를 하므로 비동기 처리를 위한 콜백 패턴에서 발생하던 콜백 헬이 발생하지 않는다.

▼ 프로미스의 정적 메서드

Promise는 주로 생성자 함수로 사용되지만 함수도 객체이므로 메서드를 가질 수 있다.

▼ Promise.resolve / Promise.reject

- 두 메서드는 이미 존재하는 값을 래핑하여 프로미스를 생성하기 위해 사용
- Promise.resolve 메서드는 인수로 전달받는 값을 resolve하는 프로미스를 생성
- Promise.reject 메서드는 인수로 전달받는 값을 reject하는 프로미스를 생성

▼ Promise.all

- 여러 개의 비동기 처리를 모두 병렬 처리할 때 사용
- 모든 프로미스가 fulfilled 상태가되는 것을 기다림

▼ Promise.race

- 프로미스를 요소로 갖는 배열 등의 이터러블을 인수로 전달받음
- Promise.all메서드처럼 모든 프로미스가 fulfilled 상태가되는 것을 기다리는 것이 아니라 가장 먼저fulfilled상태가 된 프로미스의 처리 결과를 resolve하는 새로운 프로미스를 반환
- 프로미스가 reject상태가되면 Promise.all메서드와 동일하게 처리된다.

▼ Promise.allSettled

- 프로미스를 요소로 갖는 배열 등의 이터러블을 인수로 전달받음.
- 전달받은 프로미스가 모두 settled 상태가 되면 처리 결과를 배열로 반환

- 반환한 배열에는 fulfilled 또는 rejected 상태와는 상관없이 모든 프로미스의 처리 결과가 담겨있다.

▼ 마이태로태스크 큐

```
setTimeout(() => console.log(1), 0);

Promise.resolve()
  .then(() => console.log(2))
  .then(() => console.log(3));
```

프로미스의 후속 처리 메서드도 비동기로 동작하므로 1→2→3의 순으로 출력될 것 처럼 보이지만 2→3→1로 출력된다.

왜?

프로미스의 후속 처리 베서드의 콜백 함수는 태스크 큐가 아니라 마이크로태스크 큐에 저장되기 때문!

마이크로태스크큐

- 프로미스의 후속처리메서드의 콜백 함수가 일시 저장되는 곳.
- 태스크 큐와 별도의 큐다. 태스크큐보다 우선순위가 높다.

▼ fetch

fertch 함수는 HTTP 응답을 나타내는 Response 객체를 래핑한 Promise객체를 반환한다.

fetch 함수가 반환하는 프로미스는 기보적으로 404 Not Found나 500 Internal Server Error와 같은 HTTP에러가 발생해도 에러를 reject하지 않고 불리언 타입의 ok 상태를 false로설정한 Response객체를 reoslve한다. 오프라인 드의 네트워크 장애나 CORS에러에 의해 요청이 완료되지 못한 경어에만 프로미스를 reject한다.

따라서 **fetch함수를 사용할 때는 다음과 같이 fetch함수가 반환한 프로미스가 resolve한 불리언 타입의 ok상태를 확인해 명시적으로 에러를 처리할 필요**가 있다.