45장. 프로미스

45.1 비동기 처리를 위한 콜백 패턴의 단점

▼ 비동기 함수를 호출했을 때 완료 순서

비동기 함수 내부의 비동기로 동작하는 코드는 비동기 함수가 종료된 이후에 완료

▼ 비동기 처리 결과

비동기 함수는 비동기 처리 결과를 외부에 반환할 수 없고, 상위 스코프의 변수에 할당할 수 없음

비동기 함수에 비동기 처리 결과에 대한 후속 처리를 수행하는 콜백 함수를 전달하는 것이 일반

▼ 콜백 헬이란?

콜백 함수 호출이 중첩되어 복잡도가 높아지는 현상

→ 콜백 헬은 가독성을 나쁘게 하고 실수를 유발하는 원인

▼ 에러 전파 방향

호출자 방향으로 전파

45.2 프로미스의 생성

▼ 프로미스 생성

new 연산자와 함께 호출하면 프로미스 생성 가능

▼ 프로미스 상태 결정

비동기 처리 성공: resolve 함수를 호출해 프로미스를 fulfilled 상태로 변경

비동기 처리 실패: reject 함수를 호출해 프로미스를 rejected 상태로 변경

resolve 또는 reject 함수를 호출하는 것으로 결정

즉, 프로미스는 비동기 처리 상태와 처리 결과를 관리하는 객체

45.3 프로미스의 후속 처리 메서드

▼ then 메서드

두 개의 콜백 함수를 인수로 전달 받음

→ 첫 번째 콜백 함수는 비동기 처리가 성공했을 때 호출되는 성공 처리 콜백 함수

두 번째 콜백 함수는 비동기 처리가 실패했을 때 호출되는 실패 처리 콜백 함수

then 메서드는 언제나 프로미스를 반환

⇒ 프로미스 반환 시 그 프로미스를 그대로 반환

⇒ 프로미스가 아닌 값을 반환 시 암묵적으로 resolve 또는 reject하여 프로미스를 생성해 반환

▼ catch 메서드

한 개의 콜백 함수를 인수로 전달 받음

→ 콜백 함수는 프로미스가 rejected 상태인 경우만 호출

catch 메서드는 then 메서드처럼 언제나 프로미스 반환

▼ finally 메서드

한 개의 콜백 함수를 인수로 전달 받음

프로미스의 성공(fulfilled) 또는 실패(rejected)와 상관없이 무조건 한 번 호출

→ finally 메서드는 프로미스 상태와 상관없이 공통적으로 수행해야 할 처리 내용이 있을 때 유용

finally 메서드도 then/catch 메서드와 마찬가지로 언제나 프로미스를 반환

45.4 프로미스의 에러 처리

▼ 가장 좋은 에러 처리 방법

catch 메서드를 모든 then 메서드를 호출한 이후에 호출하면 비동기 처리에서 발생한 에러뿐만 아니라 then 메서드 내부에서 발생한 에러까지 모두 캐치 가능

⇒ 따라서, 에러 처리는 then 메서드보다 catch 메서드에서 하는 것을 권장

45.5 프로미스 체이닝

▼ 프로미스 체이닝이란?

then, catch, finally 후속 처리 메서드는 언제나 프로미스를 반환하므로 연속적으로 호출 가능

45.6 프로미스의 정적 메서드

▼ Promise.resolve / Promise.reject

이미 존재하는 값을 래핑하여 프로미스를 생성하기 위해 사용

▼ Promise.all

여러 개의 비동기 처리를 모두 병렬 처리할 때 사용

▼ Promise.race

프로미스를 요소로 갖는 배열 등의 이터러블을 인수로 받음

▼ Promise.allSettled

프로미스를 요소로 갖는 배열 등의 이터러블을 인수로 전달 받음

45.7 마이크로태스크 큐

▼ 마이크로태스크 큐와 태스크 큐의 차이점

마이크로태스크 큐는 태스크 큐보다 우선순위가 높다

→ 마이크로태스크 큐가 비어있으면 태스크 큐에서 대기하고 있는 함수를 가져와 실행함

⇒ 프로미스의 후속 처리 메서드의 콜백 함수는 태스크 큐가 아니라 마이크로태스크 큐에 저장되기 때문

45.8 fetch

▼ fetch 함수란?

HTTP 요청 전송 기능을 제공하는 클라이언트 사이드 Web API

HTTP 요청을 전송할 URL과 HTTP 요처어 메서드, HTTP 요청 헤더, 페이로드 등을 설정한 객체를 전달

HTTP 응답을 나타내는 Response 객체를 래핑한 Promise 객체를 반환