day57-promise

태그	
날짜	@2022년 12월 20일

promise가 왜 필요한다.

프로미스는 주로 서버에서 받아온 데이터를 화면에 표시할 때 사용한다. 일반적으로 웹 애플리케이션을 구현할 때 서버에서 데이터를 요청하고 받아오기 위해 아래와 같은 API를 사용한다.

```js
$.get('url 주소/products/1', function(response) {
  // ...
});
```

위 API가 실행되면 서버에다가 '데이터 하나 보내주세요' 라는 요청을 보낸다. 그런데 여기서 데이터를 받아오기도 전에 마치 데이터를 다 받아온 것마냥 화면에서 데이터를 표시하려고 하면 오류가 발생하거나 빈 화면이 뜬다.

이와 같은 문제점을 해결하기 위한 방법 중 하나가 프로미스이다.

프로미스의 3가지 상태(states)

프로미스를 사용할 때 알아야하는 가장 기본적인 개념이 바로 프로미스의 상태이다.

여기서 말하는 상태란 프로미스의 처리 과정을 의미한다.

new Promise()로 프로미스를 생성하고 종료될 때까지 3가지 상태를 갖는다.

- Pending(대기) : 비동기 처리 로직이 아직 완료되지 않은 상태
- Fulfilled(이행) : 비동기 처리가 완료되어 프로미스가 결과 값을 반환해준 상태
- Rejected(실패) : 비동기 처리가 실패하거나 오류가 발생한 상태

Pending(대기)

new Promise() 메서드를 호출하면 대기(Pending) 상태가 된다.

```
new Promise();
```

new Promise() 메서드를 호출할 때 콜백 함수를 선언할 수 있고, 콜백 함수의 인자는 resolve, reject이다.

```
new Promise(function(resolve, reject){
 //...
});
```

Fulfilled(이행)

콜백 함수의 인자 resolve를 아래와 같이 실행하면 이행(Fulfilled) 상태가 된다.

```
new Promise(function(resolve, reject) {
  resolve();
});
```

그리고 이행 상태가 되면 아래와 같이 then()을 이용하여 처리 결과 값을 받을 수 있다.

```
function getData() {
  return new Promise(function(resolve, reject) {
    var data = 100;
    resolve(data);
  });
}

// resolve()의 결과 값 data를 resolvedData로 받음
getData().then(function(resolvedData) {
  console.log(resolvedData); // 100
});
```

※ 프로미스의 '이행'상태를 다르게 표현하면 '완료'이다.

Rejected(실패)

new Promise()로 프로미스 객체를 생성하면 콜백 함수 인자로 resolve와 reject를 사용할 수 있다. 여기서 reject를 아래와 같이 호출하면 실패(Rejected) 상태가 된다.

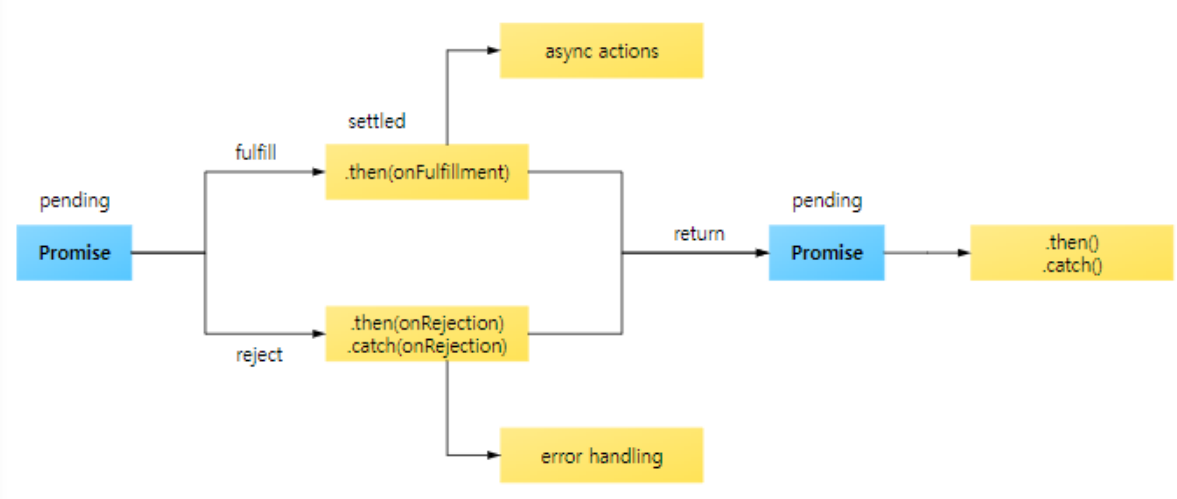
```
new Promise(function(resolve, reject) {
reject();
});
```

그리고, 실패 상태가 되면 실패한 이유(실패 처리의 결과 값)를 catch()로 받을 수 있다.

```
function getData() {
  return new Promise(function(resolve, reject) {
    reject(new Error("Request is failed"));
  });
}

// reject()의 결과 값 Error를 err에 받음
getData().then().catch(function(err) {
  console.log(err); // Error: Request is failed
});
```

- 프로미스 처리 흐름



프로미스의 에러 처리 방법

1. then() 의 두 번째 인자로 에러를 처리하는 방법

```
getData().then(
  handleSuccess,
  handleError
);
```

2. catch()를 이용하는 방법

```
getData().then().catch();
```

위 2가지 방법 모두 프로미스의 reject() 메서드가 호출되어 실패 상태가 된 경우에 실행 된다. 간단하게 말해서 프로미스의 로직이 정상적으로 돌아가지 않는 경우 호출되는 것이다.

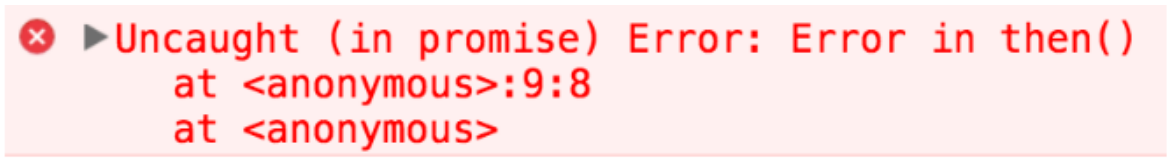
프로미스 에러 처리는 가급적 catch()를 사용

then()을 사용하는 방법도 있지만 가급적 catch()로 에러를 처리하는게 더 효율적이다.

```
// then()의 두 번째 인자로는 감지하지 못하는 오류
function getData() {
  return new Promise(function(resolve, reject) {
    resolve('hi');
  });
}

getData().then(function(result) {
  console.log(result);
  throw new Error("Error in then()"); // Uncaught (in promise) Error: Error in then()
}, function(err) {
  console.log('then error : ', err);
});
```

getDate() 함수의 프로미스에서 resolve() 메서드를 호출하여 정상적으로 로직을 처리했지만, then()의 첫 번째 콜백 함수 내부에서 오류가 나는 경우 오류를 제대로 잡아내지 못한다.



하지만 똑같은 오류를 catch()로 처리하면 다른 결과가 나온다.

```
// catch()로 오류를 감지하는 코드
function getData() {
  return new Promise(function(resolve, reject) {
    resolve('hi');
```

```
  });
}

getData().then(function(result) {
  console.log(result); // hi
  throw new Error("Error in then()");
}).catch(function(err) {
  console.log('then error : ', err); // then error :  Error: Error in then()
});
```

```
then error :  Error: Error in then()
    at <anonymous>:9:8
    at <anonymous>
```

발생 에러를 성공적으로 콘솔에 출력한 모습

따라서, 더 많은 예외 처리 상황을 위해 프로미스의 끝에 가급적 catch()를 붙히는게 좋다.

출처 : captain pangyo