day56-callback

태그	
날짜	@2022년 12월 19일

비동기 처리란

자바스크립트에서 특정 코드의 연산이 끝날 때까지 코드의 실행을 멈추지 않고 다음 코드를 먼저 실행하는 자바스크립트의 특성을 의미한다.

비동기 처리의 첫 번째 사례

비동기 처리의 가장 흔한 사례는 제이쿼리의 ajax이다.

제이쿼리로 실제 웹 서비스를 개발할 때 ajax 통신을 빼놓을 수가 없다. 보통 화면에 표시할 이미지나 데이터를 서버에서 불러와 표시해야 하는데 이때 ajax 통신으로 해당 데이터를 서버로부터 사져올수 있기 때문이다.

```
function getData() {
  var tableData;
  $.get('https://domain.com/products/1', function(response) {
    tableData = response;
  });
  return tableData;
}

console.log(getData()); // undefined
```

여기서 `$.get()`이 ajax 통신을 하는 부분입니다. `https://domain.com` 에다가 HTTP GET 요청을 날려 1번 상품(product) 정보를 요청하는 코드죠. 좀 더 쉽게 말하면 지정된 URL에 '데이터를 하나 보내주세요' 라는 요청을 날리는 것과 같습니다.

그렇게 서버에서 받아온 데이터는 `response` 인자에 담깁니다. 그리고 `tableData = response;` 코드로 받아온 데이터를 `tableData`라는 변수에 저장합니다. 그럼 이제 이 getData()를 호출하면 어떻게 될까요? 받아온 데이터가 뭐든 일단 뭔가 찍혀야겠죠. 근데 결과는 맨 아래에서 보시는 것처럼 undefined입니다. 왜 그럴까요?

그 이유는 `$.get()`로 데이터를 요청하고 받아올 때까지 기다려주지 않고 다음 코드인 `return tableData;`를 실행했기 때문입니다. 따라서, getData()의 결과 값은 초기 값을 설정하지 않은 tableData의 값 undefined를 출력합니다.

이렇게 특정 로직의 실행이 끝날 때까지 기다려주지 않고 나머지 코드를 먼저 실행하는 것이 비동기 처리이다.

자바스크립트에서 비동기 처리가 필요한 이유를 생각해보면, 화면에서 서버로 데이터를 요청했을 때 서버가 언제 그 요청에 대한 응답을 줄지도 모르는데 마냥 다른 코드를 실행 안 하고 기다릴 순 없기 때문이다. 위에선 간단한 요청 1개만 보냈는데 만약 100개를 보낸다고 생각해보면 비동기 처리가 아니고 동기 처리라면 코드 실행하고 기다리고, 실행하고 기다리고, 아마 웹 애플리케이션을 실행하는데 수십분은 걸릴 것이다.

비동기 처리의 두번째 사례

또 다른 비동기 처리 사례는 setTimeout()이다. setTimeout() 은 web API 의 한 종류 이다. 코드를 바로 실행하지 않고 지정한 시간만큼 기다렸다가 로직을 실행한다.

```
// #1
console.log('Hello');
// #2
setTimeout(function() {
  console.log('Bye');
}, 3000);
// #3
console.log('Hello Again');
```

비동기 처리에 대한 이해가 없는 상태에서 위 코드를 보면 아마 다음과 같은 결과값이 나올 거라고 생각한다.

- ‘Hello’ 출력
- 3초 있다가 ‘Bye’ 출력
- ‘Hello Again’ 출력

그런데 실제 결과 값은 아래와 같이 나온다.

- ‘Hello’ 출력
- ‘Hello Again’ 출력
- 3초 있다가 ‘Bye’ 출력

setTimeout() 역시 비동기 방식으로 실행되기 때문에 3초를 기다렸다가 다음 코드를 수행하는 것이 아니라 일단 setTimeout()을 실행하고 나서 바로 다음 코드인 `console.log('Hello`

`Again');`
으로 넘어갔습니다. 따라서, ‘Hello’, ‘Hello Again’를 먼저 출력하고 3초가 지나면 ‘Bye’가 출력됩니다.

콜백 함수로 비동기 처리 방식의 문제점 해결하기

앞에서 자바스크립트 비동기 처리 방식에 의해 야기될 수 있는 문제들을 살펴봤다. 이러한 문제들은 어떻게 해결할 수 있을까.

바로 콜백(CallBack) 함수를 이용하는 것이다. 앞에서 살펴본 ajax 통신 코드를 콜백 함수로 개선해보자.

```
function getData(callbackFunc) {
  $.get('https://domain.com/products/1', function(response) {
    callbackFunc(response); // 서버에서 받은 데이터 response를 callbackFunc() 함수에 넘겨줌
  });
}

getData(function(tableData) {
  console.log(tableData); // $.get()의 response 값이 tableData에 전달됨
});
```

이렇게 콜백 함수를 사용하면 특정 로직이 끝났을 때 원하는 동작을 실행시킬 수 있습니다

비유로 이해하는 콜백 함수 동작 방식

콜백 함수의 동작 방식은 일종의 식당 자리 예약과 같습니다. 일반적으로 맛집을 가면 사람이 많아 자리가 없습니다. 그래서 대기자 명단에 이름을 쓴 다음에 자리가 날 때까지 주변 식당을 돌아다니죠. 만약 식당에서 자리가 생기면 전화로 자리가 났다고 연락이 옵니다. 그 전화를 받는 시점이 여기서의 콜백 함수가 호출되는 시점과 같습니다. 손님 입장에서는 자리가 날 때까지 식당에서 기다리지 않고 근처 가게에서 잠깐 쇼핑을 할 수도 있고 아니면 다른 식당 자리를 알아볼 수도 있습니다.

자리가 났을 때만 연락이 오기 때문에 미리 가서 기다릴 필요도 없고, 직접 식당 안에 들어가서 자리가 비어 있는지 확인할 필요도 없습니다. 자리가 준비된 시점, 즉 데이터가 준비된 시점에서만 저희가 원하는 동작(자리에 앉는다, 특정 값을 출력한다 등)을 수행할 수 있습니다.

콜백 지옥 (Callback hell)

콜백 지옥은 비동기 처리 로직을 위해 콜백 함수를 연속해서 사용할 때 발생하는 문제이다.

```
$.get('url', function(response) {
  parseValue(response, function(id) {
    auth(id, function(result) {
      display(result, function(text) {
        console.log(text);
      });
    });
  });
});
```

웹 서비스를 개발하다 보면 서버에서 데이터를 받아와 화면에 표시하기까지 인코딩, 사용자 인증 등을 처리해야 하는 경우가 있습니다. 만약 이 모든 과정을 비동기로 처리해야 한다고 하면 위와 같이 콜백 안에 콜백을 계속 무는 형식으로 코딩을 하게 됩니다. 이러한 코드 구조는 가독성도 떨어지고 로직을 변경하기도 어렵습니다. 이와 같은 코드 구조를 콜백 지옥이라고 합니다.

콜백 지옥을 해결하는 방법

일반적으로 콜백 지옥을 해결하는 방법에는 Promise나 Async를 사용하는 방법이 있습니다. 만약 코딩 패턴으로만 콜백 지옥을 해결하려면 아래와 같이 각 콜백 함수를 분리해주면 됩니다.

```
function parseValueDone(id) {
  auth(id, authDone);
}
function authDone(result) {
  display(result, displayDone);
}
function displayDone(text) {
  console.log(text);
}
$.get('url', function(response) {
  parseValue(response, parseValueDone);
});
```