|  |  |  |
| --- | --- | --- |
| **[JavaScript]콜백함수(CallBack)** | 작성자 | 박민지 |
| 작성일 | 2021.04.14 |
|  |  |

* **콜백함수란?**

**콜백함수**는 간단하게 다른 함수에 **매개변수**로 넘겨준 함수를 말한다. 매개변수로 넘겨받은 함수는 일단 넘겨받고, 때가 되면 나중에 호출(called back)한다는 것이 콜백함수의 개념이다.

개발자는 단지 함수를 등록하기만 하고, **어떤 이벤트가 발생했거나 특정 시점에 도달 했을 때 시스템에서 호출되는 함수를** 말한다.

자바스크립트에서 함수가 일급객체이기 때문에 함수를 인자로 사용하는 콜백패턴을 사용 할 수 있다.

☞ **일급객체란**?

* 다른 객체들에 일반적으로 적용 가능한 연산을 지원하는 객체를 가리킨다.
* 일급객체가 되기 위한 조건

1. 변수나 데이터에 할당 할 수 있어야 한다.
2. 다른 함수를 인자로 전달 받는다.
3. 다른 함수의 결과로서 리턴 될 수 있다.

* **콜백함수의 예제1**

**function** returnName(callback) {  
 callback("jason");  
 console.log("만나서 반갑습니다.");  
}  
**function** sayHello(name) {  
 console.log("안녕하세요"+name+"씨");  
}  
returnName(sayHello)

**코드 해석**

**- returnName**을 보면 파리미터에 callback이라는 함수가 선언되어있다. **returnName**함수의   
파라미터로 받게되는 함수(여기선sayHello)는 “jason”이라는 string를 인자로 받게된다.

**- returnName**함수는 sayHello라는 함수를 파라미터로 가지고 호출되는데, **returnName**함수는 sayHello함수를 인자로 받아야 하기 때문에 **sayHello함수가 먼저 실행된** 뒤 **returnName**함수가 먼저 실행된다.

**출력 결과**

https://blog.kakaocdn.net/dn/d6ezDJ/btqDajawuOX/ksurS84ZuTzHrNSu1M1DSK/img.png

* **콜백함수의 예제2 \_ jQuery**

$( "#target" ).click(**function**() {  
 alert( "Handler for .click() called." );  
});

**코드 해석**

- 익명의 함수 click이라는 함수 내부에서 나중에 호출이 된다. 아무런 이름이 없지만 이 함수를 담고 있는 click이라는 함수를 언제든지 이 함수를 호출할 수 있다.

* **콜백함수의 예제3 \_ ajax**

$.ajax({  
 url : "URL",  
 type: "GET",  
 success: **function** () {  
 // 콜백 함수   
 },  
 error: **function** () {  
 // 콜백 함수   
 },  
 complete: **function** () {  
 // 콜백 함수   
 }   
})

**코드 해석**

- ajax가 성공했을 경우 success의 콜백함수가, 실패했을 경우 error의 콜백함수가, 완료됐을 경우 complete 콜백함수가 실행된다.

* **콜백함수 적용의 기본 원칙**

1. 익명의 함수를 사용

jquery에서 익명의 함수를 파라미터로 사용 한다. 이러한 방식이 가장 일반적인 패턴이다. 또 다른 보편적인 방법으로는 함수를 정의해 해당 함수의 이름을 파라미터로 넘기는 방식이다.

**var** allUserData = [];  
  
// 콘솔에 결과를 찍는 함수  
**function** logStuff (userData) {  
 **if** ( **typeof** userData === "string")  
 {  
 console.log(userData);  
 }  
 **else if** ( **typeof** userData === "object")  
 {  
 **for** (**var** item **in** userData) {  
 console.log(item + ": " + userData[item]);  
 }  
 }  
}  
  
// 두 개의 인자를 받아서 마지막에 콜백함수를 호출한다.  
**function** getInput (options, callback) {  
 allUserData.push (options);  
 callback (options);  
}  
  
// getInput 함수를 호출할 때 , 우리는 logStuf이라는 함수의 이름을 인자로 넘긴다.  
// 그래서 logStuff 은 콜백함수가 되어 getInput이라는 함수의 내부에서 동작을 할것이다.  
getInput ({name:"Rich", speciality:"JavaScript"}, logStuff);  
// name: Rich  
// speciality: JavaScript

1. 콜백함수로 파라미터 전달

콜백함수가 실행이 될 때는 그냥 일반 함수와 동일하게 동작을 한다. 그래서 우콜백함수에 파라메터를 전달할 수가 있다. 파라미터로 콜백함수를 감싸고있는 함수 내부의 어떠한 프로퍼티라도 파라메터로 전달할 수가 있다. 아래의 예제를 실행하면 options 파라메터를 콜백함수에 전달할 수 있다. 전역변수와 지역변수를 파라메터로 전달할 수 있다.

//전역변수  
**var** generalLastName = "Clinton";  
**function** getInput (options, callback) {  
 allUserData.push (options);  
   
// 전역변수를 콜백함수의 인자로 전달한다.  
 callback (generalLastName, options);  
}

1. 콜백함수가 실행된기 전에 함수임을 명확하게 하기

콜백함수가 인자로 전달되어 함수의 내부에서 실행이 될 때 전달받은 인자가 함수인자를 명확하게 정의하고 실행하는 것이 좋은 습관이다.

**function** getInput(options, callback) {  
 allUserData.push(options);  
  
// callback 이 함수 인지를 확인합니다.  
 **if** (**typeof** callback === "function") {  
  
 // callback 이 함수인지를 확인 했으니까 함수호출합니다.  
 callback(options);  
 }  
}

1. This를 사용한 메서드를 콜백으로 사용시 문제

콜백함수가 this객체를 사용하는 메서드인 경우에는 반드시 this객체의 컨택스트를 보호할 수 있도록 콜백함수를 수정해야 한다. 예를들면 전역함수에 인자로 콜백함수가 전달된 경우에는 this객체가 window객체를 가리키게 만들거나 또는 콜백함수를 감싸고 있는 메서드를 가리키게 해야한다.

let userData = {  
 signUp: '2021-04-14 15:00:00',  
 id: "minzzi",  
 name : 'Not Set',  
 setUserName: **function** (firstName, lastName) {  
  
 **this**.name = firstName + ' ' + lastName;  
 }  
}  
  
**function** getUserName(firstName, lastName, callback ) {  
 callback(firstName, lastName);  
}  
  
getUserName ("PARK", "MINJI", userData.setUserName);  
  
console.log ('1: ',userData.name);  
console.log ('2: ',window.name);

**출력 결과**

1: Not Set

2. PARK MINJI

1. Call과 apply를 통한 this보호

* call() : 첫 번째 인자로 this 객체 사용, 나머지 인자들은 , 로 구분.

**function** getUserName(firstName, lastName, callback, obj ) {  
 callback.call(obj, firstName, lastName);  
}  
  
getUserName ("PARK", "MINJI", userData.setUserName, userData);  
console.log (userData.name);

* apply() : 첫 번째 인자로 this 객체 사용, 나머지 인자들은 배열로 전달

**function** getUserName(firstName, lastName, callback, obj ) {  
 callback.apply(obj, [firstName, lastName]);  
}  
  
getUserName ("PARK", "MINJI", userData.setUserName, userData);  
console.log (userData.name);

**출력 결과(call, apply 동일)**

PARK MINJI

* **콜백함수는 클로저**

**function** callbackFunction (callback) {  
 callback();  
}  
**function** testFunction() {  
 **var** text = "callback function is closure";  
 callbackFunction(**function** () {  
 console.log(text);   
 });  
}

**코드 해석**

- 콜백함수가 실행되는 2번째 줄의 시점에 text변수는 존재하지 않는다. 하지만 콜백함수는 클로저이기 때문에 함수가 만들어진 환경을 기억하게 된다. 6번째 줄에서 함수가 만들어질 때의 환경인 4번째 줄의 text 변수의 값을 기억하고 있어, 로드로 출력하게 된 것이다.

* **콜백지옥**

비동기 호출이 자주 일어나는 프로그램의 경우 '콜백 지옥'이 발생한다. 함수의 매개변수로 넘겨지는 콜백 함수가 반복되어 코드의 들여쓰기 수준이 감당하기 힘들어질 정도로 깊어지는 현상이다.

* Promise의 return 사용하여 Promise Hell을 탈출할 수 있다.
* **콜백함수가 필요한 이유?**

**콜백함수**를 사용하는 이유는, 자바스크립트에서 비동기적 프로그래밍을 할 수 있기때문이다. 이 콜백함수는 자바스크립트에서 가장 오래된 **비동기적 매커니즘**이라고 한다.

☞ **비동기적 테크닉**?

* 싱글스레드의 멈춤을 방지한다. 즉, 블록킹을 방지하여 싱글스레드가 논블록킹으로 동작하게 한다.

☞ **비동기적 테크닉**을 사용하는 경우

1. 사용자 이벤트 처리
2. 네트워크 응답처리
3. 파일을 읽고 쓰는 등의 파일 시스템 작업
4. 의도적으로 시간지연을 사용하는 기능(알림 등)