람다 함수와 화살표 함수

소프트웨어학과
2021763013_김민영
2024.04.12(금)
웹프로그래밍응용

일반함수와 익명함수

+ 일반함수

```
function 함수명() {
    함수로직
}
```

- 일반함수와 익명함수의 차이

함수명의 유무

일반함수 : 함수명 O

익명함수 : 함수명 X

호이스팅(hoisting)

일반함수 : 호이스팅이 가능

익명함수 : 호이스팅이 불가능

+ 익명함수(Anonymous function)

```
function() {
    함수로직
}
```

익명함수의 특징

- 메모리를 차지하지 않아 한 번만 사용하고 쓰지 않을 함수에 대해서는 익명 함수를 사용한 다면, 불필요한 메모리를 줄일 수 있다.

- 메모리를 차지하지 않기 때문에 익명함수는 변수에 담아서 많이 사용한다.

```
const abc = function (){
    console.log('hi')
}
```

위와 같이 익명함수를 변수에 저장하는 방식을 리터럴 방식이라고 함.

일반함수와 익명함수

호이스팅

호이스팅은 자바스크립트에서 변수와 함수 선언이
코드 실행 전에 해당 범위의 최상단으로 끌어올려지는 것처럼 보이는 현상

호이스팅이 작동되는 원리를 3단계로 나누어본다면,
(1) 자바스크립트 엔진은 기본적으로 코드를 위에서부터 아래로 순서대로 읽는다.
(2) 함수를 사용하는 방법은 선언문 안에 작성된 코드를 호출문이 호출을 하는 방식이다.
호출문이 선언문보다 먼저 작성되면 자바스크립트 엔진은 호출할 함수를 찾지 못한다.
(3) 그러나 일반함수는 호이스팅을 지원하기 때문에 호출문이 선언문보다 먼저 선언되어도
작성된 코드 내에서 호출해야할 함수를 알아서 찾아오기 때문에 정상작동이 된다.
(+) 호이스팅을 지원하지 않는 익명함수는 오류가 발생한다.

```
hanpy1()

// 일반함수
function hanpy1(){
       console.log("hi");
}

hanpy1()
```

```
happy2()

const happy2 = function (){
     console.log('happy');
}

happy2()

// 첫번째 happy2()에서 에러 발생
// ReferenceError: Cannot access 'hanpy2' before initialization
```

람다함수(=화살표 함수)

람다 함수

- 보통 함수를 익명으로 선언하고, 함수를 다른 함수의 인자로 전달하거나 함수의 반환값으로 사용할 때 람다 함수라고 부른다.

"람다함수"와 "화살표 함수"는 기본적으로 동일한 개념을 가리킨다.
둘 다 익명 함수를 간결하게 표현하기 위한 방법으로 사용된다.
"람다 함수"는 함수형 프로그래밍에서 사용되는 용어로 더 일반적인 표현.
JavaScript에서는 주로 "화살표 함수"라고 부른다.

화살표 함수

- 화살표(=>)를 사용하여 간결하게 함수를 정의
- 함수명이 없는 익명함수로 메모리 관리에 효율적이다.

```
// 일반함수
function happy1(){
    console.log('hi');
}

// 화살표함수
const happy3 = () => {
    console.log('hi');
}
```

ES5 vs ES6

ES5 (기존의 함수 표현식)

```
var a = function () {
  return new Date()
};

var b = function (a) {
  return a * a
};

var c = function (a, b) {
  return a + b
};

var d = function (a, b) {
  console.log( a * b )
}
```

ES6 (화살표 함수)

```
let a = () => {
    return new Date()
};
let aa = () => new Date();

let b = (a) => {
    return a * a
};
let bb = a => a * a;

let c = (a, b) => {
    return a + b
};
let cc = (a, b) => a + b;

let d = (a, b) => {
    console.log( a * b )
};
```

- ES6(ECMAScript 2015)에서
  도입된 새로운 함수 선언 방식 중 하나
- function 키워드를 삭제
- 화살표(=>)를 사용하여 정의

매개변수 표현

매개변수가 하나뿐인 경우 괄호 생략 가능
매개변수가 없을 경우에는 괄호 필수

함수 몸체 표현

본문이 return [식 or 값] 뿐인 경우, { } 와 return 키워드 생략 가능
return 할 값이 객체인 경우에는 괄호 필수

람다함수(=화살표 함수)

일반함수

```
> var functionA = function(a) {
      return a *= 10;
  }
<- undefined
> console.log(functionA(10));
  100                                                    VM378:1
```

화살표함수로 변경

```
var function1 = a => a * 10;
undefined
console.log(function1(10));
100                                                      VM588:1
undefined
console.log(function1(100));
1000                                                     VM605:1
```

다양한 화살표 함수 표현

```
let sum = (a, b) => {
    let result = a + b;
    return result;
}
undefined
console.log(sum(3, 4));
7                                                        VM514:1
undefined
let sum2 = (a, b) => a + b;
undefined
console.log(sum2(3, 4));
7                                                        VM670:1
undefined
let sum3 = (a, b) => {
    return a + b;
}
undefined
console.log(sum(3, 4));
7                                                        VM893:1
```

화살표 함수에서의 this

실행 컨텍스트 생성시 this 바인딩을 하지 않음

일반함수

```
target = 'global happy';

function test(){
    console.log(this.target);
}

const object = {
    target: 'local happy',
    action: test
}

object.action();
```

결과 : "local happy"

target은 전역 변수로 local happy가 출력된다.

화살표 함수

```
target = 'global happy';

const test = () => {
    console.log(this.target);
}

const object = {
    target: 'local happy',
    action: test
}

object.action();
```

결과 : "global happy"

화살표 함수의 경우는 내부의 this는 선언한 시점에서 호출한다.

따라서 target 변수는 전역 설정한 값들이 들어오게 되는 것이다.

*이러한 화살표 함수의 this 특징 때문에 자바스크립트의 콜백 함수에서 유용히 쓰일 수도 있다

화살표 함수에서의 arguments

자바스크립트 함수의 arguments는 일반적인 함수가 호출될 때 전달된 인수들을 담고 있는 유사 배열 객체다.
화살표 함수는 일반 함수와는 다르게 모든 인수에 접근할 수 있게 해주는 유사 배열 객체 arguments를 지원하지 않는다.

일반함수

```
function argsFunc() {
  console.log(arguments);
}

argsFunc(1, 2, 3); // [1, 2, 3]
```

화살표 함수

```
let argsFunc = () => {
  console.log(arguments);
}

argsFunc(1, 2, 3); // ! Error
```

대신에 나머지 매개변수(rest parameter)라는 문법을 사용하여 인수들을 배열로 받을 수 있다.

```
let argsFunc = (...args) => {  // ... 나머지 매개변수
  console.log(args);
}

argsFunc(1, 2, 3); // [1, 2, 3]
```

화살표 함수에서의 생성자 함수

역시 화살표 함수는 this가 없기 때문에 new와 함께 실행할 수 없다. 따라서 화살표 함수는 new와 함께 호출할 수 없다.

일반함수

```
function User(name) {
  this.name = name;
}

let user = new User("Alice");
console.log(user.name); // Alice
```

화살표 함수

```
let User = (name) => {
  this.name = name;
}

let user = new User("Alice"); // TypeError: User is not a constructor
```

화살표 함수는 객체를 생성하는 용도로 사용할 수 없다. 그래서 화살표 함수는 보통 콜백 함수나 익명 함수로서 사용되는 편이다.

출처

https://velog.io/@yoon_han0/%EB%9E%8C%EB%8B%A4%EC%8B%9D%EC%9D%84-%EC%95%8C%EC%95%84%EB%B3%B4%EC%9E%90

https://zoeday.tistory.com/44

https://han-py.tistory.com/473

https://velog.io/@preace11/Javascript-%EC%9D%B5%EB%AA%85%ED%95%A8%EC%88%98-%EB%9E%8C%EB%8B%A4%EC%8B%9D-%ED%95%A8%EC%88%98-%EC%BD%9C%EB%B0%B1%ED%95%A8%EC%88%98

https://webclub.tistory.com/649

https://inpa.tistory.com/entry/JS-%F0%9F%93%9A-%EC%9E%90%EB%B0%94%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%8A%B8-%ED%99%94%EC%82%B4%ED%91%9C-%ED%95%A8%EC%88%98-%EC%A0%95%EB%A6%AC#%ED%99%94%EC%82%B4%ED%91%9C_%ED%95%A8%EC%88%98%EC%97%94_arguments_%EA%B0%80_%EC%97%86%EB%8B%A4