26장

ES6 함수의 추가기능

26.1 함수의 구분

- ES6 이전 모든 함수는 **생성자 함수**로도 호출 가능.

- 따라서 객체에 바인딩 된 **메소드**도 생성자 함수로 호출 가능.

ES6 함수의 구분	constructor	prototype	super	arguments
일반 함수(Normal)	○	○	X	○
메서드(Method)	X	X	○	○
화살표 함수(Arrow)	X	X	X	X

26.2 메서드

```
const obj = {
  x: 1,
  // foo는 메서드다.
  foo() { return this.x; },
  // bar에 바인딩된 함수는 메서드가 아닌 일반 함수다.
  bar: function() { return this.x; }
};

console.log(obj.foo()); // 1
console.log(obj.bar()); // 1
```

26.2 메서드

```
const derived = {
  __proto__: base,
  // sayHi는 ES6 메서드가 아니다.
  // 따라서 sayHi는 [[HomeObject]]를 갖지 않으므로 super 키워드를 사용할 수 없다.
  sayHi: function () {
    // SyntaxError: 'super' keyword unexpected here
    return `${super.sayHi()}. how are you doing?`;
  }
};
```

- ES6 메서드는 자신을 바인딩한 객체를 가르키는 [[HomeObject]] 를 갖는다..

26.3 화살표 함수

- Function 키워드 대신 화살표를 사용하여 함수 정의.

- 내부 동작도 기존 함수보다 더 간략하다.

ES6 함수의 구분	constructor	prototype	super	arguments
일반 함수(Normal)	○	○	X	○
메서드(Method)	X	X	○	○
화살표 함수(Arrow)	X	X	X	X

26.3 화살표 함수

- 화살표 함수는 noncunstructor이다.

```
const Foo = () => {};
// 화살표 함수는 생성자 함수로서 호출할 수 없다.
new Foo(); // TypeError: Foo is not a constructor
```

- 화살표 함수는 중복된 매개변수 이름을 선언할 수 없다.

```
const arrow = (a, a) => a + a;
// SyntaxError: Duplicate parameter name not allowed in this context
```

26.3 화살표 함수

- 함수 자체의 this, arguments, super, new.target을 갖지 않는다.

- 화살표 내부에서 this, arguments, super, new.target를 참조하면 **상위 스코프**에서 이를 찾게된다.

26.3 화살표 함수

```
class Prefixer {
  constructor(prefix) {
    this.prefix = prefix;
  }

  add(arr) {
    // add 메서드는 인수로 전달된 배열 arr을 순회하며 배열의 모든 요소에 prefix를 추가한다.
    // ①
    return arr.map(function (item) {
      return this.prefix + item; // ②
      // → TypeError: Cannot read property 'prefix' of undefined
    });
  }
}

const prefixer = new Prefixer('-webkit-');
console.log(prefixer.add(['transition', 'user-select']));
```

26.3 화살표 함수

```
class Prefixer {
  constructor(prefix) {
    this.prefix = prefix;
  }

  add(arr) {
    return arr.map(item => this.prefix + item);
  }
}

const prefixer = new Prefixer('-webkit-');
console.log(prefixer.add(['transition', 'user-select']));
// ['-webkit-transition', '-webkit-user-select']
```

26.3 화살표 함수

```
// Bad
const person = {
  name: 'Lee',
  sayHi: () => console.log(`Hi ${this.name}`)
};

// sayHi 프로퍼티에 할당된 화살표 함수 내부의 this는 상위 스코프인 전역의 this가 가리키는
// 전역 객체를 가리키므로 이 예제를 브라우저에서 실행하면 this.name은 빈 문자열을 갖는 window.name과 같다.
// 전역 객체 window에는 빌트인 프로퍼티 name이 존재한다.
person.sayHi(); // Hi
```

26.3 화살표 함수

```
// Bad
function Person(name) {
  this.name = name;
}

Person.prototype.sayHi = () => console.log(`Hi ${this.name}`);

const person = new Person('Lee');
// 이 예제를 브라우저에서 실행하면 this.name은 빈 문자열을 갖는 window.name과 같다.
person.sayHi(); // Hi
```

26.3 화살표 함수

1. this
2. super
3. arguments

모두 바인딩을 갖지 않고

상위 스코프를 참조한다.

26.4 Rest 파라미터

```
function foo(...rest) {
  // 매개변수 rest는 인수들의 목록을 배열로 전달받는 Rest 파라미터다.
  console.log(rest); // [ 1, 2, 3, 4, 5 ]
}

foo(1, 2, 3, 4, 5);
```

- Rest 파라미터는 함수에 전달되는 인수들의 목록을 배열로 받는다.

- Rest 파라미터는 함수 객체의 length에 영향을 주지 않는다.

26.4 Rest 파라미터

- Arguments 객체는 **유사배열 객체**.

- Rest 파라미터는 인수 목록을 배열로 직접 전달받음.

- 화살표 함수는 Arguments객체를 갖지 않음.

ES6 함수의 구분	constructor	prototype	super	arguments
일반 함수(Normal)	○	○	×	○
메서드(Method)	×	×	○	○
화살표 함수(Arrow)	×	×	×	×