02장 리액트 ES6 문법 엑기스

■ 학습 목표
- 이 장에서는 리액트에 필요한 자바스크립트 ES6의 주요 문법을 공부한다.

2.1 템플릿 문자열

■ ES5/ES6의 문자열 사용 방법 알아보기
- 04: 병합 연산자를 사용해 문자열과 문자열을 연결한다.
- 06: 문자열과 변수를 연결할 때도 병합 연산자를 사용한다.
- 07: 줄바꿈을 할 때는 이스케이스 시퀀스(\n)를 문자열에 포함시킨다.
- 12: 연산 결과를 괄호로 묶어 연산 구문을 먼저 실행한다.
- 17,20: 템플릿 문자열은 작은 따옴표(') 대신 백틱(`)으로 문자열을 표현한다.
- 26,27: 템플릿 문자열에 $기호를 사용하여 연산을 포함시킬 수도 있다.

[./src/02/02-2.js]

```
01  // ES5 문법
02  var string1 = '안녕하세요';
03  var string2 = '반갑습니다';
04  var greeting = string1 + ' ' + string2;
05  var product = { name: '도서', price: '4200원' };
06  var message = '제품' + product.name + '의 가격은' + product.price + '입니다';
07  var multiLine = '문자열1\n문자열2';
08  var value1 = 1;
09  var value2 = 2;
10  var boolValue = false;
11  var operator1 = '곱셈값은 ' + value1 * value2 + '입니다. ';
12  var operator2 = '불리언값은 ' + (boolValue ? '참' : '거짓') + '입니다. ';
13
14  //ES6 문법
15  var string1 = '안녕하세요';
16  var string2 = '반갑습니다';
17  var greeting = `${string1} ${string2}`;
18
19  var product = { name: '도서', price: '4200원' };
20  var message = `제품 ${product.name}의 가격은 ${product.price}입니다`;
21  var multiLine = `문자열1
22  문자열2`;
23  var value1 = 1;
24  var value2 = 2;
25  var boolValue = false;
26  var operator1 = `곱셈값은 ${value1 * value2}입니다.`;
27  var operator2 = `불리언값은 ${boolValue ? '참' : '거짓'}입니다.`;
```

2.2 전개 연산자

■ ES5/ES6의 전개 연산자 사용 방법 알아보기
- 04: 배열의 각 요소를 추출하여 새로운 배열을 만든다.
- 05: concat() 함수로 두 배열을 합친다.
- 09: || 연산자와 조합하면 추출할 배열 요소가 없을 때 기본값을 지정할 수 있다.
- 20: 두 배열 항목을 전개 연산자로 합친다.
- 25~27: 함수 인자 배열을 args 변수에 할당한다.

[./src/02/02-3.js]

```
01  // ES5 문법
02  var array1 = ['one', 'two'];
03  var array2 = ['three', 'four'];
04  var combined = [array1[0], array1[1], array2[0], array2[1]];
05  var combined = array1.concat(array2);
06  var combined = [].concat(array1, array2);
07  var first = array1[0];
08  var second = array1[1];
09  var three = array1[2] || 'empty';
10
11  function func() {
12    var args = Array.prototype.slice.call(this, arguments);
13    var first = args[0];
14    var others = args.slice(1);
15  }
16
17  // ES6 문법
18  var array1 = ['one', 'two'];
19  var array2 = ['three', 'four'];
20  var combined = [...array1, ...array2];
21  // combined = ['one', 'two', 'three', 'four'];
22  var [first, second, three = 'empty', ...others] = array1;
23  // first = 'one', second = 'two', three = 'empty', others = []
24
25  function func(...args) {
26    var [first, ...others] = args;
27  }
28
29  function func(first, ...others) {
30    var firstInES6 = first;
31    var othersInES6 = others;
32  }
33
34  // 올바르지 못한 예
35  // var wrongArr = ...array1;
```

2.3 가변 변수와 불변 변수

- 기존 자바스크립트 문법은 변수 선언에 var 키워드를 사용했지만 ES6에서는 값을 수정할 수 있는 가변 변수를 위한 let 키워드와, 값을 수정할 수 없는 불변 변수를 위한 const 키워드를 사용한다.
  - 13~21: 불변 변수로 정의된 배열이나 객체를 내장 함수로 수정하는 것을 암묵적으로 금지하여 무결성을 유지해야 한다. (개발자 스스로 불변 변수를 어떻게 관리할지 정해야 한다!)

[./src/02/02-4.js]

```
01  const num = 1;
02  num = 3; // 타입 에러가 발생합니다
03
04  const str = '문자';
05  str = '새 문자'; // 타입 에러가 발생합니다
06
07  const arr = [];
08  arr = [1, 2, 3]; // 타입 에러가 발생합니다
09
```

```
10  const obj = {};
11  obj = { name: '내 이름' }; // 타입 에러가 발생합니다
12
13  const arr2 = [];
14  arr2.push(1); // arr2 = [1]
15  arr2.splice(0, 0, 0); // arr2 = [0,1]
16  arr2.pop(); // arr2 = [1]
17
18  const obj2 = {};
19  obj2['name'] = '내이름'; // obj2 = { name: '내이름' }
20  Object.assign(obj2, { name: '새이름' }); //obj2 = { name: '새이름' }
21  delete obj2.name; //obj2 = {}
22
23  const num1 = 1;
24  const num2 = num1 * 3; // num2 = 3
25
26  const str1 = '문자';
27  const str2 = str1 + '추가'; // str2 = '문자추가'
28
29  const arr3 = [];
30  const arr4 = arr3.concat(1); // arr4 = [1]
31  const arr5 = [...arr4, 2, 3]; // arr5 = [1, 2, 3]
32  const arr6 = arr5.slice(0, 1); // arr6 = [1], arr5 = [1, 2, 3]
33  const [first, ...arr7] = arr5; // arr7 = [2, 3], first = 1
34
35  const obj3 = { name: '내이름', age: 20 };
36  const obj4 = { ...obj3, name: '새이름' }; // obj4 = { name: '새이름', age: 20}
37  const { name, ...obj5 } = obj4; // obj5 = { age: 20 }
38
39  const arr = [1, 2, 3];
40  // 가변 변수를 사용한 예
41  for (let i = 0; i < arr.length; i++) {
42    console.log(arr[i]);
43  }
44  // iterator 방식의 for-in 루프와 함께 불변 변수를 사용한 예
45  for (const item in arr) {
46    console.log(item);
47  }
48
49  // forEach 함수를 활용한 예
50  arr.forEach((item, index) => {
51    console.log(item);
52    console.log(index);
53  });
```

2.4 클래스

- ES5/ES6의 클래스 표현 방법 알아보기
  - 31: 기존 자바스크립트에서는 함수를 생성자 형태로 선언한 다음 상속이 필요한 변수나 함수를 prototype 객체에 할당하는 방법을 사용한다.
  - 68: ES6에서는 Java와 같은 문법구조를 갖는다.

[./src/02/02-6.js]

```
01  // ES5 문법
02  function Shape(x, y) {
03    this.name = 'Shape';
04    this.move(x, y);
05  }
06  // static 타입 선언 예제
```

```
07  Shape.create = function(x, y) {
08    return new Shape(x, y);
09  };
10  Shape.prototype.move = function(x, y) {
11    this.x = x;
12    this.y = y;
13  };
14  Shape.prototype.area = function() {
15    return 0;
16  };
17
18  // 혹은
19  Shape.prototype = {
20    move: function(x, y) {
21      this.x = x;
22      this.y = y;
23    },
24    area: function() {
25      return 0;
26    },
27  };
28
29  var s = new Shape(0, 0);
30  var s2 = Shape.create(0, 0);
31  s.area(); // 0
32
33  function Circle(x, y, radius) {
34    Shape.call(this, x, y);
35    this.name = 'Circle';
36    this.radius = radius;
37  }
38  Object.assign(Circle.prototype, Shape.prototype, {
39    area: function() {
40      return this.radius * this.radius;
41    },
42  });
43
44  var c = new Circle(0, 0, 10);
45  c.area(); // 100
46
47  // ES6 예제
48  class Shape {
49    static create(x, y) {
50      return new Shape(x, y);
51    }
52    name = 'Shape';
53
54    constructor(x, y) {
55      this.move(x, y);
56    }
57    move(x, y) {
58      this.x = x;
59      this.y = y;
60    }
61    area() {
62      return 0;
63    }
64  }
65
66  var s = new Shape(0, 0);
67  var s1 = Shape.create(0, 0);
68  s.area(); // 0
69
70  class Circle extends Shape {
71    constructor(x, y, radius) {
```

```
72      super(x, y);
73      this.radius = radius;
74    }
75    area() {
76      if (this.radius === 0) return super.area();
77      return this.radius * this.radius;
78    }
79  }
80
81  var c = new Circle(0, 0, 10);
82  c.area(); // 100
```

2.5 화살표 함수

- 화살표 함수(arrow function)는 ES6에 추가된 표현식을 사용하는 함수로 화살표 기호 =>로 함수를 선언한다.

[./src/02/02-5.js]

```
01  function add(first, second) {
02    return first + second;
03  }
04
05  var add = function(first, second) {
06    return first + second;
07  };
08
09  var add = function add(first, second) {
10    return first + second;
11  };
12
13  // this scope를 전달한 예
14  var self = this;
15  var addThis = function(first, second) {
16    return self.value + first + second;
17  };
18
19  var addThis = (first, second) => first + second;
20
21  function addNumber(num) {
22    return function(value) {
23      return num + value;
24    };
25  }
26  // 화살표 함수로 변환한 예
27  var addNumber = num => value => num + value;
28
29  var addTwo = addNumber(2);
30  var result = addTwo(4); // 6
31
32  // bind함수를 통해 this scope를 전달한 예
33  class MyClass {
34    value = 10;
35    constructor() {
36      var addThis2 = function(first, second) {
37        return this.value + first + second;
38      }.bind(this);
39      var addThis3 = (first, second) => this.value + first + second;
40    }
41  }
```

2.6 객체 확장 표현식과 구조 분해 할당

[./src/02/02-7.js]

```
// ES5의 예
var x = 0;
var y = 0;

var obj = { x: x, y: y};

var randomKeyString = 'other';
var combined = {};
combined['one' + randomKeyString] = 'some value';

var obj2 = {
  methodA: function() { console.log('A'); },
  methodB: function() { return 0; },
};

// ES6의 예
var x = 0;
var y = 0;
var obj = { x, y };

var randomKeyString = 'other';
var combined = {
  ['one' + randomKeyString]: 'some value',
};

var obj2 = {
  x,
  methodA() { console.log('A'); },
  methodB() { return 0; },
};
```

2.7 라이브러리 의존성 관리

2.8 배열 함수

2.9 비동기 함수

■ 비동기 함수는 비동기 처리를 위해 사용한다.

2.9.1 기존 자바스크립트의 비동기 함수 처리 방법 알아보기

■ 기존 자바스크립트는 비동기 작업을 위해 지연 작업이 필요한 함수에 setTimeout() 함수를 이용한다.

[./src/02/02-10.js]

```
// ES5의 예제
```

```
function work1(onDone) {
  setTimeout(() => onDone('작업1 완료!'), 100);
}
function work2(onDone) {
  setTimeout(() => onDone('작업1 완료!'), 200);
}
function work3(onDone) {
  setTimeout(() => onDone('작업3 완료!'), 300);
}
function urgentWork() {
  console.log('긴급 작업');
}
// 실제 비동기 함수를 사용하는 예
work1(function(msg1) {
  console.log('done after 100ms:' + msg1);
  work2(function(msg2) {
    console.log('done after 300ms:' + msg2);
    work3(function(msg3) {
      console.log('done after 600ms:' + msg3);
    });
  });
});
urgentWork();
```

2.9.2 ES6의 비동기 함수 처리 방법 알아보기

■ ES6에는 Promise 클래스 함수 표현법이 추가되었다.

[./src/02/02-10-2.js]

```
// ES6의 예제
const work1 = () =>
  new Promise(resolve => {
    setTimeout(() => resolve('작업1 완료!'), 100);
  });
const work2 = () =>
  new Promise(resolve => {
    setTimeout(() => resolve('작업2 완료!'), 200);
  });
const work3 = () =>
  new Promise(resolve => {
    setTimeout(() => resolve('작업3 완료!'), 300);
  });
function urgentWork() {
  console.log('긴급 작업');
}

const nextWorkOnDone = msg1 => {
  console.log('done after 100ms:' + msg1);
  return work2();
};

work1()
  .then(nextWorkOnDone)
  .then(msg2 => {
    console.log('done after 200ms:' + msg2);
    return work3();
  })
  .then(msg3 => {
    console.log(`done after 600ms:${msg3}`);
  });
```

```
32  urgentWork();
33  const work1and2 = () =>
34    work1().then(msg1 => {
35      console.log('done after 100ms:' + msg1);
36      return work2();
37    });
38
39  work1and2()
40    .then(msg2 => {
41      console.log('done after 200ms:' + msg2);
42      return work3();
43    })
44    .then(msg3 => {
45      console.log('done after 600ms:' + msg3);
46    });
47
48  // 아래 코드는 이해를 돕기 위한 코드입니다. 실제 코드와는 다르니 내장된 Promise를 사용해주세요.
49  // class Promise {
50  //   constructor(fn) {
51  //     function resolve() {
52  //       if (typeof this.onDone === 'function') {
53  //         this.onDone.apply(null, arguments);
54  //       }
55  //       if (typeof this.onComplete === 'function') {
56  //         this.onComplete();
57  //       }
58  //     }
59  //     function reject() {
60  //       if (typeof this.onError === 'function') {
61  //         this.onError.apply(null, arguments);
62  //       }
63  //       if (typeof this.onComplete === 'function') {
64  //         this.onComplete();
65  //       }
66  //     }
67  //     fn(resolve.bind(this), reject.bind(this));
68  //   }
69  //   then(onDone, onError) {
70  //     this.onDone = onDone;
71  //     this.onError = onError;
72  //     return this;
73  //   }
74  //   catch(onError) {
75  //     this.onError = onError;
76  //     return this;
77  //   }
78  //   finally(onComplete) {
79  //     this.onComplete = onComplete;
80  //     return this;
81  //   }
82  // }
```