JavaScript

String – 김근형 강사

String

- String
  - String 객체는 원시 타입인 문자열을 다룰 때 유용한 프로퍼티와 메소드를 제공하는 레퍼(wrapper) 객체이다.
  - 변수 또는 객체 프로퍼티가 문자열을 값으로 가지고 있다면 String 객체의 별도 생성없이 String 객체의 프로퍼티와 메소드를 사용할 수 있다.
  - 원시 타입이 wrapper 객체의 메소드를 사용할 수 있는 이유는 원시 타입으로 프로퍼티나 메소드를 호출할 때 원시 타입과 연관된 wrapper 객체로 일시적으로 변환되어 프로토타입 객체를 공유하게 되기 때문이다.

String

- String constructor
  - String 객체는 String 생성자 함수를 통해 생성할 수 있다. 이때 전달된 인자는 모두 문자열로 변환된다.

```
        let strObj = new String('Lee');
        console.log(strObj); // String {0: 'L', 1: 'e', 2: 'e', length: 3, [[PrimitiveValue]]: 'Lee'}

        strObj = new String(1);
        console.log(strObj); // String {0: '1', length: 1, [[PrimitiveValue]]: '1'}

        strObj = new String(undefined);
        console.log(strObj); // String {0: 'u', 1: 'n', 2: 'd', 3: 'e', 4: 'f', 5: 'i', 6: 'n', 7: 'e', 8: 'd',
length: 9, [[PrimitiveValue]]: 'undefined'}
```

String

- String constructor
  - String 객체는 String 생성자 함수를 통해 생성할 수 있다. 이때 전달된 인자는 모두 문자열로 변환된다.

```
var x = String('Lee');

console.log(typeof x, x); // string Lee
```

  - 일반적으로 문자열을 사용할 때는 원시 타입 문자열을 사용한다.

```
const str = 'Lee';
const strObj = new String('Lee');

console.log(str == strObj);  // true
console.log(str === strObj); // false

console.log(typeof str);    // string
console.log(typeof strObj); // object
```

String 속성

- length
  - 문자열 내의 문자 개수를 반환한다. String 객체는 length 프로퍼티를 소유하고 있으므로 유사 배열 객체이다.

```
const str1 = 'Hello';
console.log(str1.length); // 5

const str2 = '안녕하세요!';
console.log(str2.length); // 6
```

String method

- charAt
  - 인수로 전달한 index를 사용하여 index에 해당하는 위치의 문자를 반환한다.
  - index는 0 ~ (문자열 길이 - 1) 사이의 정수이다.
  - 지정한 index가 문자열의 범위(0 ~ (문자열 길이 - 1))를 벗어난 경우 빈문자열을 반환한다.

String method

- charAt

```
const str = 'Hello';

console.log(str.charAt(0)); // H
console.log(str.charAt(1)); // e
console.log(str.charAt(2)); // l
console.log(str.charAt(3)); // l
console.log(str.charAt(4)); // o
// 지정한 index가 범위(0 ~ str.length-1)를 벗어난 경우 빈문자열을 반환한다.
console.log(str.charAt(5)); // ''

// 문자열 순회. 문자열은 length 프로퍼티를 갖는다.
for (let i = 0; i < str.length; i++) {
  console.log(str.charAt(i));
}

// String 객체는 유사 배열 객체이므로 배열과 유사하게 접근할 수 있다.
for (let i = 0; i < str.length; i++) {
  console.log(str[i]); // str['0']
}
```

String method

- concat
  - 인수로 전달한 1개 이상의 문자열과 연결하여 새로운 문자열을 반환한다.
  - concat 메소드를 사용하는 것보다는 +, += 할당 연산자를 사용하는 것이 성능상 유리하다.

```
console.log('Hello '.concat('kim')); // Hello kim
```

String method

- indexOf
  - 인수로 전달한 문자 또는 문자열을 대상 문자열에서 검색하여 처음 발견된 곳의 index를 반환한다. 발견하지 못한 경우 -1을 반환한다.

```
const str = 'Hello World';

console.log(str.indexOf('l'));  // 2
console.log(str.indexOf('or')); // 7
console.log(str.indexOf('or' , 8)); // -1

if (str.indexOf('Hello') !== -1) {
  // 문자열 str에 'hello'가 포함되어 있는 경우에 처리할 내용
}

// ES6: String.prototype.includes
if (str.includes('Hello')) {
  // 문자열 str에 'hello'가 포함되어 있는 경우에 처리할 내용
}
```

String method

- lastIndexOf
  - 인수로 전달한 문자 또는 문자열을 대상 문자열에서 검색하여 마지막으로 발견된 곳의 index를 반환한다. 발견하지 못한 경우 -1을 반환한다.
  - 2번째 인수(fromIndex)가 전달되면 검색 시작 위치를 fromIndex으로 이동하여 역방향으로 검색을 시작한다. 이때 검색 범위는 0 ~ fromIndex이며 반환값은 indexOf 메소드와 동일하게 발견된 곳의 index이다.

```
const str = 'Hello World';

console.log(str.lastIndexOf('World')); // 6
console.log(str.lastIndexOf('l'));     // 9
console.log(str.lastIndexOf('o', 5));  // 4
console.log(str.lastIndexOf('o', 8));  // 7
console.log(str.lastIndexOf('l', 10)); // 9

console.log(str.lastIndexOf('H', 0));  // 0
console.log(str.lastIndexOf('W', 5));  // -1
console.log(str.lastIndexOf('x', 8));  // -1
```

String method

- replace
  - 첫번째 인수로 전달한 문자열 또는 정규표현식을 대상 문자열에서 검색하여 두번째 인수로 전달한 문자열로 대체한다. 원본 문자열은 변경되지 않고 결과가 반영된 새로운 문자열을 반환한다.
  - 검색된 문자열이 여럿 존재할 경우 첫번째로 검색된 문자열만 대체된다.

```
const str = 'Hello world';

// 첫번째로 검색된 문자열만 대체하여 새로운 문자열을 반환한다.
console.log(str.replace('world', 'Kim')); // Hello Kim

// 특수한 교체 패턴을 사용할 수 있다. ($& => 검색된 문자열)
console.log(str.replace('world', '<strong>$&</strong>')); // Hello <strong>world</strong>

/* 정규표현식
g(Global): 문자열 내의 모든 패턴을 검색한다.
i(Ignore case): 대소문자를 구별하지 않고 검색한다.
*/
console.log(str.replace(/hello/gi, 'Kim')); // Kim world

// 두번째 인수로 치환 함수를 전달할 수 있다.
// camelCase => snake_case
const camelCase = 'helloWorld';

// /.[A-Z]/g => 1문자와 대문자의 조합을 문자열 전체에서 검색한다.
console.log(camelCase.replace(/.[A-Z]/g, function (match) {
  // match : oW => match[0] : o, match[1] : W
  return match[0] + '_' + match[1].toLowerCase();
})); // hello_world

// /(.)([A-Z])/g => 1문자와 대문자의 조합
// $1 => (.)
// $2 => ([A-Z])
console.log(camelCase.replace(/(.)([A-Z])/g, '$1_$2').toLowerCase()); // hello_world

// snake_case => camelCase
const snakeCase = 'hello_world';

// /_./g => _와 1문자의 조합을 문자열 전체에서 검색한다.
console.log(snakeCase.replace(/_./g, function (match) {
  // match : _w => match[1] : w
  return match[1].toUpperCase();
})); // helloWorld
```

String method

- split
  - 첫번째 인수로 전달한 문자열 또는 정규표현식을 대상 문자열에서 검색하여 문자열을 구분한 후 분리된 각 문자열로 이루어진 배열을 반환한다. 원본 문자열은 변경되지 않는다.
  - 인수가 없는 경우, 대상 문자열 전체를 단일 요소로 하는 배열을 반환한다.

```
const str = 'How are you doing?';

// 공백으로 구분(단어로 구분)하여 배열로 반환한다
console.log(str.split(' ')); // [ 'How', 'are', 'you', 'doing?' ]

// 정규 표현식
console.log(str.split(/\s/)); // [ 'How', 'are', 'you', 'doing?' ]

// 인수가 없는 경우, 대상 문자열 전체를 단일 요소로 하는 배열을 반환한다.
console.log(str.split()); // [ 'How are you doing?' ]

// 각 문자를 모두 분리한다
console.log(str.split('')); // [ 'H','o','w',' ','a','r','e',' ','y','o','u',' ','d','o','i','n','g','?' ]

// 공백으로 구분하여 배열로 반환한다. 단 요소수는 3개까지만 허용한다
console.log(str.split(' ', 3)); // [ 'How', 'are', 'you' ]

// 'o'으로 구분하여 배열로 반환한다.
console.log(str.split('o')); // [ 'H', 'w are y', 'u d', 'ing?' ]
```

String method

- substring
  - 첫번째 인수로 전달한 start 인덱스에 해당하는 문자부터 두번째 인자에 전달된 end 인덱스에 해당하는 문자의 바로 이전 문자까지를 모두 반환한다. 이때 첫번째 인수 < 두번째 인수의 관계가 성립된다.
    - 첫번째 인수 > 두번째 인수 : 두 인수는 교환된다.
    - 두번째 인수가 생략된 경우 : 해당 문자열의 끝까지 반환한다.
    - 인수 < 0 또는 NaN인 경우 : 0으로 취급된다.
    - 인수 > 문자열의 길이(str.length) : 인수는 문자열의 길이(str.length)으로 취급된다.

String method

- substring

```
const str = 'Hello World'; // str.length == 11

console.log(str.substring(1, 4)); // ell

// 첫번째 인수 > 두번째 인수 : 두 인수는 교환된다.
console.log(str.substring(4, 1)); // ell

// 두번째 인수가 생략된 경우 : 해당 문자열의 끝까지 반환한다.
console.log(str.substring(4)); // o World

// 인수 < 0 또는 NaN인 경우 : 0으로 취급된다.
console.log(str.substring(-2)); // Hello World

// 인수 > 문자열의 길이(str.length) : 인수는 문자열의 길이(str.length)으로 취급된다.
console.log(str.substring(1, 12)); // ello World
console.log(str.substring(11)); // ''
console.log(str.substring(20)); // ''
console.log(str.substring(0, str.indexOf(' '))); // 'Hello'
console.log(str.substring(str.indexOf(' ') + 1, str.length)); // 'World'
```

String method

- slice
  - String.prototype.substring과 동일하다. 단, String.prototype.slice는 음수의 인수를 전달할 수 있다.

```
const str = 'hello world';

// 인수 < 0 또는 NaN인 경우 : 0으로 취급된다.
console.log(str.substring(-5)); // 'hello world'
// 뒤에서 5자리를 잘라내어 반환한다.
console.log(str.slice(-5)); // 'world'

// 2번째부터 마지막 문자까지 잘라내어 반환
console.log(str.substring(2)); // llo world
console.log(str.slice(2)); // llo world

// 0번째부터 5번째 이전 문자까지 잘라내어 반환
console.log(str.substring(0, 5)); // hello
console.log(str.slice(0, 5)); // hello
```

String method

- toLowerCase
  - 대상 문자열의 모든 문자를 소문자로 변경한다.

```
console.log('Hello World!'.toLowerCase()); // hello world!
```

- toUpperCase
  - 대상 문자열의 모든 문자를 대문자로 변경한다.

```
console.log('Hello World!'.toUpperCase()); // HELLO WORLD!
```

String method

- trim
  - 대상 문자열 양쪽 끝에 있는 공백 문자를 제거한 문자열을 반환한다.

```
const str = '   foo  ';

console.log(str.trim()); // 'foo'

// String.prototype.replace
console.log(str.replace(/\s/g, ''));   // 'foo'
console.log(str.replace(/^\s+/g, '')); // 'foo  '
console.log(str.replace(/\s+$/g, '')); // '   foo'

// String.prototype.{trimStart,trimEnd} : Proposal stage 3
console.log(str.trimStart()); // 'foo  '
console.log(str.trimEnd());   // '   foo'
```

String method

- repeat
  - 인수로 전달한 숫자만큼 반복해 연결한 새로운 문자열을 반환한다. count가 0이면 빈 문자열을 반환하고 음수이면 RangeError를 발생시킨다.

```
console.log('abc'.repeat(0));   // ''
console.log('abc'.repeat(1));   // 'abc'
console.log('abc'.repeat(2));   // 'abcabc'
console.log('abc'.repeat(2.5)); // 'abcabc' (2.5 → 2)
console.log('abc'.repeat(-1));  // RangeError: Invalid count value
```

String method

- includes
  - 인수로 전달한 문자열이 포함되어 있는지를 검사하고 결과를 불리언 값으로 반환한다. 두번째 인수는 옵션으로 검색할 위치를 나타내는 정수이다.

```
const str = 'hello world';

console.log(str.includes('hello')); // true
console.log(str.includes(' '));     // true
console.log(str.includes('wo'));    // true
console.log(str.includes('wow'));   // false
console.log(str.includes(''));      // true
console.log(str.includes());        // false

// String.prototype.indexOf 메소드로 대체할 수 있다.
console.log(str.indexOf('hello')); // 0
```

String method

- Unicode
  - ES6에서 유니코드와 관련된 프로퍼티와 메서드가 추가가 됨.

```
// 16진수 이스케이프 시퀀스
console.log("1:", "\x31\x32\x33");

// 유니코드 이스케이프 시퀀스
console.log("2:", "\u0031\u0032\u0033");

// 유니코드 코드포인트 이스케이프
console.log("3:", "\u{34}\u{35}\u{36}");

// U+FFFF보다 큰 코드 포인트: 코끼리
//http://unicode-table.com/en/1F418/
console.log("4:", "\u{1f418}");

//서로게이트 페어(Surrogate pair)
console.log("5:", "\uD83D\uDC18");
```

```
1: 123
2: 123
3: 456
4: 🐘
5: 🐘
```

String method

- fromCodePoint()
  - 유니코드의 코드 포인트에 해당하는 문자를 반환한다.

```
// #$%&
console.log("1:", String.fromCodePoint(35, 36, 37));

// 16진수로 지정, 49, 50, 51로 지정한 것과 같음
console.log("2:", String.fromCodePoint(0x31, 0x32, 0x33));

// 가각
console.log("3:", String.fromCodePoint(44032, 44033));

// 코끼리
//http://unicode-table.com/en/1F418/
console.log("4:", String.fromCodePoint(0x1F418));

console.log("5:", String.fromCharCode(0x1f418));

console.log("6:", String.fromCharCode(0xD83D, 0xDC18));
```

```
1: #$%
2: 123
3: 가각
4: 🐘
5: □
6: 🐘
```

String method

- codePointAt()
  - 문자열에서 파라미터에 지정한 인덱스 번째 문자의 코드 포인트 값을 반환한다.
  - 파라미터로는 문자열 인덱스 위치를 정수로 넣어준다.

```
console.log("가".codePointAt(0));

let values = "ABC";
for (var value of values){
  console.log(value, value.codePointAt(0));
};
```

```
44032
A 65
B 66
C 67
```

String method

- startWith()
  - 해당 문자열이 첫 번째 파라미터 문자열로 시작하면 true 아니면 false를 반환.
  - 두번째 파라미터에 인덱스를 넣어서 어디서부터 시작할 지를 지정할 수 있다.

```
let target = "123가나다";
console.log("1:", target.startsWith(123));

console.log("2:", target.startsWith("23"));

console.log("3:", target.startsWith("가나", 3));
```

```
1: true
2: false
3: true
```

String method

- endWith()
  - 해당 문자열이 첫 번째 파라미터 문자열로 끝면 true 아니면 false를 반환.
  - 두번째 파라미터는 해당 문자열의 인덱스 바로 전 까지를 문자열의 끝으로 지정할 때 쓴다.

```
let target = "123가나다";
console.log(target.endsWith("가나다"));

console.log(target.endsWith("가나"));

console.log(target.endsWith("가나", 5));
```

```
true
false
true
```