삼육대학교 SW융합교육원 김진호

자바스크립트 기초교육

<class>
</body>
<head>
<title>

11-5. RegExp

11-5. RegExp

정규 표현식(Regular expression)은 일정한 패턴을 가진 문자열의 집합을 표현하기 위해 사용하는 형식 언어(formal language)이다. 정규 표현식은 대부분의 프로그래밍 언어와 코드 에디터에 내장되어 있다. 문자열을 대상으로한 패턴 매칭 기능을 제공하므로 예를 들어 회원 가입 시 필요한 사용자가 입력한 비밀번호의 패턴 확인, 전화번호 유효성 확인 등의 기능에서 활용할 수 있다.

11-5. RegExp

11-5-1. RegExp

정규표현식 생성

```
const target = 'JavaScript';    // 검색 대상

// 정규 표현식 리터럴 - /pattern/플래그
let regexp = /j/i;  // 패턴 : j, 플래그 : i => 대소문자 구별 없이

// RegExp 생성자 함수 - new RegExp(pattern[, flag])
regexp = new RegExp('j', 'i');
regexp = new RegExp(/j/, 'i');
regexp = new RegExp(/j/i);          // ES6부터 가능한 표현

// test 메서드 - target 문자열에 대해 정규 표현식 regexp의 패턴을 검색하여
매칭 결과를 불리언으로 반환
console.log(regexp.test(target));
```

11-5. RegExp

11-5-2. RegExp method

RegExp.prototype.exec

■ 인수로 전달받은 문자열에 대해 정규 표현식의 패턴을 검색하여 매칭 결과를 배열로 반환한다.

```
const target = 'Java JavaScript';
console.log(/va/.exec(target));     // [ 'va', index: 2, input: 'Java
JavaScript', groups: undefined ]
// 문자열 내의 모든 패턴을 검색하는 g 플래그를 지정해도 첫 번째 매칭 결과만 반환
console.log(/va/g.exec(target));    // [ 'va', index: 2, input: 'Java
JavaScript', groups: undefined ]
// 매칭 결과가 없을 경우 null 반환
console.log(/hello/.exec(target));  // null
```

11-5. RegExp

11-5-2. RegExp method

RegExp.prototype.test

- 인수로 전달받은 문자열에 대해 정규 표현식의 패턴을 검색하여 매칭 결과를 불리언으로 반환한다.

```
const target = 'Java JavaScript';
console.log(/va/.test(target));          // true
console.log(/hello/.test(target));       // false
```

11-5. RegExp

11-5-2. RegExp method

String.prototype.match

- String 표준 빌트인 객체가 제공하는 메서드로 대상 문자열과 인수로 전달 받은 정규 표현식과의 매칭 결과를 배열로 반환한다.

```
const target = 'Java JavaScript';
console.log(target.match(/va/));        // [ 'va', index: 2, input:
'Java JavaScript', groups: undefined ]
// 문자열 내의 모든 패턴을 검색하는 g 플래그를 지정하면 모든 매칭 결과를 배열로 반환
console.log(target.match(/va/g));       // [ 'va', 'va' ]
console.log(target.match(/hello/));     // null
```

이외에 정규표현식을 사용하는 메서드

- String.prototype.replace, String.prototype.search, String.prototype.split 등이 있다. String 챕터에서 참조한다.

11-5. RegExp

11-5-3. flag and pattern

플래그 문자

- flag는 옵션이므로 선택적으로 사용할 수 있고, 순서와 상관 없이 하나 이상의 플래그를 동시에 설정할 수 있다.
- i(ignore case) : 대소문자를 구별하지 않고 패턴 검색
- g(Global) : 대상 문자열 내에서 패턴과 일치하는 모든 문자열을 전역 검색

```
let target = 'Java JavaScript';

console.log(target.match(/VA/));        // null
console.log(target.match(/VA/i));       // [ 'va', index: 2, input:
'Java JavaScript', groups: undefined ]
console.log(target.match(/VA/ig));      // [ 'va', 'va' ]
console.log('----------------------');
```

11-5. RegExp

11-5-3. flag and pattern

패턴

- 패턴은 특별한 의미를 가지는 메타문자 또는 기호로 표현할 수 있다.
- . : 임의의 문자열

```
target = 'abcdefg';
// 임의의 2자리 문자열 전역 검색
console.log(target.match(/../g));       // [ 'ab', 'cd', 'ef' ]
```

11-5. RegExp

11-5-3. flag and pattern

패턴

- {m,n} : 최소 m번, 최대 n번 반복 되는 문자열 (반복 검색)

```
target = 'a aa aaa b bb bbb ab aab abb';
// a 최소 2번 ~ 최대 3번 반복
console.log(target.match(/a{2,3}/g));        // [ 'aa', 'aaa', 'aa' ]
// b 두번 반복
console.log(target.match(/b{2}/g));          // [ 'bb', 'bb', 'bb' ]
// b 세번 이상 반복
console.log(target.match(/b{3,}/g));         // [ 'bbb' ]
```

11-5. RegExp

11-5-3. flag and pattern

패턴

- + : 앞선 패턴이 최소 한 번 이상 반복되는 문자열 (반복 검색)
- +는 {1,}과 같다

```
// b 최소 1번 이상 반복
target = 'a aa aaa b bb bbb ab aab abb';
console.log(target.match(/b+/g));
// [ 'b', 'bb', 'bbb', 'b', 'b', 'bb' ]
```

11-5. RegExp

11-5-3. flag and pattern

패턴

- ? : 앞선 패턴이 최대 한 번(0번 포함) 이상 반복되는 문자열 (반복 검색)
- ?는 {0,1}과 같다

```
target = 'soul seoul';
// s 다음 e가 최대 한번(0번 포함) 이상 반복 되고 oul이 이어지는 문자열 전역 검색
console.log(target.match(/se?oul/g));        // [ 'soul', 'seoul' ]
```

11-5. RegExp

11-5-3. flag and pattern

패턴

- | : or

```
target = 'aa bb cc dd 123 456 _@';
console.log(target.match(/a|b/g));          // [ 'a', 'a', 'b', 'b' ]
// 분해되지 않은 단어 레벨로 검색하기 위해 + 함께 사용
console.log(target.match(/a+|b+/g));        // [ 'aa', 'bb' ]
// [] 내의 문자는 or로 동작
console.log(target.match(/[ab]+/g));        // [ 'aa', 'bb' ]
// 범위를 지정하려면 -를 사용 (소문자 범위)
console.log(target.match(/[a-z]+/g));       // [ 'aa', 'bb', 'cc', 'dd' ]
// 대소문자 범위
console.log(target.match(/[A-Za-z]+/g));    // [ 'aa', 'bb', 'cc', 'dd' ]
// 숫자 범위
console.log(target.match(/[0-9]+/g));       // [ '123', '456' ]
```

11-5. RegExp

11-5-3. flag and pattern

패턴

- \d : 숫자
- \D : 숫자가 아닌 문자
- \w : 알파벳, 숫자, 언더스코어
- \W : \w의 반대

```
target = 'aa bb cc dd 123 456 _@';
console.log(target.match(/\d+/g));      // [ '123', '456' ]
console.log(target.match(/\D+/g));      // [ 'aa bb cc dd ', ' ', ' _@' ]
console.log(target.match(/\w+/g));
/*
[
  'aa',  'bb',
  'cc',  'dd',
  '123', '456',
  '_'
]
*/
console.log(target.match(/\W+/g));
/*
[
  ' ', ' ', ' ',
  ' ', ' ', ' ',
  '@'
]
*/
```

11-5. RegExp

11-5-3. flag and pattern

패턴

- [...] 내의 ^ : not

```
target = 'aa bb cc dd 123 456 _@';
console.log(target.match(/[^0-9]+/g));    // [ 'aa bb cc dd ', ' ', ' _@' ]
console.log(target.match(/[^a-z]+/g));    // [ ' ', ' ', ' ', ' 123 456 _@' ]
```

- [...] 밖의 ^ : 시작 위치 검색
- $: 마지막 위치 검색

```
target = 'https://www.google.com';
// https로 시작하는지 검사
console.log(/^https/.test(target));          // true
// com으로 끝나는지 검사
console.log(/com$/.test(target));            // true
```

11-5. RegExp

11-5-4. example

특정 단어로 시작하는지 검사

```
const url = 'https://www.google.com';
// const url = 'http://www.google.com';
// const url = 'www.google.com';

// http:// 또는 https:// 로 시작하는지 검사
console.log(/^https?:\/\//.test(url));
```

11-5. RegExp

11-5-4. example

특정 단어로 끝나는지 검사하는 경우

```
const fileName = 'test.js';
// const fileName = 'test.css';

// 파일 확장자 검사
console.log(/js$/.test(fileName));
```

11-5. RegExp

11-5-4. example

숫자로만 이루어진 문자열인지 검사

```
const target = '12345';
// const target = '12345@';

// 처음과 끝이 숫자이고 최소 한 번 이상 반복 되는 문자열과 매칭
console.log(/^\d+$/.test(target));
```

아이디로 사용 가능한지 검사

```
const id = 'hello123';
// const id = 'hello';
// const id = 'hello안녕';

// 알파벳 대소문자 또는 숫자로 시작하고 끝나며 6~12자리인지 검사
console.log(/^[A-Za-z0-9]{6,12}$/.test(id));
```

11-5. RegExp

11-5-4. example

핸드폰 번호 형식에 맞는지 검사

```
const phone = '010-1234-5678';
// const phone = '010-123-4567';
// const phone = '010-123-456';

console.log(/^\d{3}-\d{3,4}-\d{4}$/.test(phone));
```

특수 문자 포함 여부 검사

```
const target2 = 'hello#world';
// const target2 = 'helloworld';

console.log(/[^A-Za-z0-9]/gi.test(target2));
```