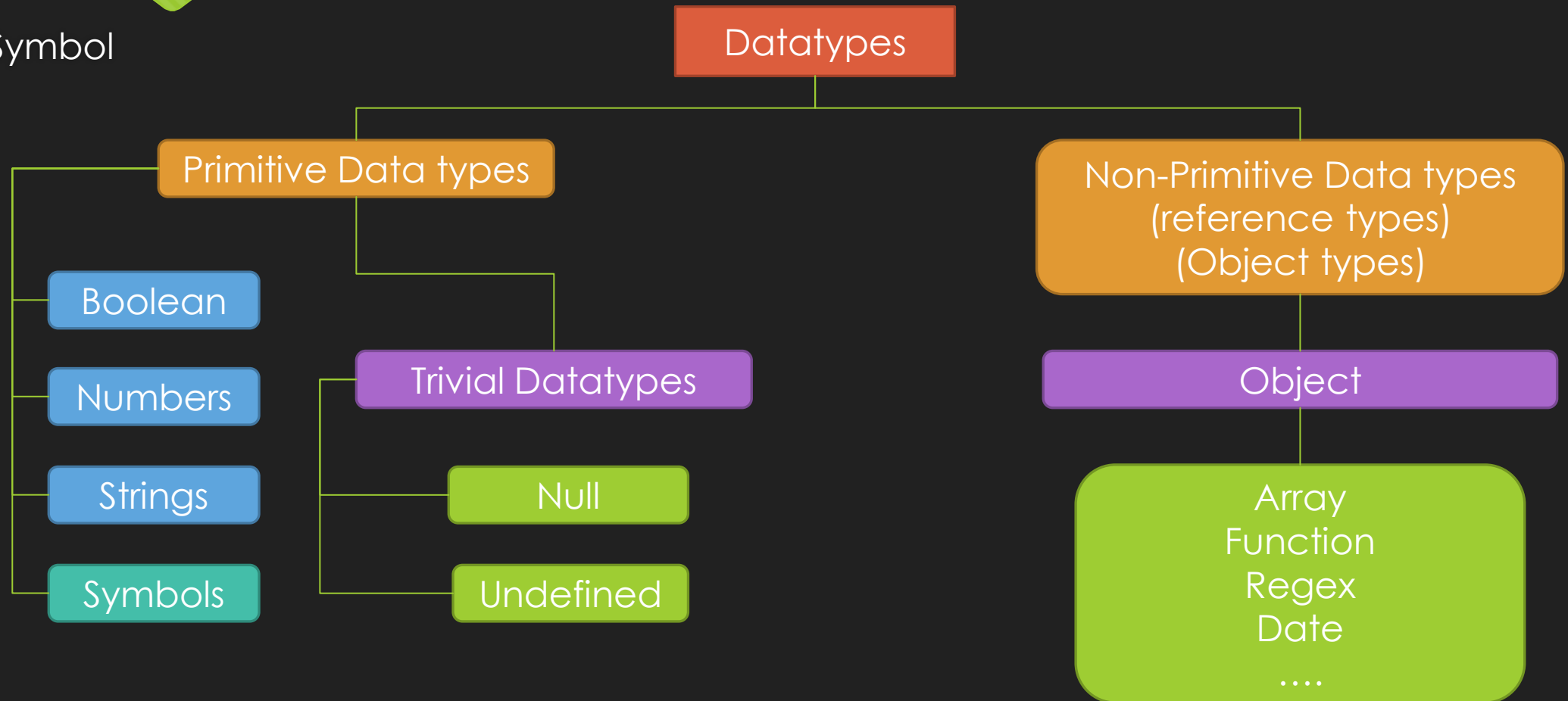


# JavaScript

Symbol – 김근형 강사

# Symbol

## ○ Symbol



# Symbol

- Symbol

- 심볼(symbol)은 ES6에서 새롭게 추가된 7번째 타입으로 변경 불가능한 원시 타입의 값이다.
- 심볼은 주로 이름의 충돌 위험이 없는 유일한 객체의 프로퍼티 키(property key)를 만들기 위해 사용한다.

구분	타입	데이터(값)
Object	Symbol	Symbol()
파라미터	String	(선택), 설명, 주석
반환	Symbol	유니크한 Symbol 값

# symbol object

- Symbol object
  - Symbol은 유일하고 변경 불가능한(immutable) 기본값 (primitive value) 이다.
  - 또한, 객체 속성의 key 값으로도 사용될 수 있다.

```
const sym = Symbol();  
console.log("1:", sym);  
console.log("2:", typeof sym);  
console.log("3:", Symbol("주석"));  
  
console.log("4:", sym == Symbol());
```



1:	Symbol()
2:	symbol
3:	Symbol(주석)
4:	false

# symbol object

## ○ Symbol 값 변경

```
let sym = Symbol();
try {
  +sym;
} catch (e) {
  console.log("+sym 사용 불가");
};

try {
  sym | 0;
} catch (e) {
  console.log("sym | 0 사용 불가");
};
```



+sym 사용 불가
sym   0 사용 불가

# symbol object

## ○ Symbol 값 변경

```
let sym = Symbol();
try {
  sym + "문자열";
} catch (e) {
  console.log("문자열 연결 불가");
};

console.log(String(sym) + "연결");
console.log(sym.toString() + "연결");
```



문자열 연결 불가
Symbol()연결
Symbol()연결

```
let sym = Symbol("123");
try {
  `${sym}`;
} catch (e) {
  console.log(`${sym} 불가`);
}
```



`\${sym} 불가`
--------------

# symbol object

- Symbol 오브젝트 생성
  - Object()의 파라미터에 Symbol 값을 지정하면 Symbol 오브젝트를 반환한다.
  - Symbol 오브젝트에 symbol 메서드, Symbol.prototype, prototype에 연결된 프로퍼티가 설정된다.

```
let sym = Symbol("123");  
const obj = Object(sym);  
console.log(obj);  
  
console.log(obj == sym);  
console.log(obj === sym);
```



```
▼ Symbol ⓘ  
  ► __proto__: Symbol  
    [[PrimitiveValue]]: Symbol(123)  
true  
false
```

# symbol object

- 오브젝트에서 Symbol 사용
  - 유일한 값을 갖는 Symbol 특성을 활용하여 Symbol 값을 오브젝트의 프로퍼티 키로 사용하면 프로퍼티 키가 중복되지 않는다.

```
let sym = Symbol("123");  
let obj = {[sym]: "456"};  
console.log(obj);
```

```
console.log(obj[sym]);  
console.log(obj.sym);
```



```
▼ Object ⓘ  
  Symbol(123): "456"  
  ► __proto__: Object  
456  
undefined
```



# symbol object

- Symbol 사용 형태
  - for~in 문에서 symbol-keyed 프로퍼티가 열거되지 않는다.

```
let obj = {nine: 999};  
obj[Symbol("one")] = 111;  
obj[Symbol("two")] = "222";  
console.log(obj);  
  
for (var key in obj){  
    console.log(key);  
};
```



```
▼ Object ⓘ  
  nine: 999  
  Symbol(one): 111  
  Symbol(two): "222"  
  ► __proto__: Object  
nine
```

# symbol Object

- Symbol 사용 형태
  - 클래스의 메서드 이름으로 Symbol 사용이 가능하다.

```
const symbolOne = Symbol("symbol one");
const symbolTwo = Symbol("symbol two");

class Sports {
  static [symbolOne]() {
    return "Symbol-1";
  }
  [symbolTwo]() {
    return "Symbol-2";
  }
}
console.log(Sports[symbolOne]());

let obj = new Sports();
console.log(obj[symbolTwo]());
```



Symbol-1

Symbol-2

# symbol Object

- Symbol 사용 형태
  - JSON.stringify()에서 Symbol 사용
    - 변환 대상에 [[sym]:"값"] 과 같이 Symbol 값을 symbol-keyed 프로퍼티로 작성한다음 이 상태에서 JSON.stringify()를 실행하면 프로퍼티 키와 프로퍼티 값이 문자열로 변환되지 않고 빈 Object가 반환된다.

```
let sym = Symbol("key");  
let result = JSON.stringify({[sym]: "값"});  
  
console.log(result);
```



{}

# symbol property

- toStringTag

- [object Object] 상태에서 Object를 Symbol.toStringTag 값으로 표시한다.

구분	타입	데이터(값)
형태		Symbol.toStringTag
파라미터	String	
반환	String	프로퍼티 값 또는 return에서 변환된 값

# symbol property

## ○ toStringTag

```
"use strict";
debugger;

let Sports = function(){};
let sportsObj = new Sports;
console.log(sportsObj.toString());

Sports.prototype[Symbol.toStringTag] = "Sports-Function";
console.log(sportsObj.toString());

let dummy;
```

[object Object]

[object Sports-Function]

# symbol property

## ○ toStringTag

```
class Book {};  
let bookObj = new Book();  
console.log(bookObj.toString());  
  
class Sports {  
  get [Symbol.toStringTag]() {  
    return "Sports-class";  
  }  
};  
let sportsObj = new Sports();  
console.log(sportsObj.toString());  
console.log(Map.prototype[Symbol.toStringTag]);
```

[object Object]

[object Sports-class]

Map

Map은 빌트인 오브젝트이며 Map 안에  
Symbol.toStringTag로 Map이 선언되어 있다

# symbol property

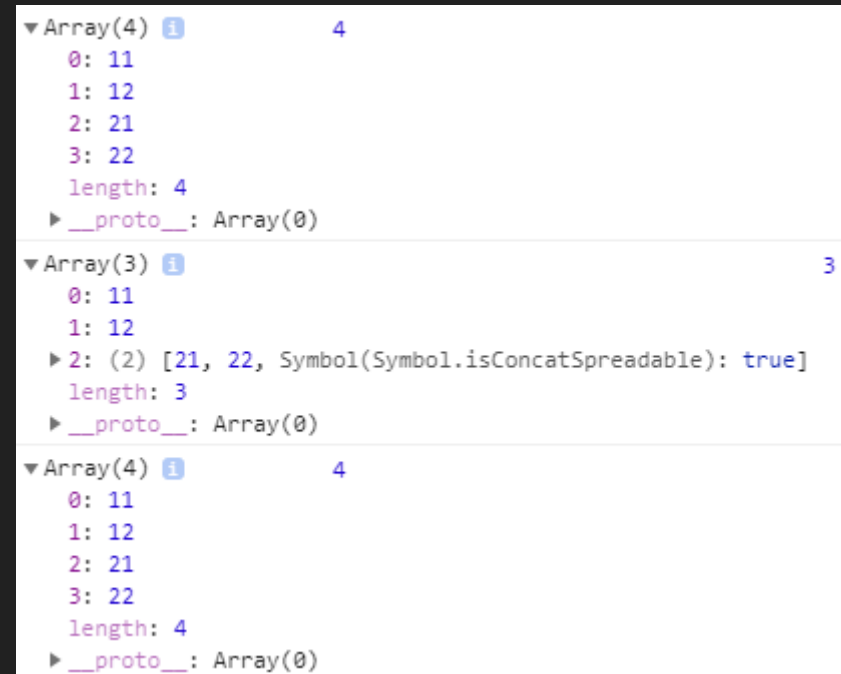
- isConcatSpreadable

- Array 오브젝트의 concat()에서 배열을 결합할 때 결합하는 배열의 펼침 여부를 지정한다.

```
let one = [11, 12], two = [21, 22];
let result = one.concat(two);
console.log(result, result.length);

two[Symbol.isConcatSpreadable] = false;
result = one.concat(two);
console.log(result, result.length);

two[Symbol.isConcatSpreadable] = true;
result = one.concat(two);
console.log(result, result.length);
```



```
▼ Array(4) 4
  0: 11
  1: 12
  2: 21
  3: 22
  length: 4
  ▶ __proto__: Array(0)

▼ Array(3) 3
  0: 11
  1: 12
  2: (2) [21, 22, Symbol(Symbol.isConcatSpreadable): true]
  length: 3
  ▶ __proto__: Array(0)

▼ Array(4) 4
  0: 11
  1: 12
  2: 21
  3: 22
  length: 4
  ▶ __proto__: Array(0)
```

# symbol property

- isConcatSpreadable
  - Array-like 오브젝트에서의 사용

```
let one = [11, 12];
let fiveSix = {
  0: "five",
  1: "six",
  length: 2
};
let result = one.concat(fiveSix);
console.log(result, result.length);

let arrayLike = {
  [Symbol.isConcatSpreadable]: true,
  0: "five",
  1: "six",
  length: 2
};
result = one.concat(arrayLike);
console.log(result, result.length);
```

```
▼ Array(3) ⓘ 3
  0: 11
  1: 12
  ▶ 2: {0: "five", 1: "six", length: 2}
     length: 3
  ▶ __proto__: Array(0)

▼ Array(4) ⓘ 4
  0: 11
  1: 12
  2: "five"
  3: "six"
  length: 4
  ▶ __proto__: Array(0)
```



# symbol property

- unscopables
  - with문에서 사용하며 값이 true이면 프로퍼티를 전개하지 않는다

축구 야구

축구

baseball is not defined

```
// "use strict"를 선언하면 with에서 에러 발생
// "use strict";
debugger;

let sports = {
  soccer: "축구",
  baseball: "야구"
};
with(sports){
  console.log(soccer, baseball);
};

sports[Symbol.unscopables] = {
  baseball: true
};

try {
  with (sports) {
    console.log(soccer);
    let value = baseball;
  }
} catch (e) {
  console.log(e.message);
};

let dummy;
```

# symbol property

## ○ species

- Symbol.species는 constructor를 반환한다.
- 다시 이야기 하면 constructor로 인스턴스를 생성하여 반환하는 것과 같다.
- Symbol.species를 오버라이드 할 수 있으며 개발자 코드로 반환되는 인스턴스를 변경할 수 있다.
- 결국 인스턴스의 메서드를 호출했을 때 인스턴스를 반환하도록 하는 것이 Symbol.species이다.

구분	타입	데이터(값)
형태		Symbol.species
파라미터		없음
반환	Instance	생성한 인스턴스

# symbol property

## ○ species

- Symbol.species는 static 액세스 프로퍼티로 getter만 있고 setter 는 존재하지 않는다.
- Array, Map, Set, Promise, RegExp, ArrayBuffer, TypedArray 오브젝트에 Symbol.species가 빌트인으로 포함되어 있다.
- 위의 빌트인 오브젝트를 상속받은 클래스에 Symbol.species를 작성하면 빌트인 오브젝트의 Symbol.species가 오버라이드 된다.
- 이를 통해 Symbol.species 에서 다른 오브젝트를 반환할 수 있다.

# symbol property

## ○ species

```
class ExtendArray extends Array {  
  static get [Symbol.species]() {  
    return Array;  
  }  
};  
let oneInstance = new ExtendArray(1, 2, 3);  
let twoInstance = oneInstance.slice(1, 2);  
console.log(oneInstance instanceof ExtendArray);  
console.log(twoInstance instanceof Array);  
console.log(twoInstance instanceof ExtendArray);  
console.log(oneInstance);  
console.log(twoInstance);
```

```
true  
true  
false  
▼ ExtendArray(3) ⓘ  
  0: 1  
  1: 2  
  2: 3  
  length: 3  
  ▶ __proto__: Array  
▼ Array(1) ⓘ  
  0: 2  
  length: 1  
  ▶ __proto__: Array(0)
```

# symbol property

- species
  - 다른 클래스의 반환

```
▼ ExtendOne(2) ⓘ  
  0: 20  
  1: 30  
  length: 2  
  ▶ __proto__: Array
```

ExtendOne

undefined

```
class ExtendOne extends Array{  
  showOne(){  
    console.log("ExtendOne");  
  }  
};  
class ExtendTwo extends Array{  
  static get [Symbol.species]() {  
    return ExtendOne;  
  }  
  showTwo(){  
    console.log("ExtendTwo");  
  }  
};  
let twoInst = new ExtendTwo(10, 20, 30);  
let threeInst = twoInst.filter(value => value > 10);  
console.log(threeInst);  
  
threeInst.showOne();  
console.log(threeInst.showTwo());
```

# symbol property

- species
  - null 반환

```
class ExtendOne extends Array{
  static get [Symbol.species]() {
    return null;
  }
};

let oneInst = new ExtendOne(10, 20, 30);
let arrayInst = oneInst.filter(value => value > 10);

console.log(arrayInst instanceof Array);
console.log(arrayInst instanceof ExtendOne);
```

true

false

# symbol property

- species

- null 반환

- oneInst.filter()를 호출하면 ExtendOne 클래스의 [Symbol.species]()가 호출되며 null을 반환할거라 생각하지만 그렇지 않고 디폴트 [Symbol.species]()가 호출된다.
    - 상속받은 Array 오브젝트의 [Symbol.species]()가 호출되며 Array 인스턴스를 생성하여 반환한다.

# symbol property

- toPrimitive
  - 오브젝트를 프리미티브 타입으로 반환한다.

구분	타입	데이터(값)
형태		Symbol.toPrimitive()
파라미터	String	프리미티브 값 변환 힌트
반환	Any	(선택) 반환 값



# symbol property

## ○ toPrimitive

- Symbol.toPrimitive() 에서 값을 반환하는 기준은 이를 호출하는 형태에 따라 결정된다.
- 엔진은 호출한 곳의 형태에 따라 Symbol.toPrimitive(hint) 파라미터에 세 가지 모드를 설정한다.
- 개발자가 작성하는 것이 아닌 엔진이 다음 기준으로 설정한다.
  1. Number 환경이면 "number"를 toPrimitive(hint) 파라미터에 설정한다.
  2. String 환경이면 "string"을 toPrimitive(hint) 파라미터에 설정한다.
  3. Number와 String 환경이 아니면 "default" 를 toPrimitive(hint) 파라미터에 설정한다.

# symbol property

## ○ toPrimitive

```
let obj = {
  [Symbol.toPrimitive](hint){
    if (hint === "number"){
      return 30;
    };
    if (hint === "string"){
      return "문자열";
    };
    return "디폴트";
  }
};
console.log("1:", 20 + obj);
console.log("2:", 20 * obj);

console.log("3:", obj + 50);
console.log("4:", +obj + 50);
console.log("5:", `${obj}` + 123);
```

1:	20디폴트
2:	600
3:	디폴트50
4:	80
5:	문자열123

# symbol property

## ○ Iterator

- 이터레이터 오브젝트를 생성하여 반환할 때 쓴다.

구분	타입	데이터(값)
형태		Symbol.Iterator()
파라미터		없음
반환	Iterator	이터레이터 오브젝트

- Symbol.Iterator 는 String, Array, Map, Set, TypedArray 오브젝트의 prototype 에 연결되어 있다.
- 해당 오브젝트의 Symbol.Iterator 를 호출하면 Iterator 오브젝트를 생성하여 반환한다.
- Object 에는 Symbol.Iterator 가 없지만 개발자 코드로 구현할 수 있다.

# symbol property

- Iterator

- Array.prototype[Symbol.iterator]

```
let numberArray = [10, 20];
for (let value of numberArray){
  console.log(value);
};
let iteratorObj = numberArray[Symbol.iterator]();

console.log(iteratorObj.next());
console.log(iteratorObj.next());
console.log(iteratorObj.next());
```

```
10
20
▼ Object i
  done: false
  value: 10
  ► __proto__: Object
▼ Object i
  done: false
  value: 20
  ► __proto__: Object
▼ Object i
  done: true
  value: undefined
  ► __proto__: Object
```

# symbol property

- Iterator
  - String.prototype[Symbol.iterator]

```
let stringValue = "1A";
for (let value of stringValue) {
  console.log(value);
}
let iterObj = stringValue[Symbol.iterator]();

console.log(iterObj.next());
console.log(iterObj.next());
console.log(iterObj.next());
```

```
1
A
▼ Object i
  done: false
  value: "1"
  ► __proto__: Object
▼ Object i
  done: false
  value: "A"
  ► __proto__: Object
▼ Object i
  done: true
  value: undefined
  ► __proto__: Object
```

# symbol property

- Iterator
  - Object 이터레이션
    - Object 에는 Symbol.iterator 가 없다.
    - 때문에 for~of 문 사용이 불가능하지만 Object 에 Symbol.iterator 를 작성하면 반복처리 가능하다.

```
▼ Object 1
  done: false
  value: 0
  ▶ __proto__: Object

▼ Object 1
  done: false
  value: 1
  ▶ __proto__: Object

▼ Object 1
  done: true
  value: undefined
  ▶ __proto__: Object
```

```
let obj = {
  [Symbol.iterator]() {
    return {
      maxCount: 2,
      count: 0,
      next() {
        if (this.count < this.maxCount) {
          return {value: this.count++, done: false};
        }
        return {value: undefined, done: true};
      }
    }
  }
};

let iteratorObj = obj[Symbol.iterator]();

console.log(iteratorObj.next());
console.log(iteratorObj.next());
console.log(iteratorObj.next());
```

# symbol property

## ○ Generator

- Object 에 Symbol.iterator 를 제너레이터 함수로 작성하면, 이터레이터로 반복 할 때마다 yeild를 수행 가능하다.

```
let obj = {};  
obj[Symbol.iterator] = function*(){  
  yield 10;  
  yield 20;  
  yield 30;  
};  
let result = [...obj];  
console.log(result);
```

```
▼ Array(3) ⓘ  
  0: 10  
  1: 20  
  2: 30  
  length: 3  
  ▶ __proto__: Array(0)
```

# symbol property

## ○ Generator

- Object 에 Symbol.iterator 를 제너레이터 함수로 작성하면, 이터레이터로 반복 할 때마다 yeild를 수행 가능하다.

```
let gen = function*() {  
  yield 10;  
  yield 20;  
};  
let genObj = gen();  
console.log(genObj.next());  
  
let iteratorObj = genObj[Symbol.iterator]();  
console.log(iteratorObj.next());
```

```
▼ Object ⓘ  
  done: false  
  value: 10  
  ▶ __proto__: Object  
  
▼ Object ⓘ  
  done: false  
  value: 20  
  ▶ __proto__: Object
```



# symbol property

- match

- match 결과를 반환한다.

구분	타입	데이터(값)
object	Symbol	Symbol.match()
파라미터	any	Symbol.match()에서 사용하려는 형태
반환	any	Symbol.match()에서 반환한 값

- String 오브젝트에서 정규 표현식을 사용할 수 있는 메서드는 match(), replace(), search(), split() 이 있다.
  - Symbol 오브젝트에 이에 대응하는 Symbol.match(), Symbol.replace(), Symbol.search(), Symbol.split() 이 있다.

# symbol property

- match

- `Symbol.match()` 는 `String.prototype.match()` 대신 호출된다.
- `Symbol.match()` 에 개발자 코드를 작성할 수 있으므로 해당 기능을 좀 더 다양하게 작성할 수 있다.
- 하지만 `match` 방법과 기준은 같아야 한다.
- `String.prototype.match()`가 호출되면 먼저 오브젝트에서 `Symbol.match` 작성 여부를 체크한다.
- 존재하면 오브젝트의 `Symbol.match()`를 호출하고, `String.prototype.match()`는 호출하지 않는다.

# symbol property

## ○ match

```
console.log("1", "Sports".match(/s/));

class MatchCheck {
  constructor(base) {
    this.base = base;
  }
  [Symbol.match](target) {
    return this.base.indexOf(target) >= 0;
  }
}

let instMatch = new MatchCheck("sports");
console.log("2:", "po".match(instMatch));
```

```
1 ▼ Array(1) ⓘ
   0: "s"
   groups: undefined
   index: 5
   input: "Sports"
   length: 1
   ▶ __proto__: Array(0)
2: true
```

# symbol property

## ○ match

```
try {  
  "ABC".includes(/ABC/);  
} catch (e) {  
  console.log("정규 표현식 작성 불가");  
}  
  
let regexpObj = /ABC/;  
regexpObj[Symbol.match] = false;  
  
console.log("/ABC/".includes(regexpObj));
```

정규 표현식 작성 불가

true

# symbol Method

- for()
  - 글로벌 Symbol 레지스트리에 Symbol 값을 저장한다.

구분	타입	데이터(값)
형태		Symbol.for()
파라미터	String	(선택) Symbol key
반환	Symbol	검색된 Symbol, 검색되지 않으면 Symbol값 생성, 저장, 반환

# symbol Method

## ○ for()

- Symbol.for()는 글로벌 Symbol 레지스트리에 {key:value} 형태로 저장한다.
- 파라미터에 지정한 문자열이 key가 되고 생성한 Symbol 값이 value가 된다.
- 글로벌 Symbol 레지스트리에 이미 key가 등록되어 있으면 Symbol값을 생성하지 않고 등록된 value 값을 반환한다.
- 글로벌 Symbol 레지스트리는 Symbol 값을 공유하기 위한 영역이다.
- Symbol은 다음과 같이 세 가지 형태로 사용할 수 있다.
  - Symbol() : Symbol 값을 생성하며 스코프 안에서 사용한다.
  - Symbol.for() : 글로벌 Symbol 레지스트리에 저장되며 전체 프로그램에서 사용한다.
  - Well-Known Symbol : 빌트인 Symbol 프로퍼티로 오버라이드하여 기능을 추가, 변경할 수 있다.

# symbol Method

## ○ for()

```
console.log(Symbol.for("sports"));  
console.log(Symbol.for("sports"));  
  
console.log(Symbol.for("ABC") === Symbol.for("ABC"));  
console.log(Symbol.for("DEF") === Symbol("DEF"));  
console.log(Symbol.for(true));
```

Symbol(sports)

Symbol(sports)

true

false

Symbol(true)

# symbol Method

- keyFor()
  - 글로벌 Symbol 레지스트리에서 프로퍼티 키 값을 반환한다.

구분	타입	데이터(값)
형태		Symbol.keyFor()
파라미터	Symbol	(선택) 검색할 Symbol
반환	String	검색된 Symbol key 값, 아니면 undefined

```
let symOne = Symbol.for("123");  
console.log(Symbol.keyFor(symOne));  
  
let symTwo = Symbol("222");  
console.log(Symbol.keyFor(symTwo));
```

123

undefined



# symbol Method

- toString()
  - Symbol을 문자열로 변환하여 반환한다

구분	타입	데이터(값)
형태		Symbol.prototype.toString()
파라미터		파라미터 없음
반환	String	변환된 문자열

```
console.log("1:", Symbol("123").toString());  
console.log("2:", Symbol.for("ABC").toString());  
console.log("3:", Symbol.iterator.toString());
```

```
1: Symbol(123)  
2: Symbol(ABC)  
3: Symbol(Symbol.iterator)
```

# symbol Method

- `valueOf()`
  - Symbol로 생성한 값을 반환한다.

구분	타입	데이터(값)
형태		<code>Symbol.prototype.valueOf()</code>
파라미터		파라미터 없음
반환	String	프리미티브 값

```
console.log(Symbol("123").valueOf());  
console.log(Symbol.for("789").valueOf());
```

`Symbol(123)`

`Symbol(789)`

# symbol Method

- `getOwnPropertySymbols()`
  - Symbol 프로퍼티를 배열로 반환한다.

구분	타입	데이터(값)
형태		<code>Object.getOwnPropertySymbols()</code>
파라미터	Object	Object, 추출대상
반환	Array	Symbol 프로퍼티

# symbol Method

## ○ `getOwnPropertySymbols()`

```
let bookObj = {book: 123};
bookObj[Symbol("one")] = 10;
bookObj[Symbol.for("two")] = 20;

let names = Object.getOwnPropertyNames(bookObj);
console.log("1:", names);

let symbolList = Object.getOwnPropertySymbols(bookObj);
console.log("2:", symbolList);

for (let sym of symbolList){
  console.log(sym.toString(), bookObj[sym]);
}

let emptyList = Object.getOwnPropertySymbols({});
console.log("5:", emptyList.length);
```

```
1: ▼ Array(1) ⓘ
   0: "book"
   length: 1
   ▶ __proto__: Array(0)

2: ▼ Array(2) ⓘ
   0: Symbol(one)
   1: Symbol(two)
   length: 2
   ▶ __proto__: Array(0)

Symbol(one) 10
Symbol(two) 20
5: 0
```

# symbol Method

- JSON.stringify()
  - 자바스크립트 형태를 JSON 형태의 문자열로 변환한다.

구분	타입	데이터(값)
형태		JSON.stringify()
파라미터	String	변환 대상
	Function	(선택) 함수 또는 배열
	String	(선택) 가독성을 위한 구분자
반환	String	변환된 문자열

# symbol Method

- JSON.stringify()
  - JSON.stringify()로 자바스크립트의 형태의 {key:value}를 JSON 형태의 문자열로 변환하면, Symbol-keyed 프로퍼티로 작성한 Symbol이 변환에서 제외된다.
  - Symbol값을 외부에 노출시키지 않으려는 의도이지만, 에러가 나지 않으므로 주의가 필요하다.

```
let result = JSON.stringify({[Symbol("one")]: "1"});  
console.log(result);  
console.log(typeof result);  
  
console.log(JSON.stringify({[Symbol.for("two")]: "2"}));
```

{}
string
{}

# symbol Method

- JSON.stringify()
  - 주석을 프로퍼티 키 값으로 활용

```
▼ Array(2) ⓘ  
  0: Symbol(one)  
  1: Symbol(two)  
  length: 2  
  ▶ __proto__: Array(0)  
{"one":10,"two":20}
```

```
let bookObj = {};  
bookObj[Symbol("one")] = 10;  
bookObj[Symbol.for("two")] = 20;  
  
let symbolList = Object.getOwnPropertySymbols(bookObj);  
console.log(symbolList);  
  
let first, second, key, keyValue = {};  
for (let sym of symbolList){  
  key = Symbol.keyFor(sym);  
  if (key){  
    keyValue[key] = bookObj[sym];  
  } else {  
    //Symbol(one)  
    first = /^Symbol\([^\)]*\)/[Symbol.replace](sym.toString(), "");  
    second = /\[\]\$/[Symbol.replace](first, "");  
    keyValue[second] = bookObj[sym];  
  }  
};  
console.log(JSON.stringify(keyValue));
```