[JS Deep Dive] 21장 . 빌트인 객체

Status	진행 중

자바스크립트 객체의 분류

표준 빌트인 객체 (네이티브 객체)

- Ecmascript 사양에 정의된 객체, 애플리 케이션 전역의 공통 기능을 제공, 표준 빌트인 객체는 전역객체의 프로퍼티로서 제공됨. 따라서 선언 없이 전역 변수처럼 언제나 참조가능

호스트 객체

- ecma 사양에 정의되어 있지 않지만, 자바스크립트 실행환경 (호스트환경. 브라우저, node.js) 에서 추가로 제공하는 객체

사용자 정의 객체

- 사용자가 직접 정의한 객체

표준 빌트인 객체

> - 자바 스크립트는 object, string 등 40여개의 표준 빌트인 객체를 제공한다.
> -개발자는 별도의 선언 없이 사용할수 있다.
> -이 객체들은 다양한 기능 (수학연산, 데이터 처리, 문자열 다루기) 등을 제공한다.
> -전역객체의 일부로 포함된다.

▼ 표준 빌트인 객체

Object, Array , String, Number 등...

- Math, Reflect, JSON
  - 인스턴스를 생성할수 없음.
  - 정적 메소드만 제공
- 나머지 표준 빌트인 객체
  - 인스턴스를 생성할수 있는 생성자 함수 객체
  - 프로토타입 메서드와 정적 메소드를 제공

- 정적메소드와 프로토 타입 메소드
  - 정적 메소드
    - 클래스나 객체 자체에 붙어있는 메소드
    - 클래스의 인스턴스가 아니라 클래스 자체에서 호출됨.
    - 예시 코드) Math 객체

```
console.log(Math.max(1, 2, 3)); // 3
console.log(Math.random());    // 0과 1 사이의 난수

// Math의 메서드는 모두 정적 메서드입니다.
const math = new Math(); // TypeError: Math is not a co
```

  - 프로토타입 메소드
    - 클래스의 인스턴스에 의해서 호출되는 메소드
    - 클래스의 prototype 객체에 정의되어, 해당 클래스의 인스턴스가 이 메소드를 상속받음.

    - 클래스를 정의하면 자동으로 prototype 객체가 생성되고, 여기에 우리가 정의한 메소드가 붙음.

- 두가지 구별하기

```
//정적 메소드: 클래스나 객체 자체의 유틸리티 기능 제공
Array.isArray([1, 2, 3]); // 정적 메서드

//프로토타입 메서드: 인스턴스가 고유하게 사용할 수 있는 기능 제공.
const arr = [1, 2, 3];
arr.push(4); // 프로토타입 메서드
```

▼ 인스턴스란?

- 클래스라는 설계도를 바탕으로 컴퓨터 저장공간에서 할당된 실제 데이터 구조 그자체임
- class 키워드로 클래스를 정의하고, new 키워드를 사용하여 이미 정의한 클래스를 기반으로 생성
- 간단한 경우에는 지금껏 우리가 배워왔던 객체 리터럴 방식을 사용해도 되지만,

  복잡하거나 재사용이 필요한 경우에는 클래스가 적합함.

▼ 코드 예제

```
class School {
  constructor(name, location) {
    this.name = name; // 학교 이름
    this.location = location; // 위치
    this.students = []; // 학생 리스트
  }

  addStudent(studentName) {
    this.students.push(studentName); // 학생 추가
    console.log(`${studentName} 학생이 ${this.name}에 등록
  }

  listStudents() {
    console.log(`${this.name}의 학생 명단: ${this.students
```

```
  }
}
// 서울고등학교와 부산중학교 인스턴스 생성
const school1 = new School("서울고등학교", "서울");
const school2 = new School("부산중학교", "부산");

// 학생 등록
school1.addStudent("철수"); // 출력: 철수 학생이 서울고등학교에
school1.addStudent("영희"); // 출력: 영희 학생이 서울고등학교에
school2.addStudent("민수"); // 출력: 민수 학생이 부산중학교에

// 학생 명단 출력
school1.listStudents(); // 출력: 서울고등학교의 학생 명단: 철수
school2.listStudents(); // 출력: 부산중학교의 학생 명단: 민수
```

원시값과 래퍼 객체

> 원시값을 객체처럼 접근하면 생성되는 임시 객체를 의미

- 래퍼객체
  - 자바스크립트 엔진은 원시값인 문자열, 숫자, 불린 값은 객체가 아니기때문에 프로퍼티나 메소드를 가질수 없는데, 마치 객체처럼 일시적으로 원시값을 연관된 객체로 변환해줌.
  - 엔진은 암묵적으로 연관된 객체를 생성, 생성된 객체로 메소드 , 프로퍼티에 접근한 뒤에 다시 원시값으로 돌린다.
  - 원시값은 자바 스크립트 엔진에 의해 생성되는 래퍼 객체에 의해 마치 객체처럼 사용할수 있음
  - 표준 빌트인 객체의 프로토 타입 메서드, 또는 프로퍼티를 참조할수 있음
  - 따라서 굳이 생성자 함수를 new 연산자와 함께 호출하여 문자열, 숫자, 불린 인스턴스를 생성할 필요가 없으며, 권장하지도 않는다.

- 그러나 문자열, 숫자, 불린, 심벌 이외의 원시값인 null, undefined는 래퍼 객체를 생성하지 않는다.

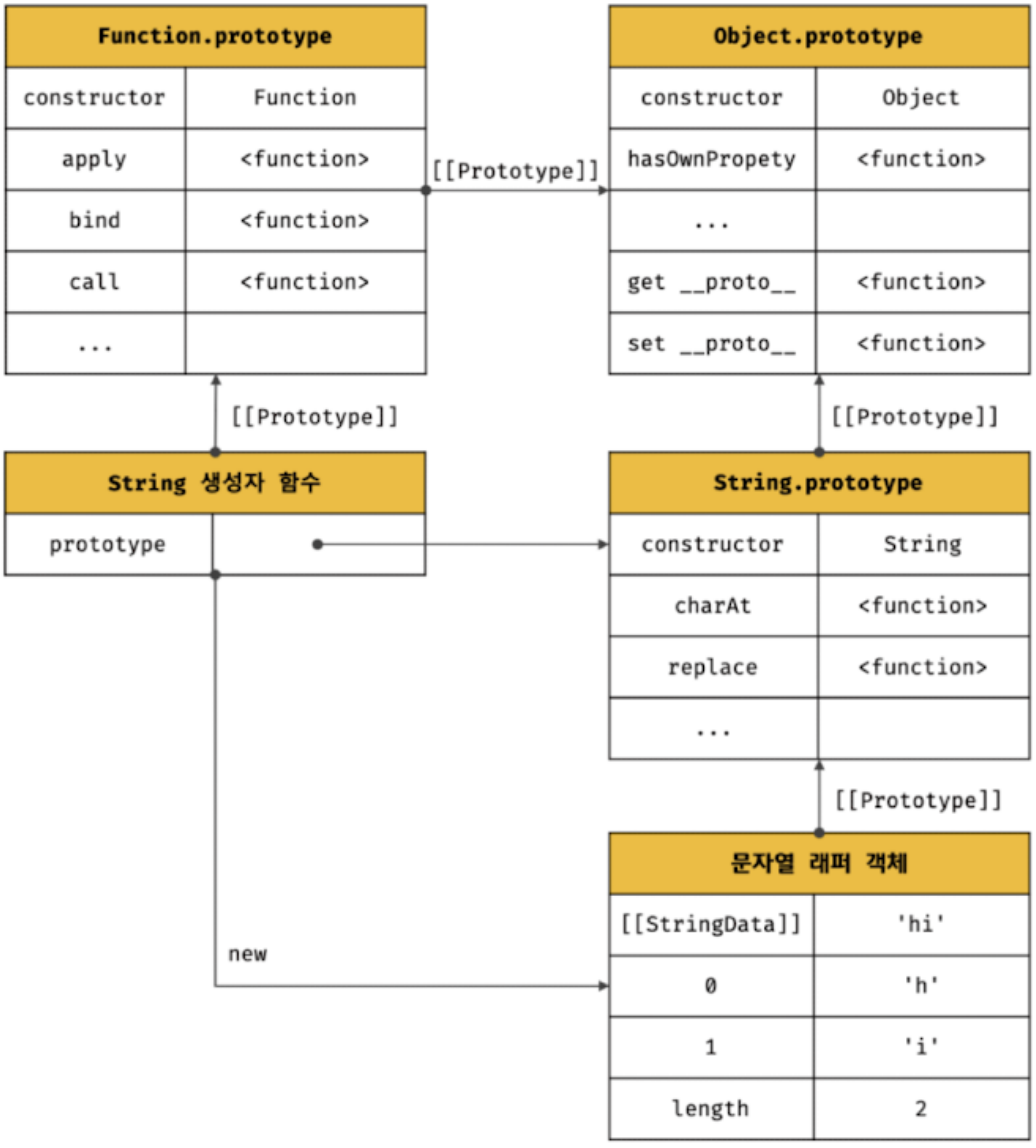

그림 21-1 문자열 래퍼 객체의 프로토타입 체인

- 래퍼 객체의 처리가 종료되면 문자열 래퍼객체의 [[stringData]] 에 할당되어있는, 식별자가 원시값을 갖도록 되돌리고 래퍼 객체는 가비지 컬렉션의 대상이 된다.

```
const str = "hello";
console.log(str.toUpperCase());
```

```
/*자바스크립트의 프로토 체인
*1. 자바스크립트는 특정 프로퍼티에 접근할때 해당 객체에 프로퍼티가 없는것을
*2. 내부 슬롯의 참조를 따라 상위 프로토타입의 프로퍼티를 순차적으로 검색
*3. 원시값은 생성자 함수로 생성된 객체가 아니므로 어느곳에서도 찾지못하고 읍
*/
'HELLO' 메소드를 호출할때 일시적으로 stirng 객체로 변환
console.log(typeof str); //string 메소드 호출후 원시값으로 돌아갔으므
```

▼ 가비지 컬렉션이란? (garbage collection, GC.)

- 메모리 관리 방법중 하나로, 프로그래머가 동적으로 할당한 메모리중 더이상 쓰지 않는 영역을 자동으로 찾아내어 해제하는 기능
- 프로그램을 개발하다보면 유효하지 않은 메모리인 가비지가 발생. JavaScript의 엔진은 불필요한 메모리를 알아서 정리해준다.

전역객체

> - 자바스크립트 엔진에 의해 어떤 객체보다도 먼저 생성되는 특수한 객체,
> - 어떤 객체에도 속하지 않은 최상위 객체
> - 의도적으로 생성할수 없으므로 생성자 함수가 제공되지 않는다
> - 전역객체의 프로퍼티를 참조할때 window(or global)을 생략할 수 있다.
> - 모든 자바스크립트 코드는 하나의 전역 객체를 공유한다.
>
> →여러개의 script 태그를 통해 코드를 분리해도 하나의 전역을 공유함.

- 전역객체 살펴보기

```
console.log(window);// 전역개체 출력

console.log(Object.keys(window)); // window 객체의 모든 속성 이름
```

- globalThis
  - ECMAScript 2020에서 도입되었다.
  - 브라우저, node.js 환경에서 전역객체를 가리키던 식별자들을 통일한 식별자.

- 전역객체의 프로퍼티
  - 표준빌트인 객체
  - 호스트 객체
  - var 키워드로 선언한 전역 변수와 전역 함수 , 암묵적 전역

- 전역 객체의 프로퍼티의 종류

  ▼ 가장 많이 사용되는 프로퍼티 2가지

  1. console
     a. 전역객체의 프로퍼티중 하나로, 디버깅과 관련된 메소드를 제공
     b. 브라우져와 Node.js 환경 모두에서 사용가능.
     c. 개발자가 정보를 출력하거나 디버깅을 할때 유용
     d. 가장 많이 사용하는 메소드 console.log / console.error, console.warn / console.info 등이 있음.
  2. document (호스트객체)
     a. 브라우져에서 제공하는 DOM 을 조작하기 위한 핵심 객체
     b. 이를 통해 HTML 요소를 검색, 추가, 수정할수 있음.
     c. 가장 많이 사용하는 메소드 getElementById/ getElementByTagName 등이 있음.
  3. 그밖에 Math, JSON 등 수많은 프로퍼티가 존재

- 전역객체의 메소드

▼ 메소드

1. setTimeout
    a. 일정 시간이 지난후에 지정된 작업을 실행하는데 사용.
    b. 비동기 작업이나 딜레이를 구현할때 자주 활용됨.
2. `setInterval` / `clearInterval`
    a. 일정 간격으로 작업을 반복하거나 중단할때 사용

```
const interval = setInterval(() => console.log("Ping
clearInterval(interval); // 반복 취소
```

▼ 그밖의 메소드

1. eval
2. isFinite
3. isNaN
4. parseFloat
5. endocdURI / decodeURI
6. endoceURIComponent / decodeURIComponent

암묵적 전역

> 선언하지 않은 식별자 이지만, 이 식별자 에 값을 할당하면 전역 객체의 프로퍼티가 되기 때문에 마치 선언된 전역 변수처럼 동작하는것을 의미.

```
var x = 10 //전역변수
function foo() {
y = 20 //선언하지 않은 식별자에 값을 할당 window.y =20
```

```
}
foo()

//선언하지 않은 식별자 y를 전역에서 참조할 수 있다.
console.log(x+y) //30
```

암묵적 전역이 일어나는 엔진 동작원리

1. foo 함수가 호출되면 자바 엔진은 y 변수에 값을 할당하기 위해 스코프 체인을 통해 선언된 변수 인지 확인.
2. y 변수의 선언을 찾을수 없으므로 에러가 발생함.
3. 하지만 엔진은 y =20을 window.y = 20으로 해석하여 전역객체에 프로퍼티를 동적 생성

   (하지만 y는 단지 전역객체의 프로퍼티로 추가 되었을뿐 y는 변수가 아니므로,

   호이스팅 또한 발생하지 않고, delete 연산자로 삭제할 수 없다)

암묵전 전역 막기

- 암묵적 전역은 코드의 혼란을 야기할수 있기 때문에 '엄격모드' 로 이를 방지하는것이 좋다.

```
"use strict" // 1. 파일의 최상단에 작성하여 파일 전체에 엄격모드  적용ㅎ
function example() {
    "use strict"; // 2. 함수 내에서만 엄격모드 적용학.
    undeclaredVar = 42; // ReferenceError 발생
}
example();

//엄격모드를 설정하지 않았을때.
function anotherExample() {
    undeclaredVar = 42; // 정상 실행 (엄격 모드 미적용)
}
anotherExample();
```

Garbage_collection- MDN 문서

strictMode - Modern Javascript Deep Dive '20장 strictMode' 참고