



Kakao Cloud😊

성함 : 문성훈 강사님

Email: moon9342@gmail.com

커리큘럼: ECMA Script → React → JAVA → DB → Servlet → Spring Boot

JavaScript = prototype 기반의 함수형 언어

JavaScript = ECMA Script + Client Side Web API(DOM, BOM)

Host API

ECMA Script = ES6 [ES5 대격변(언어, 기능)]

JQuery = 라이브러리를 사용(로직을 자신이 짠다) → 유지보수에 문제가 생김

HTML = 정형성문제(문법이 틀려도 작동)

확장성문제(태그가 정해져 있어서 유저가 태그를 만들지 못함)

XHTML 1.0 = HTML문제를 해결하기 위해 XML 추가(W3C 표준)

WHATWG = HTML 자체를 발전시킴 → HTML 5(표준화)(애플, 구글 참여)

HTML 5 = 브라우저 안에서 동작하는 웹 어플리케이션을 만드는것을 지향

HTML 5 = HTML (25%) + CSS 3 (5%) + JavaScript Web API (70%)

Package Manager : npm (모듈 간 버전을 맞춰준다)

TransPiler : Type Script

자동화 도구 : Build, Test, Deploy 자동화 → Gulp

모듈화 : WebPack

FrameWork : Angular, React, Vue.js (React !)

JAVA Script 실행하려면 JAVA Script Engine이 필요하다.(browser안에 내장)

Chrome → V8 Engine

Node.js → (V8 Engine)

Edge → ChaKra Engine

Safari → WebKit Engine

Firefox → Spider Monkey Engine

변수 = 해당 메모리 주소를 가지고 있는 식별자

식별자가 가지고 있는 값 = 메모리 주소

var → function level scope(함수를 기준으로 스코프를 잡는다.)

const, let → block level scope (블록을 기준으로 스코프를 잡는다.)

hoisting = 아래에서 선언한 변수가 위로 올라간다.

JAVA는 2패스로 동작(먼저 변수들은 다 찾아내고 메모리를 할당)

→ 변수 선언은 runtime 이전에 수행

→ 마치 변수선언이 해당 scope의 최상단으로 끌어 올려진것 같은 현상

VAR KEYWORD로 변수선언시 주의점

같은 변수의 이름이 존재하면 나중에 나오는 변수를 없는것처럼 여긴다.

VAR KEYWORD가 키워드가 없는것처럼 동작

초기화 구문이 없으면 무시, CODE ERROR가 안난다.

VAR 변수 스코프 주의 !

let

1. 중복선언 X

2. block-level scope를 생성

3. let을 이용하면 hoisting이 안되는 것처럼 보인다.(선언과 동시에 초기화 X)

4. let keyword로 변수를 선언하면 변수를 일시적으로 사용할 수 없는 구간 ⇒ Temporal Dead Zone이 생길 수 있다.

const

상수를 선언할 때 사용(변수에 대한 재할당 금지)

원시값은 불변의 특성을 가진다.

기본적인 용어정리

1.literal

→ 특정값을 표현하기 위해서 사람이 이해할 수 있는 문자, 약속된 기호를 이용하는 notation(표기법)

ex) 3 ⇒ 숫자 literal

'hello' , "hello", `hello` => 문자 literal

null → 값이 없다는 상태를 나타내는 null literal(개발자가 직접사용하는 값)

undefined → 초기화 되지 않은 변수를 나타내는 literal (JAVA Script Engine)

{'name':'kim'} → 객체 literal

[1,2,3] → 배열 literal

function(){ } → 함수 literal

2.statement(문)

→ 프로그램의 최소 실행 단위

→ 프로그램은 statement의 집합

→ ; (세미콜론) statement의 종결을 나타내는 표현

→ ASI 기능이 탑재되어 있다. (**Automatic Semicolon Insertion**) ⇒ 자동 세미콜론

→ 블록({},[])은 self closing기능을 가지고있다.

3.expression(식)

→ 평가과정을 통해 값으로 인식되는 구문

Data Type

– 총 7개의 Data Type이 있다.

1. Number Type → 정수, 실수를 구분하지 않는다.

→ 내부적으로 모든 숫자는 64-bit 실수로 처리한다.

→ Infinity : 양의 무한대

→ -Infinity : 음의 무한대

→ NaN : Not a Number(숫자가 아니라는 의미지만 NaN은 숫자 Type), 산술연산 불가

2. String Type : ''(기본형태), `` (백틱)

→ Multi Line 문자열 표현 가능

→ Expression Interpolation(표현식 삽입)

3. Boolean Type : true, false

4. Undefined Type : Undefined

5. Null Type : Null

6. Symbol Type (ES6) : Symbol 값(Unique)

→ 눈으로 확인할 수 없다(출력 X)

→ Symbol()을 이용해서 만든다.

→ 인자를 사용할 수 있지만 실제 Symbol값에 영향을 주지 않는다.

Symbol.for() → Symbol을 찾기 위한 메소드

Key값을 이용해서 Symbol을 찾거나, 못찾으면 Symbol을 생성, 등록, 리턴

→ Global Symbol Registry

Symbol.KeyFor() → Symbol을 찾기 위한 KEY를 찾는 메소드

→ Symbol값으로 Key를 찾는다.

→ 위에 6개 Data Type = Primitive Type (원시 타입)

7.Object Type → Reference Type (참조 타입)

→ 나중에 다시 얘기해요 !

* 변수에도 데이터 타입이 있나요?

JAVA Script는 변수에 값이 할당되는 시점에 Type이 결정된다.

→ Type inference (타입 추론) → dynamic typing (동적 타이핑)

→ 동적 타입이 일어나는 프로그래밍 언어

dynamic type language

weak type language

묵시적 타입변환 일어난다

엄격한 문법이 적용되지 않아서 유연한 프로그래밍가능

효율적인 프로그래밍 가능

신뢰도 하락

JAVA

→ 변수에 타입을 명시

→ 명시적 type언어 (동적 타입과 반대되는 개념)

static type language

strong type language

엄격한 문법 때문에 효율적이지 못한다.

형 변환(Operator)

⇒ String

1. String 생성자 함수 → new 없이 호출

2. `Object.prototype.toString()` 호출

3. "+" 연산자 이용

* "+" 연산자 *

피 연산자의 따라서 기능이 달라진다. (연산자 오버로딩)

ex) $2+2 = 4$, "신재민" + "위수민" = Love

`==` \Rightarrow loose equality (값만 비교)

`===` \Rightarrow Strict equality (type과 값을 같이 비교)

`typeof` \Rightarrow data type을 알려주는데 표준 data type과 완벽히 일치하지는 않는다.

ex) `typeof(null)` \Rightarrow Object (bug)