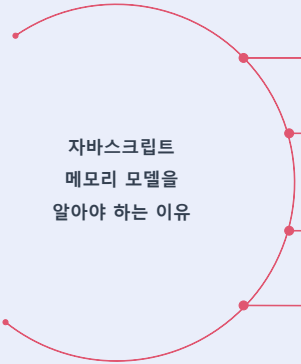
JavaScript **Memory Model**

소프트웨어학과
2021763013 김민영
2024.04.05
웹프로그래밍응용

JavaScript Memory Model

프로그램 실행 중에 메모리가 어떻게 관리되는지 설명하는 개념



- 메모리 사용량을 최적화하여 프로그램 성능을 향상
- 메모리 누수를 방지해서 프로그램 안정성을 향상
- 메모리 관련 오류를 예측하고 디버깅하는 데 도움
- 자바스크립트 코드 작동 방식을 이해

원시 타입과 참조 타입

자바스크립트의 데이터 타입을 크게 원시 타입과 참조 타입으로 구분

▼ 원시타입(primitive type)

원시 타입의 변수들은 데이터 복사가 일어날 때 메모리 공간을 새로 확보하여 독립적인 값을 저장

변경 불가능(immutable)

boolean, null, undefined, number, string, symbol

▼ 참조타입(reference type)

데이터의 복사 또는 접근을 할 때, 메모리에 직접 접근하는 것이 아닌 메모리의 주소에 대한 간접적인 참조를 통해 메모리에 접근하는 데이터 타입

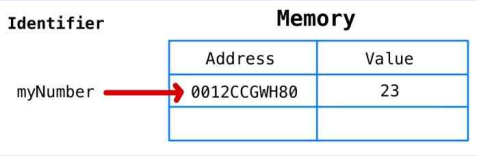
변경가능(mutable)

object, function, array etc. (원시 타입을 제외한 모든 것)

변경 불가능 (원시 타입)

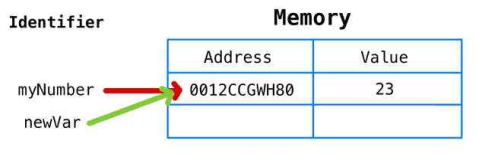
```
let myNumber = 23;
```

1) 변수의 고유 식별자를 만든다.
2) 런타임 시 메모리 주소를 할당한다.
3) 할당된 주소에 23이라는 값을 저장한다.




```
let newVar = myNumber;
```

할당된 변수를 새로운 변수에 할당하면?
> 새로운 변수도 23이라는 가진 메모리 주소를 가진다.

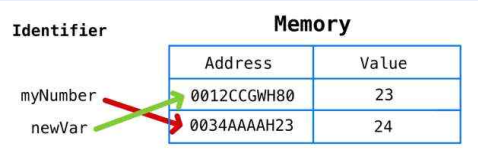


변경 불가능 (원시 타입)

```
myNumber += 1;
```

기존의 변수에 +1을 해주면?

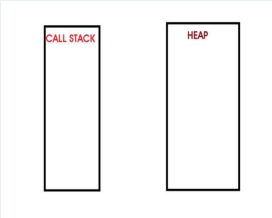
> 메모리에 새로운 주소를 할당하여 24를 저장하고
> 기존 변수가 해당 주소를 가지게한다.



- 정리

: 기존 변수와 새로운 변수가 같은 주소를 가지고 있을때 기존 변수의 값을 변경하면
새로운 주소를 할당하여 변경된 값을 저장하고 기존 변수가 해당 주소를 가지게 되는데
이는 새로운 변수가 변경 불가능(immutable)한 특징을 가진 원시 타입이기 때문이다.

Call Stack & Heap



자바스크립트의 메모리 모델을 메모리 관점에서 보면 call stack과 heap이 있다.

▼ 콜 스택(Call Stack)

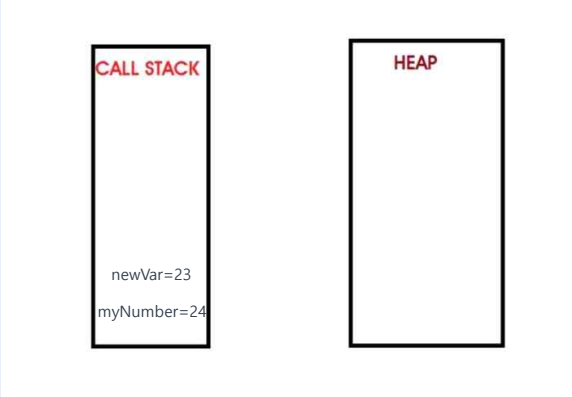
실행 중인 함수를 추적하고 계산을 수행하는 영역
함수가 호출될 때마다 새로운 프레임이 콜 스택에 추가되고, 함수가 종료될 때 해당 프레임은 제거됨
각 프레임은 함수의 지역 변수, 매개변수, 현재 실행 위치 등을 저장
원시 타입이 저장됨.

▼ 힙(Heap)

객체, 배열, 문자열 등 프로그램에서 사용하는 모든 데이터를 저장하는 영역
객체가 생성될 때 자동으로 메모리가 할당되고, 더 이상 사용하지 않을 때 가비지 컬렉터에 의해 메모리가 해제됨
참조 타입이 저장됨

변경 불가능 예제 + Call Stack & Heap

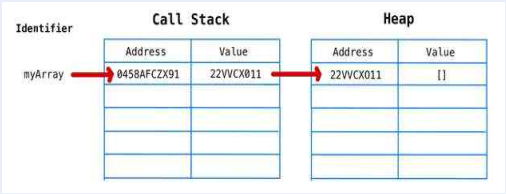
```
let myNumber = 23;

let newVar = myNumber;

myNumber += 1;
```



변경 가능 (참조 타입)

const myArray = [];

1) 변수의 고유 식별자를 만든다.
2) 런타임 시 메모리에 주소를 할당한다.
3) 런타임 시에 할당된 힙에 할당된
   메모리 주소 값을 저장한다.
4) 힙의 메모리 주소에 [ ]값을 저장한다.



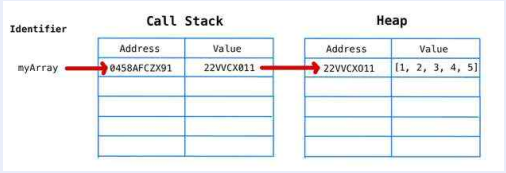
myArray.push(1); // [1]

myArray.push(2); // [1, 2]

myArray.push(3); // [1, 2, 3]

myArray.push(4); // [1, 2, 3, 4]

myArray.push(5); // [1, 2, 3, 4, 5]



할당된 변수의 값을 변경하려고 하면?

> 힙에 저장된 메모리 주소의 값은 변하지만 메모리 주소는 변경되지 않는다.

let & const

let 과 const 의 차이점은 immutable(변경 불가능)의 여부

일반적으론, 가능한 한 const 를 사용해야하며 변수가 변경 될 때만 let 을 사용해야한다.

변경 불가능 (원시 타입)

let myNumber = 23;

1) 변수의 고유 식별자를 만든다

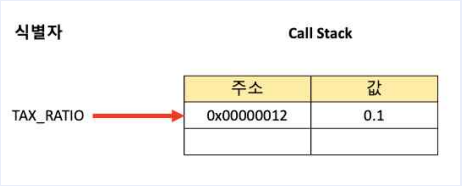
변경 가능 (참조 타입)

const myArray = [];

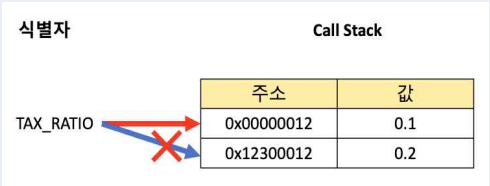
1) 변수의 고유 식별자를 만든다

▼ const로 원시 타입을 선언한 경우

const TAX_RATIO = 0.1;



TAX_RATIO = 0.2; // TypeError



원시 타입을 const로 선언하면 변경 불가능하다는 특징 때문에 결국 TypeError가 발생한다.

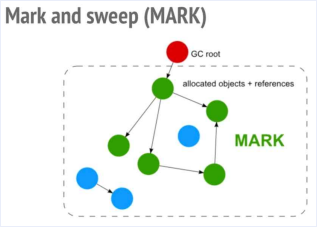
가비지 콜렉션(Garbage Collection)

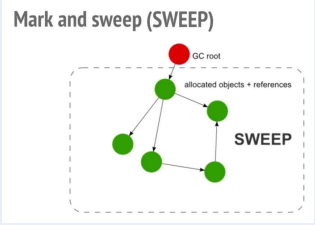
변수의 필요유무를 판단하고 메모리에서 제거

Heap 내부에는 세 개 이상의 가비지 컬렉션이 존재
각각 다른 내부 알고리즘을 사용하여 메모리를 최적화

▼ Mark-Sweep 알고리즘

대부분의 JS 엔진에서는
Mark-and-Sweep 알고리즘을 사용하여 가비지 컬렉션을 수행
1) roots라는 root 객체의 집합을 만들어 낸다.(JS에서 roots는 전역 변수들)
2) 주기적으로 roots로부터 접근 가능한 모든 객체들을 돌아다니며 Marking
3) Marking이 끝나고 메모리 내에 Marking이 되지 않은 데이터들은
사용하지 않는 데이터로 판단해 지워버린다.(Sweeping)





참조

JS Memory Model

https://velog.io/@jiseong/JS-Memory-Model

https://velog.io/@hustle-dev/JavaScript-%EB%A9%94%EB%AA%A8%EB%A6%AC-%EB%AA%A8%EB%8D%B8

https://velog.io/@code-bebop/JS-%EB%A9%94%EB%AA%A8%EB%A6%AC-%EA%B5%AC%EC%A1%B0

https://kimbangg.tistory.com/152

원시 타입, 참조 타입

https://bongra.tistory.com/462

Garbage Collection

https://velog.io/@arski/JavaScript-How-JavaScript-works-3-memory-management-how-to-handle-4-common-memory-leaks