1장. 데이터 타입

데이터 타입의 종류

1. **기본형 Primitive Type**

→ number, string, boolean, null, undefined, Symbol

2. **참조형 Reference Type**

→ Object

Array, Function, Date, RegExp, Map, WeakMap, Set, WeakSet

데이터 타입에 관한 배경지식

- **메모리와 데이터**

C/C++, 자바 - 정적 타입 언어로 데이터 타입별로 할당되는 메모리 영역 크기가 정해져있다.

ex) int - 4byte, long - 8byte, float - 4byte

자바스크립트 - 동적 타입 언어로 숫자의 경우 정수, 부동소수 구분하지 않고 8byte 이다.

바이트는 시작하는 비트의 식별자로 위치를 파악할 수 있다. ⇒ 메모리 주솟값

- **식별자와 변수**

변수 - 변할 수 있는 무언가. **데이터 자체**

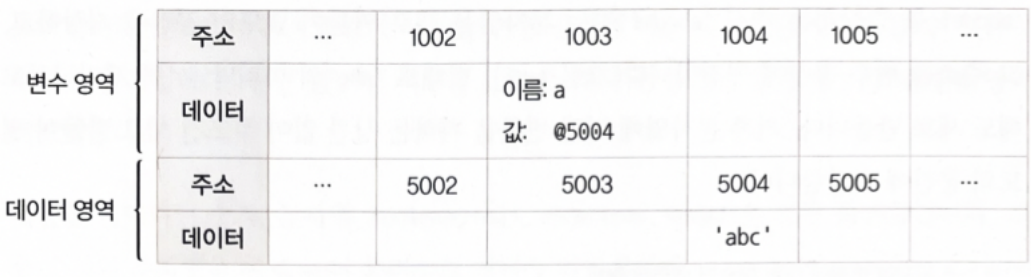
식별자 - 어떤 데이터를 식별하는데 사용하는 이름. **변수명**

변수 선언과 데이터 할당

- **기본형 데이터 선언, 할당**

```
var a = 'abc';

// 실제 실행 순서
var a;  // 변수 a 선언
a = 'abc';  // 변수 a에 데이터 할당
```



→ **순서**

1. 변수 영역에서 빈 공간(@1003)을 확보한다.
2. 확보한 공간의 식별자는 a로 지정한다.

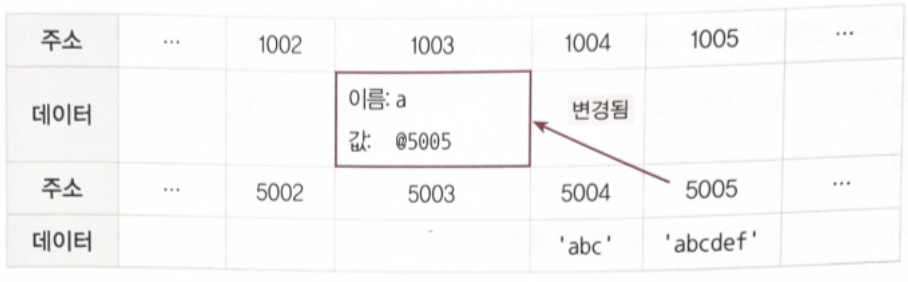
3. 데이터 영역에서 문자열 ‘abc’(리터럴)를 찾는다.
4. 없으면 데이터 영역의 빈 공간(@5004)에 문자열 ‘abc’를 저장한다.
5. 변수 영역에서 a라는 식별자를 검색한다. (@1003)
6. 앞서 저장한 문자열의 주소(@5004)를 @1003의 공간에 대입한다.

> ❗ **변수영역, 데이터 영역**이라는 말은 저자가 이 둘을 구분해서 설명하는 문서를 찾지 못해서 임의로 만든 개념이라고 한다.
> 어찌되었든 공간 하나에 이름, 값이 한번에 저장될 수도 있고, 데이터 값만 저장될 수도 있다는 것 같다.

이해를 돕기위해 만든 개념이다.

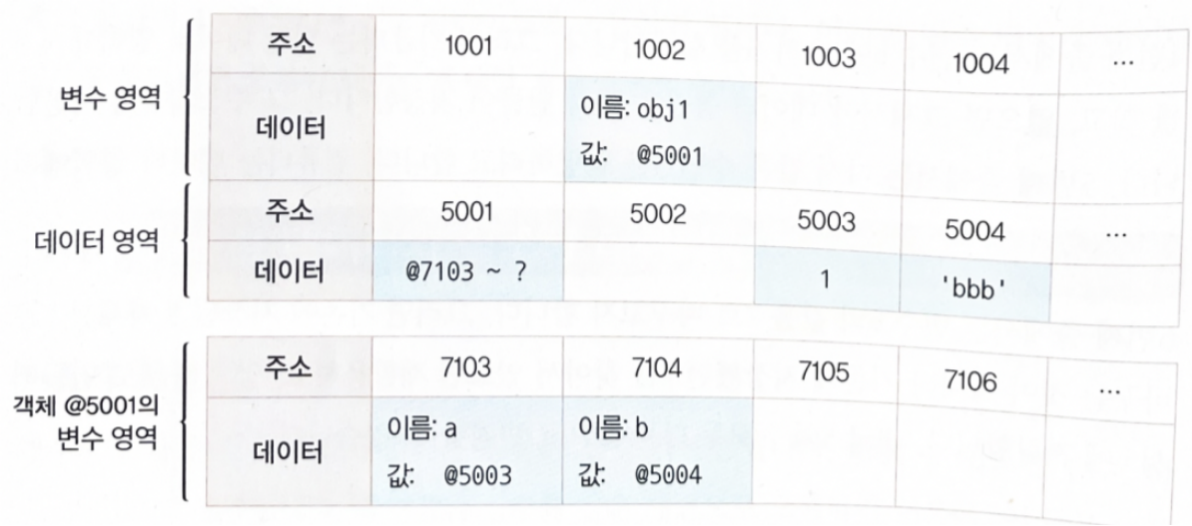
- **기본형 데이터 재할당**

```
a = 'abcdef';
```



- **참조형 데이터 선언, 할당**

```
var obj1 = {
  a: 1,
  b: 'bbb'
};
```

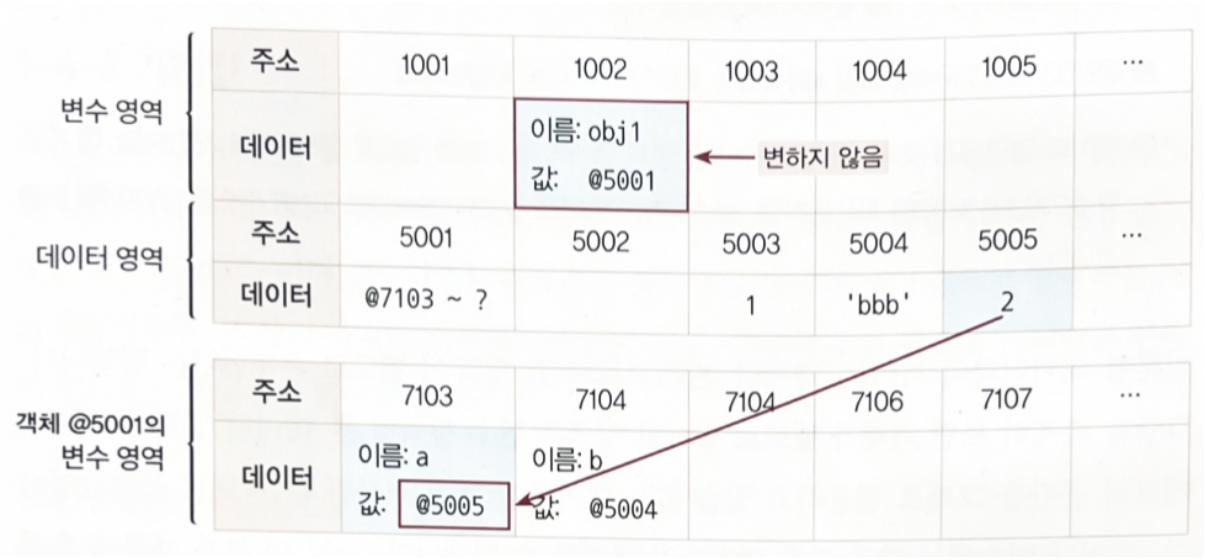


1. 변수 영역에 빈공간(@1002)확보, 확보한 공간의 식별자를 obj1로 지정
2. 임의의 데이터 저장 공간(@5001)에 데이터를 저장하려고 보니, 여러 개의 프로퍼티로 이뤄진 데이터 그룹!
   이들을 위해 별도의 변수 영역을 마련하고, 그 영역의 주소(@7103 ~ ?)를 @5001에 저장

3. @7103 및 @7104에 각각 a와 b라는 프로퍼티 이름을 지정
4. 데이터 영역에서 숫자 1을 검색, 없으면 @5003에 저장하고, 이 주소를 @7103에 저장.
   문자열 'bbb'도 마찬가지로 진행

- **참조형 데이터 재할당**

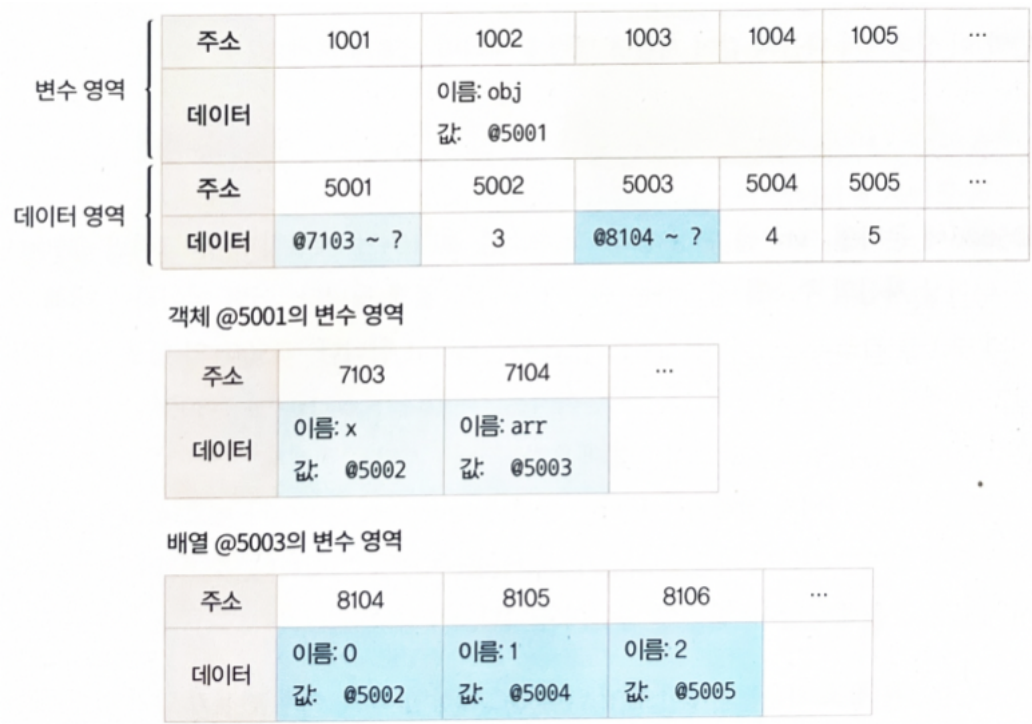
```
obj1.a = 2;
```



참조 카운트

- **중첩 객체**

```
var obj = {
  x: 3,
  arr: [3, 4, 5]
};
```



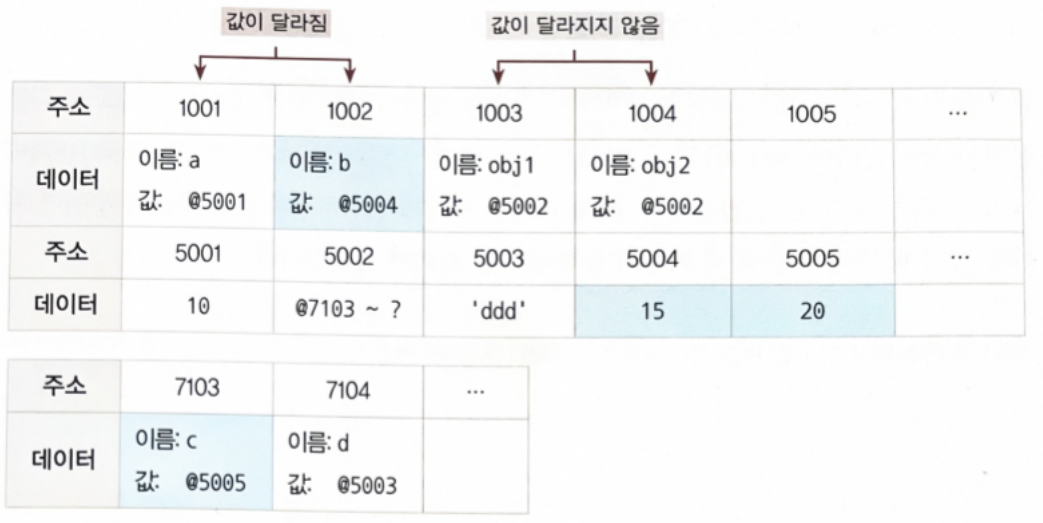
- **변수 복사**

```
// 기본형 데이터 복사
var a = 10;
var b = a;

// 참조형 데이터 복사
var obj1 = {
  c: 10,
  d: 'ddd'
};
var obj2 = obj1;
```

- **값 변경**

```
b = 15;
obj2.c = 20;
```




```
a !== b // true
obj1 === obj2 // true
```

- **공평한 값 변경**

```
b = 15;
obj2 = { c:20, d: 'ddd' };
```

불변 객체

- **불변 객체를 만드는 간단한 방법**

참조형 데이터의 가변은 데이터 자체가 아닌 내부 프로퍼티를 변경할 때만 성립한다.

데이터 자체를 새로운 데이터로 변경하고자 하면 기본형 데이터와 마찬가지로 기존 데이터는 변하지 않는다.

그래서 내부 프로퍼티를 변경할 필요가 있을 때마다 매번 새로운 객체를 만들어 재할당하기로 규칙을 정하거나 자동으로 새로운 객체를 만

→ 객체의 가변성에 따른 문제점

```
var user = {
  name: 'Joungpyo',
  gender: 'male'
};

var changeName = function (user, newName) {
  var newUser = user;
  newUser.name = newName;
  return newUser;
};

var user2 = changeName(user, 'Minsu');

console.log(user.name, user2.name);  // Minsu Minsu
```

→ 해결 방법1 - 직접 객체를 반환

```
var user = {
  name: 'Joungpyo',
  gender: 'male'
};

var changeName = function (user, newName) {
  return {
    name: newName,
    gender: user.gender
  };
};

var user2 = changeName(user, 'Minsu');

console.log(user.name, user2.name);  // Joungpyo Minsu
```

이렇게 하면 해결은 되지만 복사할 객체를 모두 하드코딩해야 하므로 객체가 클수록 비효율적이다.

→ 해결 방법2 - 기존 정보를 복사해서 새로운 객체를 반환하는 함수

```
var user = {
  name: 'Joungpyo',
  gender: 'male'
};

var copyObject = function(target) {
  var result = {};
  for (var prop in target) {
    result[prop] = target[prop];
  }
  return result;
}

var user2 = copyObject(user);
user2.name = 'Minsu';

console.log(user.name, user2.name);  // Joungpyo Minsu
```

협업하는 모든 프로그래머가 이 함수를 사용해서 객체를 복사해야 의미가 있는데, 쉽지 않다.

그래서 외부라이브러리를 사용해서 프로퍼티 변경을 막 할 수 없게끔한다.

아무튼 이렇게 간단하게 객체를 복사할 수 있지만, 이는 얕은 복사이기 때문에 완전한 복사라고 볼 수 없다.

얕은 복사와 깊은 복사

얕은 복사는 바로 아래 단계의 값만 복사하는 방법이고,

깊은 복사는 내부의 모든 값들을 하나하나 찾아서 전부 복사하는 방법이다.

얕은 복사는 중첩된 객체의 참조형 데이터가 저장된 프로퍼티를 복사할 때 그 주소값만 복사한다.

따라서 복사된 프로퍼티는 원본과 같은 데이터를 참조하고 있는 것이다.

- **얕은 복사**

```
var user = {
  name: 'Joungpyo',
  urls: {
    portfolio: 'http://github.com/abc',
    blog: 'http://blog.com',
    facebook: 'http://facebook.com/abc'
  }
};

var copyObject = function(target) {
  var result = {};
  for (var prop in target) {
    result[prop] = target[prop];
  }
  return result;
}

var user2 = copyObject(user);

user2.name = 'Minsu';
console.log(user.name, user2.name);  // Joungpyo Minsu

user.urls.portfolio = 'hello';
console.log(user.urls.portfolio, user2.urls.portfolio); // hello hello
```

→ 해결방법 1 - 중첩된 객체에 대한 깊은 복사

```
var user2 = copyObject(user);
user2.urls = copyObject(user.urls);

user.urls.portfolio = 'hello';
console.log(user.urls.portfolio, user2.urls.portfolio);
// hello http://github.com/abc
```

이렇게 객체의 프로퍼티 중에서 그 값이 기본형일 경우에는 그대로 복사하면 되지만, 참조형 데이터인 경우는 다시 그 내부의 프로퍼티들을

→ 해결방법 1 - 중첩된 객체에 대한 깊은 복사(재귀)

```
var copyObjectDeep = function(target) {
  var result = {};
  if (typeof target === 'object' && target !== null) {
    for (var prop in target) {
      result[prop] = copyObjectDeep(target[prop]);
    }
  } else {
    result = target;
  }
  return result;
};

// 참고 - typeof 명령어는 null에 대해서도 ‘object’를 반환한다.
```

→ 해결방법 2 - JSON 활용

```
var copyObjectViaJSON = function (target) {
  var jsonString = JSON.stringify(target);
  return JSON.parse(jsonString);
};
```

```
var obj = {
  a: 1,
  b: {
    c: null,
    d: [1, 2],
    func1: function() { console.log(9); }
  }
};

var obj2 = copyObjectViaJSON(obj);

obj2.a = 3;
obj2.b.c = 4;
obj2.b.d[1] = 3;

console.log(obj);
console.log(obj2);
```

객체를 JSON 문자열로 변환한 뒤 이를 다시 JSON객체로 바꾸는 방법이다.

하지만 메서드(함수)나 getter/setter 등과 같이 JSON으로 변경할 수 없는 프로퍼티들은 무시한다.

httpRequest로 받은 데이터를 저장한 객체를 복사할 때 등 순수한 정보만 다룰 때 활용하기 좋은 방법이다.