기본 타입(1)

기본 타입과 참조 타입

기본 타입	참조 타입
- number - string - boolean - undefined ...	-Object * Array * Function * Date * RegExp ...

기본 타입

- string : char 없음
- number : double, float, long, int 구분 없음
- undefined : 변수에 값이 할당되지 않은 상태 (정의되지 않은 상태)

typeof 연산자

- 변수의 데이터 타입을 string으로 리턴

기본 타입(2)

[예제 02-01]

```
01: var intValue = 100;
02: var floatValue = 10.222;
03: var strValue = "Hello jQuery";
04: var boolValue = true;
05: var undefinedValue;
06:
07: console.log(typeof(intValue));
08: console.log(typeof(floatValue));
09: console.log(typeof(strValue));
10: console.log(typeof(boolValue));
11: console.log(typeof(undefinedValue));
```

[실행 결과]

```
number
number
string
boolean
undefined
```

기본 타입(3)

[예제02-02]

```
01: var a;
02: console.log(typeof(a));
03: a = "hello world";
04: console.log(typeof(a));
05: a = 10000;
06: console.log(typeof(a));
07: a = null;
08: console.log(typeof(a));
```

[실행결과]

```
undefined
string
number
object
```

참조 타입(1)

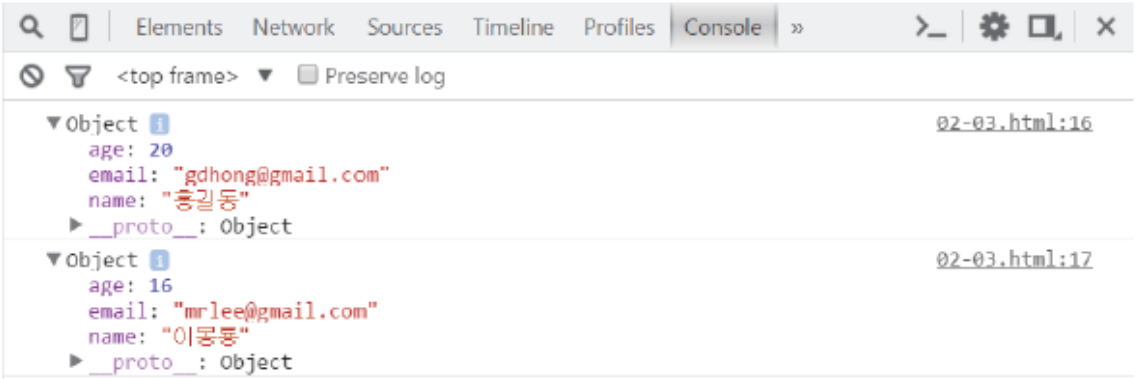
- 기본 타입을 제외한 모든 데이터 타입은 참조타입
  - 배열, 함수, Date, RegExp 등 --> Object
- 객체
  - 클래스 기반의 객체가 아님.
  - 속성 : 키와 값의 쌍 하나를 말함
    - 언제든지 추가, 삭제가 가능함.
    - 속성의 값으로 다른 객체의 참조를 가질 수도 있음
    - 속성의 값이 함수인 경우를 메서드라 부름

참조 타입(2)

객체 생성 방식

[예제 02-03]

```
01: var p1 = { name:"홍길동", "age":20 };     //객체 리터럴로 객체 생성
02: var p2 = new Object();          //Object 생성자 함수로 객체 생성
03: p2.name = "이몽룡";
04: p2.age = 16;
05: p1.email = "gdhong@gmail.com";
06: p2.email = "mrlee@gmail.com";
07: console.dir(p1);
08: console.dir(p2);
```



[**그림 02-02** : 크롬 브라우저에서 실행 결과]

배열(1)

- 자바스크립트 객체들중 하나
  - 다른 언어의 컬렉션과 유사(Java의 ArrayList와 유사)

[예제 02-04 : 배열 생성]

```
01: var a1 = [100,200,300];
02:
03: var a2 = new Array();
04: a2[0] = 100;
05: a2[1] = 200;
06: a2[2] = 300;
07:
08: console.dir(a1);
09: console.dir(a2);
```

var 키워드(1)

var?

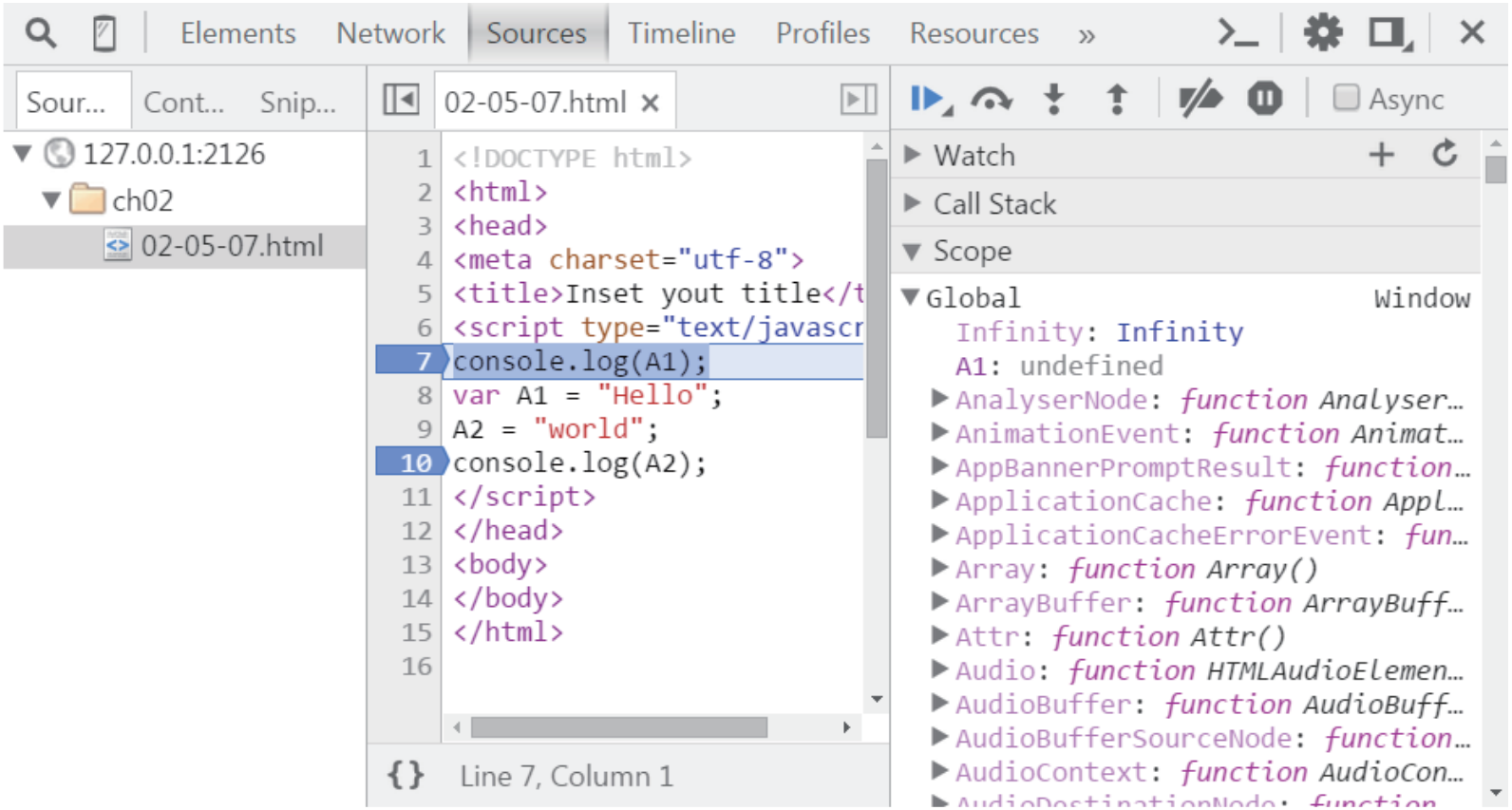
- 단지 변수를 선언하기 위한 키워드는 아님.
- 정확하게 이해하기 위해서 자바스크립트의 실행방식을 이해해야 함
  - 실행 컨텍스트와 호이스팅 개념!!
  - 실행 컨텍스트(Execution Context)는 "자바스크립트 코드가 실행되는 환경"이라고 정의할 수 있으며, 일종의 객체이다.
- 호이스팅(Hoisting)
  - JS코드가 실행되면 가장 먼저 전역 실행컨텍스트가 생성됨.
  - 그 후 선언적 방식의 함수가 생성되고, var 키워드가 사용된 변수가 미리 만들어짐
- 아래 코드는 오류가 발생하지 않음.--> 호이스팅 때문에...

[예제02-06]

```
console.log(A1);
var A1 = "Hello";
A2 = "world";
console.log(A2);
```

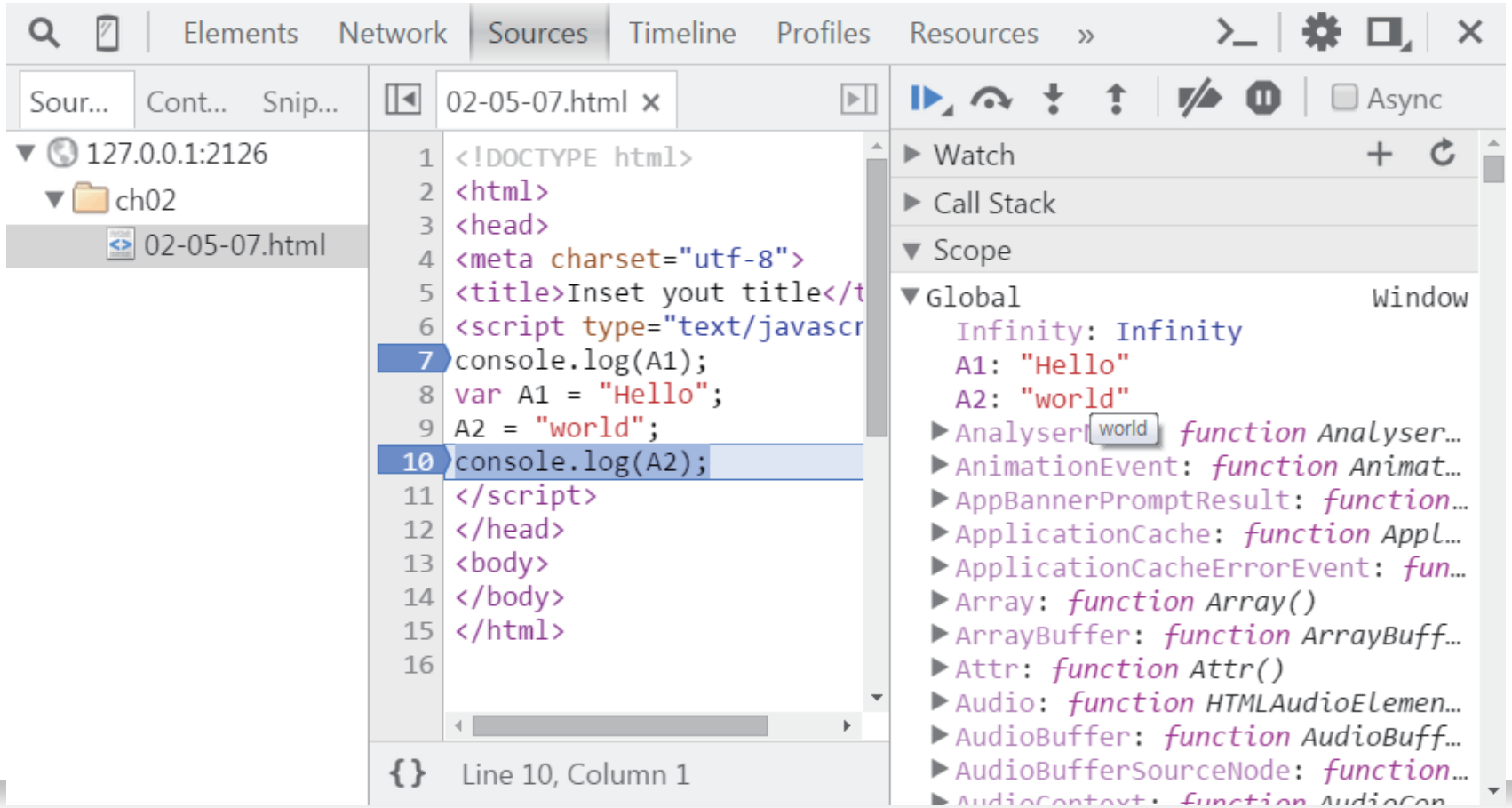
var 키워드(2)

- 앞 코드 브라우저 디버깅 화면
  - Global 아래에 A1이 미리 만들어진 것이 보인다!!



var 키워드(3)

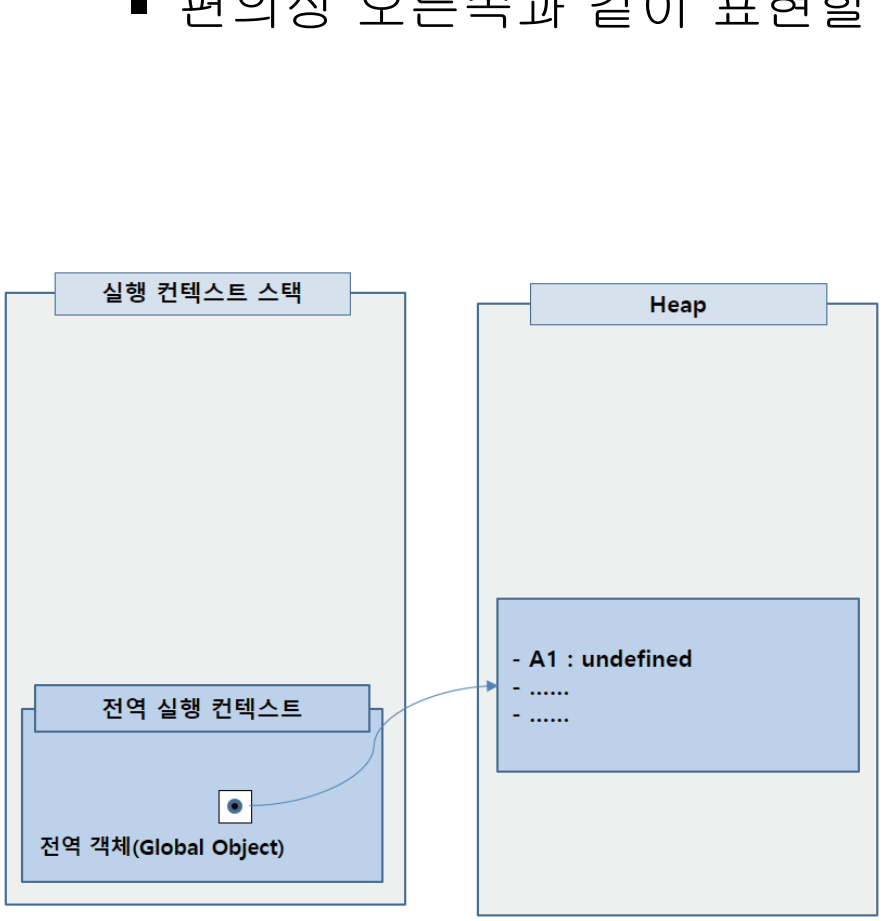
- 그림 02-04
  - 10행 까지 이동하니 A2도 만들어졌다.

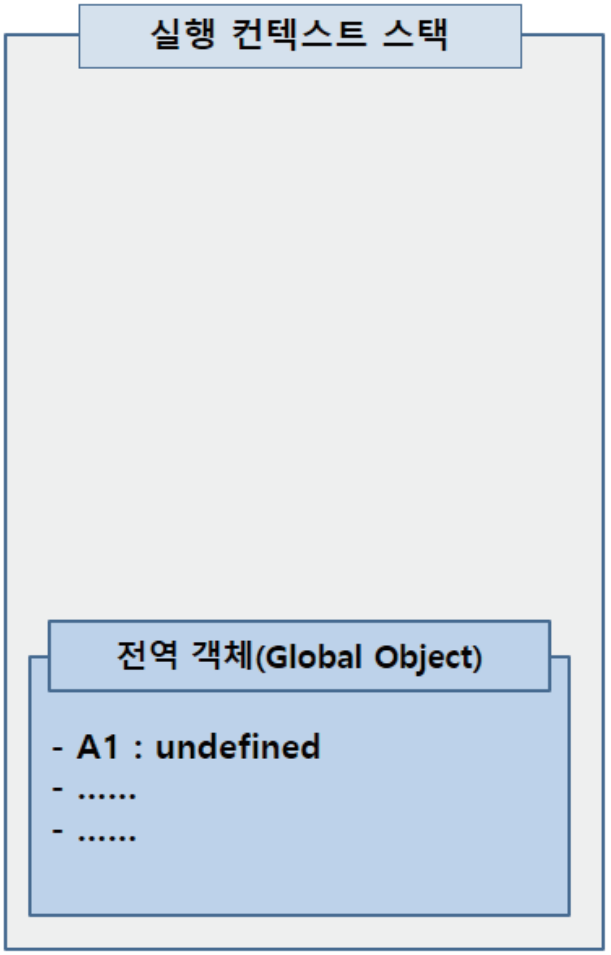


var 키워드(4)

전역 실행 컨텍스트와 전역 객체

- 편의상 오른쪽과 같이 표현할 것임





중복된 var 키워드

아래 코드는 에러가 발생하지 않음

- 호이스팅하기 때문에...

[예제02-07]

```
var A1 = "hello";
console.log(A1);
var A1 = 1000;
console.log(A1);
var A1 = true;
console.log(A1);
```

  - 호이스팅 단계에서 이미 변수가 만들어져 있다면 다시 생성하지 않음. 건너뜀

- 변수명이 중복되어도 오류가 발생하지 않으므로 주의해야 함.
  - 가능하다면 var 키워드 사용은 충돌 방지를 위해 코드 블록의 맨 앞에 배치하는 것이 바람직함.

연산자와 형변환(1)

산술 연산자와 암시적 형변환

- 연산자를 중심으로 암시적 형변환이 일어난다는 사실에 주의해야 함

[예제 02-09 : 암시적 형변환]

```
01: var a1 = "1"+"2";   // "12"
02: var a2 = 1 + "2";   // "12"
03: var a3 = "1" + 2;   // "12"
04: var a4 = 1 + 2;     // 3
05:
06: console.log(a1);
07: console.log(a2);
08: console.log(a3);
09: console.log(a4);
10: console.log("==========");
11:
12: var b1 = "2" * "3";     //6
13: console.log(b1);
14: console.log("==========");
15:
16: var c1 = parseInt("123");   //123
17: var c2 = parseInt("abc");   //NaN
18: var c3 = parseInt("123abc")     //123
19: var c4 = Number("123abc");      //NaN
20: var c5 = 123 - "23";        //100
21: var c6 = 123 - "abc";       //NaN
22:
23: console.log(c1);
24: console.log(c2);
25: console.log(c3);
26: console.log(c4);
27: console.log(c5);
28: console.log(c6);
```

[실행 결과]

```
12
12
12
3
==========
6
==========
123
NaN
123
NaN
100
NaN
```

연산자와 형변환(2)

- parseInt와 Number의 차이
  - 아래 코드를 브라우저 console에서 실행

```
> parseInt("123abc")
< 123
> Number("123abc")
< NaN
> parseInt("")
< NaN
> Number("")
< 0
```

연산자와 형변환(3)

number와 Number

```
> var a1 = 123;
undefined
> var a2 = new Number(123);
undefined
> typeof(a1);
"number"
> typeof(a2);
"object"
> a2.valueOf()
123
> a2
Number {[[PrimitiveValue]]: 123}
```

string과 String

```
> var b1 = "hello";
undefined
> var b2 = new String("world");
undefined
> b1.toUpperCase()
"HELLO"
> b2.toUpperCase()
"WORLD"
```

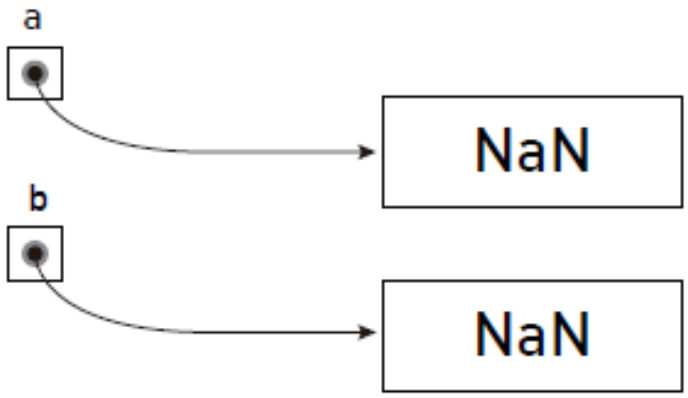
number : 기본타입

Number : 객체타입

연산자와 형변환(4)

NaN

- Not a Number :숫자가 아님
- 수치 연산에 실패한 경우에 리턴하는 number 타입의 값
- Number.NaN == Number.NaN은 false를 리턴
  - 암시적 형변환이 일어나면서 객체 타입으로 변환
  - 객체에서의 == 비교 연산자는 같은 주소를 참조하는지를 나타냄
  - NaN인지를 확인하려면 isNaN 함수 사용




```
> var a = parseInt("abc")
<- undefined
> isNaN(a)
<- true
```

연산자와 형변환(5)

비교 연산자

- ==, !=, >, <
  - 암시적 형변환한 후 값을 비교함

[예제 02-10]

```
01: console.log("" == 0)        //true
02: console.log("" == false)    //true
03: console.log(0 == false)     //true
04: console.log("123" == 123)      //true
```

```
> Number("")
< 0
> Number(false)
< 0
> Boolean("")
< false
> Boolean(0)
< false
```

- a==b의 의미

  typeof(a) == typeof(b) && a==b

!!연산자

- boolean 타입으로 변환