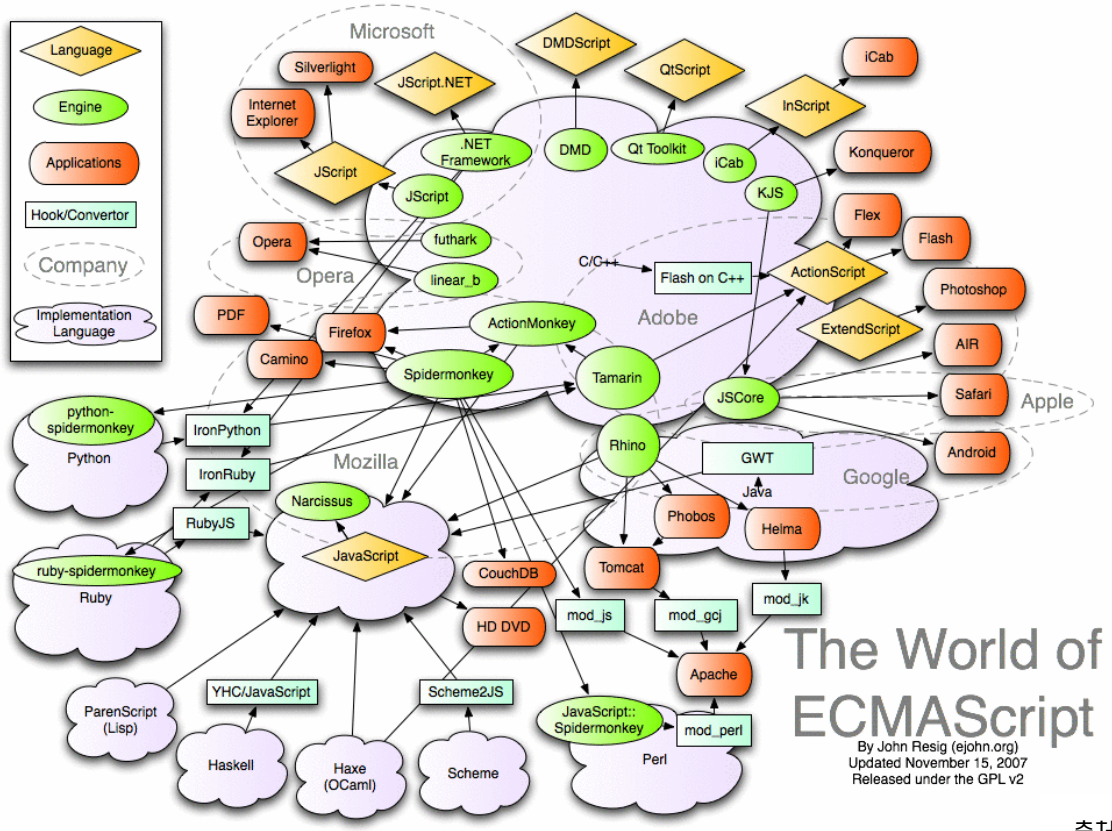
JavaScript (ver3.3.9)

목 차

1. JavaScript 소개

1.1 JavaScript 개요

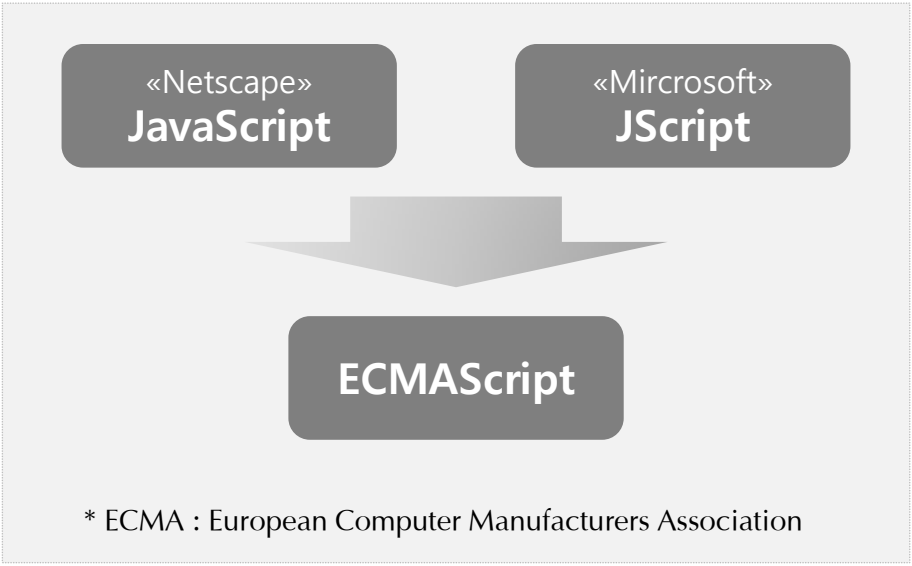
- ✓ JavaScript는 프로토타입 기반의 스크립트 프로그래밍 언어로 객체지향 개념을 지원합니다.
- ✓ 웹 브라우저가 JavaScript를 HTML과 함께 다운로드 하여 실행합니다.
- ✓ 웹 브라우저가 HTML문서를 읽어 들이는 시점에, JavaScript 엔진이 실행합니다.
- ✓ 대부분의 JavaScript 엔진은 ECMAScript 표준을 지원합니다.



출처 : http://ejohn.org/blog/the-world-of-ecmascript/

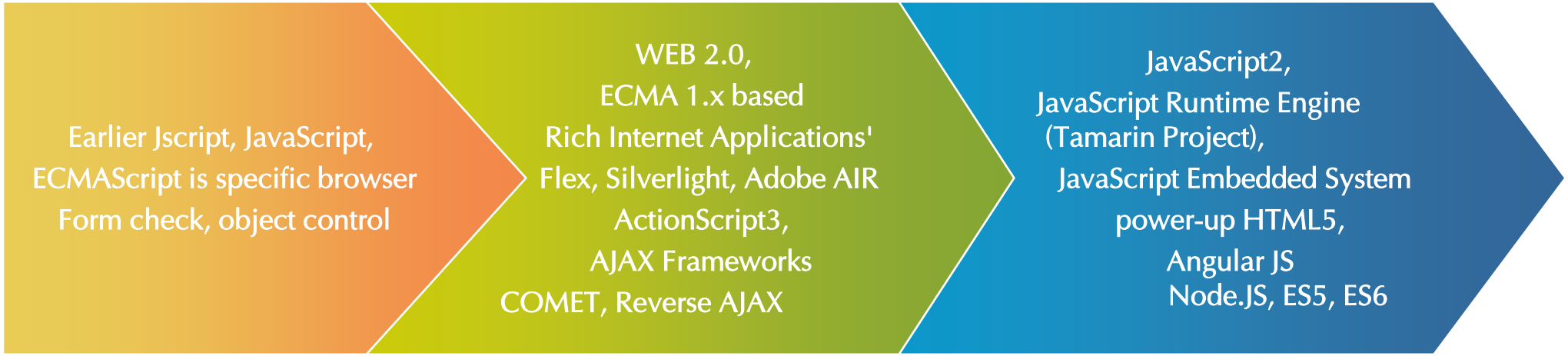
1.2 ECMAScript (ECMA: European Computer Manufacturers Association)

✓ 1996년 3월 넷스케이프는 'Netscape Navigator 2.0'을 출시하면서 JavaScript를 지원하였습니다.
✓ 마이크로소프트사는 웹에 호환되는 JScript를 개발해서 1996년 8월 인터넷 익스플로러 3.0에 포함하며 출시하였습니다.
✓ 넷스케이프는 JavaScript 표준화를 위해 기술 규격을 ECMA에 제출하였습니다.
✓ 1997년 6월 ECMA는 ECMA-262초안을 일반 회의에서 채택합니다.



1.3 JavaScript의 과거와 미래

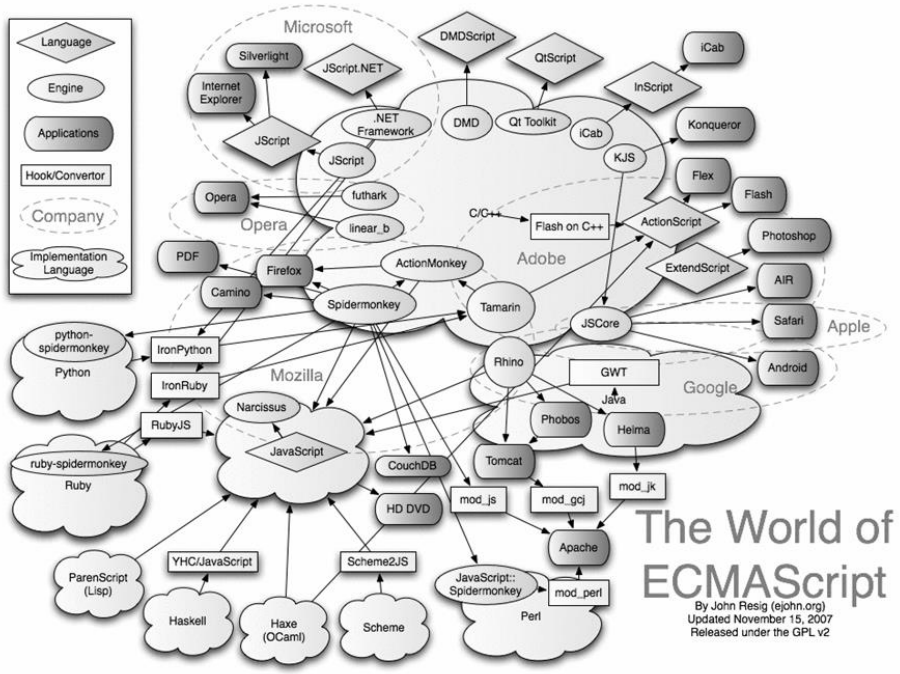
- ✓ 1998년 ECMAScript(Edition 2)를 ISO 표준으로 개정하였습니다.
- ✓ 1999년 정규 표현식, 문자열 처리, 새로운 제어문, 예외처리, 오류 정의, 포매팅 등을 추가하였습니다.(Edition 3)
- ✓ Edition 4는 정치적인 문제로 폐기되었습니다.
- ✓ Edition 5.1(2011년)를 거쳐 Edition 6(2015.06, Harmony)까지 표준화 되어 있습니다.



출처 :http://kangax.github.io/compat-table/es6/

1.4 요약

✓ 웹브라우저는 JavaScript를 HTML과 함께 다운로드하고, 브라우저의 JavaScript 엔진이 JavaScript를 실행합니다.
✓ JavaScript는 클래스가 존재하지 않는 프로토타입 기반의 객체지향 언어입니다. ← Edition6에서는 class 개념 지원
✓ 넷스케이프에서 처음 만들었으며, 이후 ECMA에서 ECMAScript라는 이름의 표준으로 정하였습니다.
✓ 각 브라우저에서는 ECMAScript 스펙을 준수하는 방식으로 JavaScript를 지원합니다.



2. JavaScript 시작하기

2.1 JavaScript 선언 (1/2)

- ✓ HTML에서 JavaScript를 사용하려면 <script> 태그를 사용합니다.
- ✓ <script> 태그는 'src'와 'type' 속성을 사용하여 JavaScript를 선언합니다. (HTML5 부터는 type 속성 생략 가능)
- ✓ src 속성은 외부의 JavaScript 파일을 HTML 문서에 포함할 때 사용하며, 생략할 수 있습니다.
- ✓ type 속성은 미디어 타입을 지정할 때 사용합니다. JavaScript 코드는 'text/javascript'로 지정합니다.

JavaScript 코드를 HTML 문서에 포함하는 방법

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JavaScript 선언</title>
<script type="text/javascript">
    // Some JavaScript Code here!
    function hello(target) {
        alert('Hello,' + target);
    }

    hello('JavaScript');
</script>
</head>
<body>
</body>
</html>
```

외부 JavaScript 파일을 HTML 문서에 포함하는 방법

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JavaScript 선언</title>
<script src="hello.js" type="text/javascript">
</script>
</head>
<body>
</body>
</html>
```

```
function hello(target) {
    alert('Hello,' + target);
}

hello('JavaScript');
```

hello.js

2.1 JavaScript 선언 (2/2)

- ✓ <script> 태그는 HTML 파일 내부의 <head>나 <body> 안 어느 곳에서나 선언 할 수 있습니다.
- ✓ 하지만 <body> 안의 끝부분에 <script> 태그를 둘 것을 권장합니다.
- ✓ <head> 안에 위치한 JavaScript는 브라우저의 각종 입/출력 발생 이전에 초기화되므로 브라우저가 먼저 점검합니다.
- ✓ <body> 안에 위치하면 브라우저가 HTML부터 해석하여 화면에 그리기 때문에 사용자가 빠르다고 느낄 수 있습니다.

```
<html>
  <head>
    <title>script tag의 위치</title>
    <!-- script tag의 위치 1 -->
    <script type="text/javascript">
      document.write("반갑습니다. 여러분!");
    </script>
  </head>
  <body>
    즐거운 하루입니다.
    <!-- script tag의 위치 2 -->
    <script type="text/javascript">
      document.write("JavaScript 공부하기 참~ 좋은 날씨입니다.");
    </script>
  </body>
</html>
```

<head>에 <script>태그를 두는 경우

<body>에 <script>태그를 두는 경우

2.2 구구단 출력

✓ 반복(loop) 문을 이용하여 브라우저 콘솔 창에 구구단을 출력하는 예제를 작성해 봅니다.
✓ 변수(variable)를 선언할 때는 숫자형(type)이든 문자형(type)이든 모두 var 키워드를 사용하여 선언합니다.
✓ for문을 사용해서 연산을 9번 반복 수행하는 코드를 작성할 수 있습니다.
✓ 연산 결과를 선언한 변수에 저장합니다. 그리고 console.log() 함수로 브라우저 콘솔 창에 결과를 출력합니다.

```
<html>
  <body>
    <script type="text/javascript">
      // 변수를 선언합니다.
      var mul = 0;

      // for 반복문을 사용해서 9번 반복합니다.
      for (var i = 1 ; i < 10 ; i++) {

        // 곱셈 연산을 수행하여 결과를 저장합니다.
        mul = 2 * i;

        // 결과를 콘솔 창에 출력합니다.
        console.log(2 + ' * ' + i + ' = ' + mul);
      }
    </script>
  </body>
</html>
```

RESULT

연산결과 :

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
```

2.3 시계 구현

- ✓ JavaScript의 Date 내장 객체로 현재 시간을 구하여, 웹 브라우저에 현재 시간을 보여주는 예제를 작성해 봅시다.
- ✓ Date 객체의 toLocalTimeString() 메소드를 호출하면 현재 시간을 돌려줍니다.
- ✓ span 태그의 Body 에 내용을 추가하기 위해 innerHTML 프로퍼티에 값을 할당합니다.
- ✓ setInterval() 함수는 주기적으로 주어진 함수를 수행합니다. 1초마다 현재 시간을 업데이트 합니다.

```
<html>
  <head>
    <script type="text/javascript">
      // 화면이 로딩되면 수행할 문구를 작성합니다.
      window.onload = function() {
        setInterval(showClock, 1000);
      };
      // 화면에 시간을 출력합니다.
      function showClock() {
        // clock 요소를 검색합니다.
        var clockEl = document.getElementById("clock");

        // 현재 시간을 계산합니다.
        var date = new Date();
        clockEl.innerHTML = date.toLocaleTimeString();
      }
    </script>
  </head>
  <body>
    <span id="clock"></span>
  </body>
</html>
```

RESULT

수행결과 :

오후 9:15:58

2.4 요약

- ✓ HTML 문서에서 JavaScript 를 사용하려면 <script> 태그를 사용합니다.
- ✓ JavaScript 코드는 HTML 파일 안에 두거나, 외부 JavaScript 파일을 HTML 문서 안에 포함할 수도 있습니다.
- ✓ script 태그는 HTML 문서의 어느 위치에서나 선언 가능하며, 일반적으로 head나 body 내부에 위치시킵니다.
- ✓ 웹 브라우저가 HTML 문서를 순차적으로 해석(parsing)하므로, script 위치에 따라 로드와 실행 시점이 달라집니다.

JavaScript 코드를 HTML 문서 내에 구현하는 방법

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JavaScript 선언</title>
<script type="text/javascript">
    // Some JavaScript Code here!
    function hello(target) {
        alert("Hello," + target);
    }

    hello("JavaScript");
  </script>
</head>
<body>
</body>
</html>
```

독립된 JavaScript 파일을 HTML 문서에 포함시키는 방법

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JavaScript 선언</title>
<script src="hello.js" type="text/javascript">
</script>
</head>
<body>
</body>
</html>
```

```
function hello(target) {
    alert("Hello," + target);
}

hello("JavaScript");
```

hello.js

3. JavaScript 기초

3.1 주석 (comment)

✓ 주석은 JavaScript 코드에 대한 부연 설명이므로 실행 코드에 포함되지 않습니다.
✓ JavaScript 주석은 한 줄 주석(Line comment)과 블록 주석(Block comment) 이 있습니다.
✓ 한 줄 주석은 // 로 표기하고, 블록 주석은 /* */ 로 표기하며, /* 부터 */ 사이에 내용을 둡니다.
✓ 가능하면 블록 주석(/* */)보다는 라인 주석(//)을 사용하는 것이 좋습니다.

```
<html>
  <head>
    <title>주석문</title>
    <script type="text/javascript">
      // 한줄 주석처리

      /*
         여러 줄의 주석처리
         C, Java와 동일함.
      */
    </script>
  </head>
  <body>
  </body>
</html>
```

3.2 변수 (variable)

- ✓ JavaScript는 변수를 선언할 때 타입을 명시하지 않은 채, var 키워드를 사용하여 선언합니다.
- ✓ 변수 이름은 함수 이름과 혼동되지 않도록 유일한 이름을 사용합니다. (변수 이름은 형용사나 명사, 함수 이름은 동사를 사용)
- ✓ JavaScript는 ECMAScript 표준에 따라 낙타 표기법(Camel case)을 사용합니다.
- ✓ 낙타 표기법이란 기본적으로 소문자로 작성하되 2개 이상의 단어일 때 단어의 첫 글자를 대문자로 표기하는 방법입니다.

```
// 변수
var myName;
var myVariable;

// 좋은 예
var listCount = 0;
var userName = "홍길동";
var selected = false;

// 나쁜 예 - 변수인지 함수인지 알기 어렵습니다.
var getListCount = 10;
var isSelected = false;
```

3.3 자료 형 (data type) (1/6)

✓ 프로그램은 정적인 데이터 값을 동적으로 변환해 가면서 원하는 정보를 얻습니다.
✓ 프로그램에서 다루는 데이터 값의 종류들을 자료 형(data type)이라고 표현합니다.
✓ JavaScript에서는 자료 형을 원시 타입(primitive type)과 객체 타입(object type)으로 분류합니다.
✓ 원시 타입에는 숫자, 문자열, boolean, null, undefined, 다섯 가지가 있습니다. 이를 제외한 모든 값은 객체 타입입니다.

자료 형	typeof 출력 값	설명
숫자형	number	정수 또는 실수형
문자열형	string	문자, 작은따옴표나 큰따옴표로 표기
부울형	boolean	참(true)과 거짓(false)
undefined	undefined	변수가 선언 되었지만 초기화 되지 않음
null	object	값이 존재하지 않음을 뜻함

3.3 자료 형 (data type) (2/6) – 숫자 (1/2)

✓ JavaScript는 숫자를 정수와 실수로 나누어서 구분하지 않습니다.

- 모든 숫자를 8 byte의 실수 형태로 처리합니다.
- 편의성을 위해 정수 리터럴과 실수 리터럴을 제공하고 있습니다.
- 숫자의 연산 처리시 실수 형태로 하기 때문에 특정 소수점을 정확하게 표현하지 못합니다.

✓ 기본 연산 기호는 Java나 C++와 같은 일반 프로그래밍 언어와 같습니다. (+, -, *, /, %)

부동소수점 오류

```
var x = 0.3 - 0.2;
var y = 0.2 - 0.1;
x == y            // false
console.log(x)    // 0.09999999999999998
console.log(y)    // 0.1
```

정확한 계산이 필요할 때

```
var x = 0.3;
var y = 0.2;

var result = (x * 10 - y * 10) / 10;
```

3.3 자료 형 (data type) (3/6) – 숫자 (2/2)

- ✓ JavaScript는 언더블로우, 오버플로우, 0으로 나누는 연산에 대해 예외를 발생 시키지 않습니다.
- ✓ JavaScript에는 숫자와 관련된 특별한 상수가 존재합니다.
  - Infinity : 무한대를 나타내는 상수로, 어떠한 수를 0으로 나누거나 Infinity를 어떠한 수로 사칙연산 한 식의 결과입니다.
  - NaN(Not a Number) : 계산식의 결과가 숫자가 아님을 나타내는 상수입니다.

언더/오버플로우

```
// 언더플로우
0 / 100;              // 0
-0 / 100;             // -0

// 오버플로우
100 / 0;              // Infinity
-100 / 0;             // -Infinity
Infinity / 100;       // Infinity
-Infinity / 100;      // -Infinity
```

NaN

```
0 / 0;                // NaN

parseInt('1A');       // 1
parseInt('A');        // NaN

new Number('1');      // 1
new Number('1A');     // NaN
```

3.3 자료 형 (data type) (4/6) - 문자열

- ✓ JavaScript에서 문자열은 16비트의 Unicode 문자를 사용합니다.
- ✓ 문자 하나를 표현하는 char와 같은 문자형은 제공하지 않습니다. 'a'와 같은 한 글자도 문자열로 표현합니다.
- ✓ 작은 따옴표(', single quotes) 또는 큰 따옴표(", double quotes)를 둘 다 사용할 수 있으며, 섞어서 쓰면 안됩니다.
- ✓ 이스케이프 시퀀스(\)도 사용할 수 있습니다.

문자열 사용

```
console.log("");
console.log('작은 따옴표로 문자열 나타내기');
console.log("3.14");
console.log('문자열 안에 포함된 "따옴표"는 이렇게...');
console.log('특수문자 사용하기\n이 것은 두 번째 줄');
```

3.3 자료 형 (data type) (5/6) – boolean, null과 undefined

- ✓ boolean은 비교 연산의 결과 값으로, true 값 또는 false 값 중 하나를 가집니다.
- ✓ null은 값이 없거나 비어있음을 뜻하고, undefined는 값이 초기화 되지 않았음(정의되지 않음)을 뜻하는 특수한 값입니다.
- ✓ null과 undefined는 의미가 비슷하지만 값을 할당 하지 않은 변수는 undefined가 할당 되고(시스템 레벨), 코드에서 명시적으로 값이 없음을 나타낼 때(프로그램 레벨)는 null을 사용합니다.

boolean, null과 undefined

```
var name;
var id = null;

var n1 = 10;
var n2 = 12;

console.log(n1 == n2); // false
console.log(name);     // undefined
console.log(id);       // null
```

3.3 자료 형 (data type) (6/6) – 자동 형변환

✓ JavaScript는 Java나 C++ 등과 같은 언어와는 달리 자료 형에 대해 매우 느슨한 규칙을 적용합니다.
✓ 어떤 자료 형이든 전달할 수 있고, 그 값을 필요에 따라 변환할 수 있습니다.
✓ 서로 다른 자료 형의 연산이 가능합니다.
✓ 모든 자료 형을 var로 선언하기 때문에 변수 선언은 쉽지만 이런 느슨한 규칙 때문에 혼란스러울 수도 있습니다.

```
1  var answer = 42;
2
3  console.log("answer " + answer);        // answer 42
4  answer = "How do you do";
5  console.log(answer);                    // How do you do
6
7  console.log("The answer is " + 42);     // The answer is 42
8  console.log(42 + " is the answer");     // 42 is the answer
9
10 console.log("37" - 7); // 30
11 console.log("37" + 7); // 377
12
13 console.log(parseInt("123.45") + 1);    // 124
14 console.log(parseFloat("123.45") + 1); // 124.45
15
16 console.log("1.1" + "1.1");             // 1.11.1
17 console.log((+"1.1") + (+"1.1"));       // 2.2
```

3.4 상수 (constant)

✓ ECMAScript6 이전까지는 상수 표현을 지원하지 않았습니다.

- 변수의 값을 변경하면 안 되는 상수와 일반 변수를 구분하고자 변수 명명 규칙을 다르게 하여 사용했습니다.
- 상수의 표기법은 모든 문자를 대문자를 사용하고 단어 사이는 '_'(언더스코어)로 표기합니다.

✓ ECMAScript6 에서는 const 키워드가 추가 되어 상수를 지원합니다.

ECMAScript6 이전

```
// 변수
var listCount = 0;
var selected = false;

// 상수 - ECMAScript6 이전 보편적 사용 방법
var LIST_COUNT = 10;
var SELECTED = false;

if (listCount < LIST_COUNT) {
    // doSomethig()
}
console.log('상수 초기값: ' + LIST_COUNT); // 10

LIST_COUNT = 20;
console.log('상수 변경값: ' + LIST_COUNT); // 20
```

ECMAScript6

```
// 변수
var listCount = 0;
var selected = false;

// 상수 - ECMAScript6
const LIST_COUNT = 10;
const SELECTED = false;

if (listCount < LIST_COUNT) {
    // doSomethig()
}
console.log('상수 초기값: ' + LIST_COUNT); // 10

LIST_COUNT = 20;
console.log('상수 변경값: ' + LIST_COUNT); // 10
```

3.5 연산자 (operator) (1/4)

- ✓ JavaScript에서 기본적으로 제공하는 약속된 문자의 표현 식을 연산자라고 합니다.
- ✓ 연산자에는 산술 연산자, 비교 연산자, 논리 연산자, 기타 연산자 등을 제공합니다.
- ✓ 표현 식에서 2개 이상의 연산자를 동시에 사용했을 경우 우선순위별로 표현 식을 해석합니다.
- ✓ 괄호를 사용하여 우선순위를 조절할 수 있습니다.

연산자	설명
++	선행 ++는 현재 값을 반환하고 값을 증가합니다. 후행 ++는 값을 증가시키고 결과를 반환합니다.
--	선행 --는 현재 값을 반환하고 값을 감소합니다. 후행 --는 값을 감소 한 후 결과를 반환합니다.
-	부호를 전환해서 결과를 반환합니다.
+	숫자로 값을 변환합니다.
~	비트단위 연산에서 사용하며 NOT 연산을 수행합니다.
!	논리연산에서 사용하며 boolean 결과를 반환합니다.
delete	프라퍼티를 제거합니다.
typeof	피연산자 타입을 알려줍니다.
void	undefined 값을 알려줍니다.
*, /, %	곱하기, 나누기, 나머지의 연산결과를 반환합니다.

연산자	설명
+, -	값이 숫자일 때는 더하기, 빼기의 연산결과를 반환합니다.
+	값이 문자열이면 문자열을 서로 결합합니다.
<<	비트연산자로 값을 왼쪽으로 이동합니다.
>>	비트연산자로 값을 오른쪽으로 이동합니다.
>>>	부호비트 확장 없이 값을 오른쪽으로 이동합니다.
<, <=, >, >=	숫자를 비교합니다.
instanceof	객체가 특정 객체의 타입인지 확인합니다.
in	프라퍼티가 존재하는지 확인합니다.
==	동등 관계인지 확인합니다.
!=	동등하지 않은지 확인합니다.

3.5 연산자 (operator) (2/4)

✓ 연산자는 연산의 대상이 되는 값에 따라서 동작이 결정됩니다.
✓ '+' 연산자는 대상의 값이 모두 숫자인 경우 산술 연산을 수행합니다.
✓ '+' 연산자는 대상 중에 문자열이 포함된 경우 모든 연산 대상을 문자열로 변환하고 문자열을 붙입니다.
✓ 연산자는 종류에 따라 1항 연산자, 2항 연산자, 3항 연산자로 구분됩니다.

연산자	설명
===	값이 일치하는지 확인합니다.
!==	값이 일치하지 않는지 확인합니다.
&	비트 단위 연산자로 AND 연산을 수행합니다.
^	비트 단위 연산자로 XOR 연산을 수행합니다.
|	비트 단위 연산자로 OR 연산을 수행합니다.
&&	AND 연산을 수행합니다.
||	|| 연산을 수행합니다.
?:	조건에 해당하는 구문을 수행합니다.
=	변수 또는 프라퍼티에 값을 할당합니다.
,	1번째 구문은 버리고 다음 구문 값을 반환합니다.

3.5 연산자 (operator) (3/4) – 활용 (1/2)

✓ a++와 a-- 는 각각 a = a + 1 과 a = a - 1 연산의 축약 형태로써 증감연산자 라고 합니다.
✓ 증감연산자가 앞에 오면 연산을 먼저 실행하고, 뒤에 오면 해당 라인을 진행 후 연산을 실행합니다.
✓ ! 연산은 NOT의 의미로써 boolean형의 값을 반대로 반환합니다.
✓ typeof는 해당 변수의 타입을 반환합니다.

```
1  var num1 = 10;
2  var num2 = 20;
3  var bool = true;
4
5  console.log("num1++ : " + num1++);
6  console.log("num1   : " + num1);
7  console.log("--num1 : " + --num1);
8  console.log("!bool  : " + !bool);
9
10
11 console.log("typeof bool : " + typeof bool);
12 console.log("typeof num1 : " + typeof num1);
13
14
15 console.log("num1 + num2 : " + (num1 + num2));
16 console.log("num1 - num2 : " + (num1 - num2));
17 console.log("num1 * num2 : " + (num1 * num2));
18 console.log("num1 / num2 : " + (num1 / num2));
```

```
num1++ : 10
num1   : 11
--num1 : 10
!bool  : false

typeof bool : boolean
typeof num1 : number

num1 + num2 : 30
num1 - num2 : -10
num1 * num2 : 200
num1 / num2 : 0.5
```

3.5 연산자 (operator) (4/4) – 활용 (2/2)

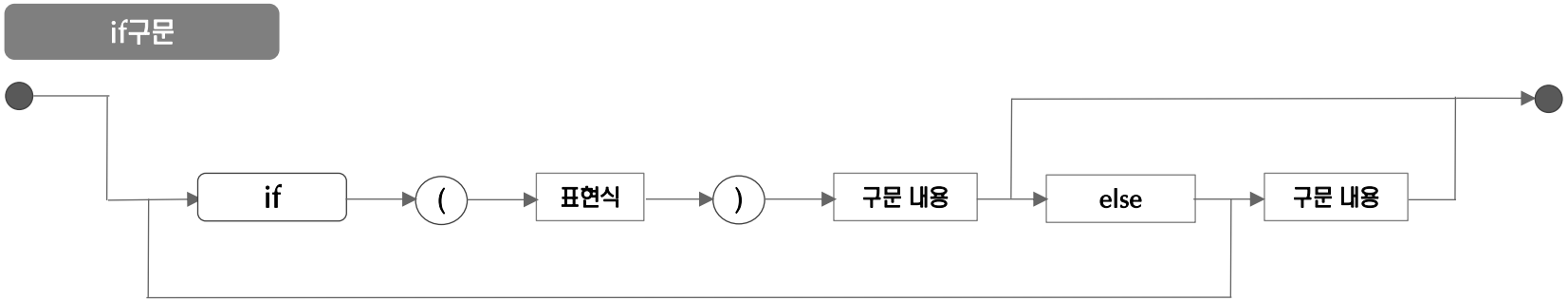
- ✓ 논리값을 비교하여 참(true)과 거짓(false)을 판단할 수 있습니다.
- ✓ 비교연산자 ==, ===의 차이점은 자료 형 까지 비교하는지 아닌지의 여부입니다.
- ✓ 비교연산자 &&은 둘 중 하나라도 거짓(false)이면 false, ||는 둘 중 하나라도 참(true)이면 true를 반환합니다.
- ✓ 3항 연산자의 ? 앞 비교 값이 참(true)이면 : 앞의 값을 반환하고, 거짓(false)이면 :뒤의 값을 반환합니다.

```
1  var num = 10;
2  var str = "10";
3
4  console.log("num == str  : " + (num == str));
5  console.log("num === str : " + (num === str));
6
7  str = "20";
8
9  console.log("num != str  : " + (num != str));
10 console.log("num !== str : " + (num !== str));
11
12 console.log("true && true  : " + (true && true));
13 console.log("true && false : " + (true && false));
14 console.log("true || true  : " + (true || true));
15 console.log("true || false : " + (true || false));
16
17 console.log("num > num ? 'true' : 'false' "
18         + (num > num ? true : false));
```

```
num == str  : true
num === str : false


num != str  : true
num !== str : true

true && true  : true
true && false : false
true || true  : true
true || false : true

num > num ? 'true' : 'false' false
```

3.6 조건문 (conditional) (1/3) – if

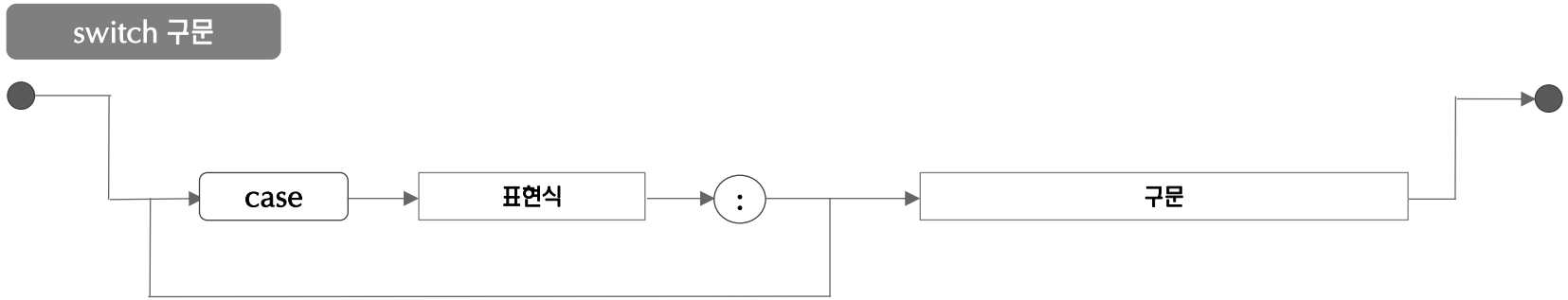
- ✓ 표현 식의 값에 따라서 특정 구문들을 실행 하거나 실행 하지 않도록 제어합니다.
- ✓ 조건 구문을 분기 구문이라고도 표현하는데 2개 이상의 경로 중에서 특정 경로를 선택할 수 있습니다.
- ✓ if 구문은 단순한 결정을 내리거나 정교한 구문들을 표현하는데 사용하는 구문입니다.
- ✓ if와 else if의 표현 식 결과 값이 참(true)일 경우 구문을 실행 합니다. 모두 거짓(false)일 경우 else문을 실행합니다.




```
if (표현식) {
  // 구문내용
}
else if (표현식) {
  // 구문내용
}
else {
  // 구문내용
}
```

3.6 조건문 (conditional) (2/3) – switch

- ✓ switch 구문은 동일한 표현식이 반복될 때 효과적인 구문입니다.
- ✓ 값이 case와 동일 할 경우 해당 구문내용을 실행합니다.
- ✓ break문은 switch 구문을 종료하며, break문이 없을 경우 다음 case를 실행합니다.
- ✓ 동일한 case가 없을 경우 default 구문을 실행합니다.




```
switch (값) {
  case 1:
  // 구문내용
  break;
  case 2:
  // 구문내용
  break;
  default:
  // 구문내용
  break;
}
```

3.6 조건문 (conditional) (3/3) – 활용

- ✓ if 구문의 예제는 변수 number1과 number2를 비교하여 해당하는 조건의 구문을 실행하는 예제입니다.
- ✓ if 조건이 거짓이면 순차적으로 else if 조건을 확인하고, 모두 거짓일 경우에 else 문을 실행합니다.
- ✓ switch 구문의 예제는 변수 number1의 값을 비교하여 해당하는 조건의 구문을 실행하는 예제입니다.
- ✓ case 문에 break를 빠뜨리지 않도록 주의해야 합니다.

```
var number1 = 10;
var number2 = 20;

if (number2 > number1) {
  console.log("number2 > number1");
}
else if (number2 < number1) {
  console.log("number2 < number1");
}
else {
  console.log("else…");
}
```

```
number2 > number1
```

```
var number1 = 20;

switch (number1) {
  case 10:
    console.log("number1 is 10");
  break;
  case 20:
    console.log("number1 is 20");
  default:
    console.log("default…");
}
```

```
number1 is 20
default…
```

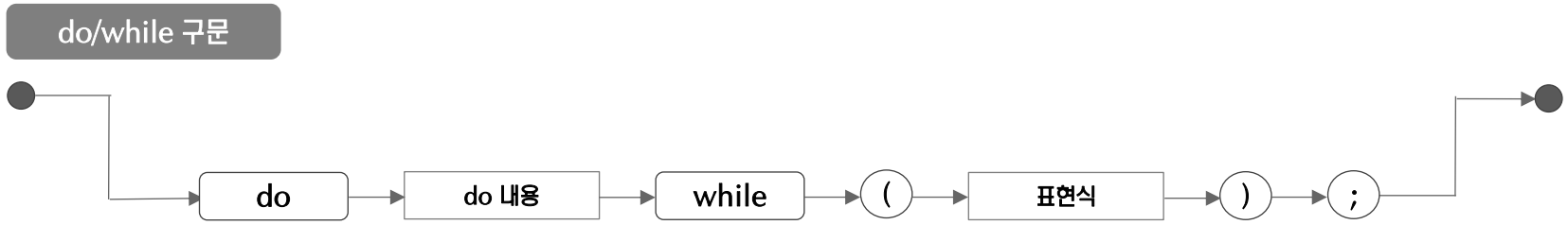
3.7 반복문 (loop) (1/4) – while

- ✓ 표현 식의 값이 참일 때 선언된 구문을 수행합니다.
- ✓ 표현 식의 값이 거짓일 때 while문을 종료합니다.
- ✓ while 구문은 무한 반복에 빠지는 상황에 쉽게 노출됩니다.
- ✓ 표현 식의 값이 변수를 참조 하고 변수가 거짓이 될 수 있는 상황을 만들어서 무한 반복 상황을 피하도록 합니다.


```
while (표현식) {
  // 구문내용
}
```

3.7 반복문 (loop) (2/4) – do/while

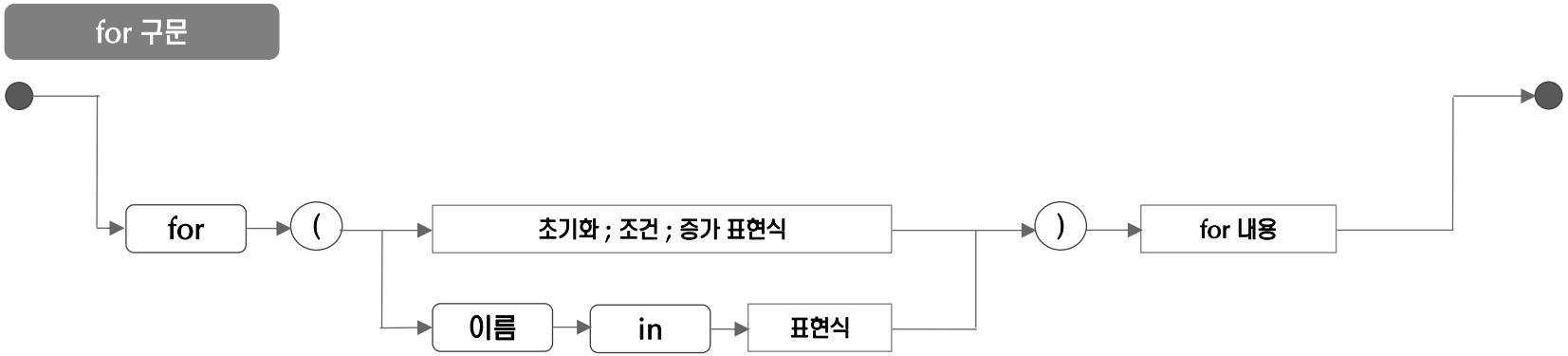
- ✓ 표현 식의 값을 확인하는 시점이 구문의 마지막이므로 최소 한번 이상 반복 구문이 수행됩니다.
- ✓ while과 문법적인 차이만 있을 뿐 수행 방법은 비슷합니다.
- ✓ 최초 1회 내용(body)를 수행하고 표현 식의 값이 참일 때 다음 내용을 수행합니다.
- ✓ 표현 식의 값이 거짓일 때 do문을 종료합니다.




```
do {
  // 구문내용
} while (표현식);
```

3.7 반복문 (loop) (3/4) – for

- ✓ 프로그래밍 언어에서 가장 많이 사용하는 구문입니다.
- ✓ for 구문은 카운터 변수를 사용하는 구문과 in 키워드를 사용하는 구문으로 분류됩니다.
- ✓ 카운터 변수를 사용하는 for 구문은 카운터 변수가 표현 식에 명시된 조건에 이르렀을 때 반복을 끝냅니다.
- ✓ in 키워드를 사용하는 for 구문은 배열 또는 객체가 가진 프라퍼티를 순회하면서 키(인덱스)값을 조회합니다.




```
for (초기화; 조건; 증감 표현식;) {
  // 구문내용
}
```

```
for (var 변수 in 배열 혹은 객체) {
  // 구문내용
  // 변수에는 배열의 인덱스
  // 객체의 프라퍼티 명이 담김
}
```

3.7 반복문 (loop) (4/4) – 활용

- ✓ 1부터 10까지 숫자의 합을 계산해 봅시다. 반복 문을 사용하여 계산합니다.
- ✓ 다른 반복 문과는 다르게 do-while문은 꼭 한번은 실행되어야 할 때 사용합니다.
- ✓ break문을 만나면 표현식이 참이더라도 반복문을 종료합니다.
- ✓ continue문을 만나면 현재 반복문 블록의 아래 로직은 수행하지 않고 위로 올라가서 다음 반복을 계속합니다.

```
var number = 1,
    sum = 0;

while (number <= 10) {
  sum += number; // sum = sum + number;
  number++;
}
console.log(sum);
```

55

```
var sum = 0;


for (var number = 1; number <= 10; number++) {
  sum += number; // sum = sum + number;
}
console.log(sum);
```

55

```
var number = 1,
    sum = 0;

do {
  sum += number; // sum = sum + number;
  number++;
} while (number <= 10);
console.log(sum);
```

55

```
var number = 1,
    sum = 0;
while (number <= 10) {
  sum += number; // sum = sum + number;
  if (number == 5) {
    break;
  }
  number++;
}
console.log(sum);
```

15

3.8 함수 (function) (1/3) – 선언, 호출

- ✓ JavaScript에서 함수는 일급(first-class) 객체입니다.
- ✓ 함수를 변수, 객체, 배열 등에 저장할 수 있고 다른 함수에 전달하는 전달 인자 또는 리턴 값으로 사용할 수 있습니다.
- ✓ 함수는 프로그램 실행 중에 동적으로 생성할 수 있습니다.
- ✓ 함수 정의 방법은 함수 선언문, 함수 표현식, Function 생성자(constructor) 함수 세가지 방식이 있습니다.

```
// 함수 선언문
function 함수이름(매개변수1, 매개변수2, … , 매개변수n) {
  //함수 내용
}
```

```
// 함수 표현식
var 함수이름 = function (매개변수1, 매개변수2, … , 매개변수n) {
  //함수 내용
}
```

```
// Function 생성자 함수
var 함수이름 = new Function("매개변수1", "매개변수2" , … , "매개변수n" , "함수내용");
```

```
// 함수호출
함수이름(매개변수1, 매개변수2, … , 매개변수n);
```

3.8 함수 (function) (2/3) – 매개변수

- ✓ 함수의 정의 부분에 외부로부터 전달받을 변수를 매개변수(parameter)라고 합니다.
- ✓ 함수를 호출할 때 전달하는 값을 전달인자(argument)라고 합니다.
- ✓ JavaScript에서 함수 정의 시 매개변수에 대한 타입은 명시하지 않습니다.
- ✓ 함수를 호출할 때, 정의된 매개변수와 전달인자의 개수가 일치하지 않더라도 호출이 가능합니다.

```
function getName(p1, p2) {

  if ( p2 === undefined ) p2 = [];
  for (var property in p1) {
      p2.push(property);
  }
  return p2;
}

var a = getName(p1);
getName(p3, p2);
```

3.8 함수 (function) (3/3) – 활용

✓ 다음은 함수의 간단한 예제입니다.
✓ 1부터 매개변수 number 까지의 합을 구하는 예제 입니다.
✓ 각각 함수 선언문과 함수 표현식, Function 생성자 함수 호출입니다.
✓ 방식은 달라도 함수 호출방식은 똑같습니다.

```
// 함수 선언문
function func1(n) {
  var sum = 0;

  for (var number = 1; number <= n; number++) {
    sum += number;
  }
  console.log(sum);
}
func1(10);
```

55

```
// 함수 표현식
var func2 = function (n) {
  var sum = 0;

  for (var number = 1; number <= n; number++) {
    sum += number;
  }
  console.log(sum);
}
func2(10);
```

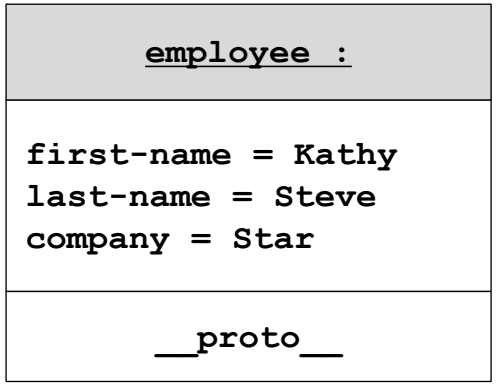
55

```
// Function 생성자 함수
var func3 = new Function("n",
    "var sum = 0; " +
    "for (var number = 1; number <= n; number++) {" +
    " sum += number; " +
    "} " +
    "console.log(sum);" );
func3(10);
```

55

3.9 객체 (object) (1/4) – 개요

- ✓ 객체는 이름과 값으로 구성된 프라퍼티의 집합입니다.
- ✓ 문자열, 숫자, boolean, null, undefined를 제외한 모든 값은 객체입니다.
- ✓ 전역 객체를 제외한 JavaScript 객체는 프라퍼티를 동적으로 추가하거나 삭제할 수 있습니다.
- ✓ JavaScript 객체는 프로토타입(prototype)이라는 특별한 프라퍼티를 포함합니다.



3.9 객체 (object) (2/4) – 속성 값 조회

- ✓ 객체는 마침표(.)를 사용하거나 대괄호([])를 사용해서 속성 값에 접근합니다.
- ✓ 객체에 없는 속성에 접근하면 undefined를 반환합니다.
- ✓ 객체 속성 값을 조회 할 때 || 연산자를 사용하는 방법도 많이 사용합니다.
  - 예 : var middle = employee['middle-name'] || 'none';

```
<html>
<body>
  <script type="text/javascript">
    // 객체 리터럴
    var empty_object = {};
    var employee = {
       "first-name" : "Kathy",
       "last-name" : "Steve",
       "company" : "Star"
    };

    // 객체의 속성에 접근하는 두 가지 방법
    console.log(employee.company);     // 1. dot 표기법
    console.log(employee["company"]); // 2. [] 표기법

    // 다음과 같이 속성명에 연산자가 포함된 경우, []표기법으로만 접근가능
    console.log(employee["first-name"]);
    console.log(employee["last-name"]);
  </script>
</body>
</html>
```

3.9 객체 (object) (3/4) – 속성 값 변경

✓ 속성 값을 변경할 때는 마침표(.)나 대괄호([])를 사용합니다.

- 예: server['port'] = '8080';
- 예: server.port = '8080';

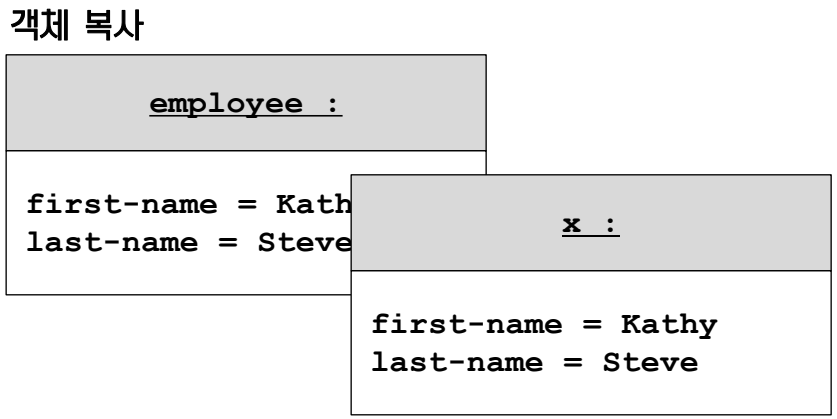
✓ 객체에 값을 할당하는 속성이 없을 경우, 그 속성이 추가됩니다.

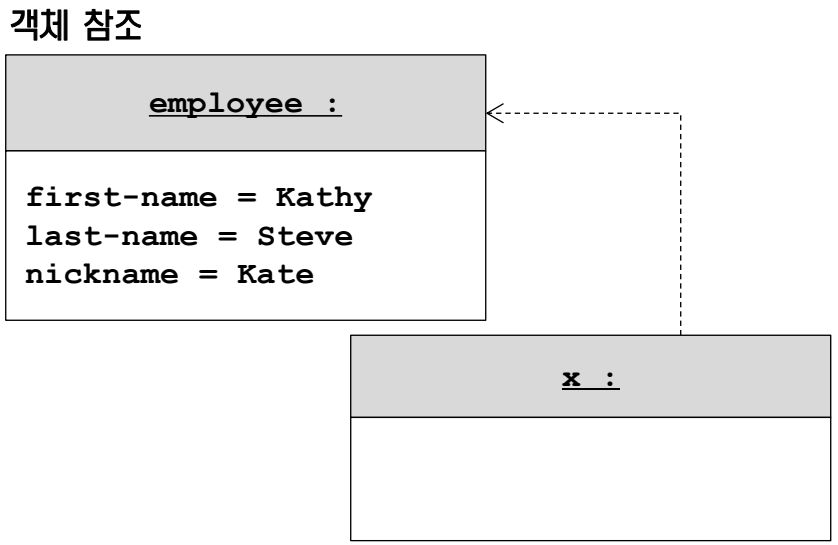
- 예 : server['region'] = 'Asia';

```
<html>
<body>
  <script type="text/javascript">
    // 객체 리터럴
    var empty_object = {};
    var employee = {
       "first-name" : "Kathy",
       "last-name" : "Steve",
       "company" : "Star"
    };

    // 속성 값의 변경
    employee.company = 'Nextree';
    employee['first-name'] = 'Gildong';
    employee['last-name'] = 'Hong';

    console.log(employee.company);
    console.log(employee["first-name"]);
    console.log(employee["last-name"]);
  </script>
</body>
</html>
```

3.9 객체 (object) (4/4) – 참조

- ✓ 객체는 복사되지 않고 참조 됩니다.
- ✓ JavaScript에서 원시(Primitive) 데이터 타입이 아닌 모든 값은 참조 타입입니다.
- ✓ 참조 타입은 Object, Array, Date, Error를 포함합니다.
- ✓ 타입 확인 방법으로는 typeof 연산자가 있습니다. (null은 원시 타입이지만 typeof 연산자에서 object를 반환합니다.)





3.10 요약

- ✓ JavaScript의 변수는 var를 사용하여 선언하며 타입이 없습니다.
- ✓ 자료형은 원시타입(Primitive Type)과 객체타입(Object Type)으로 분류합니다.
- ✓ 함수는 일급객체(First-class Object)로, 변수에 저장하거나 함수의 전달인자 또는 반환 값으로 사용할 수 있습니다.
- ✓ 객체는 이름과 값으로 구성된 프라퍼티의 집합이며 프라퍼티를 동적으로 조작할 수 있습니다.

```
// 함수 선언문
function 함수이름(매개변수1, 매개변수2, … , 매개변수n) {
  //함수 내용
}
```

```
// 함수 표현식
var 함수 명 = function (매개변수1, 매개변수2, … , 매개변수n) {
  //함수 내용
}
```

```
// Function 생성자 함수
var 함수 명 = new Function("매개변수1", "매개변수2" , … , "매개변수n" , "함수내용");
```

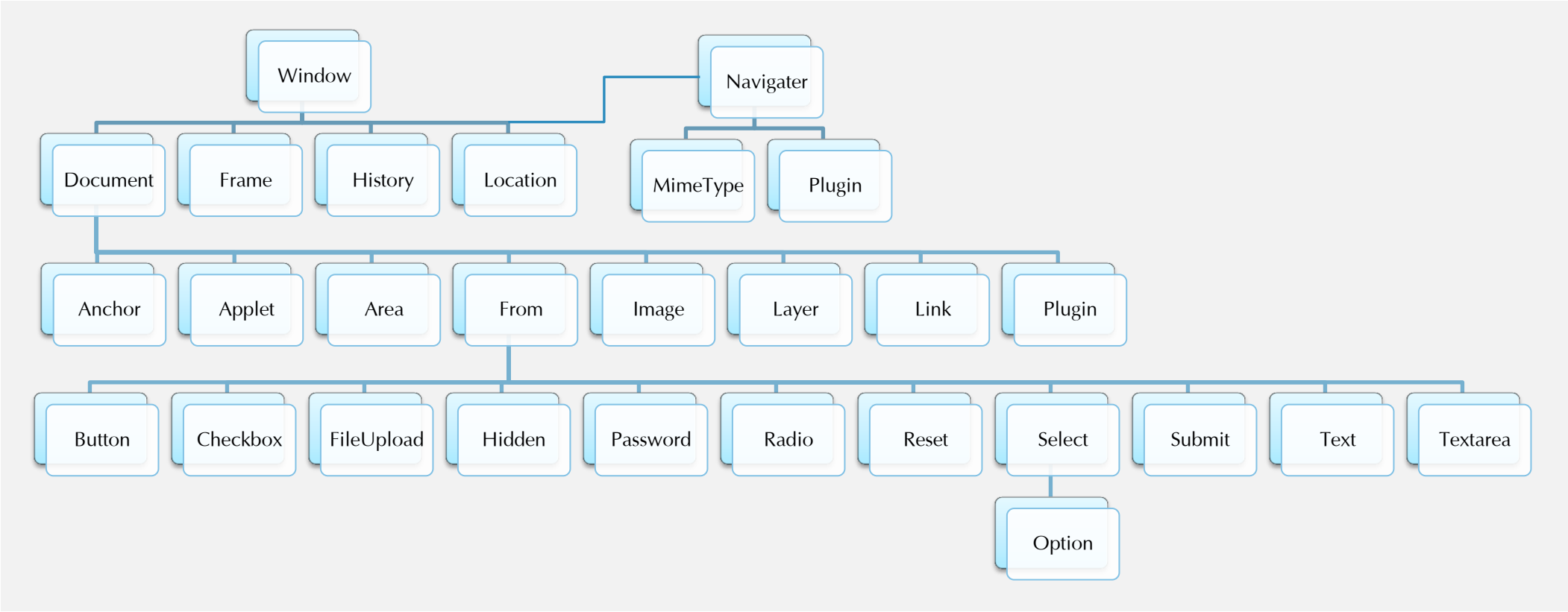
```
// 함수호출
함수 명(매개변수1, 매개변수2, … , 매개변수n);
```

```
<html>
<body>
  <script type="text/javascript">
     // 객체 리터럴
     var flight = {
           airline : "phonex",
           number : 123,
           departure : {
             IATA : "INC",
             time : "",
             city : "Inchon"
           },
           arrival : {
             IATA : "LAX",
             time : "2004-09-23 10:42",
             city : "Los Angeles"
           }
         };
  </script>
</body>
</html>
```

4. 웹 브라우저와 Window 객체

4.1 Window 객체 개요

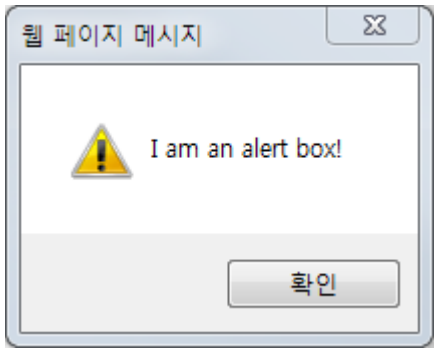
- ✓ Window 객체는 웹 브라우저에서 작동하는 JavaScript의 최상위 전역객체입니다.
- ✓ Window 객체에는 브라우저와 관련 된 여러 객체와 속성, 메소드가 있습니다.
- ✓ JavaScript에서 기본으로 제공하는 프라퍼티와 함수도 포함되어 있습니다.(Number 객체, setInterval() 함수 등)
- ✓ BOM(Browser Object Model)으로 불리기도 합니다.

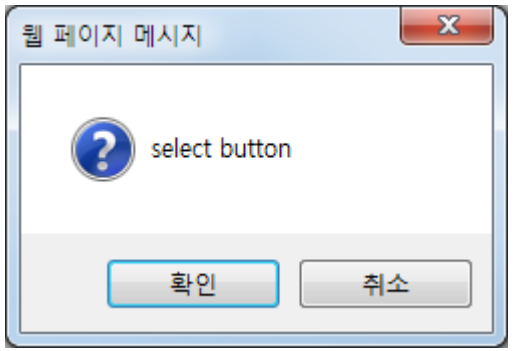


4.2 Window 객체 사용 (1/3) – alert, confirm, prompt

- ✓ Window 객체의 메소드를 호출하면 브라우저에서 제공하는 창을 띄울 수 있습니다.
- ✓ alert() : 브라우저의 알림창을 띄웁니다.
- ✓ confirm() : 브라우저의 확인/취소 선택창을 보여줍니다.
- ✓ prompt() : 브라우저의 입력 창을 띄워 사용자의 입력을 받습니다.

```
function myAlert() {
    alert("I am an alert box!");
}

function myConfirm() {
    if (confirm("select button")) {
        console.log("OK");
    } else {
        console.log("Cancel");
    }
}

function myPrompt() {
    console.log(prompt("input text", ""));
}
```





4.2 Window 객체 사용 (2/3) – Navigator

- ✓ Navigator 객체는 브라우저의 정보가 내장된 객체입니다.
- ✓ Navigator의 정보로 서로 다른 브라우저를 구분할 수 있으며, 브라우저 별로 다르게 처리할 수 있습니다.
- ✓ HTML5에서는 위치 정보를 알려주는 역할도 담당합니다.

```
console.log("Browser CodeName  : " + navigator.appCodeName);
console.log("Browser Name      : " + navigator.appName);
console.log("Browser Version   : " + navigator.appVersion);
console.log("Browser Enabled   : " + navigator.cookieEnabled);
console.log("Platform          : " + navigator.platform);
console.log("User-agent header : " + navigator.userAgent);
```

```
Browser CodeName  : Mozilla
Browser Name      : Netscape
Browser Version   : 5.0 (Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.81 Safari/537.36
Browser Enabled   : true
Platform          : MacIntel
User-agent header : Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.81 Safari/537.36
```

4.2 Window 객체 사용 (3/3) – Location, History

✓ Location 객체를 이용하여 현재 페이지 주소(URL)와 관련 된 정보들을 알 수 있습니다.

- location.href : 프라퍼티에 값을 할당하지 않으면 현재 URL을 조회하고 값을 할당하면 할당 된 URL로 페이지를 이동합니다.
- location.reload() : 현재 페이지를 새로 고칩니다.

✓ History 객체는 브라우저의 페이지 이력을 담는 객체입니다.

- history.back() / history.forward() : 브라우저의 뒤로 가기/앞으로 가기 버튼과 같은 동작을 합니다.

Location 객체

```
▼ Location {ancestorOrigins: DOMStringList, href: "http://www.nextree.co.kr/p4150/"}
  ▶ ancestorOrigins: DOMStringList
  ▶ assign: function () { [native code] }
    hash: ""
    host: "www.nextree.co.kr"
    hostname: "www.nextree.co.kr"
    href: "http://www.nextree.co.kr/p4150/"
    origin: "http://www.nextree.co.kr"
    pathname: "/p4150/"
    port: ""
    protocol: "http:"
  ▶ reload: function reload() { [native code] }
  ▶ replace: function () { [native code] }
    search: ""
  ▶ toString: function toString() { [native code] }
  ▶ valueOf: function valueOf() { [native code] }
  ▶ __proto__: Location
```

```
console.log(location.href);
location.href = "http://www.nextree.co.kr";
```

```
history.back();
history.forward();
```

4.3 새 창 열기 (1/3)

✓ Window 객체의 open() 메소드를 사용하면 새 창을 열 수 있습니다.

✓ window.open('페이지 URL', '창이름', '특성', 히스토리 대체여부)

- 창이름(string) : open 할 대상(_blank, _self 등) 지정 혹은 창의 이름
- 특성(string) : 새로 열릴 창의 너비, 높이 등의 특성
- 히스토리 대체 여부(boolean) : 현재 페이지 히스토리에 덮어 쓸지 여부 (true / false)

창 특성			
width	픽셀	All	창의 너비
height	픽셀	All	창의 높이
top	픽셀	All	창의 세로(y) 좌표 위치
left	yes | no	All	창의 가로(x) 좌표 위치
menubar	yes | no	IE, Firefox, Opera	메뉴 표시줄
status	yes | no	IE, Firefox, Opera	상태 표시줄
scrollbars	yes | no	IE, Firefox, Opera	스크롤바
toolbar	yes | no	IE, Firefox	도구모음
resizable	yes | no	IE 전용	창 크기 조절 가능여부
location	yes | no	Opera 전용	주소입력란

4.3 새 창 열기 (2/3) – 창 열고 닫기

- ✓ 이벤트를 이용하여 특정 시점에 창을 열 수 있습니다.
  - 페이지 로딩 완료 후 새 창 열기, 클릭할 때 새 창 열기 등
- ✓ window객체의 close() 메소드로 현재 창을 닫을 수 있습니다.
- ✓ 특히 브라우저에 내장 된 창이 아닌 JavaScript로 자체 구현한 팝업에서 필요합니다.

창 열기

```
<!DOCTYPE html>
<html>
<body>
<button onclick="windowOpen()">
    버튼 창열기
</button>
<a href="javascript:windowOpen()">링크 창열기</a>
<script>
  function windowOpen() {

    window.open("./S403_2_window_close.html",
       "subPopup", "width=300, height=200");
  }
</script>
</body>
</html>
```

창 닫기

```
<!DOCTYPE html>
<html>
<body>
  <a href="javascript:windowClose ()">
      함수이용해서 닫기
  </a><br/>
  <a href="javascript:window.close()">
      메소드 이용 닫기
  </a>

  <script>
    function windowClose () {
        window.close()
    }
  </script>
</body>
</html>
```

4.3 새 창 열기 (3/3) – 부모 창 컨트롤

- ✓ window 객체의 opener 속성을 이용하면 부모 창(새 창을 연 창)을 컨트롤 할 수 있습니다.
  - 부모 창에 값 전달
  - 부모 창을 새로 고침 하거나 페이지 이동
- ✓ opener 객체는 부모 창의 window 객체입니다.

```
<html>
<head>
<script type="text/javascript">
    // 부모 창에 값 전달 후 창 닫기
    function setOpenerData (data) {
        window.opener.setData(data);
        window.close();
    }
    // 부모 창 새로고침 후 창 닫기
    function reloadOpener() {
       opener.location.reload()
       self.close()
    }
</script>
</head>
```

```
<body>
    <a href="javascript:setOpenerData('value1')">
        선택1
    </a>
    <a href="javascript:setOpenerData('value2')">
        선택2
    </a>
    <a href="javascript:reloadOpener()">
        부모창 새로고침 후 닫기
    </a>
</body>
</html>
```

4.4 Window 객체 프라퍼티

- ✓ Window 객체는 웹 브라우저에서 구동 되는 JavaScript의 전역 객체입니다.
- ✓ 다음 장에서 살펴볼 document 객체는 HTML 문서와 관련된 객체로, 가장 많이 사용하는 객체입니다.
- ✓ screen 객체는 화면의 가로, 세로 크기 정보를 알 수 있습니다.
- ✓ pageYOffset등과 scroll() 메소드를 이용하면 현재 화면의 크기를 계산하여 페이지 단위로 스크롤 제어가 가능합니다.

설명			
self	현재 창 자신, window와 같음	statusbar	창의 상태 바
document	document 객체	toolbar	창의 툴 바
history	history 객체	personalbar	창의 퍼스널 바
location	location 객체	scrollbars	창의 스크롤 바
opener	open()으로 열린 창에서 볼 때 자기를 연 창	innerHeight	창 표시 영역의 높이(픽셀) (IE지원 안함)
parent	프레임에서 현재프레임의 상위프레임	innerWidth	창 표시 영역의 너비(픽셀) (IE지원 안함)
top	현재프레임의 최상위프레임	outerHeight	창 바깥쪽 둘레의 높이 (IE지원 안함)
frames	창안의 모든 프레임에 대한 배열정보	outerWidth	창 바깥쪽 둘레의 너비 (IE지원 안함)
locationbar	location 바	pageXOffset	현재 나타나는 페이지의 X위치 (IE지원 안함)
menubar	창 메뉴 바	pageYOffset	현재 나타나는 페이지의 Y위치 (IE지원 안함)

4.4 Window 객체 메소드 (1/2)

✓ 브라우저에서 버튼으로 제공하는 기능인 find, stop, print와 같은 메소드도 있습니다.

✓ move 메소드로 현재 열려 있는 창의 위치를 이동할 수 있습니다.

	설명	
alert()	경고용 대화상자를 보여줌	
confirm()	확인, 취소를 선택할 수 있는 대화상자를 보여줌	
prompt()	입력창이 있는 대화상자를 보여줌	
open()	새로운 창을 오픈	
scroll()	창을 스크롤 함	
find()	창안에 지정된 문자열이 있는지 확인. 있으면 true 없으면 false	IE지원 안함
stop()	불러오기를 중지	IE지원 안함
print()	화면에 있는 내용을 프린터로 출력	
moveBy()	창을 상대적인 좌표로 이동. 수평방향과 수직방향의 이동량을 픽셀로 지정	
moveTo()	창을 절대적인 좌표로 이동. 창의 왼쪽 상단 모서리를 기준으로 픽셀을 지정	

4.4 Window 객체 메소드 (2/2)

✓ resize 메소드로 현재 열려 있는 창의 크기를 조절할 수 있습니다.

✓ window 객체에는 브라우저와 관련 된 메소드 뿐만 아니라 순수 JavaScript에서 필요한 객체나 메소드도 있습니다.

- setTimeout() 메소드와 setInterval() 메소드로 함수를 특정 시간 후 혹은 특정 시간마다 호출할 수 있습니다.
- eval() 메소드는 문자열로 된 JavaScript 코드를 해석한 후 실행합니다.

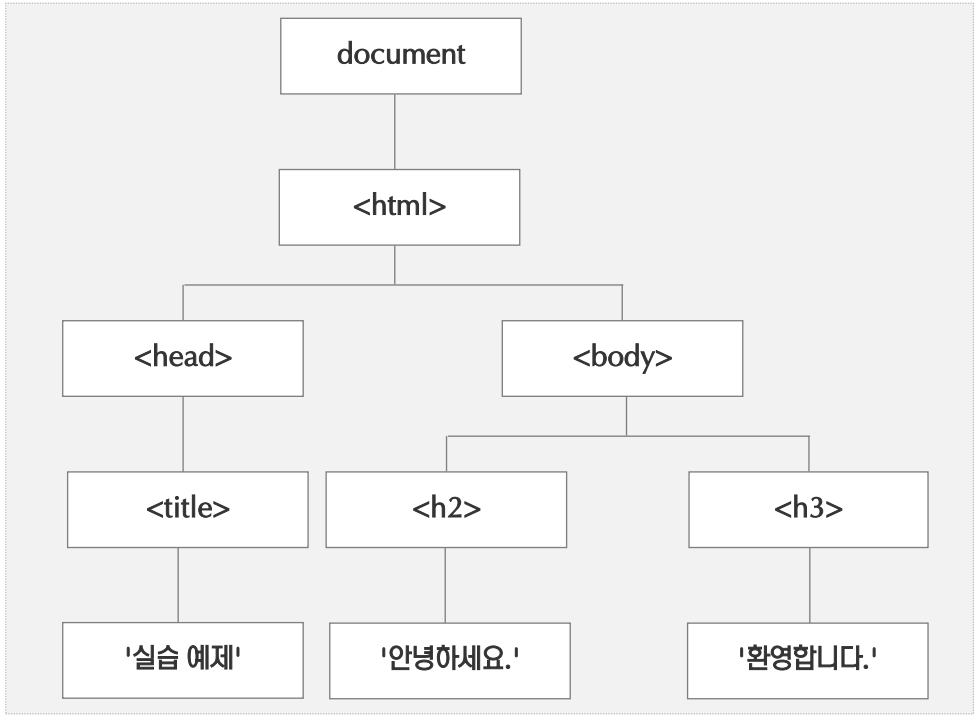
	설명
resizeBy()	창의 크기를 상대적인 좌표로 재설정
resizeTo()	창의 크기를 절대적인 좌표로 재설정, 창 크기를 픽셀로 지정
scrollBy()	창을 상대적인 좌표로 스크롤, 창의 표시영역의 수평방향과 수직방향에 대해 픽셀로 지정
scrollTo()	창을 절대적인 좌표를 스크롤 창의 왼쪽 상단 모서리를 기준으로 픽셀로 지정
setTimeout()	지정한 밀리초 시간이 흐른 후에 함수를 호출
clearTimeout()	setTimeout 메소드를 정지
setInterval()	지정한 밀리초 주기마다 함수를 반복적으로 호출
clearInterval()	setInterval 메소드의 정지
eval()	문자열을 JavaScript 코드로 변환하여 실행

5. HTML과 DOM

5.1 DOM 개요 (1/2)

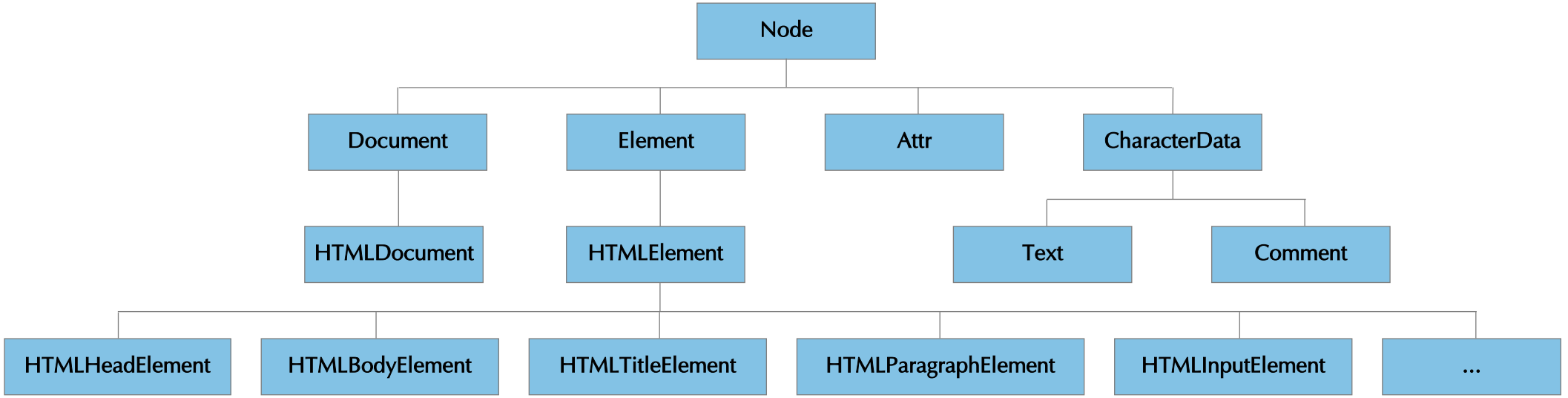
- ✓ DOM(Document Object Model)은 HTML과 XML 문서의 구조를 정의하고 API를 제공합니다.
- ✓ DOM은 객체의 계층 구조로 HTML을 표현합니다.
- ✓ HTML 계층 구조의 제일 위에는 document 노드가 있습니다.
- ✓ 그 아래로 HTML 태그나 요소(element)들을 표현하는 노드와 문자열을 표현하는 노드 등이 있습니다.

```
<html>
  <head>
    <title>실습  예제</title>
  </head>
  <body>
    <h2>안녕하세요.</h2>
    <h3>환영합니다.</h3>
  </body>
</html>
```



5.1 DOM 개요 (2/2) - 문서계층구조

- ✓ Document는 HTML 또는 XML 문서를 표현합니다.
- ✓ HTMLDocument는 HTML 문서와 요소 만을 표현합니다.
- ✓ HTMLElement의 하위 타입은 HTML 단일 요소나 요소 집합의 속성에 해당하는 JavaScript 프라퍼티를 정의합니다.
- ✓ Comment 노드는 HTML이나 XML의 주석을 표현합니다.

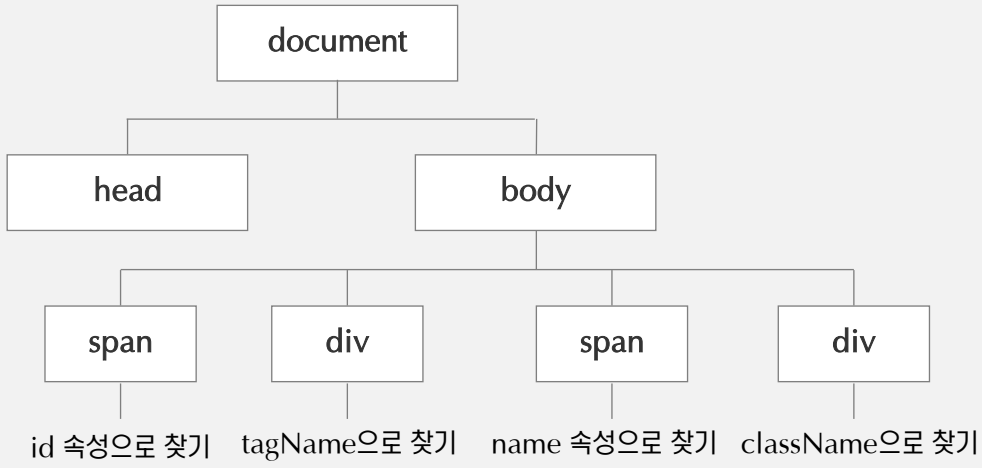


5.2 DOM 제어 (1/3)

- ✓ 웹 브라우저는 HTML(구조), CSS(표현), JavaScript(기능)의 협업을 통해 동적인 화면을 표현합니다.
- ✓ JavaScript를 사용하여 DOM을 검색하거나 제어할 수 있습니다.
  - HTML 요소(element), HTML 요소의 속성, CSS 스타일 변경, 이벤트를 제어를 할 수 있습니다.
- ✓ JavaScript는 id, name, HTML 태그 이름, class 등으로 문서 요소를 검색할 수 있습니다.

```
<!DOCTYPE html>
<html>
<body>
  <span id="content1-id">content1</span>
  <div>content2</div>
  <span name="content3-name">content3</span>
  <div class="content4-class">content4</div>

  <script>
    // 1. id속성으로 찾는 방법
    document.getElementById("contnet1-id");
    // 2. tagName으로 찾는 방법
    document.getElementsByTagName("div");
    // 3. name 속성으로 찾는 방법
    document.getElementsByName("content3-name");
    // 4. className으로 찾는 방법
    document.getElementsByClassName("content4-class");
  </script>
</body>
</html>
```

HTML의 계층 구조



5.2 DOM 제어 (2/3)

✓ DOM을 조회 할 때 id로 찾으면 가장 빨리 결과를 얻을 수 있습니다.
✓ HTML 표준 스펙 상, 같은 HTML 문서 안에 중복 되는 id가 있으면 안됩니다.
✓ 검색한 문서에 innerText나 innerHTML 속성으로 요소를 변경하거나 추가할 수 있습니다.

```
<!DOCTYPE html>
<html>

<body>
  <div id="div1">div1 content</div>

  <script>
    var element1 = window.document.getElementById("div1");
    element1.innerText = "new content";
  </script>

</body>
</html>
```

RESULT

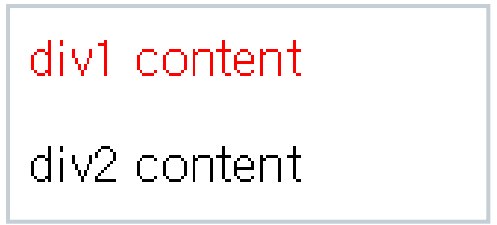
new content

5.2 DOM 제어 (3/3)

- ✓ JavaScript로 HTML 요소의 속성을 변경할 수 있습니다. → getAttribute('속성명'), setAttribute('속성명', '값')
- ✓ 문서 요소에서 style.color로 문서 요소(element)의 색을 변경할 수 있습니다.
- ✓ className 속성으로 문서 요소의 CSS 클래스를 변경할 수 있습니다.
- ✓ style.backgroundColor로 문서 요소의 배경색을 변경할 수 있습니다.

```
<!DOCTYPE html>
<html>

<body>
  <div id="div1">div1 content</div></br>
  <div>div2 content</div>

  <script type="text/javascript">
    document.getElementById("div1").style.color = "red";
  </script>

</body>
</html>
```

RESULT



5.3 이벤트 개요 (1/6)

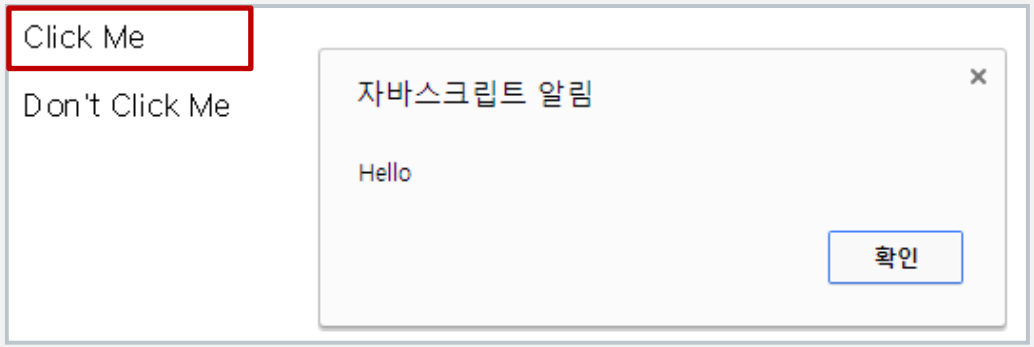
✓ 웹 페이지에서 여러 종류의 상호작용이 있을 때 마다 이벤트가 발생합니다.
✓ 사용자가 마우스를 클릭 하였을 때, 키보드를 눌렀을 때, 등과 같이 다양한 종류의 이벤트가 있습니다.
✓ JavaScript를 사용하여 DOM에서 발생하는 이벤트를 감지하여 이벤트에 대응하는 여러 작업을 할 수 있습니다.

이벤트 발생 종류
웹 페이지가 로딩되었을 때
페이지를 스크롤 했을 때
브라우저 창의 크기를 조절했을 때
마우스를 클릭했을 때
키보드로 키를 입력했을 때
Form이 Submit되었을 때
Input의 내용이 변경되었을 때
마우스를 움직여서 Element 를 이동할 때

5.3 이벤트 개요 (2/6) – 예제

✓ click 이벤트는 사용자가 마우스를 클릭 했을 때 발생합니다.
✓ 특정 DOM 요소에 한하여 click 이벤트를 제어할 수 있습니다.
✓ "Click Me"라는 문자열을 담고 있는 요소 영역을 클릭할 경우에만 "Hello" 알림 창이 표시됩니다.

[Click Me 영역을 클릭 시 이벤트 발생]



5.3 이벤트 개요 (3/6) - 마우스 이벤트

✓ 웹 초기에는 load, click 등 소수의 이벤트만 사용할 수 있었습니다.
✓ 마우스 이벤트는 웹 애플리케이션에서 가장 많이 사용하는 이벤트입니다.
✓ 마우스 이벤트 핸들러에 전달되는 이벤트 객체에는 마우스 위치와 버튼 상태 등의 정보들을 담고 있습니다.

이벤트	설명
onclick	마우스로 Element를 클릭 했을 때 발생합니다.
ondbclick	마우스로 Element를 더블 클릭 했을 때 발생합니다.
onmouseup	마우스로 Element에서 마우스 버튼을 올렸을 때 발생합니다.
onmousedown	마우스로 Element에서 마우스 버튼을 눌렀을 때 발생합니다.
onmouseenter	마우스를 움직여서 Element 밖에서 안으로 들어 올 때 발생합니다.
onmouseleave	마우스를 움직여서 Element 안에서 밖으로 나갈 때 발생합니다.

5.3 이벤트 개요 (4/6) – 키보드 이벤트

✓ 키보드의 커서가 웹 브라우저에 나타나는 지점에서 키보드를 조작할 때 이벤트가 발생합니다.
✓ 키보드 조작은 운영체제에 영향을 받으므로 특정 키가 이벤트 핸들러에게 전달되지 않을 수 있습니다.
✓ 키보드 커서가 나타내는 요소가 없다면 document에서 이벤트가 발생합니다.

이벤트	설명
onkeypress	키보드가 눌려 졌을 때 발생합니다. (키를 눌렀다가 떼면 발생)
onkeydown	키보드를 누르는 순간 발생합니다.
onkeyup	키보드 키가 올려 질 때 발생합니다.

5.3 이벤트 개요 (5/6) – Frame 이벤트

- ✓ Frame 관련 이벤트는 특정 DOM 문서에 관련된 이벤트가 아니라 Frame 자체에 대한 이벤트입니다.
- ✓ Frame 이벤트 중에서는 load 이벤트를 가장 많이 사용합니다.
- ✓ load는 문서 및 자원들이 모두 웹 브라우저에 탑재되면 이벤트를 수행합니다.
- ✓ unload는 사용자가 브라우저를 떠날 때 이벤트가 발생하지만, 사용자가 브라우저를 떠나는 것을 막을 수는 없습니다.

이벤트	설명
onload	document, image, frame 등이 모두 로딩 되었을 때 발생합니다.
onabort	이미지 등의 내용을 로딩하는 도중 취소 등으로 중단 되었을 때 발생합니다.
onerror	이미지 등의 내용을 로딩 중 오류가 발생 했을 때 발생합니다.
onresize	document, element의 크기가 변경 되었을 경우 발생됩니다.
onscroll	document, element가 스크롤 되었을 때 발생합니다.

5.3 이벤트 개요 (6/6) – 폼(form) 이벤트

- ✓ form 관련 이벤트는 웹 초기부터 지원되어 여러 웹 브라우저에서 가장 안정적으로 동작하는 이벤트입니다.
- ✓ 자주 사용되는 이벤트로, 폼이 전송될 때에는 submit 이벤트가 발생합니다.
- ✓ form을 초기화할 때는 reset 이벤트가 발생합니다.
- ✓ submit과 reset은 이벤트 핸들러에서 취소할 수 있습니다.

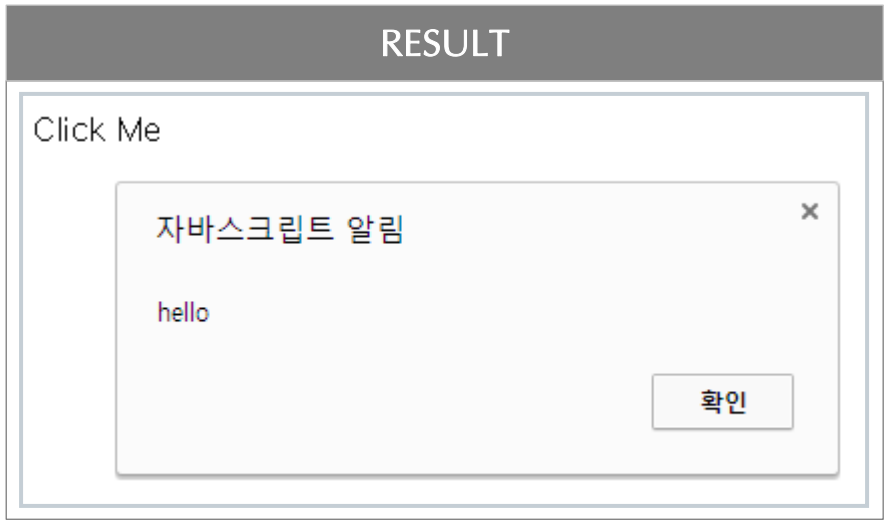
이벤트	설명
onsubmit	form이 전송될 때 발생합니다.
onreset	입력 form이 reset 될 때 발생합니다.
onblur	input과 같은 요소 등에서 입력 포커스가 다른 곳으로 이동할 때 발생합니다.
onchange	입력 내용이 변경 되었을 때 발생합니다.
onfocus	input과 같은 요소에 입력 포커스가 들어 올 때 발생합니다.
onselect	input, textarea에 입력 값 중 일부가 마우스 등으로 선택될 때 발생합니다.

5.4 이벤트 처리 (1/4) – 등록 (1/3)

✓ 이벤트를 감지하고, 대응하는 작업을 등록하는 방법은 여러 가지가 있습니다.
✓ 어떤 이벤트를 처리할 작업을 등록하는 것을 '이벤트 핸들러(혹은 리스너)를 등록한다'라고 표현합니다.
✓ JavaScript의 초기에는 HTML 요소의 내부에 직접 이벤트 핸들러를 등록하여 사용했습니다.
✓ 이러한 방식은 HTML 코드를 JavaScript 코드가 침범한다는 문제가 있습니다.

```
<!DOCTYPE html>
<html>

<body>

    <div onclick="alert('hello')">Click Me</div>

</body>
</html>
```

RESULT

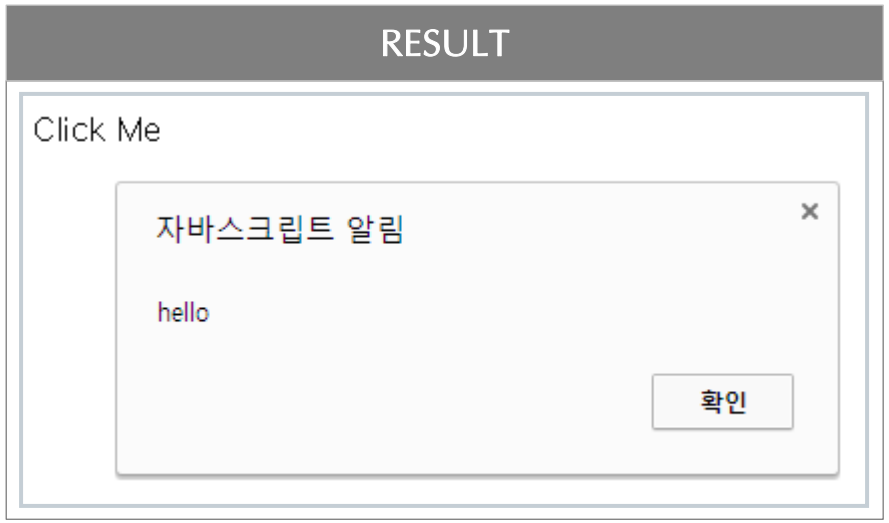


5.4 이벤트 처리 (2/4) – 등록 (2/3)

✓ HTML에 직접 이벤트 핸들러를 등록하는 방법 대신에 JavaScript에서 이벤트 핸들러를 등록하는 방법이 있습니다.
✓ JavaScript에서 이벤트 핸들러를 등록함으로써 HTML코드와 JavaScript 코드를 분리할 수 있습니다.
✓ 이벤트 대상이 되는 특정 DOM을 선택하고 이벤트 핸들러를 등록합니다.
✓ 'div1' 요소(element)에 클릭 이벤트가 발생하면 핸들러로 등록한 함수가 실행됩니다.

```
<!DOCTYPE html>
<html>
<body>
  <div id="div1">Click Me</div>

  <script type="text/javascript">

    document.getElementById("div1").onclick = function() {
      alert("hello");
    };

  </script>

</body>
</html>
```

RESULT



5.4 이벤트 처리 (3/4) - 등록 (3/3)

✓ 2000년에 발표된 DOM 레벨2 이벤트 명세의 addEventListener 를 이용하여 좀더 세밀한 이벤트 제어가 가능합니다.
✓ 전달인자의 첫 번째에는 이벤트 이름, 두 번째에는 이벤트 핸들러, 세 번째에는 캡쳐링 여부를 사용할 수 있습니다.
✓ 첫 번째 전달인자의 이벤트 이름에는 'on'을 제거한 이벤트 이름을 사용합니다.
✓ [주의 !!] 이 방식은 Internet Explorer 9 이전 버전에서는 사용할 수 없습니다.

```
<!DOCTYPE html>
<html>
<body>
  <div id="div1">Click Me</div>

  <script type="text/javascript">
    // DOM 레벨2 이벤트 명세 방법
    // 이벤트명에 on을 생략 (Internet Exploler 9 이전 버전 사용불가)
    document.getElementById("div1").addEventListener("click", fn_click, false);

    // Internet Exploler 9 이전 버전에서 사용하기 위한 방법
    document.getElementById("div1").attachEvent("onclick", fn_click);

    // 이벤트 핸들링 함수
    function fn_click() {
      alert("hello");
    }
  </script>
</body>
</html>
```

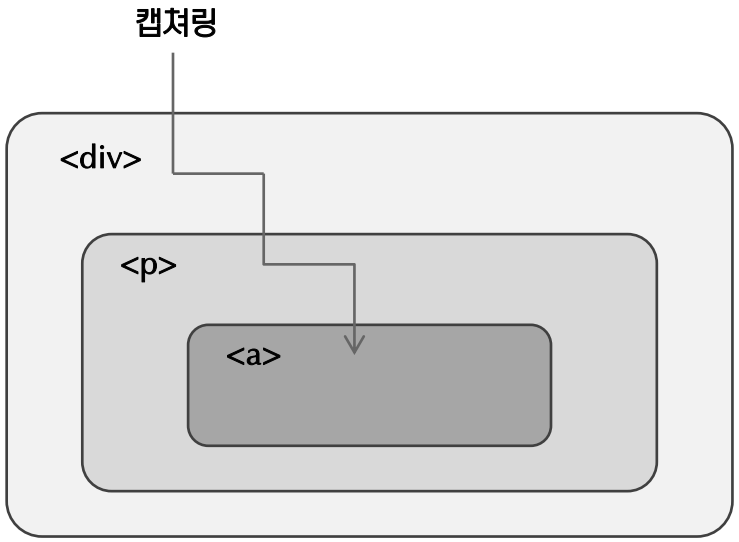
5.4 이벤트 처리 (4/4) – 버블링과 캡쳐링

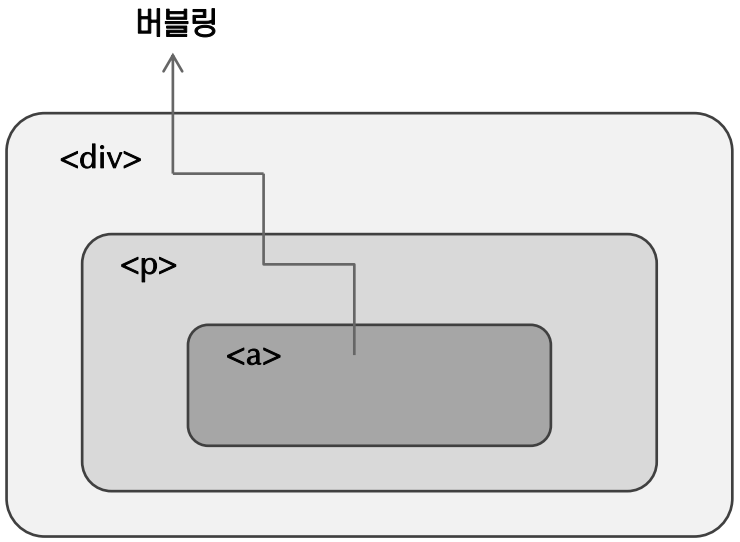
✓ **이벤트가 발생한 요소를 포함하는 부모 HTML로부터 이벤트 근원지인 자식요소까지 검사하는 것을 캡쳐링이라 합니다.**

- 이벤트 캡쳐링에서 캡쳐속성의 이벤트 핸들러가 등록되어 있으면 수행됩니다.

✓ **이벤트 발생 요소부터 요소를 포함하는 부모요소까지 올라가면서 이벤트를 검사하는 것을 이벤트 버블링이라 합니다.**

- 이벤트 버블링에서 버블속성의 이벤트 핸들러가 등록되어 있으면 수행됩니다.





5.5 이벤트 활용

- ✓ 하나의 DOM 엘리먼트에 복수의 이벤트 핸들러를 등록할 수 있습니다.
- ✓ 마우스가 특정 DOM 엘리먼트 영역 안으로 들어온 경우 mouseenter 이벤트가 발생합니다.
- ✓ 반대로 마우스가 특정 DOM 엘리먼트 영역 밖으로 나간 경우 mouseleave 이벤트가 발생합니다.
- ✓ 태그에 mouseenter, mouseleave 2가지 이벤트 핸들러를 등록합니다.

```
<!DOCTYPE html>
<html>
<body>
  <span id="span1" style="background:black;color:white">span1</span>

  <script>
    var span1 = document.getElementById("span1");

    // mouseenter 이벤트핸들러 등록
    span1.addEventListener("mouseenter", fn_mouseEnter, false);
    // mouseleave 이벤트핸들러 등록
    span1.addEventListener("mouseleave", fn_mouseLeave, false);
    // mouseenter 핸들러 함수
    function fn_mouseEnter() {
      span1.style.backgroundColor = "red";
    }
    // mouseleave 핸들러 함수
    function fn_mouseLeave() {
      span1.style.backgroundColor = "blue";
    }
  </script>
</body>
</html>
```

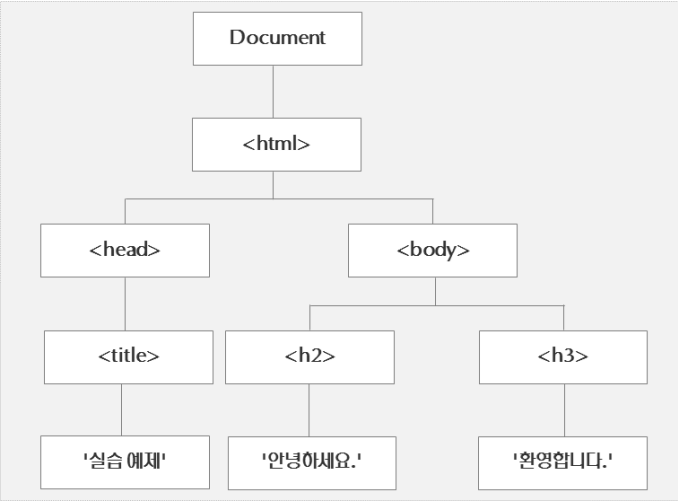
Default 상태

mouseenter 이벤트 발생 시

mouseleave 이벤트 발생 시

5.6 요약

- ✓ DOM은 HTML 문서의 내용을 조작할 수 있는 API로, HTML을 계층구조 형식의 객체로 표현합니다.
- ✓ DOM으로 HTML 문서의 검색과 조작(추가, 수정, 삭제)을 할 수 있습니다.
- ✓ DOM에서 발생되는 이벤트에 대한 핸들러(리스너)를 등록하여 특정 이벤트에 대응하는 작업을 할 수 있습니다.
- ✓ 핸들러 등록은 HTML태그의 on 속성에 명시하는 방법과 JavaScript에서 DOM 검색 후 등록하는 방법이 있습니다.



이벤트 발생 종류
웹 페이지가 로딩되었을 때
페이지를 스크롤 했을 때
브라우저 창의 크기를 조절했을 때
마우스를 클릭했을 때
키보드로 키를 입력했을 때
Form이 Submit되었을 때
Input의 내용이 변경되었을 때
마우스를 움직여서 Element 를 이동할 때

토론

✓ 질문과 대답

✓ 토론