ECMAScript 6

백명숙(myvega2k@gmail.com)

1장. JavaScript 개요

1. 자바스크립트 개요

1995년 Netscape사에서 개발됨.

원래 LiveScript로 불리워지기로 되어 있었지만 Java언어의 성공에 편승하려고 이름을 Javascript로 정함.

모든 웹 브라우저가 같은 Javascript을 사용하지 않는다.

유럽 컴퓨터 제조회사(ECMA)에서 ECMAScript 이름으로 표준화.

ECMA(European Computer Manufacturer Association: 유럽 전자계산기 공업회)

https://www.ecma-international.org/publications-and-standards/standards/ecma-262/

2. 자바스크립트 특징

웹 문서를 동적으로 바꾸거나 사용자가 입력한 폼 데이터의 유효성 검사기능을 포함하는 동적인 웹 어플리케이션 개발이 가능.

프로토타입 및 클래스 기반의 객체기반 프로그래밍 지원.

HTML5의 대부분의 기능이 자바스크립트를 활용함.

일반적으로 웹 페이지를 위한 스크립트 언어로 알려져 있으나 node.js 처럼 많은 비 브라우저 환경에서도 사용된다.

웹 브라우저에서 실행하면 브라우저에 내장된 자바스크립트 엔진에서 실행됨.

브라우저	랜더링엔진	자바스크립트 엔진
익스플로러	Trident	jscript(9은 챠크라)
파이어폭스	Gecko	traceMonkey
크롬	Webkit	V8

3. 자바스크립트 사용 방법

Case 1: body 태그내의 임의의 위치

CASE 1

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  :
<script>
</script>
  :
</body>
</html>
```

Case 2: head 태그내

CASE 2

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<script>
</script>
</head>
<body>
  :
</body>
</html>
```

3. 자바스크립트 사용 방법

Case 3: body 태그내의 마지막 위치

CASE 3

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  :
  :
<script>
</script>
</body>
</html>
```

Case 4: 외부 파일

CASE 4

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<script src="XXX.js"></script>
</head>
<body>
  :
</body>
</html>
```

XXX.js

4. 자바스크립트 주석

가. 한 줄 주석문

```
// 문장
```

나. 여러 줄 주석문

```
/*
    문장1;
    문장2;
 */
```

ECMAScript 6

ECMAScript (ES)는 ECMAScript International에서 표준화 한 스크립팅 언어 사양으로서 JavaScript. Jscript, ActionScript 등은 ECMAScript 스펙 적용을 받는다.

Online Archives

- ECMA-262 5.1 edition, June 2011
- ECMA-262, 6th edition, June 2015
- ECMA-262, 7th edition, June 2016
- ECMA-262, 8th edition, June 2017
- ECMA-262, 9th edition, June 2018
- ECMA-262, 10th edition, June 2019
- ECMA-262, 11th edition, June 2020

Standard ECMA-262
6th Edition / June 2015

ECMAScript® 2015 Language Specification

This is the HTML rendering of *ECMA-262 6th Edition, The ECMAScript 2015 Language Specification.*

The PDF rendering of this document is located at http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf.

The PDF version is the definitive specification. Any discrepancies between this HTML version and the PDF version are unintentional.

ECMAScript 6의 주요 기능

Support for constants (상수지원)
Block Scope (블록 범위)
Arrow Functions (화살표 기능)
Extended Parameter Handling (확장 매개변수 처리)
Template Literals (템플릿 리터럴)
Enhanced Object Properties (향상된 개체 속성)
De-structuring Assignment (비 구조화)
Modules (모듈)
Classes (classes)
Iterators (이터레이터)
Generators (제너레이터)
Collections (컬렉션)
New built in methods for various classes (새로운 메서드)
Promises (프라미스)
등의 객체지향 프로그래밍의 다양한 특징 반영됨.

2장. 식별자와 데이터형, 변수

1. 식별자(identifier)

식별자는 자바스크립트 코드 내에서 사용되는 각각의 단어를 의미한다.

1) 시스템 정의 식별자

자바스크립트 내부에서 먼저 정의한 식별자로서 '예약어', '키워드'라고 부른다.

[JavaScript에서의 예약어]

break	case	catch	continue	default	delete
do	else	finally	for	function	if
in	instanceof	new	return	switch	this
throw	try	typeofvar	void	while	with

2) 사용자 정의 식별자

개발자가 필요에 의해서 정의한 식별자 로서 변수,함수,생성자, 객체 정의시 사용할 이름을 의미한다.

2. 사용자 정의 식별자 명명 규칙

첫 문자는 반드시 영문자, _ , $ 문자로 시작되어야 하고
그 다음부터는 숫자와 영문자를 혼합해서 사용 가능하다.
자바스크립트는 대소문자를 구별한다.
예약어는 사용 불가하다.

기술 방법	개요	예
camelCase 기법	앞 단어 첫 문자는 소문자, 그 이후의 단어의 첫 문자는 대문자	lastName
Pascal 기법	모든 단어의 첫 문자는 대문자	LastName
언더스코프 기법	모든 단어의 첫 문자는 소문자, 단어 간은 '_'로 연결	last_name

3. 자바스크립트의 데이터형(Data Type)

데이터 형은 자바스크립트 언어가 처리할 수 있는 데이터 종류를 의미한다.

1) 기본 데이터형 (Primitive Data Type: PDT)

수치 데이터: 정수와 실수 (“number”)
문자 데이터: 문자와 문자열 (“string”) 및 이스케이프 문자
논리 데이터: true 또는 false (“boolean”)
undefined: 변수가 초기화되지 않은 상태
null: 객체 없음(초기화는 된 상태)
infinite: 양의 무한대

2) 참조 데이터형 (Reference Data Type; RDT)

객체형 (배열)과 함수형.
객체 표현은 {key:value} 형식
배열 표현은 [값, 값2,..] 형식

3. 자바스크립트의 데이터형(Data Type)

분류	데이터형	개요
기본형	수치형(number)	$\pm 4.9406564584124654 \times 100^{-324}$ ~ $\pm 1.79769313486231570 \times 10^{308}$
	문자열형(string)	작은 따옴표/큰 따옴표로 감싸인 0 개 이상의 문자 집합
	논리형(boolean)	true(참)/false(거짓)
	특수형(null/undefined)	값이 미 정의된 것을 나타냄
참조형	배열(array)	데이터의 집합(각 요소에는 인덱스 번호로 접근 가능)
	객체(object)	데이터의 집합(각 요소에는 이름으로 접근 가능)
	함수(function)	일련의 처리(절차)의 집합

변수 개요

프로그램에서 사용하는 데이터(literal)를 저장하기 위한 용도로 사용된다.
저장된 데이터는 언제든지 변경이 가능하기 때문에 '변경이 가능한 수'
즉 변수라고 부른다. 변경이 불가능한 수는 '상수'라고 부른다.
기본형 데이터를 저장하면 '기본형 변수' 라고 하고 참조형 데이터를 저장하면 '참조형 변수'라고 한다.

변수 사용

가. 변수 선언 (초기값은 undefined 값이 할당)

```
var 변수명;
let 변수명;
```

나. 변수 초기화

```
변수명 = 값;  //런타임시에 지정된 값으로 할당된다.
```

변수 특징

데이터형을 지정하지 않는다.(실행단계에서 데이터형이 지정됨)
var 변수는 함수단위로 scope가 정해지고 let 변수는 블록단위로 scope가 정해진다.
데이터 타입을 철저하게 검사하지 않기 때문에 데이터 타입 변환이 자유롭다.
실행할 때 변수에 저장되는 데이터형에 의해서 변수 타입이 결정된다.
typeof 연산자를 사용하여 저장된 데이터 타입을 확인할 수 있다.

예>

```
var  test  = 10;   //   number 타입

test = “홍길동”;  //  string 타입

test   =  true;      //  boolean 타입
```

6. 데이터 자동 형변환

자바스크립트의 데이터는 필요에 의해서 자동으로 형변환 될 수 있다.

1) *,/,- 사용하는 경우 (+제외)

예>

var test = “100” * 2; ➔ 200

2) boolean 값으로 자동 변환

다음과 같은 5가지 데이터는 필요에 의해서 false 값으로 자동 변환된다.
이외의 데이터는 필요시 true 값으로 변환된다.

0
‘’
NaN
null
undefined

7. 데이터 형 검사

프로그램 실행중 변수에 설정된 데이터의 타입을 검사하기 위한 방법으로서 실행결과는 문자열로 반환된다.

문법

```
typeof (변수)  또는  typeof 변수
```

```
예> var name = "홍길동";      typeof (name) 결과:    'string'
    var age = 20;             typeof (age) 결과:     'number'
    var array = [1,2];        typeof (array) 결과:   'object'
    var obj = {};             typeof (obj) 결과:     'object'
    var a = false;            typeof (a) 결과:       'boolean'
    var b = null;             typeof (b) 결과:       'object'
    var c;                    typeof (c) 결과:       'undefined'
    var k = function(){};     typeof (k) 결과:       'function'
```

8. 호이스팅 (hoisting)

var 키워드를 사용하는 자바스크립트 코드는 함수안에 있는 모든 변수를 함수 맨 꼭대기로 '끌어올린'것처럼 동작하며 이것을 호이스팅(hosting)이라고 부른다. 이때 선언만 되고 초기화는 아직 안된 상태이기 때문에 undefined가 저장된다.

원본 코드

```
function f(){
   console.log( scope );      // undefined
   var scope = "local";
   console.log(scope);        // local
```

호이스팅 코드

```
function f(){
    var scope;
    console.log( scope );     // undefined
    scope = "local";
   console.log(scope);        // local
```

9. var vs let

var 키워드

변수 선언시 사용하며 변수명 중복 사용 가능
블록 스코프가 아닌 함수 스코프(function scope)를 따른다.
호이스팅(hoisting) 되어 처리된다.

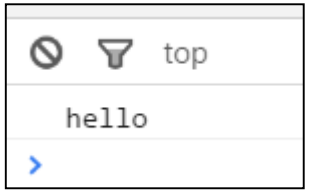
let 키워드

var 키워드의 문제점 해결 목적으로 등장
변수명 중복 사용 불가
let 키워드를 사용하면 함수 스코프가 아닌 블록 스코프를 따른다.
let 변수는 호이스팅(hoisting)되지 않는다.

9. var vs let

var 키워드

```
<script type="text/javascript">

      if(true){
          var mesg = "hello";
      }
      console.log(mesg);
</script>
```




```
for(var i = 0 ; i < 10 ; i++){
    console.log("aaa");
}
console.log(i);
```

let 키워드

```
<script type="text/javascript">

      if(true){
          let mesg = "hello";
      }
      console.log(mesg);
</script>
```


```
for(let i = 0 ; i < 10 ; i++){
    console.log("aaa");
}
console.log(i);
```

const 키워드

상수 작성시 사용.
const 변수는 선언만 할 수 없으며 반드시 초기화 필요.
const 변수는 값을 변경할 수 없는 것만 제외하고 let 변수와 기능이 동일.

```
const mesg = "hello";
try{
    mesg = "world";
}catch(e){
    console.log("상수값 변경 불가");
}
```

```
Navigated to http://localhost:8(
html
상수값 변경 불가
```

3장. 연산자

1. 산술 연산자

구분	연산자	의미
산술 연산자	+	더하기
	-	빼기
	*	곱하기
	/	나누기
	%	나머지 값 구하기

주의할 점

문자열과 다른 데이터형이 + 연산을 하면 문자열이 연결된다.

예> var test = “ABCD” + 23; ➔ “ABCD23”
var test2 = “30” + 10; ➔ “3010”

2. 대입 연산자

구분	연산자	의미
대입 연산자	=	연산자를 중심으로 오른쪽 변수값을 왼쪽 변수에 대입한다.
	+=	왼쪽 변수에 더하면서 대입한다.
	-=	왼쪽 변수값에서 빼면서 대입한다.
	*=	왼쪽 변수에 곱하면서 대입한다.
	/=	왼쪽 변수에 나누면서 대입한다.
	%=	왼쪽 변수에 나머지 값을 구하면서 대입한다.

예>

```
var n = 10;
    n += 5;        //   n = n + 5;   동일
```

3. 비교 연산자

- 변수나 상수의 값을 비교할 때 쓰이는 연산자로서 결과는 항상 true 또는 false인 논리값(boolean)을 반환한다.

구분	연산자	의미
비교 연산자	〉	크다.
	〈	작다.
	〉=	크거나 같다.
	〈=	작거나 같다.
	= =	피연산자들의 값이 같다.
	!=	피연산자들의 값이 같지 않다.

4. 동등 비교

1) == , !=

equal 연산자로서 값만 비교한다.

```
예>  var  w = 10;
     var  w2 = "10";
     console.log( w == w2 );  //  true
```

2) === , !==

identical 연산자로서 값과 데이터 타입까지 비교한다.

```
예>  var  w = 10;
     var  w2 = "10";
     console.log( w  === w2 );  // false
```

---

undefined 비교할 때는 반드시 === 연산자를 사용해야 된다.

```
var x ;
if( x === undefined ){ }
```

6. 논리 연산자

- 기본적으로 true나 false인 논리 값을 사용하여 연산하는 연산자이다.

구분	연산자	의미	설명
논리 연산자	&&	and(논리곱)	주어진 조건들이 모두 true일 때만 true를 나타낸다.
	||	or(논리합)	주어진 조건들 중 하나라도 true이면 true를 나타낸다.
	!	not(부정)	true는 false로 false는 true로 나타낸다.

자바스크립트에서는 다른 프로그램언어와 다르게 논리값만 논리연산자를 사용하지 않고 다른 데이터도 논리 연산자를 사용할 수 있다.
최종적으로 반환되는 값은 true/false값이 아닌 좌측 또는 우측 피연산자의 최종 평가값이다.

좌객체 || 우객체

==> 좌객체가 참이면 우객체를 평가하지 않고 좌객체 값을 리턴 한다.
좌객체가 거짓이면 우객체 값을 리턴한다.

예> console.log (“123” || 0); ➔ “123”

예>
어떤 함수가 인자를 받을 때 반드시 배열을 받기로 가정한 경우?

```
function x( a ){
    var arr = a || [] ;
    …
}
x();
x([1,2]);
```

7. 증감 연산자

- 1씩 증가 또는 감소시키는 연산자이다.
- 주의할 점은 다른 연산자와 같이 사용시 전치 및 후치 표현에 따라서 결과값이 다르게 산출된다.

구분	연산자	의미
증감 연산자	++	1씩 증가시킨다.
	--	1씩 감소시킨다.

예>

```
var n = 10;

var n2 = n++;   //  ++n;
console.log(n , n2 );
```

8. 3항 연산자

- 하나의 조건을 정의하여 만족 시에는 '참값'을 반환하고 아니면 '거짓값'을 반환하여 단순 비교에 의해 변화를 유도하는 연산자이다.

구분	연산자	의미	구성
조건 연산자	? :	제어문의 단일 비교문과 유사하다.	조건식 ? 참값 : 거짓값

예>

```
var n = 10;

var result = (n > 4)? true: false;
```

9. spread 연산자

- '… 배열' 형식으로 사용되고 배열요소의 값을 펼치는 효과를 얻는다.
- 일반적으로 함수의 파라미터로 사용되거나 중첩 배열에서 사용된다.

```
function aa(x,y,z){
   console.log(x+y+z);
}

var x = [10,20,30];

aa(x); //undefined
aa(...x);  //60
aa(...[10,20,30]); //60
```

```
<script type="text/javascript">

      var aa = [1,2,3, ...[10,9,8]];
      console.log(aa);
      for(let i=0 ; i < aa.length; i++){
          console.log(aa[i]);
      }
</script>
```

```
▶ [1, 2, 3, 10, 9, 8]
1
2
3
10
9
8
```

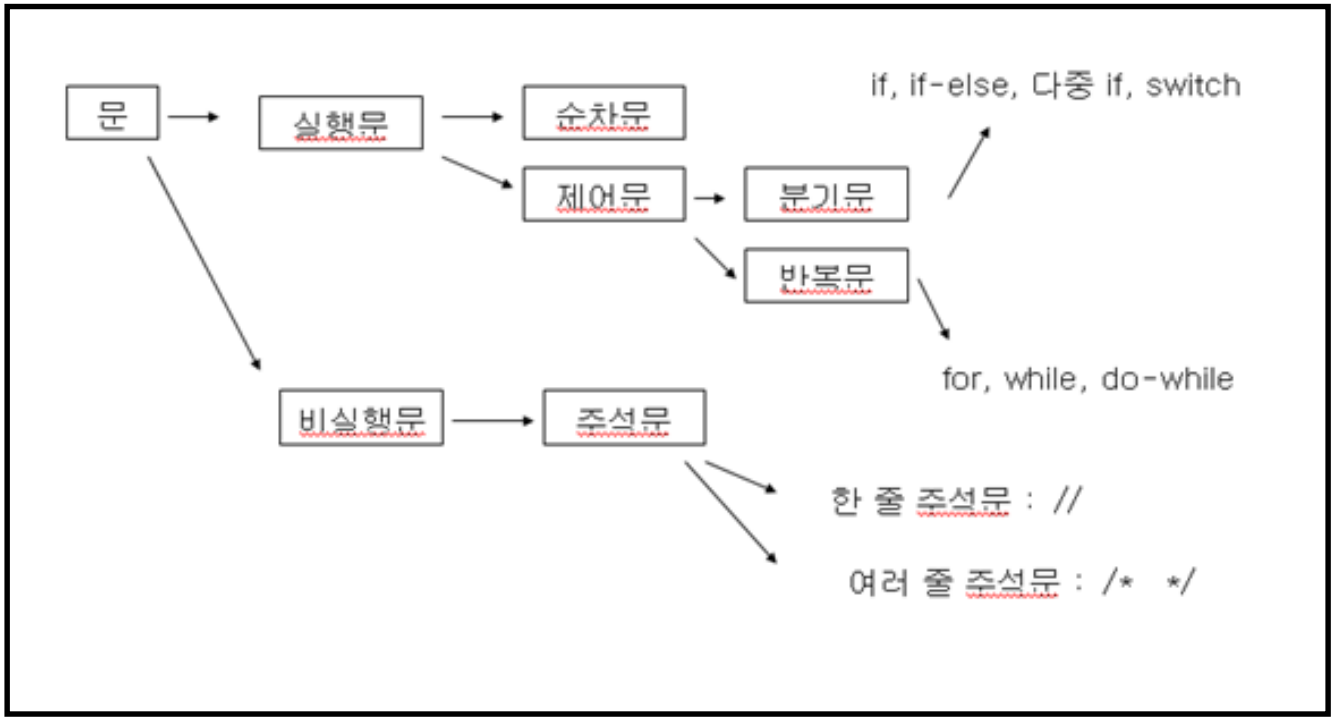
9. 연산자 우선 순위

우선순위	연산자
높다	배열([]), 괄호(())
	증가 연산자(++), 감소 연산자(--), 단항 뺄셈(-), 반전(~), 부정(!)
	곱셈(*), 나눗셈(/), 나머지(%)
	더하기(+), 뺄셈(-), 문자열 결합(+)
	쉬프트(<<, >>, <<<)
	비교 (< , <=, >=, >)
	등가(==), 부등가(!=), 동치(====), 비동치(!==)
	AND (&)
	XOR (^)
	OR (|)
	논리AND(&&)
	논리OR (||)
낮다	조건(?:)
	대입 (=), 복합 대입 (+=, -=등)
	콤마(,)

4장. 문장

1. 문장의 종류

문장(statement)은 프로그램을 개발하기 위해서 자바스크립트 소스코드에 입력시키는 코드를 의미한다.



2. 비 실행문 (주석문)

- 실제 프로그램에 영향을 주지 않으며 단지 소스코드의 기능이나 동작을 설명하기 위해 사용되는 문장이다.

주석 종류	의미	설명
// 주석문	단행 주석처리	현재 행에서 //의 뒷문장부터 주석으로 처리된다.
/* 주석문 */	다행 주석처리	/*에서 */ 사이의 문장이 주석으로 처리된다.

순차문

- 소스코드내의 문장 중에서 순차적으로 실행되는 문장을 의미한다.
- 반드시 ;(세미콜론)으로 끝나며 소스코드의 대부분이 순차문에 해당된다.

제어문

- 프로그램의 흐름에 영향을 주고 특정 조건에 따라 제어가 가능하도록 사용하는 문장이 바로 제어문이다.
- 모든 제어문은 중첩이 가능하다.

3. 제어문 종류

분기문 (조건문)

주어진 조건의 결과에 따라 실행 문장을 다르게 하여 전혀 다른 결과를 얻기 위해 사용되는 제어문이다.
if문, if~else문, 다중 if문, switch문이 있다.

반복문

특정한 문장을 정해진 규칙에 따라 반복 처리하기 위한 제어문이다.
for문, while문, do~while 문이 있다.

break문

반복문 내에서 사용되며 반복문을 빠져 나갈 때 사용되는 제어문이다.
switch문에서 사용시 switch 블록을 빠져나간다.

continue문

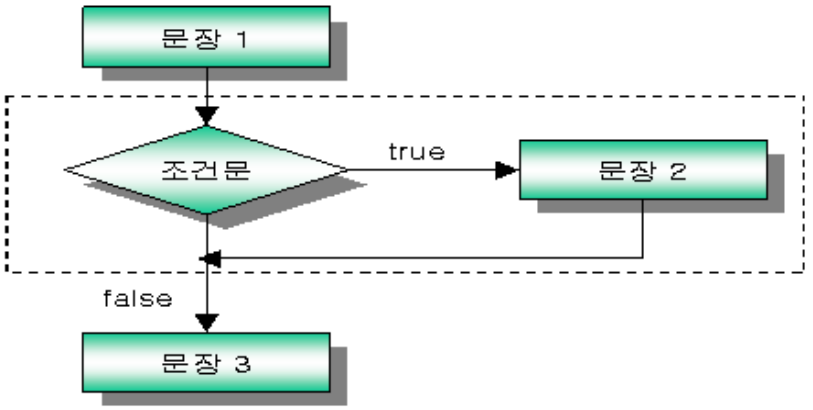
반복문 내에서 사용되며 현재 진행되는 반복 회차를 포기하고 다음 회차로 이동한다.

4. 단일 if 문

주어진 조건을 만족하는 경우에만 특정 문장을 수행하도록 제어하는 문이다.

문법

```
    문장1;
if(조건식){
    문장2;
 }
    문장3;
```



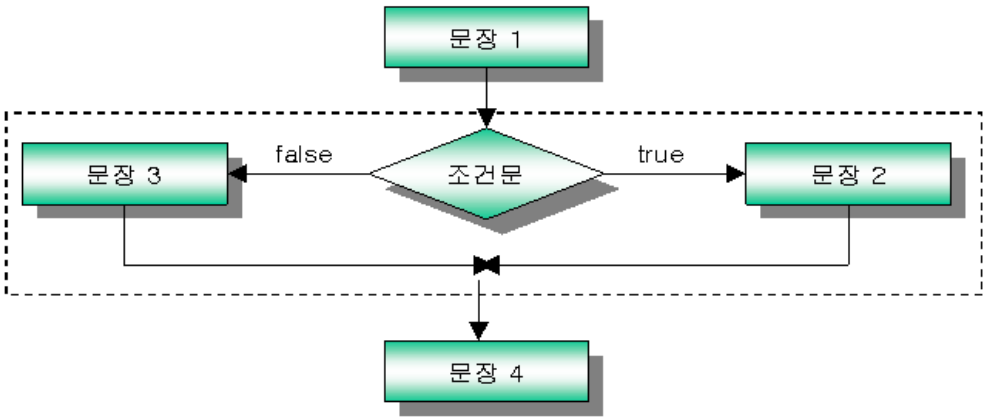
var 변수는 블록단위로 관리되지 않고 함수단위 scope로 동작된다.

5. if ~ else 문

조건식의 결과에 따라서 실행되는 문장이 서로 다른 경우에 사용한다.

문법

```
  문장1;
if(조건식){
    문장2;
}else{
    문장3;
}
   문장4;
```

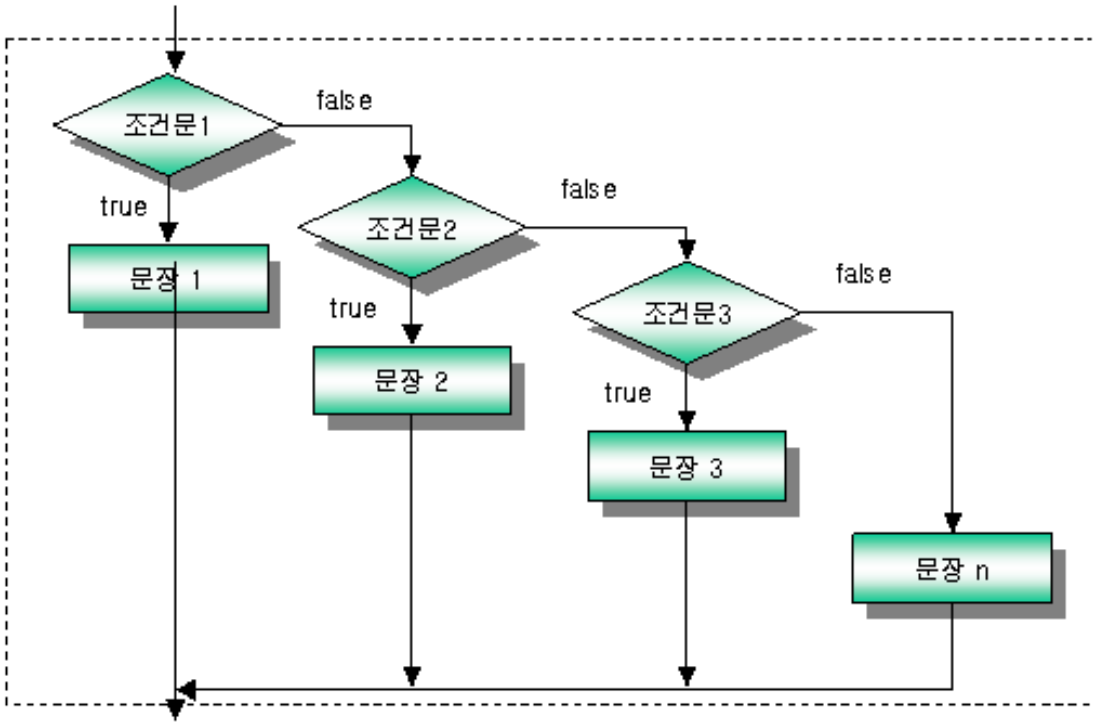


6. 다중 if ~ else 문

비교할 조건식이 여러 개인 경우에 사용된다.

문법

```
if(조건식1){
   문장1;
 }else if(조건식2){
   문장2;
 }else if(조건식3){
    문장3;
 }else{
     문장n;
  }
```



7. switch 문

다중 if ~ else 문과 비슷한 용도로 사용되며 일반적으로 동등비교인 경우에 사용된다.

지정된 인자값과 동일한 조건값을 가진 case 문이 실행되고 이후에 break문에 의해서 switch문을 빠져나온다.

break 문은 필수가 아닌 옵션이다.

문법

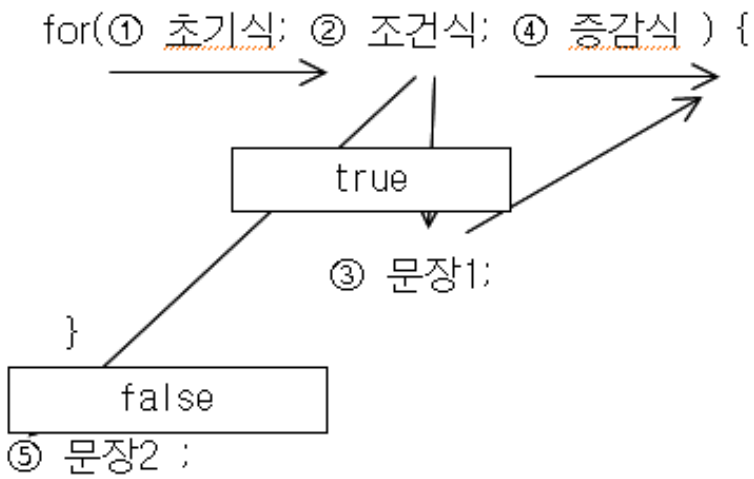
```
switch(인자값) {
   case 조건값1 : 실행문;  break;
   case 조건값2 : 실행문;  break;
   case 조건값3 : 실행문;  break;
   default : 실행문;
 }
```

8. for 문

초기식, 조건식, 증감식으로 구성되고 지정된 횟수만큼 반복 처리된다.
일반적으로 반복횟수가 예측 가능할 때 사용된다.

문법

```
for(초기식;조건식;증감식){
      문장1;
}
 문장2;
```



for문과 문법적인 형태만 다르며 동일한 방식으로 동작한다.

차이점은 for문은 초기식,조건식,증감식이 지정된 위치가 존재하지만 while문은 조건식만 정해져 있기 때문에 초기식과 증감식은 적당한 위치에 명시적으로 지정해야 된다.

반복횟수가 예측 불가능한 경우에 사용된다.

문법

```
초기식;
while(조건식){
        문장;
        증감식;
}
```

10. do~while 문

while문과 비슷하지만 차이점은 조건식이 일치하지 않더라도 반드시 한번은 문장이 실행된다.

따라서 조건이 일치하지 않더라도 적어도 한번은 꼭 수행되어야 하는 경우에 사용된다.

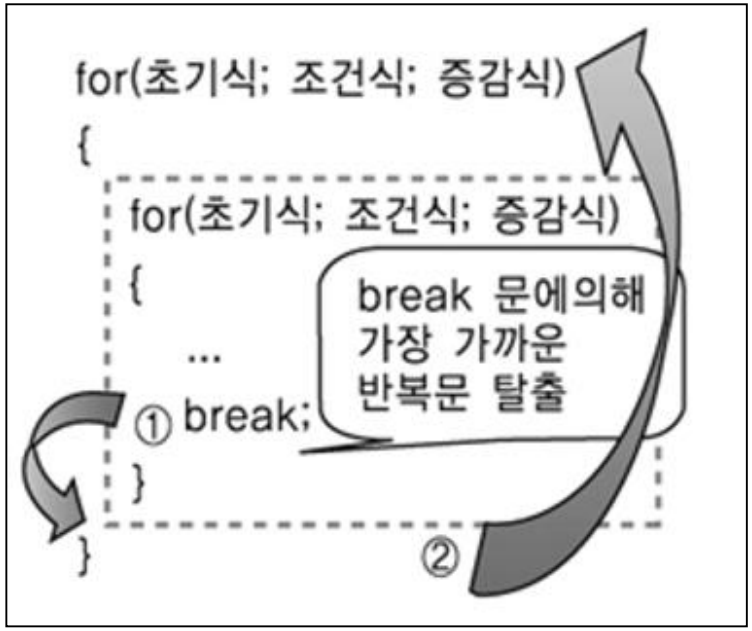
문법

```
초기식;
do{
        문장;
        증감식;
}while(조건식);
```

반복문 내에 또다른 반복문을 중첩으로 지정하여 처리하는 문이다.
for문, while문, do~ while문 모두 사용 가능하다.

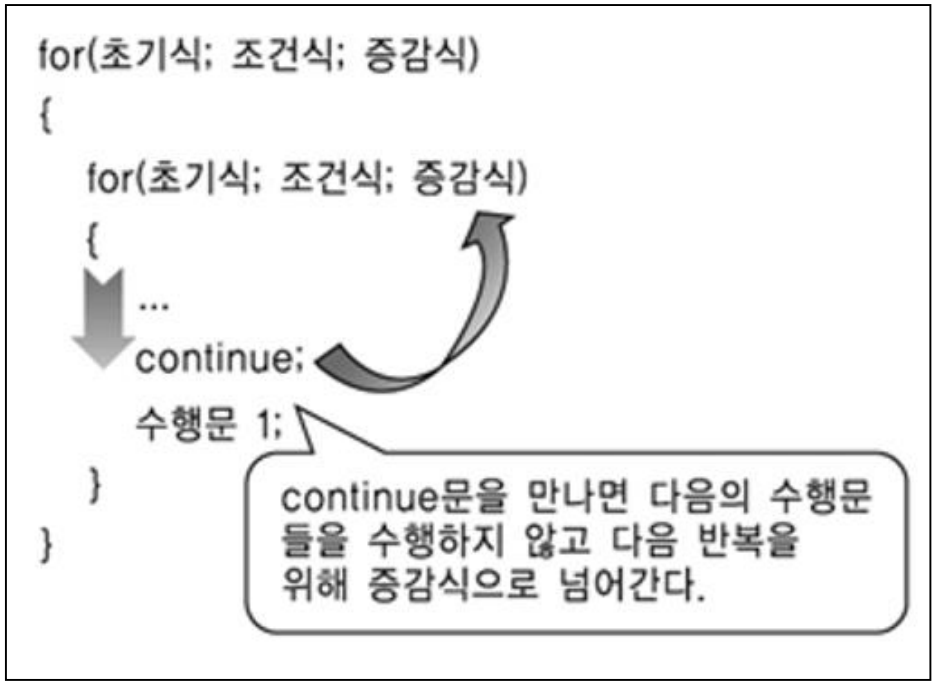
```
for(초기식1 ; 조건식1 ; 증감식1) {
        명령어1;
    for(초기식2 ; 조건식2 ; 증감식2){
        명령어2;
    }
}
명령어3;
```

가장 가까운 반복문을 빠져 나올 때 사용되는 제어문이다.



13. continue 문

반복문을 탈출하기 위해 사용되는 것이 아니라 continue문 이하의 수행문들을 skip하고 다음 회차의 반복을 수행하기 위한 제어문이다.



5장. 내장 객체 및 JSON

1. 객체의 종류

1) 시스템 정의 객체 (내장 객체)

데이터 관련 객체

String , Number , Date , Array, Boolean , Object , Math, RegExp

브라우저 관련 객체

Window, Screen , Location , History, Navigator

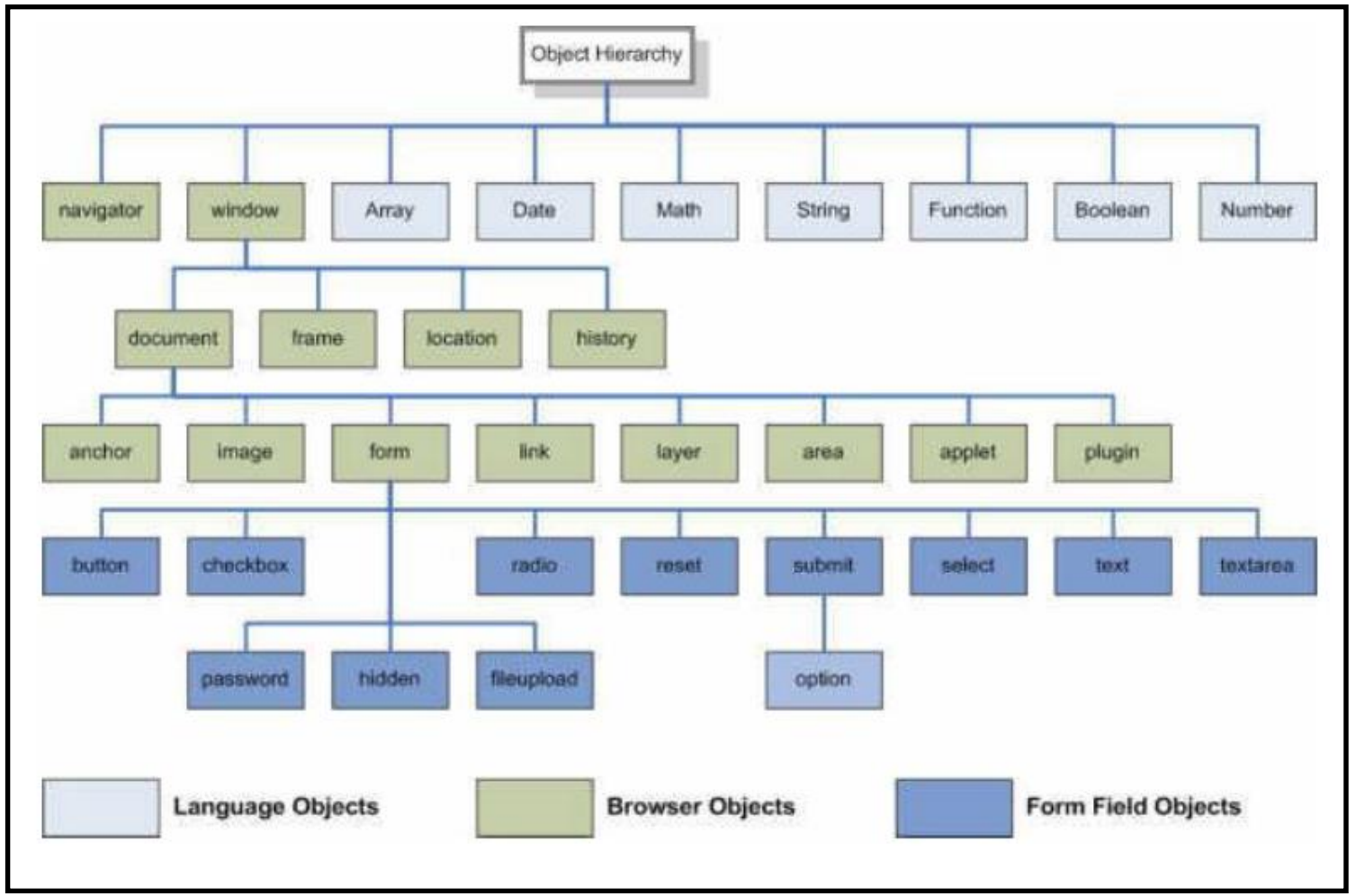
HTML 관련 객체

DOM(Document Object Model) 객체

2) 사용자 정의 객체

데이터 관리 목적으로 생성

2. 자바스크립트의 내장객체



문자열을 처리하는 객체로서 2가지 생성방법이 제공된다.

문법1

```
let str = new String("hello");
```

문법2

```
let str = "hello";
```

String객체 요소

속성	설명
constructor	String 객체의 원본함수(생성자)를 반환한다.
length	문자열의 길이를 반환한다.
prototype	String 객체의 prototype를 반환한다.

함수명	설명
charAt(index)	명시된 index에 해당하는 문자를 반환한다.
concat(s,s2,s3,…)	지정한 s, s2 문자열등을 연결하여 반환한다.
indexOf(str)	지정한 str에 해당하는 값의 index값을 반환한다. 일치하는 str이 없으면 -1 값을 반환한다.
split(구분자 , [개수])	지정한 문자열에서 특정 구분자를 기준으로 문자열을 분리하여 얻고자 할 때 사용한다.
toLowerCase()	지정한 문자열을 모두 소문자로 변경하여 반환한다.
toLowerCase()	지정한 문자열을 모두 대문자로 변경하여 반환한다.
substring(start,end)	지정한 문자열을 start 부터 end-1 까지 부분열을 반환한다.
substr(start, len)	지정한 문자열을 start 부터 개수가 len을 반환한다.
replace(패턴, sss)	지정한 문자열에서 일치하는 패턴에 해당하는 값을 sss로 변경하여 반환한다.
includes(str)	대상 문자열에 지정된 문자열 포함 여부 판별할 때 사용한다
startsWith(값), endsWith(값)	대상 문자열이 지정된 문자열로 시작(끝)하는지 판단한다.

4. Number 객체

수치 데이터를 처리하는 객체로서 기본데이터를 wrapping하는 객체이다.

문법1

```
let n = new Number(100);
```

문법2

```
let n = 100;
```

속성/메소드	설명
MIN_VALUE	자바스크립트에서 표현할 수 있는 최소값
MAX_VALUE	자바스크립트에서 표현할 수 있는 최대값
NaN	Not a Number
toFixed([자릿수])	값을 지정된 소수점 자릿수만큼 문자열로 리턴하는 메서드.
toString([진법])	Number객체를 문자열로 변경하여 리턴하는 메서드.
valueOf()	Number객체에서 기본형으로 값을 리턴하는 메서드.
Number.isNaN(값)	값이 NaN 인지 판별하는 메서드
Number.parseInt(값)	수치 데이터로 변경

날짜와 시간 데이터를 처리하는 객체이다.

문법

```
let  n = new Date([값]);
```

메서드	설명
getFullYear()	현재 년도를 반환한다.
getMonth()	현재 월을 반환한다. 반드시 +1를 해야 된다.
getDate()	현재 일을 반환한다.
getDay()	현재 요일에 해당하는 정수값을 반환한다. 0은 일요일,1은 월요일, 6을 토요일.
getHours()	현재 시간을 반환한다. 24시간 표현법
getMinutes()	현재 분을 반환한다.
getSeconds()	현재 초를 반환한다.
setXXX(값)	값을 설정하는 함수 (년,월,일,시간,분,초, 요일등)

6. Boolean 객체

불린 데이터가 아닌 값을 불린 데이터로 변경하는 객체이다.

문법

```
let  n = new Boolean(값);
```

메서드	설명
toString()	불린값을 문자열로 변경하여 반환한다. (“true”/”false”)
valueOf()	불린 객체값을 기본값으로 반환한다. (true/false)

배열 데이터를 관리하기 위한 객체로서 다양한 생성방법이 제공된다.

문법1

```
let  배열명 = new Array(값, 값2, 값3);
```

문법2

```
let  배열명 = [값, 값2, 값3];
```

각 배열의 요소를 접근하는 방법은 배열명[index] 형식을 사용하고
index는 0부터 시작, 배열의 크기는 배열명.length 이다.
배열은 객체로 처리되고 배열에 저장할 수 있는 데이터 타입은 제한이 없다.

메서드	설명
push(값,[값1,값2])	배열에 새로운 데이터 추가하는 메서드.
pop()	배열의 마지막 요소를 제거한다.
reverse()	배열요소 순서를 거꾸로 변환하여 반환한다.
indexOf(값)	지정한 값에 해당하는 index값을 반환한다.
slice(start, end)	배열로부터 지정한 start와 end 요소를 반환한다.
splice(index,delCount[,값1,값2])	배열에 값을 저장하거나 삭제하는 메서드이다. 지정한 index위치에 값을 저장한다. delCount는 삭제할 개수다.
sort(function(a,b){ return a-b;})	배열에 저장된 알파벳 또는 숫자를 정렬한다. function내에서 return a-b는 오름차순 정렬이고 return b-a는 내림차순 정렬이다.
join(separator)	배열과 지정된 separator를 join하여 반환한다.
Arrays.from(값)	지정된 값을 분해하여 생성한 새로운 배열을 반환한다.
Arrays.of(값,값2,..)	지정된 값들을 이용하여 생성한 새로운 배열을 반환한다.
fill(값, start, end)	index 범위의 값을 지정한 값으로 변경한다.
entries()	배열을 {key:value} 쌍의 형식으로 반환한다.

메서드	설명
arr.forEach(함수)	배열을 순회하는 함수로서 전달된 함수에 3가지 인자를 넘긴다. arr.forEach(function(value,idx,arr){})
arr.map(함수)	배열을 순회하면서 각 요소값을 가공하여 반환하는 함수. result = arr.map(function(v){return v*2;}
arr.filter(함수)	배열을 순회하면서 조건과 일치하는 값만 반환하는 함수. result = arr.filter(function(v){return 조건식;}
arr.every(함수)	배열의 모든 요소에 대하여 true인 경우에 every()함수는 true값을 반환한다. 빈 배열인 경우에는 true 반환. result = arr.every(function(x){return 조건식;})
arr.some(함수)	배열의 일부 요소에 대하여 true인 경우에 some()함수는 true값을 반환한다. 빈 배열인 경우에는 false 반환. result = arr.some(function(x){return 조건식;})

8. 브라우저 관련 객체 (Browser Object Model:BOM)

Window 객체

브라우저 창이 열릴 때마다 매번 만들어지는 개체로서 브라우저 창 안에 존재하는 모든 요소의 최상위 객체이다.

Navigator 객체

브라우저와 관련된 정보를 포함하는 객체이다.

Screen 객체

화면정보와 관련된 객체이다.

History 객체

사용자가 방문했던 URL 정보를 포함하는 객체이다.

Location 객체

현재 방문한 URL 정보를 포함하는 객체이다.

9. Window 객체

브라우저 창이 열릴 때마다 매번 생성되는 객체로서 브라우저창안에 존재하는 모든 요소의 최상위 객체이다.(전역객체, 글로벌 객체, 루트객체)

속성	설명
document	window에서 보여지는 현재 문서를 의미한다.
name	window의 이름을 반환한다.
history	window의 history객체를 의미한다.
location	window의 location객체를 의미한다.
navigator	window의 navigator객체를 의미한다.
screen	window의 screen객체를 의미한다.
opener	새롭게 생성된 window에서 이전 window의 참조값을 반환한다.
parent	현재 window의 부모 window 객체를 반환한다.
self	현재 window의 참조값을 반환한다.
top	가장 최상위 window의 참조값을 반환한다.

메서드	설명
alert()	경고창을 보여줄 때 사용한다.
open()	새로운 window창을 보여줄 때 사용한다.
close()	현재 window창을 닫는다.
moveTo()	명시된 위치로 현재 window를 움직인다.
print()	현재 window의 내용을 출력한다.
setInterval()	명시된 시간 후 반복적으로 특정 작업 수행.
setTimeout()	명시된 시간 후 한번 특정 작업 수행.
confirm()	다이얼로그 창을 보여줄 때 사용한다.
focus()	현재 window에 포커스를 지정할 때 사용한다.
blur()	현재 window의 포커스를 제거할 때 사용한다.

브라우저와 관련된 정보를 관리하는 객체이다.

속성	설명
appName	브라우저의 이름을 반환한다.
appVersion	브라우저의 버전을 반환한다.
cookieEnabled	브라우저의 쿠키 사용여부를 반환한다.
language	브라우저의 language를 반환한다.
onLine	브라우저의 online 여부를 반환한다.
product	브라우저의 engine명을 반환한다.
userAgent	서버에서 브라우저에 보내진 user-agent 헤더정보를 반환한다.

브라우저 화면(screen) 정보를 관리하는 객체이다.

속성	설명
availHeight	screen의 높이를 반환한다. (taskbar 제외)
availWidth	screen의 너비를 반환한다. (taskbar 제외).
height	screen의 높이를 반환한다. (taskbar 포함)
width	screen의 너비를 반환한다. (taskbar 포함).

사용자가 방문한 URL의 히스토리를 관리하는 객체이다.

속성/메서드	설명
length	히스토리 리스트에 저장된 URL의 개수.
back()	히스토리 리스트에서 이전 URL을 로드한다.
forward()	히스토리 리스트에서 다음 URL을 로드한다.
go(number|URL)	히스토리 리스트에서 명시된 위치의 URL을 로드한다.

현재 URL에 관한 정보를 관리하는 객체이다.

속성	설명
host	현재 URL의 포트번호,호스트명을 반환하거나 설정한다.
href	전체 URL 정보를 반환하거나 설정한다.
hostname	현재 URL의 호스트명을 반환하거나 설정한다.
origin	현재 URL의 포트번호,호스트명, 프로토콜을 반환한다.
port	현재 URL의 포트번호를 반환하거나 설정한다.
protocol	현재 URL의 프로토콜을 반환하거나 설정한다.

메서드	설명
assign(URL)	새로운 문서를 로드 한다.
reload([false|true])	현재 문서를 리로딩 한다. false인 경우에는 캐시로부터 리로딩되고 true인 경우에는 서버에서 리로딩한다.
replace(URL)	새로운 URL로 현재문서를 변경한다.

객체 리터럴 이용 (JSON 표현식)

```
let  객체명 ={
          프로퍼티명: 값,
          메서드명: function(){}
};

객체명.프로퍼티명
객체명.메서드명()
```

샘플

```
let person  = {
     name:"홍길동",
    "age": 20 ,
     setName:function(n){ this.name = n},
    "getName":function(){ return this.name;}
  };
```

중첩 표현식

```
let  객체명 =
        프로퍼티명: 값,
        프로퍼티명: {
                        프로퍼티명:값,
                        프로퍼티명:값2
                    },
        프로퍼티명:값
    };
```

샘플

```
let person  = {

     name:"홍길동",
    "age": 20 ,
     parent: {
                  name:"홍대감",
                  job:"사또"
                }
};

 //중첩 프로퍼티 접근
person.parent.name
person["parent"]["name"]
```

여러 문자열 혼합 가능

```
//1. 객체의 key값을  문자열과 문자열 혼합하여 사용 가능.
let mesg = { ['one'+'two']:100};
console.log( mesg,  mesg.onetwo);
```

```
▼Object {onetwo: 100} i  100
    onetwo: 100
  ▶__proto__: Object
```

변수 사용가능

```
//2. 객체의 key값을  문자열과 변수값 혼합하여 사용 가능.
let xyz = "sport";
let mesg2 = { [xyz] :'축구', [xyz+"01"]:'야구',[xyz+"02"]:'농구'};
console.log(mesg2);
```

```
▼Object {sport: "축구", sport01: "야구", sport02: "농구"}
    sport: "축구"
    sport01: "야구"
    sport02: "농구"
  ▶__proto__: Object
```

15. JSON.stringify(값) 과 JSON.parse(값)

JSON 객체와 문자열 간의 변경

```
var n=[10,20,30];  // ==> "[10,20,30]"
var m = JSON.stringify(n); //[10] =>"[10]"
var m2 = JSON.parse(m);   // "[10]"-->[10]

var n2 ={"name":"홍길동","age":20};
var k = JSON.stringify(n2);
var k2 = JSON.parse(k);
```

6장. 함수 (function)

1. 함수 (function)의 역할

1) 일반적인 호출 가능한 형식의 함수

가장 일반적으로 인식하는 기능처리 함수역할.

2) 값으로의 함수 (일급객체)

변수에 할당 가능.
다른 함수의 인자로 전달 가능.
다른 함수의 리턴값으로 사용 가능.

3) 다른 객체를 생성할 수 있는 함수

new 이용하여 객체를 생성.
‘생성자 함수 ' 라고 부른다.

2. 함수 [function] 정의 방법

1) 함수 선언식 (이름있는 함수)

문법

```
function  함수명( [매개변수1,매개변수2,..] ){

        문장;
        [return 리턴값;]
}
```

함수가 생성되는 시점은 프로그램의 파싱 단계에서 생성된다.
따라서 함수 정의 전에 호출이 가능하다.

2) 함수 표현식 (익명 함수)

문법

```
let  변수명 = function ( [매개변수1,매개변수2,..] ){

      문장;
      [return 리턴값;]
 }
```

변수명을 사용하여 함수를 호출한다.
함수가 생성되는 시점은 프로그램의 실행단계에서 생성된다.
따라서 함수 정의 전에 호출이 불가능하다.

3. 함수의 매개변수와 인자

함수에서 정의된 매개변수(parameter)와 호출하는 인자(argument) 갯수가 달라도 함수 호출이 가능하다.
기본적으로 자바스크립트는 함수 호출시에 파라미터의 갯수를 체크하지 않는다.
만약 전달되는 가변길이의 파라미터의 값을 출력하고자 할 때는 arguments 내장 변수를 사용하면 된다. (배열로 관리)

예>

```
function  one( x , y , z){
    for( var i = 0 ; i < arguments.length ; i++){
       console.log( arguments[i]);
    }
}

 one(10,20);
 one(10,20,30);
 one(10,20,30,40);
```

4. default 파라미터

전달된 값이 없거나 정의되지 않은 경우 매개 변수를 임의의 기본값으로 초기화 할 수 있다.

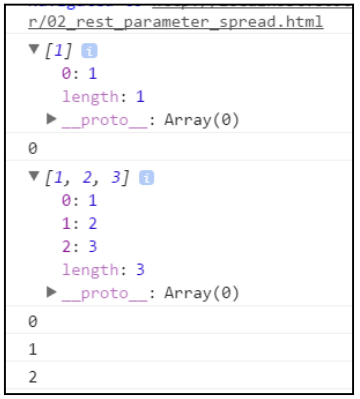
```
<script type="text/javascript">

        function aaa(a=1, b='홍길동'){
            console.log(a,b);
        }
        aaa();
        aaa(100);
        aaa(200,"유관순");
</script>
```

```
Navigated to http://localhost:8090/
r/01_default_parameter.html
1 "홍길동"
100 "홍길동"
200 "유관순"
```

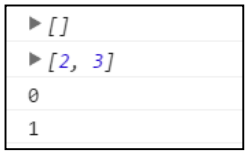
5. rest 파라미터

spread 연산자라고 하는 '…' 이용.
내부적으로 배열(Array)로 처리한다.
따라서 Array객체에서 제공하는 메서드를 사용할 수 있다.

```
<script type="text/javascript">

        function aaa(...xxx){
            console.log(xxx);
            for(let i in xxx){
                console.log(i);
            }
        }
        aaa(1);
        aaa(1,2,3);
</script>
```




```
  function bbb(n,...xxx){
    console.log(xxx);
    for(let i in xxx){
        console.log(i);
    }
  }
    bbb(1);
    bbb(1,2,3);
```



rest 파라미터는 함수의 매개 변수 목록에서 마지막에 있어야 된다.

6. 고급 함수 형태

1) 중첩 함수

특정 함수에서만 사용하는 기능이라면 외부에 노출시키지 않고
내부에 정의해서 사용 가능하다.

2) 콜백 함수

특정함수 호출시 트리거(trigger)형태로 자동으로 다른 함수를 호출하는
방식의 함수를 의미한다.

3) arrow 함수

일반적인 함수 표현식을 function 키워드 없이 => 이용하여 표현한 방법이다.

```
([param1, parma2,…param n] )=>statement;
```

4) generator 함수

function* 함수명(){} 형식이고 next() 함수를 호출하여 함수내 코드를
단계적으로 실행 시키는 함수를 의미한다.

특정 함수에서만 사용하는 기능이라면 외부에 노출시키지 않고
내부에 정의해서 사용할 수 있다.
이때 내부함수는 외부함수에서 호출해서 사용한다.

함수 선언식

```
// y 함수는 외부에서 접근 불가

    function x(arg){
        function y(n){
            return n*10;
        }
       return y(arg);
    }

// 외부에서 접근 방법
     var result = x(3);
     console.log(result);
```

함수 표현식

```
// b 함수는 외부에서 접근이 불가

     var a = function(arg){
         var b = function(n){
                 return n*100;
          }
         return b(arg);
     };

// 외부에서 접근 방법
     var result = a(2);
     console.log(result);
```

중첩함수 특징은 내부함수에서는 내부에 정의된 변수를 접근할 수 있다는 것이다. 외부에서는 접근할 수 없는 내부의 변수값을 내부함수가 접근할 수 있다는 것을 이용하면 객체지향의 은닉화(encapsulation) 기법을 적용할 수 있다.

은닉화

```
var k = function(){
          var d = 10;
          var k2 = function(){
              return  d * 2;
          }
         return k2();
      }

console.log("k() 호출 : " + k() ); //20
```

특정함수 호출시 트리거(trigger)형태로 자동으로 다른 함수를 호출하는 형태의 함수를 의미한다.
구현하는 방법은 호출함수에서 함수 호출시 콜백함수명을 같이 전달한다.

```
function call(info){
   info();
}

var callback = function(){
    console.log("callback");
}

call(callback)
```

콜백함수 구조는 이벤트 처리시 매우 많이 사용되는 구조이다.

9. 자동 호출 함수 (즉시 호출 함수)

프로그램이 실행될 때 자동으로 실행되는 함수로서
거의 대부분의 javascript 프레임워크에서 사용됨.

```
(function() {
   var msg = "Hello World"
   console.log(msg)
})()
```

arrow 함수는 일반적인 함수 표현식을 function키워드 없이 => 이용하여 표현한 방법이다.

```
([param1, parma2,…param n] )=>statement;
```

1) 파라미터 없고 리턴값 없는 함수 형태

```
//1. 함수 표현식 이용
 var a = function(){
     console.log("a");
  }
a();
```

```
//람다 표현식
var a2 = ()=>{
    console.log("a2");
};
a2();
```

2) 파라미터 있고 리턴값 없는 함수 형태

```
//1. 함수 표현식 이용
 var a = function(x){
     console.log("a" + x);
  }
a(10);
```

```
 //람다 표현식
 var a2 = (x)=>{
     console.log("a2"+x);
 };
 a2(10);
```

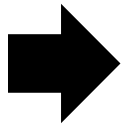
```
// 파라미터가 한개이면 () 생략 가능.
var a3 = x =>{
    console.log("a3"+x);
};
a3(10);
```

```
//2. 함수 표현식 이용
 var k = function(x,y){
     console.log("k" + x+"\t"+y);
  }
k(10,20);
```

```
// 파라미터가 여러개인 경우에는 () 생략불가
var k2 = (x,y)=>{
    console.log("k2" + x+"\t"+y);
}
k2(10,20);
```

3) 파라미터 있고 리턴값 있는 함수 형태

```
//1. 함수 표현식 이용
 var a = function(x){
      return x +10;
  }
console.log(a(10));
```



```
//람다 표현식
var a2 = (x)=>{
    return x +10;
};
console.log(a2(10));
```

```
var a3 = x=>{
    return x +10;
};
console.log(a3(10));
```

```
// return 및 {} 생략
var a4 = x=> x + 10;
console.log(a4(10));
```

* arrow 함수 사용시 주의할 점

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

arrow 함수는 객체의 this를 바인딩 하지 않는다. 따라서 객체의 메서드를 생성할 때는 사용하지 않도록 한다.

```
this.n=20;
// 익명 함수
 var a = {
      n:10,
      b: function(){
          console.log(this.n); //10
      }
 }
a.b();

 // arrow 함수
 var a2 = {
          n:10,
          b: ()=>{
              console.log(this.n); // 20
          }
      }
 a2.b();
```

11. generator 함수

generator 함수는 function* 이용하여 선언한 함수를 의미한다.
일반적으로 함수를 호출하면 함수가 실행되지만 generator 함수는 실행되지 않고 generator 객체를 생성하여 반환한다.

문법

```
function*  함수명(){}
```

샘플

```
<script type="text/javascript">

        function* a(){
            console.log("1");
        }
        var x = a();  // x가 generator 객체이다.
        console.log(typeof a);  // function
        console.log(typeof x);  // object

</script>
```

함수 블록이 실행 안됨.

generator 함수내의 코드를 실행하기 위하여 generator 객체의 next() 메서드를 호출해야 된다.

next()함수

```
<script type="text/javascript">

        function* a(){
            console.log("1");
            console.log("2");
            console.log("3");
        }
        var x = a();
        x.next();

</script>
```

```
Navigated to http://localhost:8
erator2.html
1
2
3
```

generator 함수 안에 yield 키워드를 작성하면 함수 블록내의 모든 코드를 실행하지않고 yield 단위로 나누어 실행 가능하다.
따라서 yield가 여러 개가 작성되어 있으면 yield 수만큼 next() 메서드를 호출해야 generator 함수 전체를 실행하게 된다.

yield

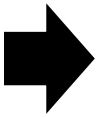
```
<script type="text/javascript">

        function* a(){
            console.log("1");
            yield console.log("2");
            console.log("3");
        }
        var x = a();
        x.next();
        x.next();

</script>
```

```
Navigated to http://localhos
erator3_yield.html
1
2
3
```

yield 키워드 다음에 설정한 표현식을 next() 메서드를 호출할 때 리턴 가능하며 리턴 형태는 { value:값 , done:불린값 } 이다.
yield가 수행되었으면 false 값 반환하고 수행되지 않았으면 true 값을 반환한다.

next()함수 반환값

```
<script type="text/javascript">

        function* a(k,k2){
            console.log("1");
            yield k+k2;
            yield '홍길동';
            console.log("end");
        }
        var x = a(10,20);
        console.log(x.next());
        console.log(x.next());

</script>
```

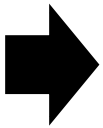


```
Navigated to http://localhost:8090/01ECMAS
erator3_yield2.html
1
▶ Object {value: 30, done: false}
▶ Object {value: "홍길동", done: false}
```

generator 함수내의 iterator를 종료하기 위하여 return() 함수를 사용한다.
반환값은 {value: undefined, done: true}

return()함수

```
<script type="text/javascript">

        function* a(k,k2){
            console.log("1");
            yield k+k2;
            yield '홍길동';
            console.log("end");
        }
        var x = a(10,20);
        console.log(x.next());
        console.log(x.return());
        console.log(x.next());

</script>
```



```
Navigated to http://localhost:8090/01ECMA
erator4_return.html
1
▶ Object {value: 30, done: false}
▶ Object {value: undefined, done: true}
▶ Object {value: undefined, done: true}
```

alert 함수

경고창을 화면에 출력할 때 사용한다.
예> alert(값)

setTimeout 함수

지정된 시간이 지나서 함수를 한번 실행한다.
예> setTimeout(function(){ 문장;}, delay);

setInterval 함수

지정된 시간이 지나서 반복적으로 함수를 실행한다.

```
예> var interval = setInterval(function(){ 문장;}, delay);
    clearInterval(interval);
```

7장. 이벤트 처리

자바스크립트 이벤트 모델

자바스크립트는 이벤트 드리븐 (event-driven) 모델에 기반한다.

웹 페이지안에서 발생한 여러 가지 사건(이벤트)에 따라 대응하는 방법을 (이벤트 핸들러) 이용한 처리이다.

이벤트에 대응해 처리를 담당하는 함수를 이벤트 핸들러라고 한다.

이벤트 발생 형태

가. 애플리케이션 사용자가 발생시키는 이벤트

예> 버튼클릭, 키보드 입력 등

나. 애플리케이션 스스로가 발생시키는 이벤트

예> 페이지 로드 등

2. 주요 이벤트

분류	이벤트명	발생 타이밍	주된 대상 요소
읽기	abort	이미지의 로딩이 중단되었을 때	img
	load	페이지, 이미지의 로딩 완료 시	body, img
	unload	다른 페이지로 이동할 때	body
마우스	click	마우스 클릭 시	-
	dblclick	더블클릭 시	-
	mousedown	마우스 버튼을 눌렀을 때	-
	mousemove	마우스 포인터가 이동했을 때	-
	mouseout	요소에서 마우스 포인터가 떨어졌을 때	-
	mouseover	요소에 마우스 포인터가 겹쳤을 때	-
	mouseup	마우스 버튼을 떼어 놓았을 때	-
	contextmenu	context menu가 표시되기 전	body
키	keydown	키를 눌렀을 때	-
	keypress	키를 누른 상태	-
	keyup	키를 떼어 놓았을 때	-
폼	change	내용이 변경되었을 때	input(text), select
	reset	리셋 버튼이 눌렸을 때	form
	submit	서브밋 버튼이 눌렸을 때	form
포커스	blur	요소로부터 포커스가 벗어났을 때	-
	focus	요소가 포커스되었을 때	-
그 외	resize	요소의 사이즈를 변경했을 때	-
	scroll	스크롤했을 때	body

3. 주요 이벤트 핸들러

이벤트 핸들러	기능 설명
onAbort	사용자의 작업을 빠져나오는 이벤트
onBlur	입력 폼 양식을 다른 곳으로 이동하는 이벤트
onChange	입력 폼 양식을 변경해 주는 이벤트
onClick	입력 폼 양식을 마우스로 클릭해 주는 이벤트
onDblClick	마우스를 더블 클릭해 주는 이벤트
onDragDrop	마우스를 드래그하여 끌어주는 이벤트
onError	이미지나 윈도우를 읽는 도중 에러가 발생할 때 수행하는 메소드
onKeyDown	사용자가 키를 눌렀을 때 발생하는 이벤트

3. 주요 이벤트 핸들러

이벤트 핸들러	기능 설명
onKeyPress	사용자가 키를 눌렀다가 놓았을 때 발생하는 이벤트
onKeyUp	사용자가 키를 눌렀다가 떼었을 때 발생하는 이벤트
onLoad	이미지나 문서 프레임 등을 로드 시키는 이벤트
onMouseDown	사용자가 마우스 버튼을 눌렀을 때 발생하는 이벤트
onMouseMove	마우스를 이동시키는 이벤트
onMouseOut	링크나 클라이언트 측에서 마우스를 옮기는 이벤트
onMouseOver	마우스를 링크나 클라이언트 측으로 옮기는 이벤트
onMouseUp	사용자가 마우스를 놓는 이벤트
onMove	윈도우나 프레임을 옮기는 이벤트
onReset	폼을 리셋 시키는 이벤트
onResize	윈도우나 프레임의 크기를 재조정하는 이벤트
onSelect	필드나 영역을 선택하는 이벤트
onSubmit	폼을 보내는 이벤트
onUnload	문서나 프레임 세트를 종료하는 이벤트
onFocus	문서나 윈도우, 폼 요소 등에 입력 포커스를 설정하는 이벤트

이벤트 모델의 핵심은 '이벤트' 와 '이벤트 핸들러' 이다.
HTML의 어떤 요소에서 발생하는 이벤트에 대해서 어떤 이벤트 핸들러를 사용할지를 정해서 구현한다.

1) DOM Level 0

인라인 이벤트 모델 방식과 고전 이벤트 모델 방식이 해당된다.
각 요소에 이벤트 타입별로 하나의 이벤트 핸들러만 지정할 수 있다.
고전 이벤트 모델은 이벤트 핸들러 해제가 가능하지만 인라인 이벤트 모델은 해제가 불가능하다.

2) DOM Level 2

표준 이벤트 모델 방식이다.
각 요소에 이벤트 타입별로 여러 개의 이벤트 핸들러를 지정할 수 있다.

DOM Level 0 에서 제공하는 이벤트 핸들러의 이름은 주요 이벤트명 앞에 on접두사를 지정하여 사용된다.
예> onload, onclick 등

인라인 이벤트 모델

```
<script>
function btn_onclick(){
   window.alert('안녕?');
}
</script>
<input type="button" value="다이얼로그 표시" onclick="btn_onclick()">
```

고전 이벤트 모델

```
<script>
window.onload = function() {
        document.getElementById('btn').onclick = function(){
                window.alert('안녕?');
        };
};
</script>
<input id="btn" type="button" value="다이얼로그 표시">
```

2000년 11월 W3C에서 이벤트 처리 표준안으로 발표.
함수의 파라미터로 이벤트 정보가 전달된다.

표준 이벤트 모델

다음과 같은 이벤트 연결/해제 메서드들을 모든 DOM 객체들이 지원한다.

- addEventListener(eventName, handler, useCapture) → 연결
- removeEventListener(eventName, handler) → 해제

* useCapture
- false : 기본값, 이벤트 버블링으로 이벤트 전파.
- true : 이벤트 캡쳐링으로 이벤트 전파.

7. DOM Level 2 이벤트 전파

이벤트가 처음 발생되면 DOM의 최상위 객체인 document 객체로 이벤트가 전달된다.

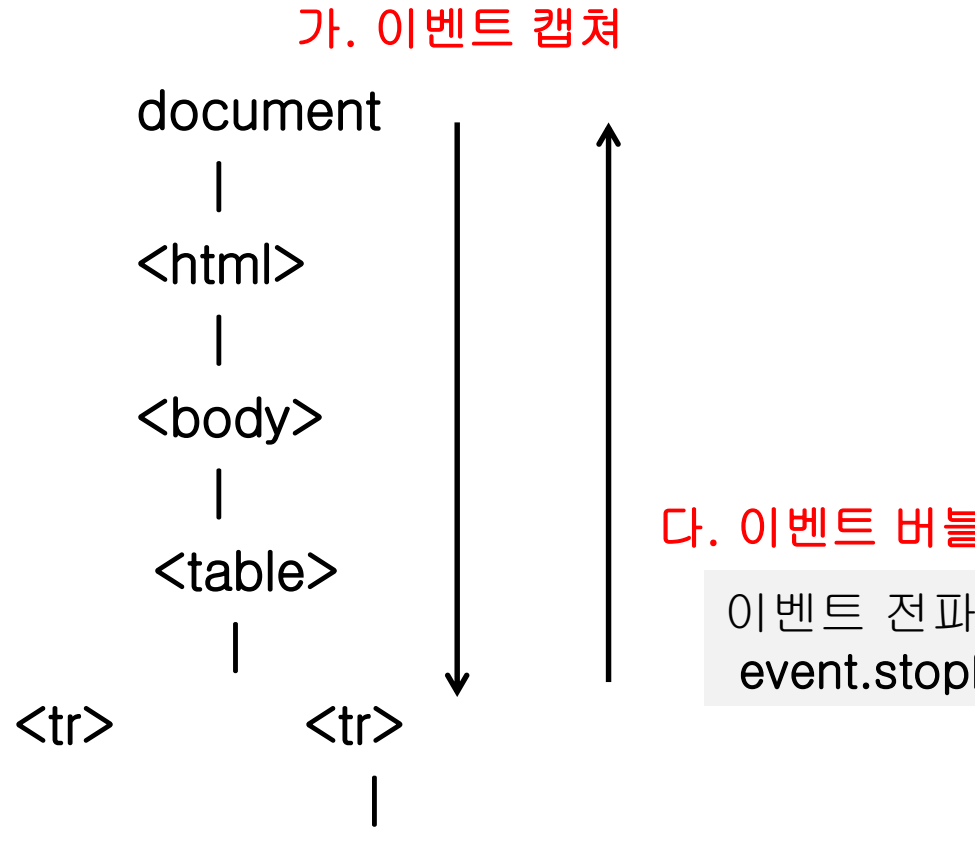
자바스크립트에서는 발생된 이벤트를 다음 과정을 거쳐 전파시킨다.

가. 이벤트 캡쳐(캡쳐링)
부모로부터 자식으로 전파.

나. 이벤트 버블 (버블링): JavaScript의 기본 처리 방식
자식부터 부모로 전파.

7. DOM Level 2 이벤트 전파

예>



이벤트 전파를 막아야 된다.
event.stopPropagation() 함수 사용.

8장. DOM

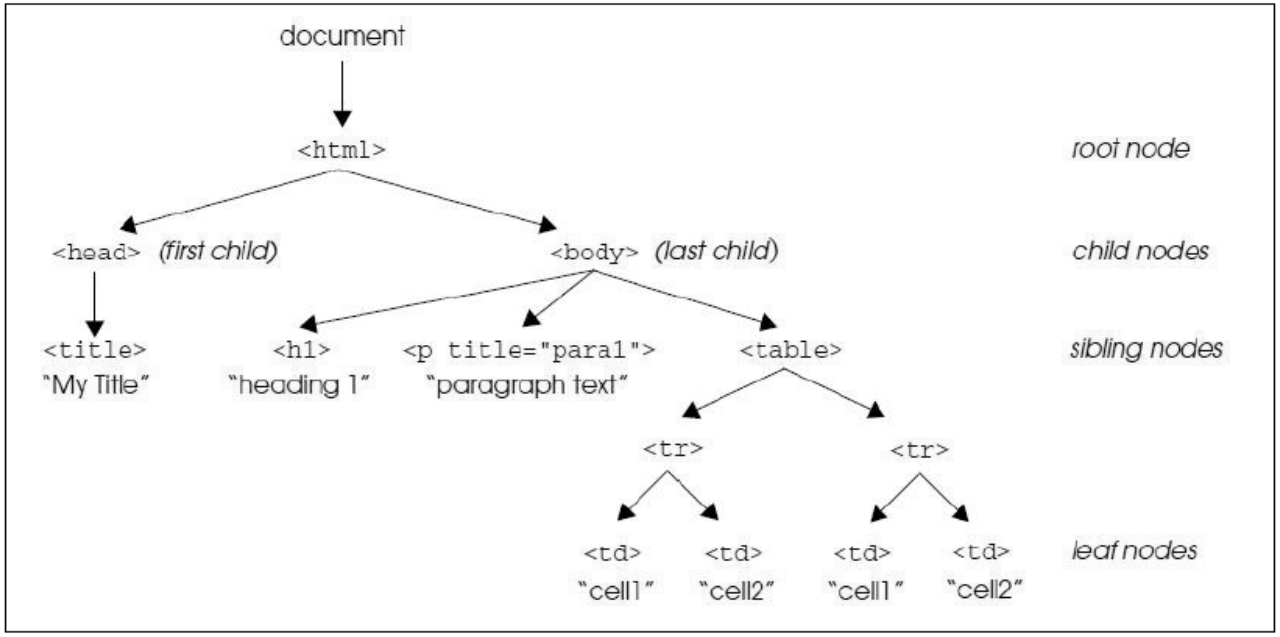
1. DOM (Document Object Model) 개요

웹 페이지의 HTML문서 구조를 객체로 표현하기 위한 계층 구조로서 문서를 나타내는 최상위 객체는 document 객체이다.

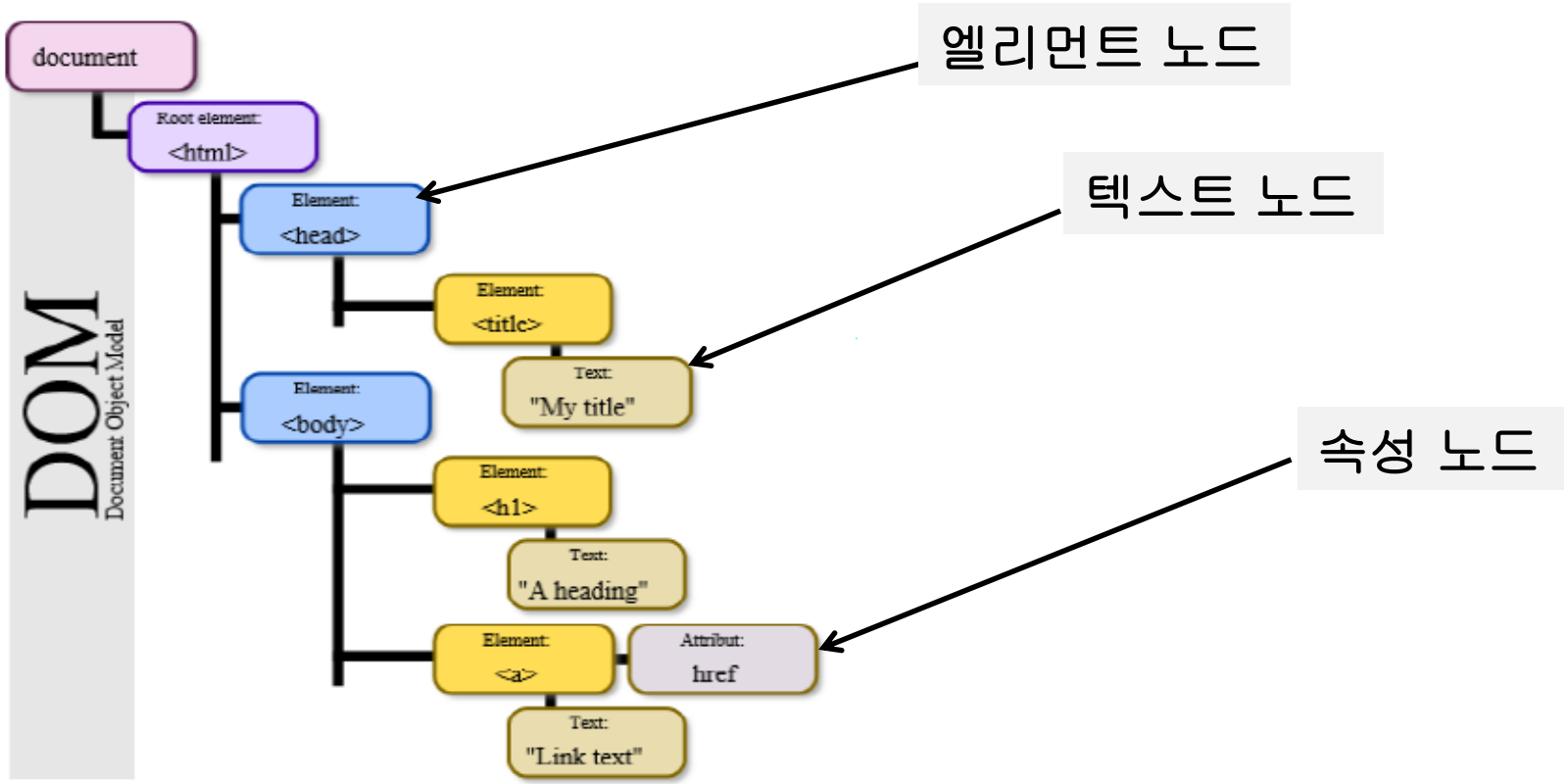
웹 페이지가 로드 될 때 웹 브라우저는 페이지의 DOM을 생성하고 트리 구조로 관리한다.

각각을 노드(node)라고 하며 엘리먼트 노드, 텍스트 노드, 속성 노드가 있다.

DOM은 플랫폼/언어 중립적으로 구조화된 문서를 표현하는 W3C의 공식 표준안 이다.



2. DOM 표현 방법


document
Root element:
<html>
Element:
<head>
Element:
<title>
Text:
"My title"
Element:
<body>
Element:
<h1>
Text:
"A heading"
Element:
<a>
Attribut:
href
Text:
"Link text"
DOM
Document Object Model
엘리먼트 노드
텍스트 노드
속성 노드

3. 노드(node) 속성 및 메서드

속성	기능 설명
firstChild	첫 번째 자식 노드의 요소를 반환
lastChild	마지막 자식 노드의 요소를 반환
nextSibling	현재 자식 노드에서 같은 레벨의 요소 다음 자식 노드를 반환
nodeName	노드의 이름을 반환
nodeType	1은 요소, 2는속성, 3은 내용으로 노드의 형을 반환
nodeValue	내용에 노드 값을 지정
ownerDocument	노드가 소속되어 있는 문서의 최상위 요소를 반환
parentNode	현재 노드가 소속되어 있는 요소를 반환
previousSibling	현재 자식 노드와 같은 레벨 요소의 이전 자식 노드를 반환

메소드	기능 설명
appendChild(새로운 노드)	자식 노드의 리스트 끝에 새 노드를 추가
cloneNode(옵션)	노드를 복사
hasChildNodes()	노드가 자식이면 true 발생
getAttribute(속성이름)	현재 노드의 속성 값을 반환
hasAtrributes()	노드에 속성이 정의되어 있는 경우 Boolean 값을 반환
hasChildNodes()	노드에 자식 노드가 정의되어 있는 경우 Boolean 값을 반환
insertBefore(새로운 노드, 현재 노드)	자식 목록에 새로운 노드를 삽입
removeChild(자식 노드)	자식 목록에 현재 노드를 삭제
setAttributeNode(속성 참조)	현재 노드에 속성을 생성하거나 지정

DOM 핵심작업

자바스크립트를 이용한 페이지의 모든 태그 변경 가능
자바스크립트를 이용한 페이지의 모든 속성 변경 가능
자바스크립트를 이용한 페이지의 모든 CSS 스타일 변경가능
자바스크립트를 이용한 페이지의 모든 이벤트 처리 가능. ..

DOM 핵심 API

순회 기능: childNodes, firstChild, lastChild, nextSibling, previousSibling
선택 기능: getElementById(), getElementsByTagName(),
getElementsByClassName(),querySelector(), querySelectorAll()
생성/추가/삭제:createElement(), createTextNode(), appendChild(),
removeChild(),createAttribute() , setAttribute(), getAttribute()
내용저장/조회: innerText, innerHTML

5. DOM 객체 접근 방법

1) 직접 접근 방법

id값으로 접근

document.getElementById(id값) : Node

tag명으로 접근

document.getElementsByTagName(tag명) : NodeList

class속성으로 접근

document.getElementsByClassName(class명) : NodeList

css selector 사용

document.querySelectorAll(selector) : NodeList

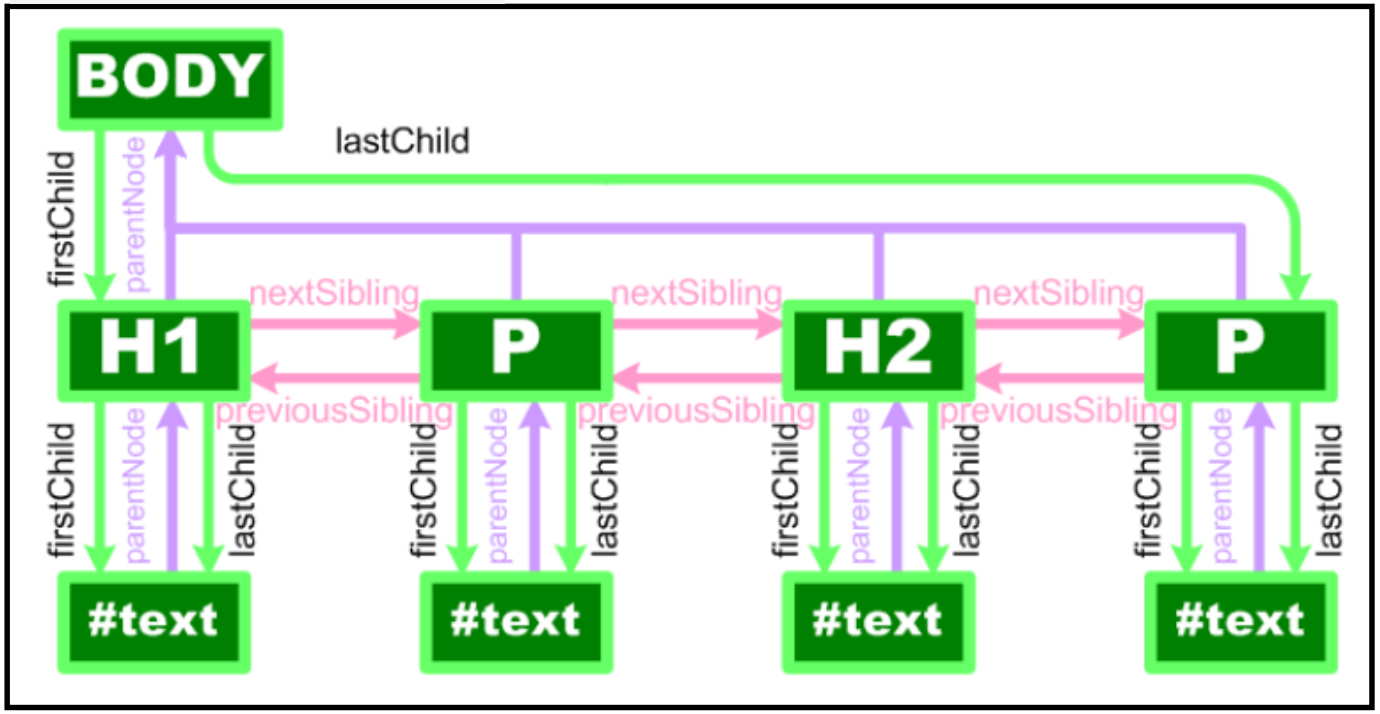
document.querySelector(selector) : Node

2) 노드워킹 접근 방법

상대적인 위치로 접근하는 방법이다.

예> 첫 번째 자식, 형제, 마지막 자식 등

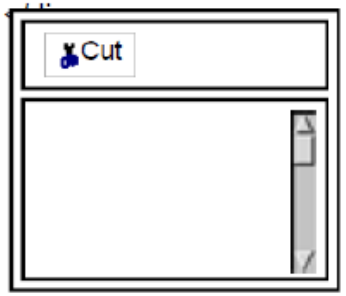
노드워킹 접근 방법


BODY
lastChild
firstChild
parentNode
nextSibling
nextSibling
nextSibling
H1
P
H2
P
previousSibling
previousSibling
previousSibling
firstChild
parentNode
lastChild
#text
#text
#text
#text

Declarative HTML vs. Procedural DOM

HTML

```
<div id="main">
  <div id="toolbar">
    <button>
      <img src="cut.png"></img>
      Cut
    </button>
  </div>
  <textarea id="editor"></textarea>
```



Document Object Model (DOM) in Javascript

```
var main = document.createElement("div");
main.setAttribute("id", "window");

  var toolbar = document.createElement("div");
  toolbar.setAttribute("id", "toolbar");

    var button = document.createElement("button");
      var img = document.createElement("img");
      img.setAttribute("src", "cut.png");
      button.appendChild(img);

      var label = document.createTextNode("Cut");
      button.appendChild(label);
    toolbar.appendChild(button);
window.appendChild(toolbar);

var editor = document.createElement("textarea");
editor.setAttribute("id", "editor");
window.appendChild(editor);
```

7. DOM이 모두 로딩 되기를 기다리기

DOM 처리작업 시 주의해야 될 점은 DOM이 완전히 로딩되기 전에 자바스크립트 코드가 실행 될 수 있다는 것이다.

head 태그에 포함된 스크립트와 외부 파일에서 불러온 자바스크립트는 HTML DOM이 실제로 생성되기 전에 실행된다.

따라서 head나 외부파일에서 실행된 모든 스크립트는 DOM에 접근할 수 없다.

해결 방법

가. 페이지가 완전히 로딩되기를 기다리기

```
window.onload = 함수;
```

나. body 태그 맨 끝에 스크립트 설정하기

```
<script>…</script>
</body>
```

9장. 객체분해할당과 template 리터럴

1. 배열 객체분해할당 (destructuring)

객체분해할당은 객체의 구조를 해체하는 것을 의미한다.
배열이나 JSON객체의 데이터를 해체하여 다른 변수로 추출할 수 있다.
이러한 이유 때문에 '분할 할당'이라고도 한다.

가. 배열 객체분해 할당

양쪽 모두 배열형식으로 지정해야 되고 인덱스를 기준으로 값을 할당한다.

```
// 1. 배열
let one,two;
[one,two] = [100,200];
console.log(one,two);
```

```
var [,,kkk] = [10,20,30]
console.log(kkk)
```

```
// default value
let a2,b2,c2;
[a2,b2,c2=999] = [100,200];
console.log(a2,b2,c2);  // 100 200 999
```

```
let a,b,c;
[a,b,c] = [100,200];
console.log(a,b,c);  // 100 200 undefined
```

```
[a, b, ...rest] = [10, 20, 30, 40, 50];
console.log(rest);
```

나. JSON 객체분해 할당

key값을 기준으로 값을 할당한다.
미리 선언된 변수를 사용하기 위해서는 반드시 () 안에 코드를 작성해야 된다.

```
//2. 객체
let a,b;
({a,b} = {a:100,b:200});
console.log(a,b);

let {a2,b2} = {a2:100,b2:200};
console.log(a2,b2);
```

```
let {a3,b3} = {a3:'100',xxx:'200'};
console.log(a3,b3);   // 100  undefined
```

```
//default value
let {a4,b4=999} = {a4:'100',xxx:'200'};
console.log(a4,b4);   // 100  999
```

다. 파라미터 객체분해 할당

호출 받는 함수의 파라미터를 객체 분해 할당 형태로 작성하면
함수에서 분해된 값을 사용할 수 있다.

```
//3. parameter destructuring
function a({x,y}){
    console.log( x, y);
}
a({x:100,y:200});


function a2([x,y]){
    console.log( x, y);
}
a2(['A','B']);
```

```
//default value
function a3([x,y,z='hello']){
    console.log( x, y, z);
}
a3(['A','B']);
```

4. template 리터럴

문자열 처리를 보다 더 효율적으로 처리하기 위한 방법.
`` (back-tick) 이라는 역따옴표를 사용하여 표현한다.
`` 안에 ${exp} 형식의 특별한 표현식을 삽입할 수 있다.

결국 문자열과 ${exp} 형식의 값을 ``으로 감싸 표현하는 리터럴 값을 'template 리터럴' 이라고 한다.

```
let mesg =`hello`;
console.log("1:" , mesg );
```

```
let a2 = 10;
let b2 = 20;
console.log("4:" , `합계:${a2+b2}` );
```

```
let a = '홍길동';
let b = `이순신`;
let result = `이름은 ${a}이고 별명은 ${b}이다`;
console.log("3:" , result);
```

```
1: hello
2: hello
world
3: 이름은 홍길동이고 별명은 이순신이다
4: 합계:30
```

10장. 클래스 (class)

가. 클래스 선언

```
//1. 클래스 선언식
 class Person{
      setName(name){
          this.name = name;
      }
      setAge(age){
          this.age = age;
      }
      getName(){
          return this.name;
      }
      getAge(){
          return this.age;
      }
 }
```

나. 객체생성

```
let p = new Person();
p.setName("홍길동");
p.setAge(20);
console.log(p.name , p.age);
console.log(p.getName(),p.getAge());
```

```
Navigated to http://1
홍길동 20
홍길동 20
```

set 및 get 키워드 추가

```
//1. 클래스 선언식
class Person{
    set setName(name){
        this.name = name;
    }
    set setAge(age){
        this.age = age;
    }

    get getName(){
        return this.name;
    }
    get getAge(){
        return this.age;
    }
}
```

```
let p = new Person();
//p.setName("홍길동");
//p.setAge(20);
p.setName="홍길동";
p.setAge=20;
console.log(p.name , p.age);
console.log(p.getName,p.getAge);
```

```
홍길동 20
홍길동 20
```

2. 생성자

클래스 인스턴스를 생성하고 생성한 인스턴스를 초기화 역할.
오버로딩 생성자 안됨. 반드시 constructor 는 단 하나만 지정 가능.
만약 명시적으로 constructor를 지정하지 않으면 default 생성자가 생성됨.

```
class Person{

    // 생성자 ( 클래스당 하나씩 )
    constructor(name,age){
        console.log("constructor(name,age)");
        this.name = name;
        this.age = age;
    }
    setName(name){
        this.name = name;
    }
    setAge(age){
        this.age = age;
    }

    getName(){
        return this.name;
    }
    getAge(){
        return this.age;
    }
}
```

```
let p = new Person();
console.log(p.name , p.age);
console.log(p.getName(),p.getAge());

let p2 = new Person("홍길동",20);
console.log(p2.name , p2.age);
console.log(p2.getName(),p2.getAge());
```

```
Navigated to http://loca
constructor(name,age)
undefined undefined
undefined undefined
constructor(name,age)
홍길동 20
홍길동 20
```

상속시 생성자 고려할 점

1.자식 클래스와 부모 클래스 양쪽 constructor를 작성하지 않아도 인스턴스 생성된다. (default constructor 사용)

2. 부모 클래스에만 constructor를 작성하면, 자식 클래스의 'default 생성자'가 호출되고 부모 클래스의 constructor가 호출된다.

3. 자식 클래스에만 constructor를 작성하면 자식 클래스의 constructor가 호출되고 반드시 부모 constructor를 명시적으로 호출해야 된다.

4. 자식과 부모 클래스 양쪽 constructor를 작성하면 자식 constructor가 호출되지만 반드시 부모 constructor를 명시적으로 호출해야 된다.

4. 메서드 오버라이딩 (method overriding)

```
class Employee{
    constructor(name,salary){
        this.name = name;
        this.salary = salary;
    }
    getEmployee(){
        return this.name+"\t"+this.salary;
    }
}

class Manager extends Employee{
     constructor(name,salary,depart){
        super(name,salary); // 반드시 명시적으로 호출
        this.depart = depart;
    }
     getEmployee(){
        return super.getEmployee()+"\t"+ this.depart;
    }
}
```

```
let man = new Manager("홍길동",2000,"관리");
console.log(man.getEmployee());
```

```
Navigated to http://lo
홍길동    2000    관리
```

5. static 메서드

```
class Employee{
    getEmployee(){
        return "hello";
    }
    static getXXX(){
        return "world";
    }
}
```

일반 메서드는 객체 생성후에 인스턴스 변수로 참조

static 메서드는 클래스명.메서드 형식으로 참조

```
console.log(Employee.getXXX());

let emp = new Employee();
console.log(Employee.getXXX(), emp.getEmployee());
```

```
world
world hello
```

11장. 모듈(module)

1. module 개요

기존 javascript에서는 <script src=“”> 형태로 파일 단위로만 재사용할 수 있었고 또한 하나의 javascript에서 다른 javascript를 사용할 수 없었다. ES6에서 module의 개념이 도입되면서 이런 부분이 개선되었다.

모듈 활용

module에서는 공유 대상을 제공하는 export와 활용하는 import가 사용된다.

```
export { name1, name2, …, nameN };
export { variable1 as name1, variable2 as name2, …, nameN };
export let name1, name2, …, nameN; // 또는 var
export let name1 = …, name2 = …, …, nameN; // 또는 var, const
```

```
export default expression;
export default function (…) { … } // 또는 class, function*
```

```
import { export } from "module-name";
import { export as alias } from "module-name";
import { export1 , export2 } from "module-name";
import { export1 , export2 as alias2 , [...] } from "module-name";
```

여러 항목 export

다양한 여러 항목을 공유하기 위해서는 export를 사용한다.

```
// 선언과 동시에 export
export let name = '홍길동';

export function add(a, b) {
    return a + b;
}

// 선언과 별개로 export
let addr = "seoul";
let age = 30;

export { addr, age };
```

위의 javascript는 name,add,addr,age 라는 4가지를 export 하고 있다.

단일 항목 export

한가지 항목만을 export할 때에는 export default를 사용한다.

```
export default {
    sayHello : {
        kor(name) {
            console.log(`안녕 ${name}`);
        },
        eng(name) {
            console.log(`Hello ${name}`);
        }
    }
}
```

export 한 내용을 사용하기 위해서는 import 명령을 사용한다.

```
import {
    name,
    add,
    addr as myaddr, // 다른 이름으로 변경 가능
    age
} from "./multi_export.js";

import greeting from "./default_export.js";

console.log(name, myaddr, age);
console.log(add(10, 20));
greeting.sayHello.kor("홍길동");

export {name, myaddr, age, greeting}
```

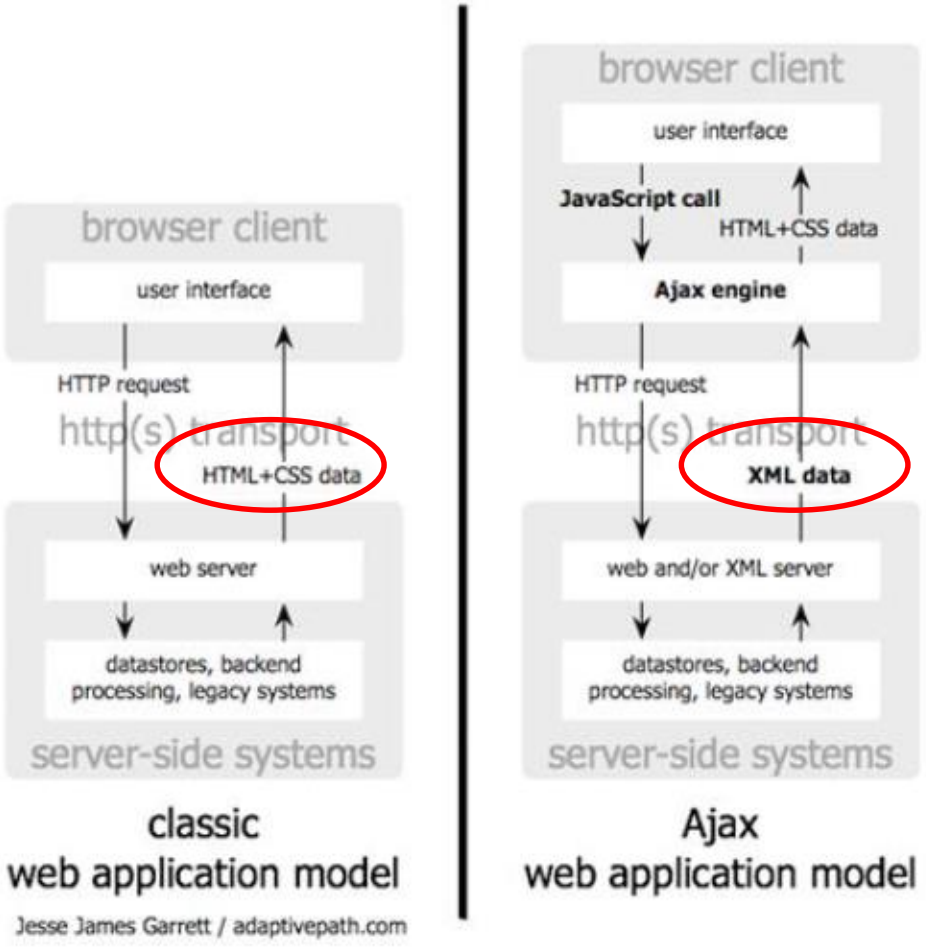
5. html에서 활용

export 한 내용을 html에서 사용하기 위해서는 <script> 태그에 type=module 속성을 추가해주어야 한다.

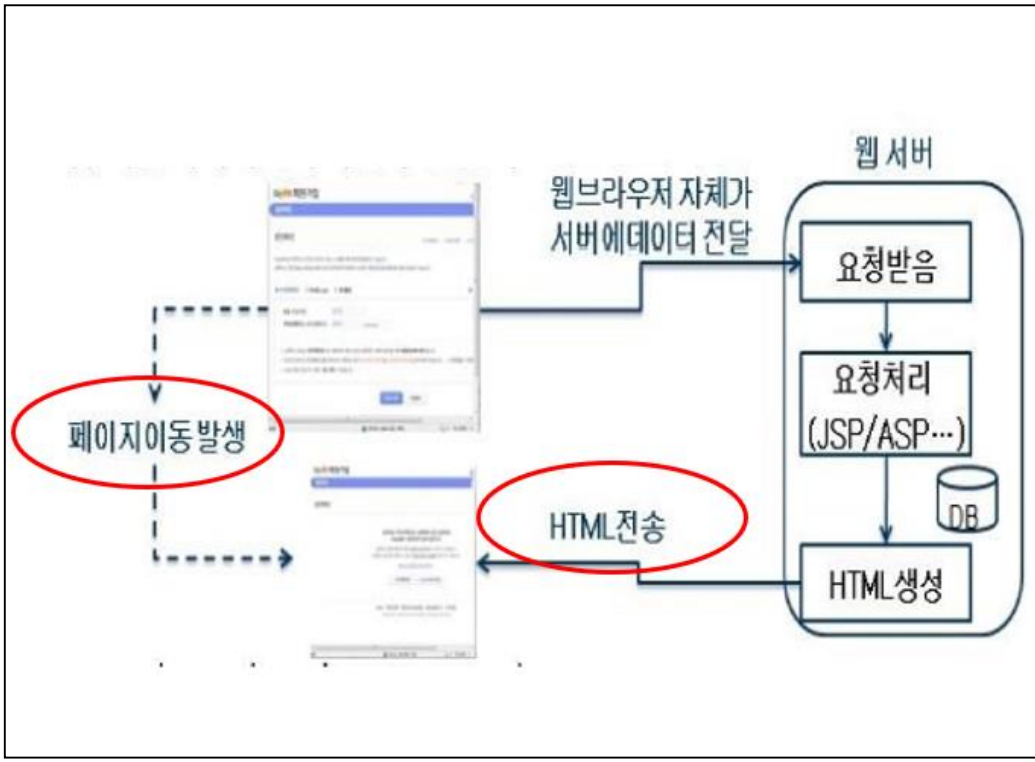
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Document</title>
  </head>

  <body></body>
  <script type="module">
    import { name, greeting } from "./import_test.js";
    console.log(name);
    greeting.sayHello.eng("hong");
  </script>
</html>
```

12장. Ajax

'Asynchronous Javascript and XML' 의 약자로서 비동기 방식으로 처리하는 새로운 웹 통신 방식을 의미한다.



2. 기존 방식의 웹 처리

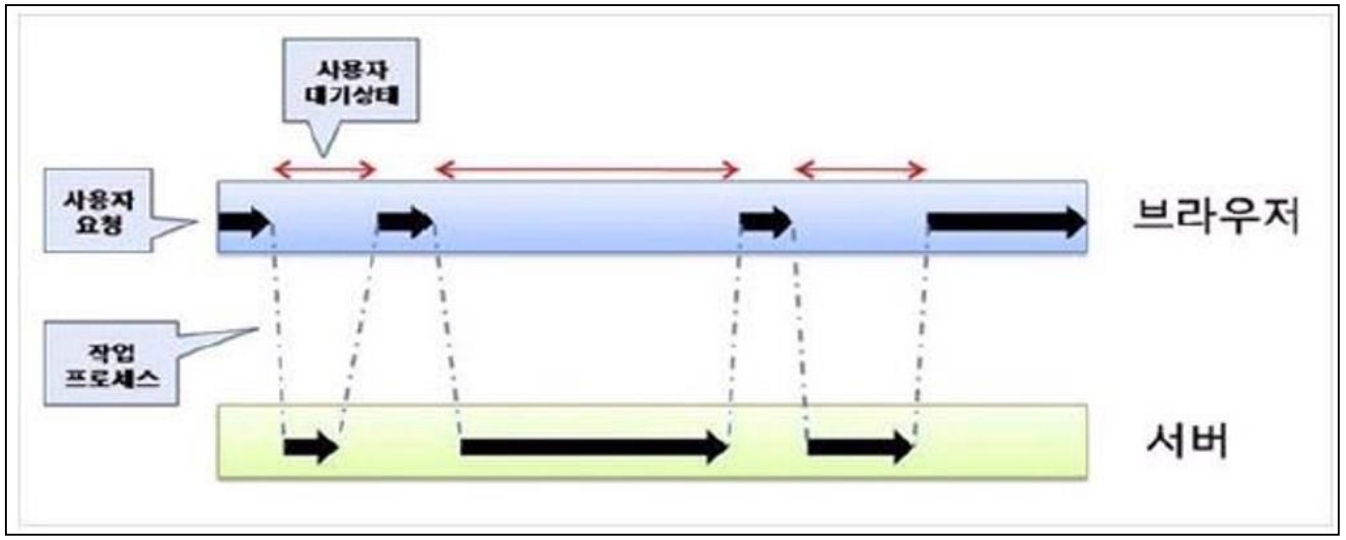


- 웹 브라우저가 웹 서버에 요청을 전송

- 웹 서버는 JSP 등의 서버 측 어플리케이션을 사용해 사용자의 요청을 처리한 뒤, 처리결과를 HTML로 생성해 웹 브라우저에 전송

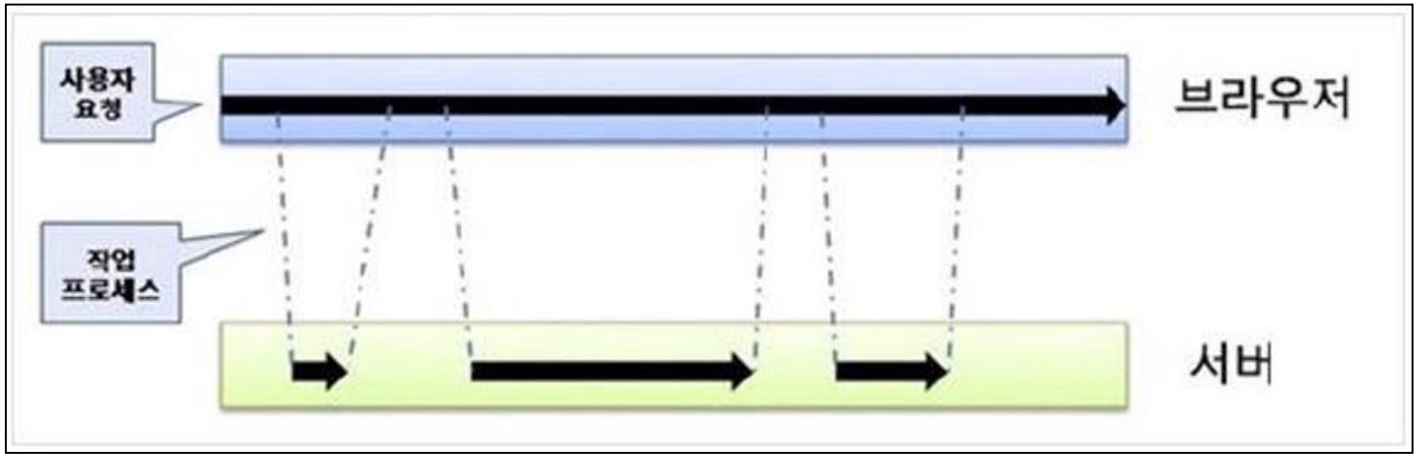
- 웹 브라우저는 응답으로 받은 HTML을 분석한 뒤 그 내용을 화면에 랜더링함

HTML 태그의 submit를 사용하여 서버에 값을 전달하게 된다.
이때 사용되는 페이지는 다음 그림과 같이 서버와 통신하고 response값이 넘어오기 전까지 동기상태인 무한 대기상태로 존재한다.

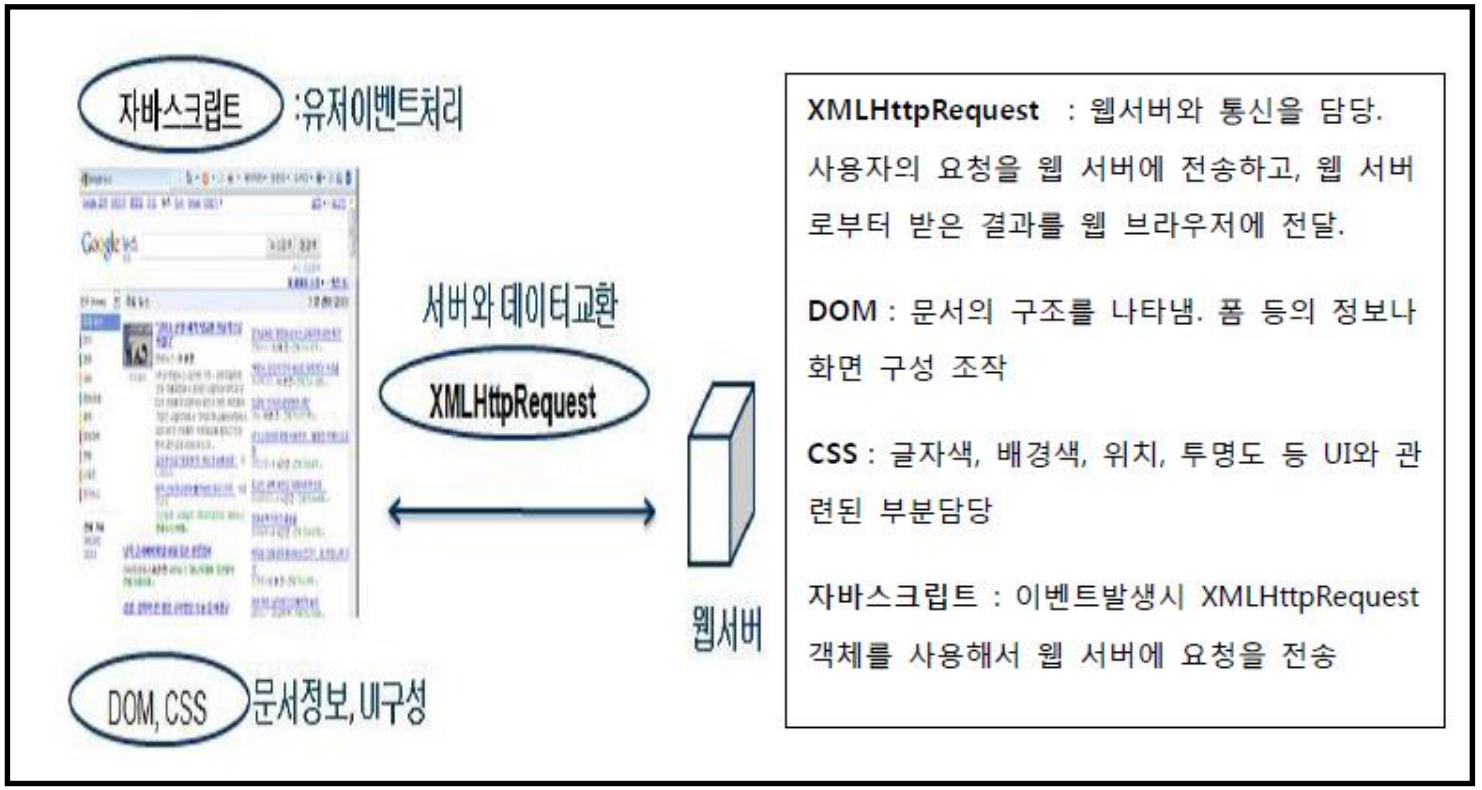


3. Ajax 처리 방식

Ajax를 이용하면 자바스크립트를 통해서 서버와 통신할 수 있으며 중요한 것은 서버와 통신하는 중에도 다른 작업을 수행할 수 있다는 것이다.
또한 Ajax를 사용하면 페이지 하나로 페이지 이동 없이 값을 보내고 받아올 수 있으며 서버와 통신하는 중에도 다른 일을 수행할 수 있다.
이것을 비동기 처리하고 한다.



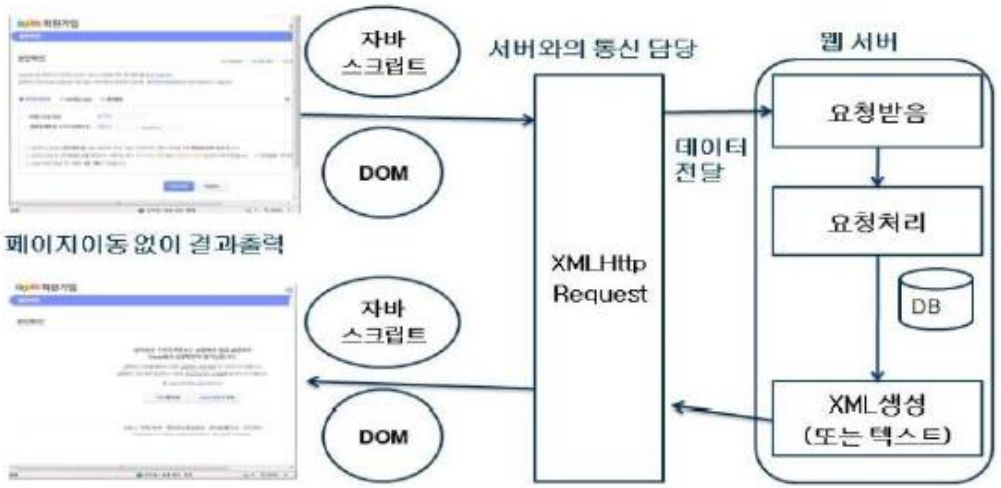
3. Ajax 처리 방식


자바스크립트 :유저이벤트처리
서버와 데이터교환
XMLHttpRequest
웹서버
DOM, CSS 문서정보, UI구성
XMLHttpRequest : 웹서버와 통신을 담당. 사용자의 요청을 웹 서버에 전송하고, 웹 서버로부터 받은 결과를 웹 브라우저에 전달.
DOM : 문서의 구조를 나타냄. 폼 등의 정보나 화면 구성 조작
CSS : 글자색, 배경색, 위치, 투명도 등 UI와 관련된 부분담당
자바스크립트 : 이벤트발생시 XMLHttpRequest 객체를 사용해서 웹 서버에 요청을 전송

3. Ajax 처리 방식

Ajax 기술을 구현하는 기본적인 순서는 다음과 같다.

- XMLHttpRequest 객체를 생성
- 콜백 함수(서버 응답 시의 처리 내용)를 등록
- 서버에 비동기 통신을 개시



4. XMLHttpRequest 객체

서버측과의 비동기 통신을 담당한다.
XMLHttpRequest 객체를 사용함으로써 지금까지 웹 브라우저가 실행해 온 서버와의 통신을 javascript가 담당할 수 있다.

```
var req = new XMLHttpRequest();
```

분류	멤버	개요
프로퍼티	onreadystatechange	통신 상태가 변화된 타이밍에 호출되는 이벤트 핸들러
	readyState	HTTP 통신 상태를 취득
	status	HTTP Status 코드를 취득
	statusText	HTTP Status의 상세 메시지를 취득
	responseText	응답 본체를 plaintext로 취득
	responseXML	응답 본체를 XML(XMLDocument 객체)로 취득

메서드	abort()	현재의 비동기 통신을 중단
	getAllResponseHeaders()	수신한 모든 HTTP 응답 헤더를 취득
	getResponseHeader(header)	지정한 HTTP 응답 헤더를 취득
	open(...)	HTTP 요청을 초기화
	setRequestHeader(header, value)	요청 시에 송신하는 헤더를 추가
	send(body)	HTTP 요청을 송신(인수 body는 요청 본체)

5. readyState 과 status 속성

readyState 값 및 status 값을 사용하여 통신 상태를 확인한다.

readyState 프로퍼티의 값

반환값	개요
0	미초기화(open 메서드가 호출되지 않음)
1	로드 중(open 메서드는 호출됐지만, send 메서드는 호출되지 않았다)
2	로드 완료(send 메서드는 호출됐지만, 응답 스테이터스/헤더는 미취득)
3	일부 응답을 취득(응답 스테이터스/헤더만 취득, 본체는 미취득)
4	모든 응답 데이터를 취득 완료

status 프로퍼티의 반환 값

반환값	개요
200	OK(처리 성공)
401	Unauthorized(인증이 필요)
403	Forbidden(액세스가 거부되었다)
404	Not Found(요청된 자원이 존재하지 않는다)
500	Internal Server Error(내부 서버 에러)
503	Service Unavailable(요구한 서버가 이용 불가)

가. GET 방식 기본 코드

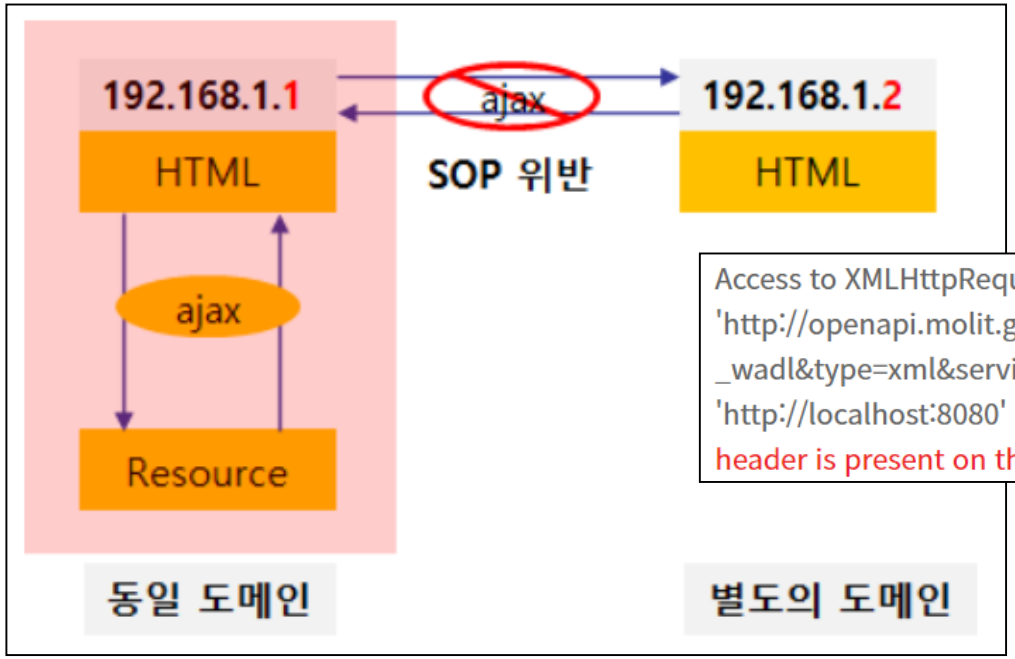
```
function formSubmit(){
    httpRequest = getXmlHttpRequest();
    httpRequest.onreadystatechange = callFunction;
    httpRequest.open("GET", "ajax3.jsp?name=hong" , true );
    httpRequest.send(null);
}
```

나. POST 방식 기본 코드

```
function formSubmit(){
    httpRequest = getXmlHttpRequest();
    httpRequest.onreadystatechange = callFunction;
    httpRequest.open( "POST" , "ajax3.jsp" , true );
    var  sendString = "name=hong";
    httpRequest.setRequestHeader("Content-Type",
                                 "application/x-www-form-urlencoded");
     httpRequest.send( sendString );
}
```

SOP(Same Origin Policy: 동일 근원 정책)

자바스크립트는 웹에서 심각한 보안상의 문제를 발생시키곤 한다.
그래서 다른 사이트의 자바스크립트에서 접근하지 못하도록 처리함.
즉 동일 근원(Same Origin)에서만 접근이 가능하게 하는 정책이다.

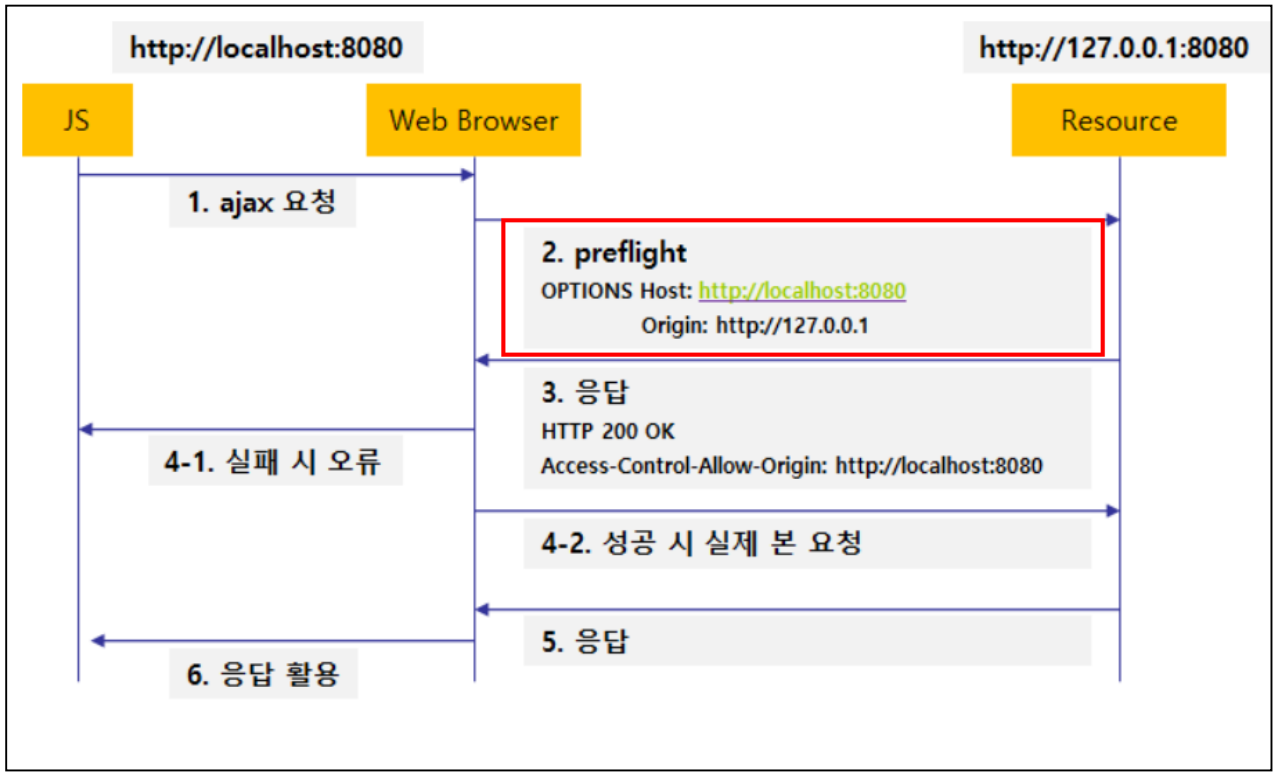


Access to XMLHttpRequest at 'http://openapi.molit.go.kr/OpenAPI_ToolInstallPackage/service/rest/RTMSOBJSvc/getRTM_wadl&type=xml&serviceKey=서비스키&LAWD_CD=11110&DEAL_YMD=201512' from origin 'http://localhost:8080' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

위에서 보듯이 동일 도메인에서 ajax 호출은 전혀 제한이 없다.
하지만 별도의 도메인에 ajax 요청을 하게 되면 SOP 위반에 의해 오류를 발생시킨다.

SOP 진행 과정

SOP를 처리하기 위해 브라우저는 요청 URL이 다른 도메인일 경우 본 요청 전에 preflight라는 사전 요청을 보내 호출 가능한 권한이 있는지 확인한다. 이를 위해 OPTIONS 메서드로 본 요청과 동일한 경로로 요청을 날려본다. 이과정에서 권한이 없으면 본 요청을 날리지 않고 오류를 발생시킨다.



처리 방법

서버로 날아온 preflight 요청을 처리하여 웹 브라우저에서 실제 요청을 날릴 수 있도록 해준다.
preflight 요청을 받기 위해 OPTIONS 메서드의 요청을 받아서 컨트롤해야 된다.

모든 요청의 응답에 아래 header 추가

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET,POST,PUT,DELETE,OPTIONS
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: Origin,Accept,X-Requested-With,Content-Type,Access-Control-Request-Method,Access-Control-Request-Headers,Authorization
```

Thank you