

객체지향 언어로 써의 JavaScript

- intro

조용준
(stgray22@gmail.com)

JavaScript is

❖ 저평가된 언어

- The World's most misunderstood Programming language

- ◆ Douglas Crockford(2001. <http://javascript.crockford.com/javascript.html>)

- JavaScript를 언어적으로 배울 필요가 있을까?

- ◆ 필요한 기능은 웹에서 검색 후 복사 → 사용 → 잘 돌아가면 끝
 - 거의 복-붙용 언어

- 객체지향이긴 한가?

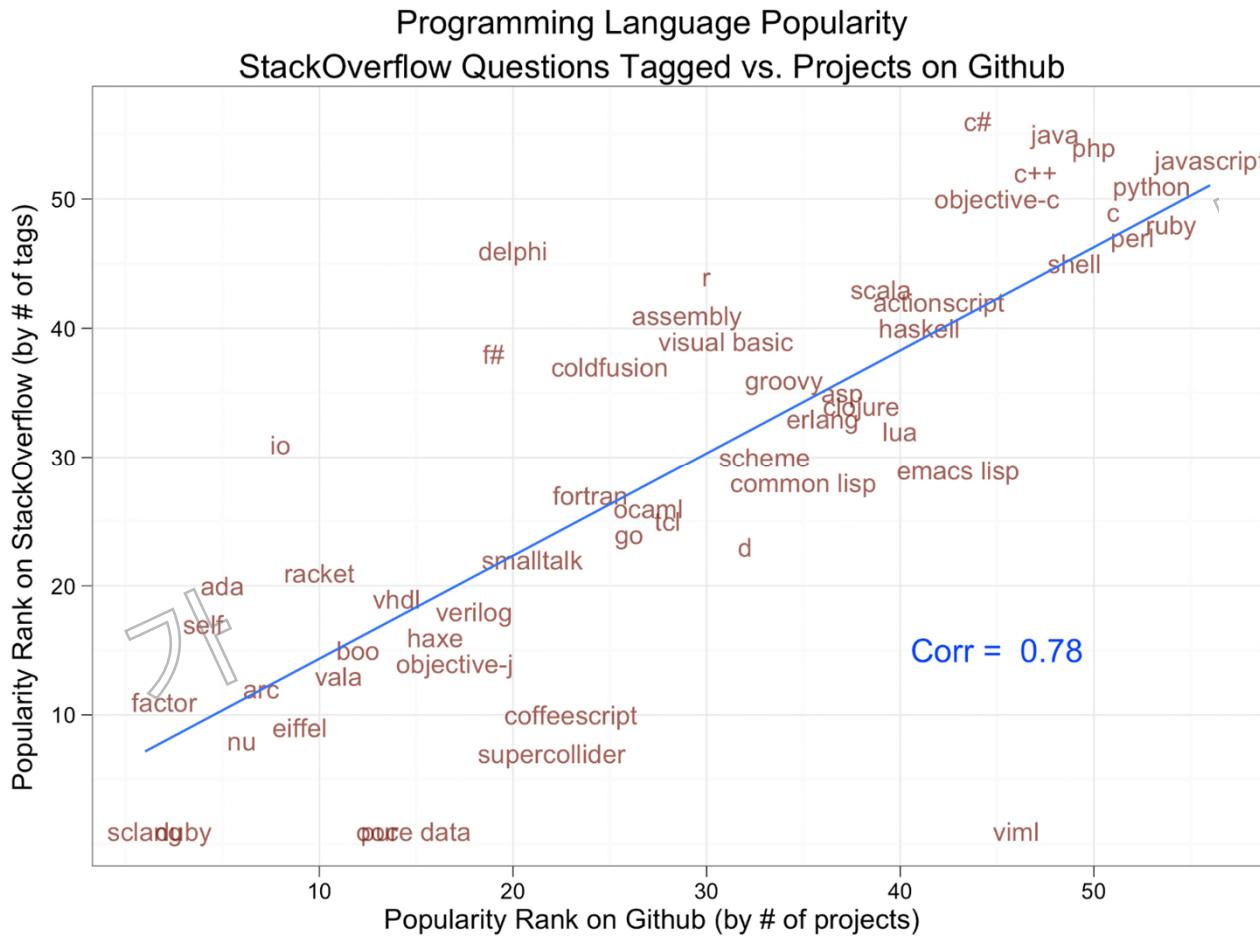
- ◆ 기능 위주의 사용으로 클래스를 사용할 필요가 없었음
 - HTML 요소에 접근을 위한 DOM 사용법 정도 알면 충분

- ◆ 기본적으로 객체지향의 필수인 클래스라는 것이 존재하지 않음
 - Prototype 기반으로 동작
 - ECMA 6 버전부터 class 등장

JavaScript is

❖ 달라지는 위상

● 가장 많이 사용되는 언어



JavaScript is

❖ JavaScript의 특징

● 인터프리터 언어

- ◆ 컴파일의 결과물이 있지 않고 실행하면서 한 줄씩 해석

● 스펙과 구현은 별개

- ◆ 스펙 주관은 ECMA이며 2015.6월 ECMA6 발표
- ◆ 스펙의 구현은 각 브라우저 벤더 → 브라우저마다 다르게 동작하는 문제 발생

● JavaScript is a OOP language

- ◆ 찾아보지 않았을 뿐 자바스크립트 역시 객체지향의 특성을 포함함
- ◆ boolean, number, string, null, undefined 외의 모든 것들을 객체로 표현
- ◆ 클래스 기반이 아닌 함수(function)과 프로토타입(prototype) 기반으로 동작

● 함수

- ◆ 일반적인 기능으로서의 함수 외에 생성자, 객체로써 동작
- ◆ 일급 객체로 다뤄지며 JavaScript를 이해하는데 가장 중요

● 프로토타입

- ◆ 상위 객체에 접근할 수 있는 링크로 상속을 이용한 객체 확장의 근간

JavaScript is

❖ 활용 범위

● 웹 클라이언트

- ◆ 전통적인 JavaScript 활용 분야
- ◆ HTML, CSS와 함께 웹 클라이언트를 위한 핵심 요소
- ◆ 과거 단순히 사용자를 위한 이벤트 처리가 목적이었으나
- ◆ HTML5에서부터 다양한 API의 처리를 위해 사용처가 증대됨



● 서버 개발

- ◆ Node.js의 출현으로 서버 개발 용도로 활용
- ◆ express, socket.io 등으로 손쉬운 개발 가능



● 애플리케이션 개발

- ◆ 크롬OS, 웹OS 등 브라우저 기반 OS에서 애플리케이션 개발을 위한 핵심 요소
- ◆ Cordova, PhoneGap 등 Hybrid 모바일 애플리케이션의 제어
- ◆ Atom 등 일반 PC용 애플리케이션 제어



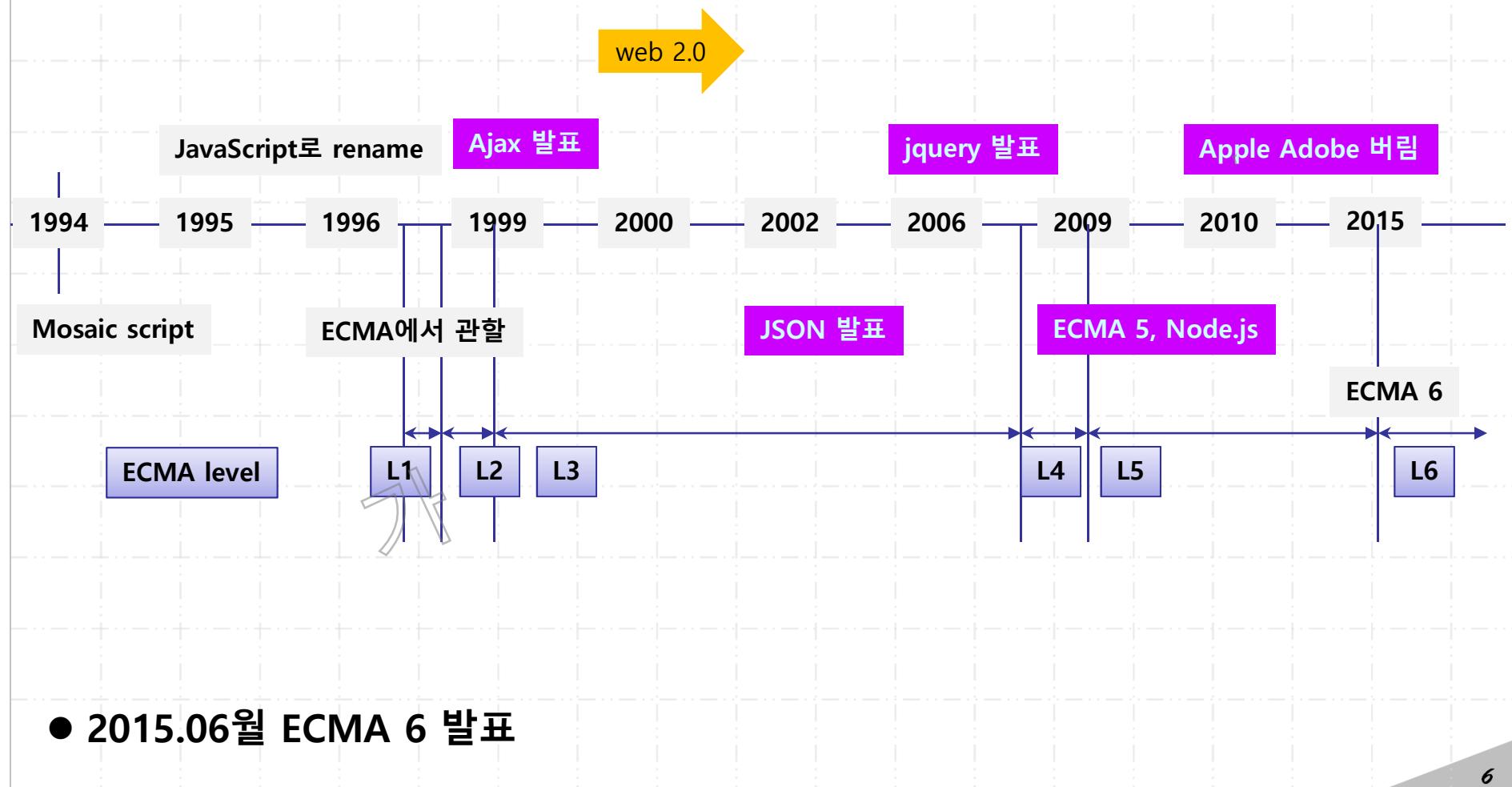
● 모든 범위에서 JavaScript의 활용 범위가 증가

5

JavaScript is

❖ JavaScript의 역사

● JavaScript의 공식 명칭은 ECMAScript



JavaScript is

❖ 참조 사이트

- w3schools.com : <http://www.w3schools.com/js/>

- ◆ Javascript tutorial 등 제공



초급

- Mozilla Developer Network : <https://developer.mozilla.org>

- ◆ spec에 대한 가장 상세한 설명, 예문



중,고급

- ecmascript.org : <http://www.ecmascript.org/>

- ◆ 공식적인 JavaScript의 고향

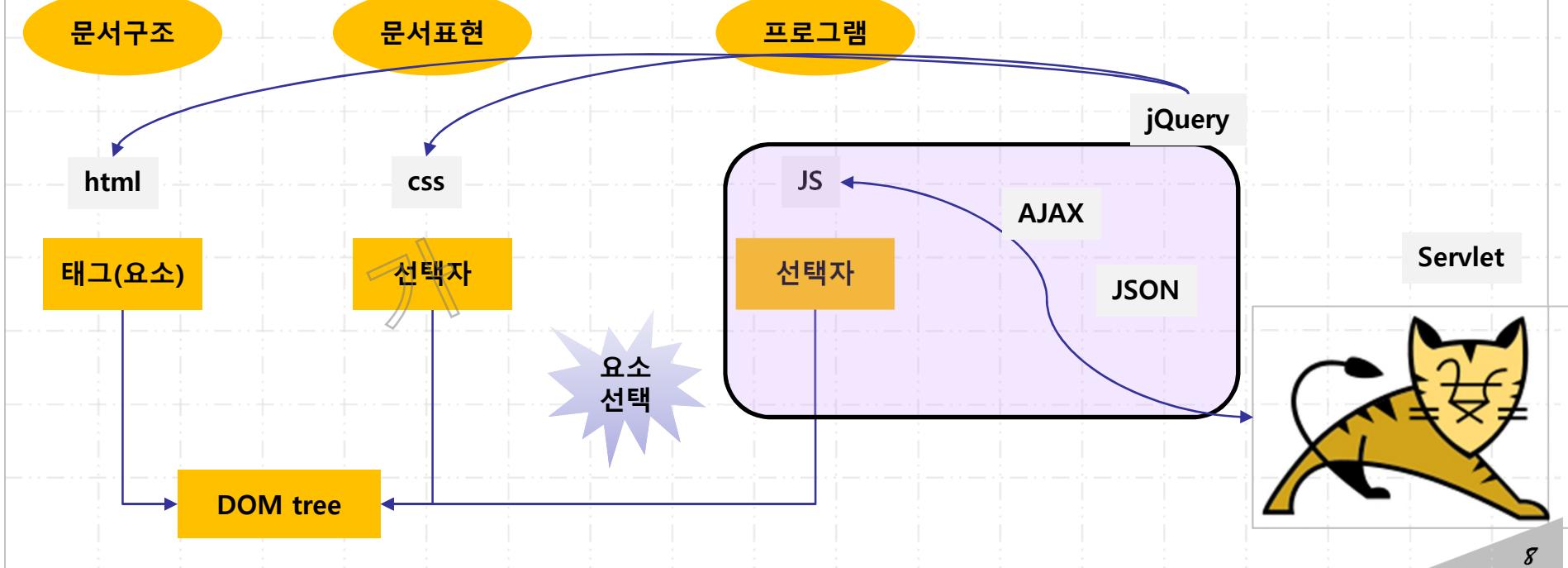
- ECMA6 주요 특징

- ◆ <https://github.com/lukehoban/es6features/blob/master/README.md>

JavaScript is

❖ 웹을 위한 JavaScript에 사용되는 기술들

- HTML: 기본 tag 사용법
- CSS : selector 사용법
- JavaScript : 기본 문법, 객체지향 개념, JavaScript 객체, 이벤트 처리
- 기타 : JSON, Servlet(for AJAX)



JavaScript 기본

- 기본 문법과 타입

가

조용준
(stgray22@gmail.com)

Hello JS

❖ JavaScript 사용 선언 3가지

1. Html 파일 내에 <script> 태그를 이용한 영역 표시

```
<script>
  console.log("Hello JavaScript!!");
</script>
```

◆ 선언 위치

- 과거에는 주로 <head>영역의 </title> 아래 위치
- Script 코드의 양이 많을 경우 로딩 속도 지연의 원인이 되므로 body 아래에 두기도 함

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript</title>
  <script>
    console.log("Hello JavaScript!! - title");
  </script>
</head>
<body>
  <h1>Hello</h1>
</body>
<script>
  console.log("Hello JavaScript!! - body");
</script>
</html>
```

가

DOM 로딩 시
점 주의!!

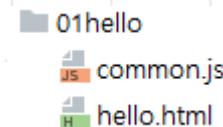
Hello JS

❖ JavaScript 사용 선언 3가지

2. 외부 파일 링크

- ◆ 공통되는 스크립트를 별도의 파일로 분리하여 여러 html 페이지에서 재사용 가능
- ◆ 서버에서 다운로드 되는 데이터의 양도 줄어들게 됨
- ◆ 확장자는 .js 로 지정
- ◆ 3rd party library 사용시 필수

```
</body>
<script src="common.js"></script>
</html>
```



◆ CDN (Content Delivery Network) 방식

- http 서비스를 이용해 Script 제공 서비스

◆ 외부 및 내부 선언 방법을 함께 많이 사용함

```
<script src='http://code.jquery.com/jquery.min.js'></script>

<script src="common.js"></script>

<script>
  console.log("Hello JavaScript!!!");
</script>
```

CDN 방식

Hello JS

❖ JavaScript 사용 선언 3가지

3. JavaScript 의사 프로토콜

- ◆ <a> 태그의 href 속성에 JavaScript:~ 형식으로 스크립트 삽입
- ◆ 작은 스크립트를 처리할 때 사용

```
<body>
  <h1>Hello</h1>
  <a href="JavaScript:sayHello()">sayHello</a>
</body>

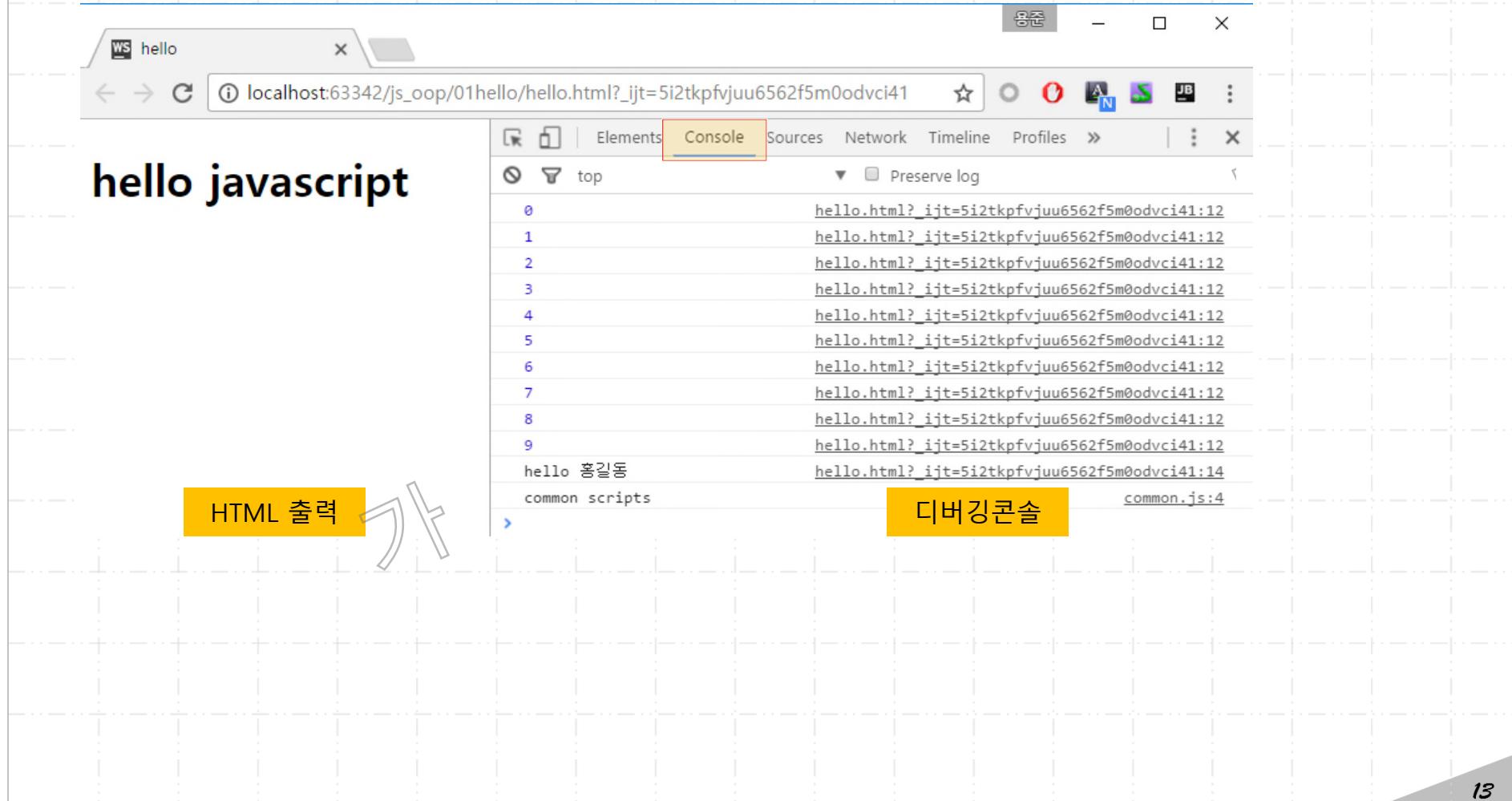
<script>
  function sayHello(){
    alert("Hello");
  }
</script>
```

가

Hello JS

❖ 개발자 도구의 사용

- chrome에서 실행 후 F12 를 누르면 개발자 도구 활성화



기본 특징

❖ 기본 특징

- html 과 달리 대소 문자를 구별한다.
- 일반적으로 실행문의 마지막에는 가급적 세미콜론(;)을 붙여준다.
 - ◆ 줄바꿈 만으로도 잘 동작하지만 대부분 언어처럼 관례상 ; 을 붙여 줌

```
<script>
    document.write("hi");
    document.write("bye");
</script>
```

```
<script>
    document.write("hi")
    document.write("bye")
</script>
```

- 큰따옴표("")와 작은따옴표('')가 겹치지 않도록 주의한다.

```
document.write("Hello 'java script'", "<br>"); : 0
document.write("Hello 'java script' "); : X
```

가

기본 특징

❖ 주석

- 다른 언어와 유사한 수준의 주석 사용

```
//single line comment  
  
/* multi  
line  
comment  
*/  
  
/* /* 중첩된 multi line comment*/ SyntaxError */
```

가

변수 선언

❖ 변수의 선언

● 변수 선언 지시어

◆ 생략

- 처음 값이 대입되는 시점에 묵시적으로 변수 선언
- 전역 변수로 취급되는 window의 멤버 변수로 취급되므로 가급적 사용하지 말 것
- 변수의 scope는 애플리케이션 전체
- strict mode에서 runtime error 발생

◆ var

- 가장 많이 사용되는 변수 선언 지시어로 변수의 scope는 선언된 함수 영역
- 변수의 중복 선언 가능

◆ let ES6

- 앞으로 많이 사용되기를 희망하는 지시어로 변수의 scope는 선언된 {} 영역
- 변수의 중복 선언 시 오류 발생

◆ const ES6

- 상수 선언으로 일반적으로 대문자로 작성
- 새로운 값이 대입되거나 재 선언될 수 없으며 변수 선언 시 반드시 값으로 초기화 되어야 함
- 변수의 scope는 let 변수와 동일

변수 선언

❖ 변수의 선언

```
function test() {  
    if (true) {  
        globalData = "globalData";  
        var varData = "varData";  
        let letData = "letData";  
        const constData = "constData";  
        //constData = 30;// const 변수 재할당 불가  
        console.log("g:" + globalData + ", v:" + varData + ", l+" + letData + ", c:" + constData);  
    }  
    console.log("g:" + globalData + ", v:" + varData);  
    //console.log("l+" + letData + ", c:" + constData);// let, const변수는 선언된 영역에서만 사용  
}  
  
test();  
console.log("g:" + globalData);  
//console.log("v:" + varData); // var 변수는 함수 내부에서 사용
```

가

변수 선언

❖ 변수의 선언

● 키워드와 함께 의미 있는 변수의 이름 지정

- ◆ `let age;`

- ◆ 변수 선언 시 타입을 명시하지 않음

- 정확한 변수의 타입은 런타임에 결정
- weakly typed language: 선언된 변수에 어떤 타입의 값이든 할당 가능

- ◆ 선언만 하고 초기화 하지 않을 경우는 `undefined` 초기화 → 정상 상태

- 선언되지 않은 변수 사용 시 Reference Error 예외 발생

● 선언과 함께 리터럴 할당 가능

- ◆ `let age = 100;`

- 리터럴: 변수에 할당되기 전의 순수한 값 자체

- ◆ 단 `const` 키워드가 사용된 경우 선언과 할당을 반드시 같이 처리

- blank final 형태는 사용 불가

● `let`과 `const`는 해당 scope에서 재 선언 불가

- ◆ `var` 키워드의 경우 재 선언 시 기존을 엎어 씀 → 주의

변수 선언

❖ 식별자

● 변수나 function 등에 사용하는 이름

- ◆ 예약어를 사용할 수 없음
- ◆ 숫자로 시작하면 안됨
- ◆ 특수문자는 _와 \$만 허용됨
- ◆ 공백 문자를 포함할 수 없음

JavaScript 예약어				
break	default	function	return	var
case	delete	if	switch	void
catch	do	in	this	while
const	else	instanceof	throw	with
continue	finally	let	try	
debugger	for	new	typeof	
class	enum	export	extends	import
super				
implements	interface	package	private	protected
public	static	yield		

● naming convention

- ◆ 생성자 함수의 이름은 항상 대문자로 시작
- ◆ 변수, 인스턴스, 함수의 이름은 항상 소문자로 시작
- ◆ 여러 단어로 이뤄진 식별자는 camel case 적용
 - isPowerOn
- ◆ 상수 작성 시 대문자에 under bar 활용
 - LAST_NAME

변수 타입

❖ 변수의 타입

- 기본형(Primitive Data Type) – 단순한 데이터 저장
- 참조형(Reference Data Type) – 객체에 대한 참조

● 다른 언어와의 차이점

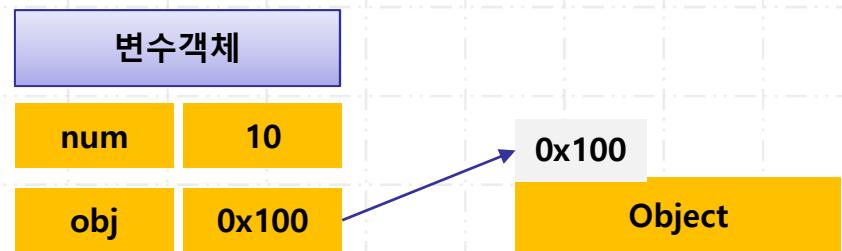
◆ 기본형을 참조형처럼 다룰 수 있음

- 기본형에 참조형의 내장 함수 적용 가능
- auto-boxing & unboxing

◆ 스택과 힙으로 메모리를 구별하지 않음

- 실행 컨텍스트 별 변수객체에서 변수를 관리하며(스택 역할)
- 기본형의 경우 변수 객체에 바로 값이 저장
- 참조형의 경우 메모리에 있는 실제 객체의 포인터 저장

```
<script>
  let num = 10;
  let obj = new Object();
</script>
```



20

변수 타입

❖ 기본형

타입	설명 및 예시	
number	정수 또는 부동소수점 실수 값	<code>let num = 100;</code> <code>let num2 = 3.14;</code>
string	작은 따옴표 또는 큰 따옴표로 감싼 문자	<code>let str = "Hello JavaScript";</code> <code>let str = 'Hi There';</code>
boolean	true와 false의 진리 값	<code>let bool = true;</code>
undefined	변수를 선언만 하고 아무것도 할당하지 않거나 undefined를 할당한 경우	<code>let str;</code>
null	변수에 null 값이 할당된 상태	<code>let str = null;</code>

가

변수 타입

❖ 참조형

- 위의 기본형을 제외하는 모든 데이터 타입
- 주로 사용되는 참조형의 리터럴

◆ 배열 리터럴

- []로 여러 자료를 묶어서 표현

// 배열 리터럴

```
let coffees = ['아메리카노', '카페라떼', '카페모카']; // length가 3인 배열
```

```
coffees = ['자바', , '브라질']; // length가 3이며 coffees[1] = undefined, not good
```

◆ 객체 리터럴

- 중괄호({ })로 묶인 0개 이상의 속성: 값 쌍의 모임

// 객체 리터럴

```
let person ={name:'andy', age:30};
```

```
console.log(person.name);
```

```
console.log(person['name']);
```

- 객체의 요소에 접근할 때 dot notation은 물론 배열 접근법도 사용 가능

변수 타입

❖ 기본형의 종류 확인

● `typeof`

- ◆ 변수의 종류를 알기 위해 사용하는 연산자
- ◆ 변수에 저장된 데이터의 타입을 문자열 형태로 리턴

```
let userName = "홍길동";
let age = 10;
let isOn = true;
let flag;
let money = null;

console.log(userName, typeof userName);
console.log(age, typeof age);
console.log(isOn, typeof isOn);
console.log(flag, typeof flag);
console.log(money, typeof money);
```

가

```
홍길동 : string
10 " : " number"
true " : " boolean"
undefined " : " undefined"
null " : " object"
```

변수 타입

❖ 숫자 타입 (number)

- 정수, 실수의 구별이 없으며 2진수(0b), 8진수(0o), 10진수, 16진수(0x) 사용 가능
- Infinity : JavaScript가 표현할 수 있는 한계를 넘어서는 값
 - ◆ Infinity는 어떤 값과 산술 연산을 해도 Infinity
- NaN : not a number
- isFinite(x)
 - ◆ x값이 표현 가능한 자릿수인지 리턴(NaN, Infinity, -Infinity에서 false, 나머지 true)
- isNaN(x)
 - ◆ 숫자 값이 NaN인지 확인해서 리턴

```
function check(x) {  
    console.log(x + " : " + eval(x) + ", " + (typeof eval(x)));  
}  
  
check("1/0 ");  
check("10*'F'");  
console.log(isFinite(10));  
console.log(isNaN(10 - 'a'));
```

◆ eval()

- 파라미터 문자열의 "" 를 제외하고 실행하는 함수 → 실무에서는 절대 사용 금지!!

변수 타입

❖ 숫자 타입 (number)

● Number 타입

- ◆ 객체로 기본형인 숫자를 객체화하기 위한 wrapper(참조형)
- ◆ 기본형, 참조형, 숫자 타입의 문자간 형 변환을 위한 함수 제공
- ◆ 하지만 굳이 Number로 변경하지 않아도 Number에 정의된 함수 사용 가능

```
function check(data) {  
    console.log(data , (typeof data));  
}  
  
let org = "10";  
check(org);  
  
let wrapper = new Number(org);  
check(wrapper);  
console.log(wrapper.toFixed(2));  
  
let toNum = wrapper.valueOf();  
check(toNum);  
console.log(toNum.toFixed(1));
```

string → Number

```
10 string  
Number {[[PrimitiveValue]]: 10} "object"  
10.00  
10 "number"  
10.0
```

◆ Number.parseInt(문자형_숫자 [, 진법])

```
console.log(Number.parseInt("123cm", 10));  
console.log(Number.parseInt("123a1", 10));  
console.log(Number.parseInt("q1231", 10));
```

25

조용준(stgray22@gmail.com)

변수 타입

❖ 문자열 (string)

- "" 또는 ''를 이용해서 표현 가능
- 숫자와 마찬가지로 wrapper로 String 제공

```
let str = "Hello JavaScript World";
console.log(str, typeof str);
console.log(str.toUpperCase());
```



```
let strObj = String(str);
console.log(strObj, typeof strObj);
console.log(strObj.toLowerCase());
```

```
Hello JavaScript World 06 string type casting.html:12
string
HELLO JAVASCRIPT WORLD 06 string type casting.html:13
06 string type casting.html:15
String {0: "H", 1: "e", 2: "l", 3: "l", 4: "o", 5: " ",
6: "J", 7: "a", 8: "v", 9: "a", 10: "S", 11: "c",
12: "r", 13: "i", 14: "p", 15: "t", 16: " ", 17: "W",
18: "o", 19: "r", 20: "L", 21: "d", length: 22,
[[PrimitiveValue]]: "Hello JavaScript World"}
"object"
hello javascript world 06 string type casting.html:16
```

- 문자열 조작을 위한 다양한 API 제공

- ◆ `substring()`
- ◆ `toLowerCase()`
- ◆ `toUpperCase()`
- ◆ `replace()`
- ◆ `concat()`

가

변수 타입

❖ boolean

- 리터럴로는 true | false 만 사용
- boolean 타입은 Boolean 타입과 호환됨
- 생성 시 false로 인식하는 경우: 0, null, ""(빈 문자열), false, undefined, NaN

- ◆ var data = Boolean("false"); → true
- ◆ var data = Boolean(); → false

```
let bool = true;  
console.log(bool, typeof bool);  
  
let boolObj = Boolean(bool);  
console.log(boolObj, typeof boolObj);
```

```
boolObj = Boolean("Hello");  
console.log(boolObj, typeof boolObj);  
  
boolObj = Boolean(false);  
console.log(boolObj, typeof boolObj);
```

```
true "boolean"  
Boolean {[[PrimitiveValue]]: true} "object"  
Boolean {[[PrimitiveValue]]: true} "object"  
Boolean {[[PrimitiveValue]]: false} "object"
```

변수 타입

❖ undefined

- 변수를 선언만하고 아직 값을 할당하지 않은 상태 – 정상
- 객체의 미 정의된 속성을 참조하려 할 때
- 함수에서 값이 반환되지 않은 경우
- 사용처에 따라 여러 형태로 변환됨

변환 타입

반환 값

boolean

false

숫자

NaN

문자열

"undefined"

```
let v1;  
let obj = {};  
  
if (v1) {  
    console.log("v1 is true");  
} else {  
    console.log("v1 is false");  
}
```

가

28

조용준(stgray22@gmail.com)

변수 타입

❖ null

- 객체가 없음을 명시적으로 할당되는 값

- undefined와 null

- ◆ 일반적인 값의 비교에서는 동일

- ◆ undefined의 타입은 undefined

- ◆ null의 타입은 object

```
let v1;  
console.log(v1);
```

```
let v2 = null;  
console.log(v2);
```

```
console.log(v1==v2);
```

```
console.log(v1===v2);
```

```
undefined  
null  
true  
false
```

- ◆ undefined는 보통 '변수의 초기화 상태'의 의미를 가지며 null은 'G.C 요청'

- 객체를 다 썼으니 Garbage Collection이 처리해도 좋음을 명시적으로 표시하는 역할로 사용

```
v1 = new Object();  
// v1을 이용한 프로그래밍  
v1 = null; // 다 썼으니 청소할 것.
```

변수 타입

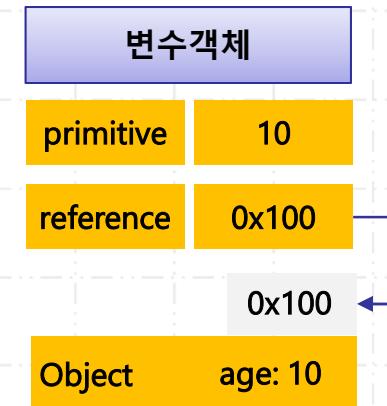
❖ 참조 타입

- 참조 타입: JavaScript가 객체지향임을 나타내는 증거
 - ◆ 객체 리터럴을 이용하거나 new 키워드로 생성자 함수를 호출 해 인스턴스 생성
- 참조 변수: 참조 타입의 메모리를 참조하는 변수
- 참조 타입의 종류 확인
 - ◆ instanceof 연산자를 이용해 실제 객체 타입 검토

```
let primitive = 10;
let reference = {};
console.log(typeof reference, reference instanceof Object);

primitive = 100;
reference.age = 10;
```

가



- JavaScript에서도 G.C가 존재하므로 특별히 객체를 삭제할 필요는 없음
 - ◆ 하지만 가급적 명시적으로 null을 할당하는 것을 권장

변수 타입

❖ 참조 타입

● 객체 리터럴을 이용한 객체 생성

- ◆ 객체를 저장할 때는 중괄호({ })를 이용

● 속성

- ◆ 속성(data) 표현 → **JSON 표현법**

- ◆ 속성의 값이 function 타입일 수 있으며 객체의 동작을 나타냄

- ◆ 즉 자바스크립트에서는 function(=함수 = 메서드)도 하나의 속성

● 속성에 접근하기 위해 . notation 적용

- ◆ 참조_타입_변수.속성이름

- ◆ 참조_타입_변수['속성이름'] 처럼 배열 접근 방식 사용 가능

● 속성의 추가와 제거

- ◆ 일반 언어의 객체와 달리 동적으로 속성 추가/수정/삭제 가능

```
let obj = {};  
  
obj.customProperty = "추가";           //속성 추가, 수정  
delete obj.customProperty;           //속성 삭제
```

변수 타입

❖ JSON?

- **JavaScript object notation :**

- ◆ JavaScript에서 객체를 표현하는 기법
- ◆ <http://www.JSON.org/>

- **주요 특징**

- ◆ 데이터 크기가 작음

- XML에서 사용되던 태그가 없이 간단한 기호만 사용
 - 전송해야 할 데이터의 양도 줄어들고 파싱을 위한 부담도 감소함

- ◆ 쉽게 읽고 쓸 수 있음

- 작성 문법이 단순하고 간단
 - 순수 텍스트로 구성되므로 쉽게 쓰고 이해할 수 있음

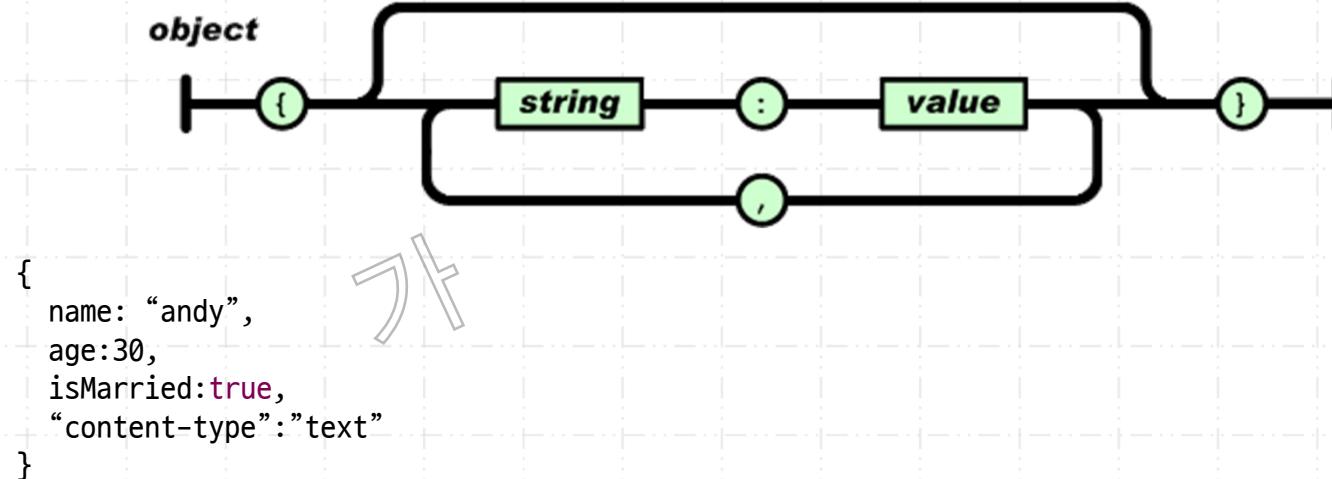
- ◆ 언어, OS에 독립적

- XML처럼 개발 시스템, 사용 언어에 독립적으로 동작
 - 이-기종 시스템 간 최적의 데이터 교환 방식

변수 타입

❖ JSON object

- 객체를 표현할 때는 전체적으로 중괄호를 이용해서 묶음
- name과 value의 쌍으로 이뤄진 데이터이며 쿠론(:)으로 구분
 - ◆ name은 문자열이며 데이터의 순서는 무관
 - name에는 ""를 생략할 수 있으나 식별자로 사용할 수 없는 문자가 있을 경우는 필수
 - { "content-type": "text" }
 - ◆ name/value의 쌍은 콤마(,)로 구분한다.

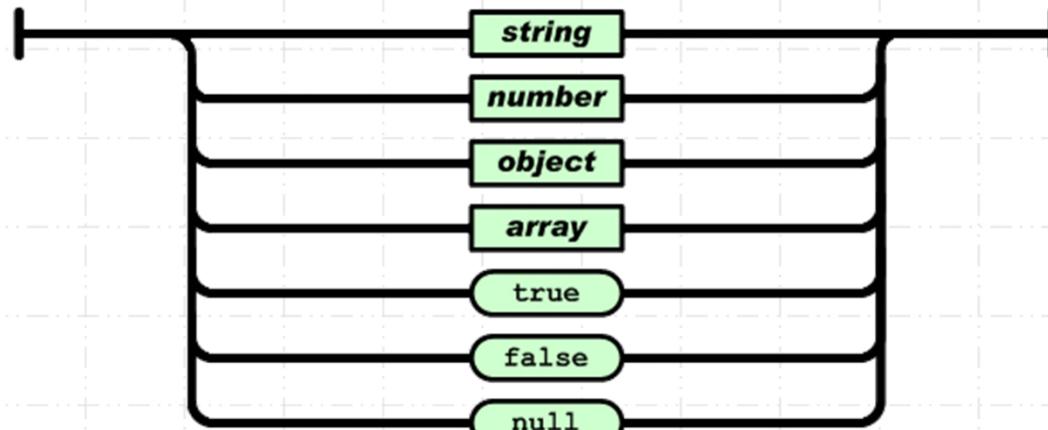


변수 타입

❖ JSON value

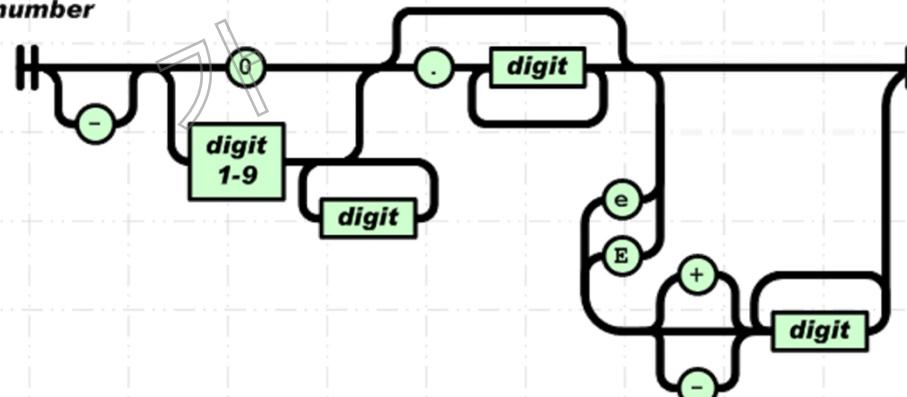
- 문자열만 쌍따옴표(" ")로 묶어 주며 나머지는 그대로 표현

value



- 숫자는 8진수와 16진수를 사용하지 않는다.

number

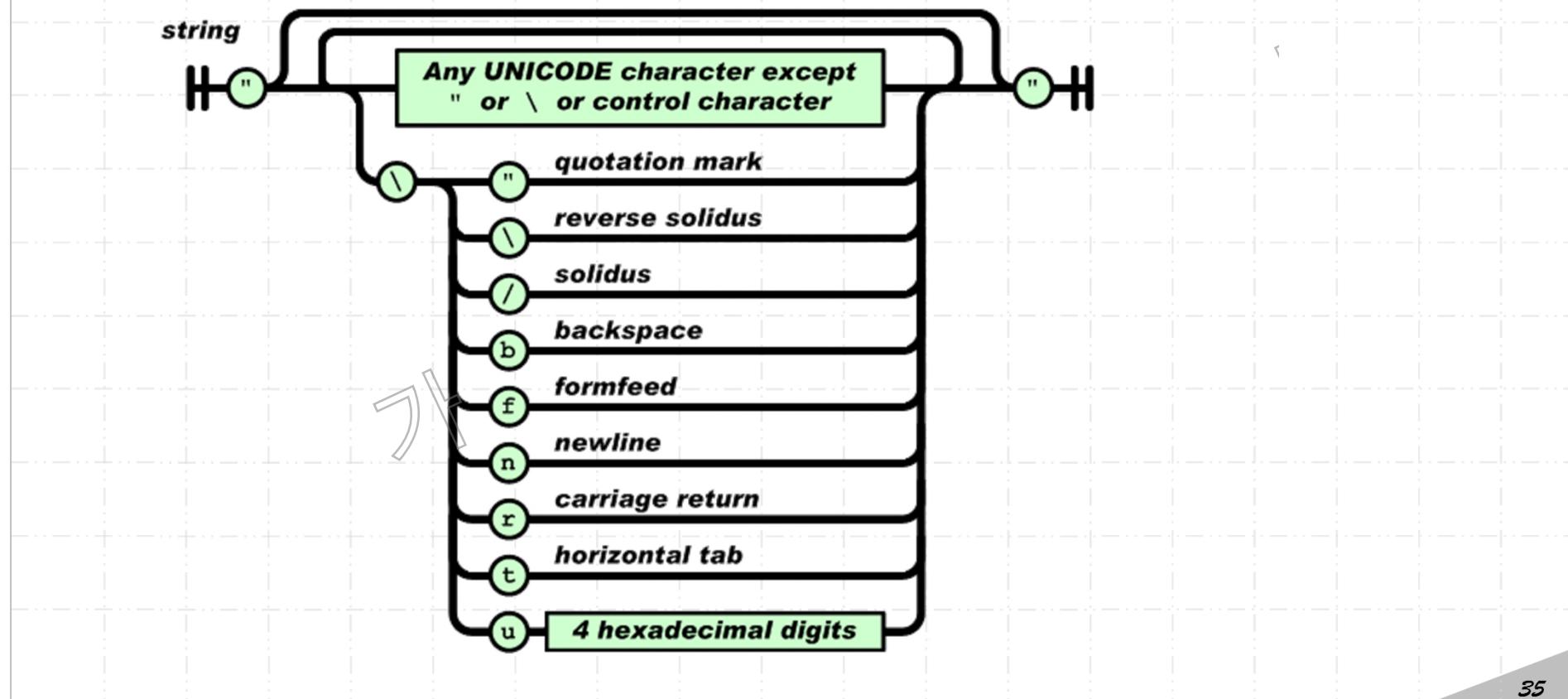


34

변수 타입

❖ JSON value

- string은 쌍 따옴표로 묶어서 표현
 - ◆ 0개 이상의 모든 Unicode 표현 가능
 - ◆ " 나 \ 는 자체 표현(escape sequence)을 위해 사용 금지

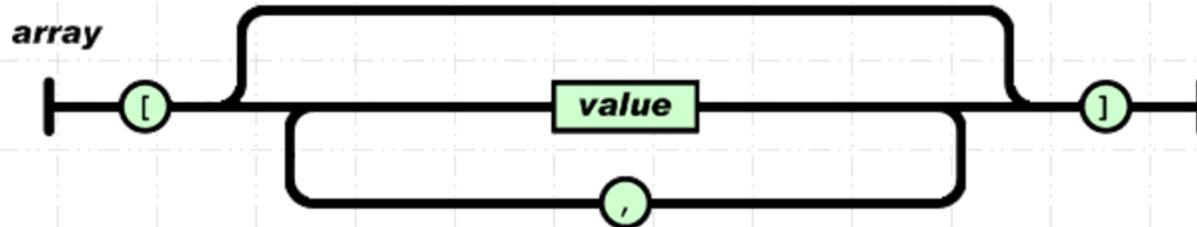


35

변수 타입

❖ JSON Array

- 배열 자체는 대괄호로 묶어주며 콤마(,)를 이용해서 value들을 구별



- ◆ 두 개의 object를 가진 배열을 만들고 members라는 키와 매핑 시켜 JSON 객체를 생성

```
let members = {  
    "members" : [ {  
        "name" : "andy",  
        "age" : 30,  
        "married" : true  
    }, {  
        "name" : "kate",  
        "age" : 25,  
        "married" : false  
    } ]  
}
```

변수 타입

❖ 참조 타입과 this

- this는 현재 객체를 참조하는데 사용되는 키워드

```
let person = {  
    name: "andy",  
    age: 20,  
    info: function(){  
        return this.name+" : "+this.age;  
    }  
};  
  
console.log(person.info());
```

가

JavaScript 기본

- 연산자

가

조용준
(stgray22@gmail.com)

연산자

❖ 산술연산

- 숫자 값을 피 연산자로 갖고 결과로 하나의 숫자를 반환
 - ◆ +, -, *, /, %
- 나눗셈 연산의 결과는 부동 소수점
 - ◆ 정수끼리의 연산도 실수 반환
- 어떤 타입과도 연산이 가능
 - ◆ 피 연산자들을 상황에 따라 형 변환 후 연산 진행

구분	문자열	boolean	참조형
+ 결합 연산자로 작용			toString() 결과와 결합 연산
-, *, /, % 숫자형 문자는 숫자로 변환		true → 1, false → 0로 변환	valueOf() 결과와 연산

- ◆ 연산이 부적절할 경우 NaN 리턴

연산자

❖ 비교 연산자

● 피연산자를 크기를 비교해서 논리값 반환

- ◆ 문자열 비교: 유니코드 값을 이용해 사전 순서로 비교
- ◆ 타입이 다른 경우 피연산자를 숫자로 변환해서 비교 → 변경 불가 시 NaN
 - NaN : 어떤 값과 어떤 비교를 해도 false 리턴

구분	문자열	boolean	참조형
문자와 비교	문자열(알파벳 순)으로 비교		
그 외	숫자로 변경 후 비교 변경 불가시 → NaN → false	true → 1, false → 0로 변환	valueOf() 결과와 연산에 따른


```
showResult("3 > 4");
showResult("10" > "2");
showResult("10" > 2);
```

X

```
console.log(new Date(2000, 1, 1).valueOf());
showResult('new Date(2000,1,1)<new Date(2000, 1, 2)');
showResult('new Date(2000,1,1)<"1949330800000"');
```

```
3 > 4 false
"10" > "2" false
"10" > 2 true
```



```
new Date(2000,1,1)<new Date(2000, 1, 2) true
new Date(2000,1,1)<"1949330800000" true
```

40

연산자

❖ 비교 연산자 - 동등 비교 연산자

● ==, !=과 ===, !==

- ◆ equal(==): 피연산자의 타입이 다를 경우 타입을 먼저 일치시켜서 내용 비교
- ◆ strict equal(===): 타입 변환을 하지 않음
 - strict 모드 사용 권장

● undefined와 null의 == 비교 : true 리턴

```
showResult("true==1");
showResult("false=='0'");
showResult("true==='1");
showResult("false==='0'");

showResult("undefined==null");
showResult("undefined====null");
```

```
true==1 true
false=='0' true
true==='1 false
false==='0' false
undefined==null true
undefined====null false
```

● 연계 연산



- ◆ 이전 연산의 결과(true/false)를 숫자로 type casting 후 비교

- 언제나 생각할 것은 true = 1, false = 0

```
showResult("1 < 2 < 3");
showResult("1 < 2 < false");
```

연산 방향

```
1 < 2 < 3 true
1 < 2 < false false
```

연산자

❖ 논리 연산자

● 비교와 할당이 동시에 발생

- ◆ 연산의 결과는 단순히 true / false가 아니라 최종 평가 값이 할당됨
- ◆ 연산 시 false로 인식하는 경우 : 0, null, ""(빈 문자열), false, undefined, NaN

● || 연산

- ◆ 좌측의 연산 결과가 true이면 좌측 값, false이고 우측의 값 리턴

```
4 || undefined  
0 || undefined
```

```
4 || undefined 4  
0 || undefined undefined
```

- ◆ 크로스 브라우징 관련해서 브라우저 지원 여부 파악을 위해 많이 사용됨

Chrome

- window.console
- window.superConsole

IE

- window.console

```
var myConsole = window.superConsole || window.console;  
  
myConsole.log("Hello MyConsole");
```

● && 연산

- ◆ 좌측의 연산 결과가 false면 바로 좌측의 결과, true이면 우측의 값 리턴

```
false && "Hello"  
true && 'Hello'
```

```
false && "Hello" false  
true && 'Hello' Hello
```

JavaScript 기본

- 제어문

조용준
(stgray22@gmail.com)

제어문

❖ 조건문 - if

- 조건식에 true / false 뿐 아니라 다른 값들도 사용 가능
 - ◆ 0, null, "" (공백문자), undefined, NaN 등 일 경우 false, 나머지는 true
- if, if ~ else, if,~ else if ~ else 구문

```
if(조건식){  
    do something..// 첫 번째 조건식이 true일 경우 실행할 문장  
}else if(조건식){  
    do something.. // 두 번째 조건식이 true 일 경우 실행할 문장  
}...  
else{  
    do something.. // 모든 조건이 false 일 경우 실행할 문장  
}
```

```
let num = Math.floor(Math.random() * 100); // 0 ~ 99까지의 정수  
  
let result;  
if (num % 2 === 0) {  
    result = "짝수";  
} else {  
    result = "홀수";  
}
```

제어문

❖ 다양한 형태의 for 문장

● for - in

◆ 증감식과 조건식을 생략하고 목시적으로 전체 키의 길이만큼 자동 반복

- for-in 문장은 배열의 원래 요소(index로 접근하는) 이외의 추가적인 속성까지 출력

◆ 객체가 가지는 properties 조회할 때 properties 개수 만큼 반복

- 내부적으로는 속성들을 모두 배열의 형태로 가지고 있음

```
for(let key in window){  
    console.log(key+ " : "+window[key]);  
}
```

window가 가지고 있는
property의 목록을 대상으로 진행

● 배열의 접근과 for - of

◆ for-in 문과 마찬가지로 내부 요소에 대한 자동 반복 지원

- 배열, Map, Set, 유사배열 객체의 요소를 출력하는 경우 적합
- 객체의 속성은 출력하지 않음

```
let array = ["Hello", "JavaScript", "World"];  
  
for(let data of array){  
    console.log("for-of: "+data + " : ");  
}
```

제어문

❖ for in vs for of

● for in의 반복 대상:

- ◆ 객체의 속성 중 enumerable 속성이 true인 내용
- ◆ 속성의 이름을 리턴하며 이를 통해 속성 값에 접근

● for of의 반복 대상:

- ◆ iterable 객체
- ◆ 속성 값을 바로 리턴

● iterable 객체란?

- ◆ String, Array, Map, Set, TypedArray, Argument 처럼 이터러블 프로토콜을 갖는 객체

● 이터러블 프로토콜

- ◆ Symbol.iterator 속성을 갖는 객체
- ◆ Symbol.iterator 속성에 next() 메서드를 가지고 있으며 이에 의해 다음 요소 출력

제어문

❖ for in vs for of

```
let array = ["Hello", "JavaScript", "World"];
array.custom="customProp";
```

배열 객체에 custom 속성 추가

```
for(let i=0; i<array.length; i++){
    console.log("org: "+i+" : "+array[i]);
}

for(let index in array){
    console.log("for-in: "+index+" : "+array[index]);
}

for(let data of array){
    console.log("for-of: "+data + " : ");
}
```

3개

4개

3개

배열에는 적합
치 않음

가

제어문

❖ 예외 처리 구문

- 목적: 프로그램 작성 오류, 파라미터 입력 오류 등에 대한 대처 기능
- 예외의 유형
 - ◆ ECMA Script에서 정의한 예외
 - Error 객체: 예외에 대한 정보를 담고 있는 객체의 최상위 객체
 - ◆ 사용자 정의 예외: 단순한 숫자 값부터 객체까지 모든 것을 예외로 사용 가능
- throw 문장
 - ◆ 예외를 발생시킬 때 사용

```
throw "Error2"; // String type
throw 42; // Number type
throw true; // Boolean type
throw {toString: function() { return "some exception!"; } };
```

제어문

❖ 예외 처리 구문

● try ~ catch ~ finally 구문을 이용해 예외 감지 및 처리

◆ try : 해당 블록에서 예외가 발생하는지 감지

- 예외 발생 시 Error 객체를 생성해서 catch 블록의 인자로 넘겨줌

◆ catch : 발생한 예외(Error 객체)에 대한 처리

- 예외가 발생하지 않으면 동작하지 않음

◆ finally : 예외발생 여부, try 및 catch의 return과 상관 없이 반드시 실행되는 블록

- 자원 반납 등의 역할에 유용
- finally에서의 리턴은 try와 catch의 리턴을 덮어쓰게 됨

```
try {
    console.log("자원 사용");
    console.log(notdefined);
    //throw new Error("에러가 발생했습니다.");
    //throw 404;
} catch (err) {
    console.log("에러 처리 : " + err.toString() + " : " + err.name + " : " + err.message);
} finally {
    console.log("언제나 실행 - 자원 반납");
}
```

JavaScript 객체지향 프로그래밍

- function

조용준
(stgray22@gmail.com)

function

❖ 역할

- 흔히 말하는 메서드 - 일을 수행하는 단위 블록
- javascript의 function은 일반 메서드와 다른 개념
 - ◆ method < function
- function의 역할

역할 구분	설명
메서드	<p>호출 가능한 루틴으로서의 함수</p> <ul style="list-style-type: none">- runtime에 () 를 붙여주면 실행- 일반적으로 알고 있는 기능
★ 객체(변수)	<ul style="list-style-type: none">- 인자로 전달 가능- 변수에 할당 가능- 다른 함수의 반환 값으로 사용 가능
생성자	<p>다른 객체를 생성할 수 있는 요소 함수</p> <ul style="list-style-type: none">- new와 함께 사용될 때일반적으로 대문자로 시작

```
function sayHello(name) {  
    this.message = "Hello";  
    console.log(this.message + " " + name)  
}  
sayHello("jang");
```

```
function getMyParseInt(x) {  
    console.log(x);  
    return x || window.parseInt;  
}  
var myParseInt = getMyParseInt(Number.parseInt);  
console.log(myParseInt("123cm"));
```

```
function Student(name) {  
    this.name = name;  
}  
console.log(new Student("홍길동").name);
```

function

❖ 함수 리터럴

- `function function_name(parameter_name, ...){
 ...
 return value;
}`

- parameter
 - ◆ function이 호출될 때 값을 전달받을 인자로 이름만 표시하며 local 변수로 취급됨
 - ◆ var | let 키워드를 사용하지 않음

- return
 - ◆ return을 실행하면 function이 종료됨
 - ◆ function을 호출한 곳으로 결과값을 돌려주기 위해서는 반드시 return 문장 필요
 - ◆ 하지만 return 하는 타입은 명시하지 않음

function

❖ 다양한 정의 방법

● 선언을 이용한 정의

```
function add(x, y) {  
    return x + y;  
}
```

● 리터럴을 이용한 함수 표현식 – anonymous function

- ◆ 바로 변수에 할당하거나 파라미터로 전달, 또는 즉시 실행

```
let minus = function(x, y) {  
    return x - y;  
};
```

```
(function(name){  
    console.log("Hello "+name)  
})("바로실행");
```

● Function의 생성자 사용

```
var multi = new Function("x", "y", "return x * y;");  
  
console.log(multi(3, 4));
```

◆ 이 방식을 주로 사용하지는 않음

- 내부적으로 eval()을 사용하는 등 적절치 못함

◆ 이를 통해서 알 수 있는 것은?

function도 하나의
객체다!

function

❖ 다양한 정의 방법

● 애로우 함수로 정의하기 ES6

◆ 기본 구문

```
(parameter list) => {  
    // do something  
}
```

- ◆ function 키워드를 생략하고 =>로 대체
- ◆ 구현부가 한 문장일 경우 {} 생략 가능
- ◆ 실행문이 return 문장 하나로 구성된 경우 return 생략 가능
- ◆ 파라미터가 하나일 경우 () 생략 가능
 - 단 파라미터가 없을 경우는 생략 불가

```
let arrowFunc = (p1, p2) => {  
    return p1 + p2;  
};
```

```
arrowFunc = (p1, p2) => p1 + p2;
```

```
arrowFunc = p => "Hello " + p;
```

메서드로서의 function

❖ 메서드로서의 function

● function의 기본 역할은 중복 코드의 제거

```
let dan = 3;
```

```
for (let i = 1; i < 10; i++) {  
    console.log(dan + "*" + i + "=" + (dan * i));  
}
```

```
dan = 5;
```

```
for (let i = 1; i < 10; i++) {  
    console.log(dan + "*" + i + "=" + (dan * i));  
}
```

로직의 반복 → 수정 요소의 반복

```
function gugu(dan){  
    for (let i = 1; i < 10; i++) {  
        console.log(dan + "*" + i + "=" + (dan * i));  
    }  
}
```

```
gugu(4);
```

```
gugu(7);
```

로직의 모듈화

메서드로서의 function

❖ 메서드로서의 function

- runtime에 function 타입의 변수에 () 를 붙여주면 실행

```
function add(x, y) {  
    return x + y;  
}  
  
add(2, 3);
```

- 호출과 인자 전달

- ◆ 파라미터의 개수가 맞지 않아도 호출됨

- 전달되지 않은 파라미터는 undefined로 처리되며 선언되지 않은 변수는 무시됨

- ◆ arguments: 모든 변수들을 배열과 유사한 형태(Array-like)로 저장하는 내장 객체

```
function minus(a, b) {  
    console.log(arguments, Array.isArray(arguments));  
    for ( let param in arguments) {  
        console.log(param, arguments[param]);  
    }  
    return a - b;  
}  
  
console.log(minus(1, 2));  
console.log(minus(1, 2, 3));  
console.log(minus(1));
```

Array-like

: 배열은 아니지만 배열처럼 length를 가지고 있으며
0<= key < length 범위에서 순차적으로 증가하는 key를 속성으로
가짐

메서드로서의 function

❖ 메서드로서의 function

● 가변길이 인수(rest parameter) 사용 ES6

```
function varArgs(...nums){  
    console.log(nums, Array.isArray(nums));  
    let sum = 0;  
    for(let num of nums){  
        if(typeof num ==="number"){  
            sum+=num;  
        }else{  
            console.log("숫자가 아님: "+num)  
        }  
    }  
    console.log("총 합: "+sum);  
}  
  
varArgs(1,4,5,9);
```

● 가변길이 인수 vs arguments

가변 길이 인수

win!!

배열 → 정렬 등 배열의 모든 기능 사용 가능

함수 선언 부만으로 필요한 파라미터 파악 가능
→ 명확한 소스 코드 분석 가능

Arguments

유사 배열 → 배열의 기능 사용에 한계

선언부로는 파라미터 파악 불가능
→ 소스 코드 분석의 어려움

메서드로서의 function

❖ 메서드로서의 function

● 인수의 디폴트 값 사용 가능 ES6

```
function defaultParam(name= "아무개"){
    console.log(name+"님 반갑습니다.")
}

defaultParam("홍길동")
defaultParam();

function required(){
    throw new Error("필수 입력 항목입니다.");
}

function useRequired(name=required()){
    console.log(name+"님 반갑습니다.")
}

useRequired("홍길동");
useRequired();
```

예외 처리를 통한 필수 입력 체크

가

메서드로서의 function

❖ 메서드로서의 function

- iterable 타입 반환 객체의 요소들을 개별 변수에 할당 **ES6**

- ◆ destructuring assignment

```
function getMinMax(...nums) {  
  return [Math.max(...nums), Math.min(...nums)];  
}  
  
let result = getMinMax(1, -10, 5, 7, -4, 10);  
let max = result[0];  
let min = result[1];  
console.log(result, max, min);  
  
let [min2, max2] = getMinMax(1, -10, 5, 7, -4, 10);  
console.log(min2, max2);
```

before

after

가

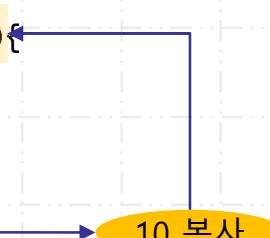
메서드로서의 function

❖ 파라미터의 타입에 따른 동작 변화

● call by value!!

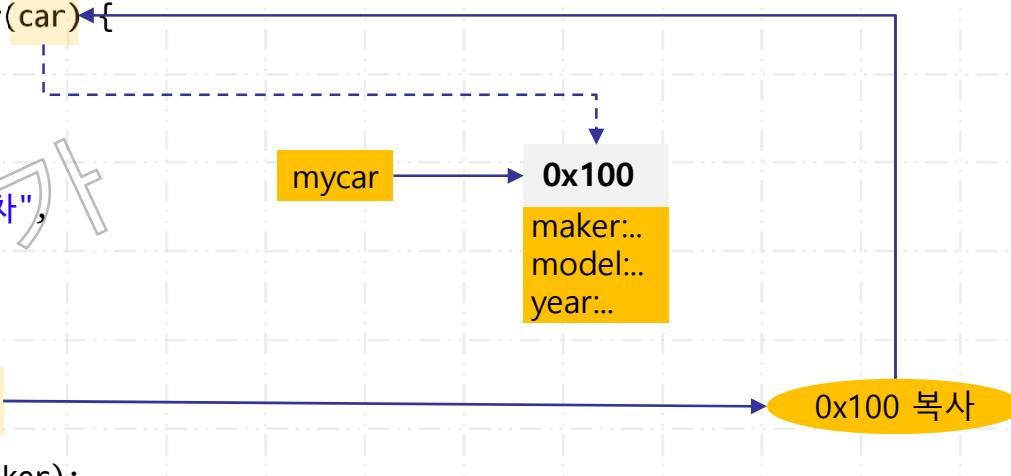
◆ 기본형 파라미터

```
function changeAge(age){  
    age+=100;  
}  
  
let age = 10;  
changeAge(age);  
console.log(age);
```



◆ 참조형 파라미터

```
function changeMaker(car){  
    car.maker = "KIA";  
}  
  
let mycar = {  
    maker : "기아자동차",  
    model : "쏘렌토",  
    year : 2016  
};  
  
changeMaker(mycar);  
  
console.log(mycar.maker);
```



60

객체로서의 function

❖ 객체로서의 function

● first class citizen

- ◆ function은 메모리상에 객체로 존재
- ◆ 변수에 할당되어 참조 값으로 사용 됨
- ◆ 함수의 반환 값이나 다른 함수 호출 시 인자로 사용됨

```
function sayHi(name) {  
  console.log("Hi " + name);  
}  
  
console.log(typeof sayHi, sayHi instanceof Object);  
  
sayHi("hong");  
  
let sayHi2 = sayHi;  
sayHi2("lim");  
  
function paramFunc(someFunc, name){  
  someFunc(name);  
}  
  
paramFunc(sayHi, "jang");
```

sayHi의 타입은? Object의 객체인가?

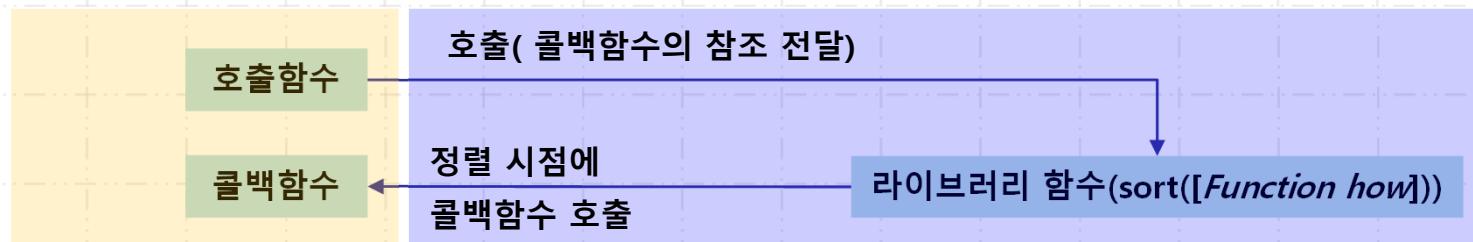
값으로써 할당

파라미터로 function 전달

객체로서의 function

❖ 인자로 활용된 function - 콜백 구조

- framework에 runtime에서 사용할 function을 넘겨줌



◆ 콜백 함수: 나중에 호출되어야 하는 함수

- 배열의 sort 적용

```
Array jsArray.sort(Function funct)
function sort(funct)
See Also:
  Array
Parameters:
  {Function} funct
Since:
  Standard ECMA-262 3rd. Edition
  Level 2 Document Object Model Core Definition.
@returns
  {Array}
Press 'F2' for focus
```

```
let arr = ["alpha", "zz", "dream is come true", "hi"];
arr.sort();
console.log(arr);
```

어떻게 정렬할 것인가?

```
arr.sort(
  function(first, second) {
    return first.length - second.length;
});
console.log(arr);
```

음수: 자리 유지
0: 무시
양수: 자리 바꿈

62

조용준(stgray22@gmail.com)

생성자로서의 function

❖ 생성자로서의 function

```
let hong = {  
    name : "홍길동",  
    age : 30,  
    info : function() {  
        return this.name+", "+this.age;  
    }  
};  
  
console.log(hong.info());
```

객체는 만들어지지만..

임꺽정은? 장길산은?

● 객체를 찍어내는 생성자 역할로의 function

- ◆ 작성: 생성자를 뜻하는 의미로 첫 글자를 대문자로 하며 나머지 부분은 일반 함수와 동일
- ◆ 호출: new 키워드와 함께 function 호출

```
function Person(name, age) {  
    this.name = name;  
    this.age = age;  
    this.info = function() {  
        return this.name + " : " + this.age;  
    };  
}  
  
let jang = new Person("장길산", 20);  
console.log(jang.info());  
let lim = new Person("임꺽정", 25);  
console.log(lim.info());
```

Hoisting

❖ java script 실행 단계 이해와 호이스팅(hoisting)

- 파싱과 실행 두 단계를 거친다.
- 파싱 : 프로그램 전역 레벨에서의 파싱

let은 hoisting의 대상이 아님

- ◆ 어떤 함수에도 포함되지 않은 var 변수의 인지 및 undefined 초기화
- ◆ 전역레벨에서 이름있게 정의된 함수에 대해 함수명과 동일한 변수 생성
 - 변수에 실행 코드가 담긴 함수에 대한 참조로 초기화

- 실행 : 할당(=)등 실행문을 실행하다가 함수 호출을 만나면 해당 함수 레벨의 파싱 단계 반복
 - ◆ 함수 코드에서 함수의 지역 변수와 함수의 변수를 정의하고 나서야 함수 코드 실행

```
console.log(square(4));
var square = 0;

function square(x){
    return x * x;
}

console.log(square);
//console.log(square());
```

파싱

실행

1. square 변수 인지 → undefined 초기화
2. square이름의 함수 인지 → 변수 생성
3. square 실행코드가 담긴 함수 블록 생성
4. reference를 square에 할당

1. console.log(square(4)) 실행
2. square function 블록 파싱
3. square 실행
4. square에 0 할당

Hoisting

❖ java script 실행 단계 이해

```
console.log("num1 : "+num1);
//console.log(num2);
console.log("square : "+square);

var num1 = 10;
console.log("num1 : "+num1);
num2 = 20;
console.log("num2 : "+num2);

console.log("square 호출 : "+square(10));

var square = 0;
function square(x){
  return x * x;
}

console.log(square);

//console.log(square(4));
```

X

```
var square2 = function(){
  return x * x;
}

console.log("square2 : "+square2);
```

파싱 단계

```
num1 : undefined
square : function square(x){
  return x * x;
}
square2 : undefined
```

실행 단계

```
console.log("num1 : "+num1);
//console.log(num2);

console.log("square : "+square);

num1 = 10;           num1 : 10
console.log("num1 : "+num1);

num2 = 20;           num2 : 20
console.log("num2 : "+num2);

console.log("square 호출 : "+square(10));

square = 0;          square : 0
console.log(square);
//console.log(square(4));
```

.....

JavaScript 변수의 Scope

❖ 유효범위(Scope)

- 선언된 변수 또는 매개변수를 참조할 수 있는 범위 → 메모리에서의 생존 기간
- 유효 범위에 따라 전역 변수 / 지역 변수로 구분
- 전역 변수
 - ◆ 전역 영역에 선언된 변수로 전역 객체의 속성
 - 웹 환경에서는 Window가 전역 객체
 - ◆ 스크립트 전체에서 참조되는 것으로 스크립트 내 어느곳에서든 참조 가능
 - ◆ 주의! var나 let 키워드를 사용하지 않은 경우 전역 변수가 됨
 - 원치 않게 window의 속성을 업데이트 하지 않도록 주의 필요!!

● 로컬 변수

- ◆ function 내부에서 변수 선언자(var 등)와 함께 선언된 변수 또는 파라미터 변수
- ◆ 변수의 scope는 선언된 function 또는 block 내부로 한정되며 외부에서 접근 불가

JavaScript 변수의 Scope

❖ JavaScript 변수 Scope의 특징

● 로컬 변수와 전역 변수

전역 객체의 속성

```
<script>
  function myFunction() {
    var localVar1 = 10;
    localVar2 = 20;
    console.log("function 내부 : " + localVar1 + ", " + localVar2);
    alert = console;
  }
  var rootVar1 = 1;

  rootVar2 = 2;
  console.log("전역 영역: " + rootVar1 + " : " + rootVar2);

  myFunction();
  console.log("전역 영역 : " + localVar2 + " : " + window.localVar2);

  alert.log("Hello");
</script>
```

전역 변수에 대한 재 정의 발생

전역 범위

지역 범위

JavaScript - jQuery 기초

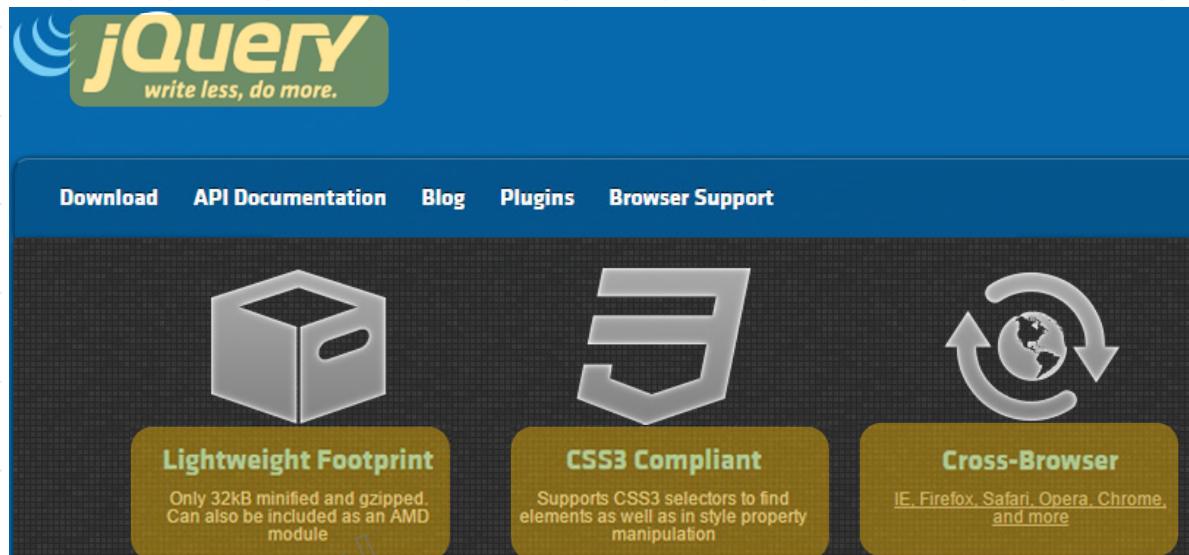
가

조용준
(stgray22@gmail.com)

jQuery?

❖ jQuery란?

- 빠르고 작고 기능이 풍부한 JavaScript 라이브러리
 - ◆ 선수 과정으로 JavaScript에 대한 사전 이해 필요



● 특징

- ◆ DOM과 관련된 처리, 일관된 이벤트 연결을 쉽게 구현
- ◆ 시각적 효과를 쉽게 구현
- ◆ Ajax 애플리케이션을 쉽게 개발
- ◆ 호환성 문제점 해결(크로스 브라우징에 대한 처리 기본 내장)

jQuery?

❖ jQuery 설치

● 라이브러리 다운로드 방식

- ◆ <http://jquery.com/download>

- ◆ 버전

- compressed는 주석, whitespace를 모두 제거 → 가독성 '0'으로 production 용
- uncompressed는 주석 등이 보존된 development 용
- slim: ajax 기능이 제거된 버전 – 용량 감소

- ◆ 다운로드 후 WebContent아래 js 폴더 생성 후 저장

- ◆ html 또는 스크립트에서 jQuery 라이브러리 참조

- `<script src="../js/jquery-2.1.1.js"></script>`

● 네트워크 전송 방식(CDN 방식 : content delivery network)

- ◆ 다운로드 하지 않고 네트워크에서 참조하는 방식

- ◆ code.jquery.com 참조

```
<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
```

jQuery?

❖ jQuery 설치

● jQuery 사용을 위한 준비 작업

```
<script type="text/javascript" src="../js/jquery-3.2.0.js"></script>

<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>

<script>
$(document).ready(function() {
    console.log("$", $);
});
</script>

$ = function ( selector, context ) {
    // The jQuery object is actually just the init constructor 'enhanced'
    // Need init if jQuery is called (just allow error to be thrown if not included)
    return new jQuery.fn.init( selector, context );
}

$ 문자는 jQuery 함수의 별칭
```

jQuery?

❖ \$(document).ready() 함수

- DOM을 사용하기 위해서는 문서 로딩이 먼저 종료되어야 함

- ◆ JavaScript: window.onload 함수 이용

- ◆ jQuery: \$(document).ready(function(){...}) 이용

- 문서 loading이 완료되면 ready 안의 function을 실행하라!!

- ◆ (document).ready는 기본적으로 생략 가능 \$(function(){ // do something});

```
<script type="text/javascript" src="../../js/jquery-3.2.0.js"></script>
<script>
    $(document).ready(function() {
        $("#userid").val("Hello");
    });
    $(function() {
        $("#userid").val("jQuery");
    });
    $("#userid").val("world");
</script>

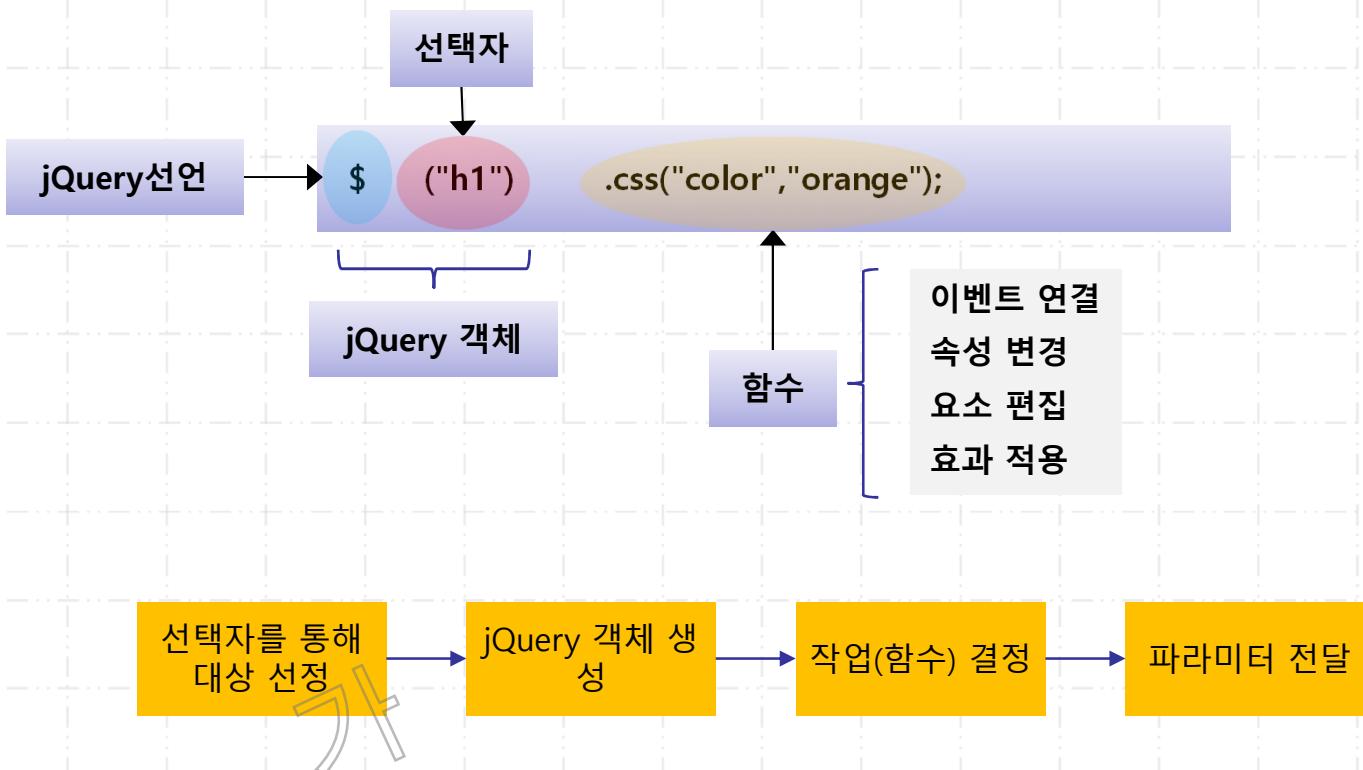
<body>
    <input type="text" id="userid">
</body>
<script>
    $("#userid").val("world 2");
</script>
```

아직 DOM 구성 이전
→ 이벤트로 처리

또는 태그가 모두 끝난 후 사용

jQuery

❖ 기본 사용법



● 기존 처리 방식과의 차이

- ◆ 정적: 스타일(CSS) : `h1 { color: orange; }`
- ◆ 동적: 자바스크립트(JS) : `document.querySelector("h1").style.color = "orange";`
- ◆ 동적: 제이쿼리(jQuery) : `$("h1").css("color", "orange");`

73

선택자

❖ 주요 선택자

● jQuery selector

◆ http://www.w3schools.com/jquery/jquery_ref_selectors.asp

● 직접 선택자

종류	예	설명
전체 선택자	<code>\$("")</code>	모든 요소 선택
아이디 선택자	<code>\$("#id_name")</code>	id 속성이 매핑된 요소 선택
클래스 선택자	<code>\$(".class_name")</code>	class 속성이 매핑된 요소 선택
태그 명 선택자	<code> \$("p")</code>	문서에 있는 모든 <p> 요소 선택
그룹 선택자	<code> \$(".class_name, #id_name, p")</code>	지정된 요소들을 한번에 선택
선택자의 복합 사용	<code> \$("p.class_name") \$("p#id_name")</code>	p 중에 클래스 이름 및 id 매핑

선택자

❖ 주요 선택자

● 계층 선택자

종류	예	설명
하위 자손 선택자	<code>\$("#wrap li")</code>	wrap id의 하위에 있는 li 모두 선택(depth 무관)
자식 요소 선택자	<code>\$("#wrap > li")</code>	wrap id의 직계 자식만으로 한정 선택

- ◆ 자손: depth를 따지지 않고 등장하는 하위 요소들
- ◆ 자식: 지정한 태그의 바로 다음 depth에 등장하는 하위 요소들

● 동위(동생) 선택자

종류	예	설명
바로 동생 선택자	<code>"label + input")</code>	<label> 바로 다음에 오는 <input> 요소 선택
모든 동생 선택자	<code>("#target ~ li")</code>	target id의 동생 요소 선택

선택자

❖ 주요 선택자

● 속성 선택자 [...]

종류	예	설명
요소[속성]	<code>\$("input[type]")</code>	타입 속성이 선언되어있는 input 요소
요소[속성=값]	<code>\$("input[type=값]")</code>	타입 =값 인 input 요소
요소[속성!=값]	<code>\$("input[type!=값]")</code>	타입과 값이 다른 input 요소
요소[속성~=값]	<code>\$("input[type~="값"]")</code>	타입의 값이 단어수준으로(공백구분) 포함된 input 요소
요소[속성*=값]	<code>\$("input[type*="값"]")</code>	속성 안의 값이 특정 값을 포함하는 요소 선택
요소[속성^=값]	<code>\$("input[type^="값"]")</code>	속성 안의 값이 특정 값으로 시작하는 요소 선택
요소[속성\$=값]	<code>\$("input[type\$="값"]")</code>	속성 안의 값이 특정 값으로 끝나는 요소 선택
요소[속성]..[속성]	<code>\$("input[type^="값"][type\$="값"])</code>	여러 조건들을 and 조건으로 묶어서 처리

선택자

❖ 필터 선택자

● 위치 기반 필터 선택자

- ◆ 부모를 기준으로 하며 위치 index는 0 기준
- ◆ index가 음수일 경우 뒤에서부터의 순서

종류	예	설명
요소:odd	<code>\$("li:odd")</code>	홀수 번째 등장하는 요소
요소:even	<code>\$("li:even")</code>	짝수 번째 등장하는 요소
요소:first	<code>\$("p:first")</code>	<p>가 부모에 처음에 등장하는 경우. eq(0)와 동일
요소:last	<code>\$("p:last")</code>	<p>가 부모의 맨 마지막에 등장하는 경우
요소:eq(n)	<code>\$("p:eq(3)")</code>	부모 태그를 기준으로 <p> 중에서 index가 3인 요소
요소:gt(n)	<code>\$("p:gt(3)")</code>	부모 태그를 기준으로 index가 3보다 큰 <p>요소
요소:lt(n)	<code>\$("p:lt(3)")</code>	부모 태그를 기준으로 index가 3보다 작은 <p>요소

선택자

❖ 필터 선택자

● 위치 기반 필터 선택자

종류	예	설명
요소:first-child	<code>\$("p:first-child")</code>	모든 자식들 중 <p>가 1번째로 등장한 경우(장자 선택)
요소:last-child	<code>\$("p:last-child")</code>	모든 자식들 중 <p>가 마지막에 등장한 경우(막내 선택)
요소:nth-child(숫자n+숫자)	<code>\$("p:nth-child(2n+1)")</code>	자식들 중 <p>가 홀수 번째 등장한 경우
요소:nth-child(odd)	<code>\$("p:nth-child(odd)")</code>	자식들 중 <p>가 홀수 번째 등장한 경우
요소:nth-child(even)	<code>\$("p:nth-child(even)")</code>	자식들 중 <p>가 짝수 번째 등장한 경우
요소:first-of-type	<code>\$("li:first-of-type")</code>	자식 들 중 처음 등장한 경우
요소:last-of-type	<code>\$("li:last-of-type")</code>	자식 들 중 마지막에 등장한 경우
요소:nth-of-type(숫자n+숫자)	<code>\$("li:nth-of-type(2n)")</code>	자식 중 짝수 번째 등장한 경우

◆ 숫자 n은 0 부터 시작하는 정수 값

- n은 index가 아님!!

◆ nth-child에서는 index가 1부터 시작

- :even, :odd의 index가 0부터 시작하는 것과 다름 주의

선택자

❖ 필터 선택자

● 폼 필터 선택자

종류	예	설명
:input	<code>\$(":input")</code>	사용자에게 입력을 받는 모든 요소 선택 – select 포함 <code>\$("input")</code> 은 <input> 태그만 선택
:type_name	<code">(":password")</code">	특정 타입의 input 요소 선택 checkbox, button, file, image, radio, submit, text, hidden
요소:checked	<code>\$(":radio:checked")</code>	radio 중 체크된 입력 양식 선택
요소:disabled	<code>\$(":input:disabled")</code> ,	input 요소 중 비 활성화된 입력 양식 선택
요소:enabled	<code>\$(":enabled")</code>	활성화된 입력 양식 선택
요소:selected	<code>\$("option:selected")</code>	option 객체 중 선택된 요소 선택
요소:focus	<code>\$("input:focus")</code>	현재 포커스를 가지고 있는 입력 양식 선택

선택자

❖ 필터 선택자

● 내용 필터

종류	예	설명
요소:contains(문자열)	<code>\$(“p:contains(‘first’)”)</code>	문자열 'first'를 포함하는 <p> 요소
요소:empty	<code>\$(:empty)</code>	자식 요소가 없고 텍스트가 비어있는 요소
요소:parent	<code>\$(“td:parent”)</code>	td 중 자식 요소가 존재하거나 text 값을 가지고 있는 요소 선택
요소:has(자손요소)	<code>\$(“div:has(hr)”)</code>	<hr>을 자손으로 가지고 있는 <div>요소

● 기타

종류	예	설명
요소:not(선택자)	<code>\$(“div:not(.intro)”)</code>	클래스가 intro가 아닌 <div>요소
:header	<code>\$(“:header”)</code>	모든 헤더(<h1>, <h2>,...) 선택

가

Traversing API

❖ Traversing API

- 문서의 특정 요소들을 탐색하는 역할의 API
- 선택자는 DOM을 대상으로 선택
 - ◆ Traversing API는 선택자를 통해 요소를 선택한 후 filter 역할
 - ◆ 선택자를 통해 바로 원하는 요소에 접근하기 어려울 경우 사용
- 많은 경우 jQuery의 필터 선택자와 유사한 기능
 - ◆ API는 체이닝 형태로 2차, 3차 탐색 가능
- 주요 구분
 - ◆ filtering
 - ◆ 기타 traversing
 - ◆ tree traversing

Traversing API - Filtering

❖ Filtering 관련 함수

● eq(index)

- ◆ 선택된 여러 요소 중 지정된 index와 일치하는 요소 반환
- ◆ :eq(index)와 기능면에서 동일하지만 다른 함수와 chaining 가능
- ◆ index는 0부터 시작하며 음수의 경우 뒤에서 부터의 순서
- ◆ 활용 예

```
// <li> 중에서 3번째 요소  
$selected = $("li").eq(3);
```

● has(selector)

- ◆ 선택된 요소 중 selector를 포함하는 요소 반환
- ◆ 활용 예

```
// tr요소 중 th를 가진 요소 선택  
$selected = $("tr").has("th");
```

Traversing API - Filtering

❖ Filtering 관련 함수

● filter(expr)

◆ 선택된 여러 요소 중 지정된 expr에 부합하는 요소 반환

- expr은 selector 또는 jQuery 표현식 사용, 함수 이용

◆ 함수 사용의 경우

- callback 함수로 index와 요소(element)를 받아 처리 가능
- function 내부에서 true가 반환되면 선택

◆ 활용 예

```
// p 중 짝수 번째 요소  
$selected = $("p").filter(":even");
```

```
// h3 중 3의 배수 번 index 요소들 선택  
$("h3").filter(function(index) {  
    return index % 3 == 0;  
})
```

Traversing API - Filtering

❖ Filtering 관련 함수

● not(expr)

◆ filter와 반대되는 개념으로 expr과 일치하지 않는 요소 선택

- expr은 selector 또는 jQuery 표현식 사용, 함수 이용

◆ 함수 사용의 경우

- callback 함수로 index와 요소(element)를 받아 처리 가능
- function 내부에서 true가 반환되면 선택

◆ 활용 예

```
// p 중 짝수 번째가 아닌 요소  
$selected = $("p").not(":even");
```

```
// h3 중 3의 배수 번 index 요소들을 제외하고 선택  
$("h3").not(function(index) {  
    return index % 3 == 0;  
})
```

Traversing API - Filtering

❖ Filtering 관련 함수

● is(expr)

◆ 선택된 요소에 expr에 해당하는 요소가 하나라도 있으면 true 그렇지 않으면 false 반환

- 주로 if 문장과 함께 사용됨
- expr은 selector 또는 jQuery 표현식 사용, 함수 이용

◆ 함수 사용의 경우

- callback 함수로 index와 요소(element)를 받아 처리 가능
- function 내부에서 true가 반환되면 선택

◆ 활용 예

```
// 선택된 요소에 체크된 요소가 있는지 여부를 console에 출력하시오.  
if ($selected.is(":checked")) {  
    console.log("체크된 요소가 있음");  
}  
  
// 선택된 요소에 값이 1234인 요소가 있는지 여부를 console에 출력하시오.  
if ($selected.is(function(index, elem) {  
    return $(elem).val()=='1234';  
})) {  
    console.log("값이 1234인 요소가 있음");  
}
```

val(): input 요소의 값

◆ is vs has?

- 해당 요소 리턴: has, 해당 요소가 있는지 리턴: is

Traversing API - Filtering

❖ Filtering 관련 함수

● first(), last()

- ◆ 선택된 요소 중 각각 첫 번째 요소와 마지막 요소 반환
- ◆ 활용 예

```
// disabled된 상태의 요소 중 처음 요소  
$selected = $(":disabled").first();  
// disabled된 상태의 요소 중 마지막 요소  
$selected = $(":disabled").last();
```

● map(fn)

- ◆ 1차 선택자에 의해 반환된 요소들을 함수(fn)을 이용해 가공 후 결과를 배열로 반환
- ◆ 활용 예

```
// li 요소들의 텍스트를 대괄호로 감싼 내용을 출력하시오.  
var array = $("li").map(function(index, elem){  
    return "["+$("li").text()+""];  
});  
  
console.log(array[0]);
```

text(): 요소의 문자열 내용

Traversing API - Filtering

❖ Filtering 관련 함수

● slice(start [, end])

- ◆ 1차 선택자에 의해서 반환된 전체 요소 중에서 index 기준으로 부분 요소를 얻을 때 사용
 - index는 0 부터 시작
- ◆ 범위는 start \leq x < end이며 end 생략 시 마지막 요소까지

```
// li 요소 중 2<=x<4인 범위의 요소  
$selected = $("li").slice(2,4);
```

가

Traversing API - Tree Traversal

❖ Tree Traversal 관련 함수

- 필터와 달리 계층(hierarchy)관계로 DOM을 탐색하는 API

- **parent([selector])**

- ◆ 선택된 요소에서 selector와 일치하는 부모 요소 반환

- ◆ 활용 예

```
// 아이디 선택자 : id가 org인 요소의 부모  
var selected = $("#org").parent();
```

- **offsetParent()**

- ◆ 선택된 요소의 style 속성 중 position 속성이 relative 또는 absolute인 가장 가까운 부모

- ◆ 해당 요소가 없을 경우는 <html>까지 올라감

- ◆ 활용 예



```
// id가 org인 요소의 조상 중 relative 또는 absolute인 요소  
$selected = $("#org").offsetParent();
```

Traversing API - Tree Traversal

❖ Tree Traversal 관련 함수

● parents([selector])

- ◆ 선택된 요소에서 selector와 일치하는 조상 요소들 반환

- ◆ 활용 예

```
// 아이디 선택자 : id가 org인 요소의 조상 중 p 또는 div인 요소  
$selected = $("#org").parents("p, div");
```

● closest(selector)

- ◆ 선택된 요소에서 selector에 부합되는 가장 가까운 조상 요소 하나 반환

- ◆ 활용 예

```
// legend의 조상들 중 가장 가까운 fieldset 요소  
$selected = $("legend").closest("fieldset");
```

● closest vs parents

closest

현재 요소에서 시작해서 탐색

selector와 일치하는 요소 하나를 찾을 때까지 탐색

parents

현재 요소의 부모 요소에서 시작해서 탐색

루트 요소까지 각각의 상위 요소를 모두 탐색

Traversing API - Tree Traversal

❖ Tree Traversal 관련 함수

● children([selector])

- ◆ 선택된 요소에서 selector에 부합되는 자식 요소 반환
 - selector 생략 시 전체 자식 반환

◆ 활용 예

```
// <ul>의 자식들  
selected = $("ul").children();
```

● find(selector)

- ◆ 선택된 요소에서 selector와 일치하는 자손 요소

◆ 활용 예

```
// body 하위의 p 요소  
$selected = $("body").find("p");
```

filter()

- 선택된 요소 중 내가 원하는 요소 검색

● siblings([selector])

- ◆ 선택된 요소에서 selector와 일치하는 모든 형제 요소(선택 요소는 제외됨)

◆ 활용 예

```
// 두 번째 <li>의 형제들  
$selected = $("li:nth-child(2)").siblings();
```

90

Traversing API - Tree Traversal

❖ Tree Traversal 관련 함수

● next([selector]), nextAll([selector])

- ◆ 선택된 요소 중 selector와 일치하는 바로 다음 동생 요소 또는 동생들

- ◆ 활용 예

```
// 두 번째 <li>의 동생  
$selected = $("li:nth-child(2)").next();  
// 두 번째 <li>의 동생들  
$selected = $("li:nth-child(2)").nextAll();
```

● prev([selector]), prevAll([selector])

- ◆ 선택된 요소 중 selector와 일치하는 바로 이전 형 또는 형들

```
// 두 번째 <li>의 형  
$selected = $("li:nth-child(2)").prev();  
// 두 번째 <li>의 형  
$selected = $("li:nth-child(2)").prevAll();
```

Traversing API - Miscellaneous

❖ Miscellaneous Traversing 관련 함수

● add(expr)

- ◆ 기존 검색 요건에 expr을 추가함
- ◆ expr에는 selector, elements, html, selection 등이 올 수 있음
- ◆ 활용 예

```
// 모든 h1 태그와 b 태그  
$selected = $("h1").css("font-style", "italic").add("b").css("border", "1px solid red");
```

● addBack([selector])

- ◆ 필터링 이전의 최초 요소까지 포함해서 반환
- ◆ 활용 예

```
// div의 자손 중 id가 name인 요소와 div 자신  
$selected = $("div").find("#name").addBack();
```



Traversing API - Miscellaneous

❖ Miscellaneous Traversing 관련 함수

● contents()

- ◆ text를 포함한 모든 자식 요소 반환
- ◆ 일반적으로 filter, find 등과 함께 사용됨
- ◆ 활용 예

```
// body의 자손 중 ul의 모든 자식 요소(텍스트 포함)  
$selected = $("body").find("ul").contents();
```



```
[0] ""  
[1] <li>헐크</li>  
[2] ""
```

● end()

- ◆ traverse 함수를 사용해 필터링하기 전의 상태로 복귀하는 함수
 - traverse 는 단방향으로 진행되므로 end는 현재의 traverse를 끝내고 처음으로 되돌아간다는 의미
- ◆ 활용 예

```
// ul의 자손 중 짹수번째 index 요소의 배경은 blue, 홀수번째 요소를 yellow로 처리  
$("ul").find(":even").css("background-color", "blue").end()  
    .find(":odd").css("background-color", "yellow");
```

기타 유용한 함수

❖ \$.each(obj, function), selection.each(function)

● 배열 또는 Map 형태의 반복되는 데이터를 순회하는 함수

- ◆ `$.each(object, function(index, item){ })`
- ◆ `$(selector).each(function(index, item){ })`
 - object: 반복될 대상 – 배열 또는 Map
 - function – object의 item 들에 대해서 처리할 callback

● callback 동작

- ◆ object에 있는 item의 개수만큼 순차적으로 호출됨
- ◆ `function(key, item)`
 - key: 배열 계열에서는 해당 아이템의 index, Map 계열에서는 key 값
 - item: key에 해당하는 element 또는 value

◆ 반복을 그만둘 경우 callback에서 false 리턴

```
var array = [ "Hello", "JavaScript", "world" ];  
$.each(array, function(index, item) {  
    console.log(index, item);  
});  
  
$(array).each(function(index, item){  
    console.log(index, item);  
});
```

array가 단순 객체일 경우
- for in 처럼 객체의 속성 조회

기타 유용한 함수

❖ selection.get(index) 또는 selection[index]

- selection이 가지고 있는 원래의 DOM 요소 리턴

- ◆ .eq(index)와 유사하지만 dom 객체인지 jquery 객체인지가 차이

```
let $buttons = $("[type=button]");
console.log($buttons.length);
for(let i =0; i<$buttons.length; i++){
  console.log($buttons.get(i), $buttons.eq(i));
}
```

```
var $h1 = $("h1");
$h1.eq(0).css({background:"yellow"});
```

```
var elem = $h1.get(0);
elem.style.color="blue";
```

```
<input type="button" id="clickMe" value="클릭">
▶ jQuery.fn.init [input#clickMe, prevObject: jQuery.fn.init(4)]
<input type="button" id="off" value="클릭그만">
▶ jQuery.fn.init [input#off, prevObject: jQuery.fn.init(4)]
<input type="button" id="offAll" value="색깔그만">
▶ jQuery.fn.init [input#offAll, prevObject: jQuery.fn.init(4)]
<input type="button" id="one" value="음모">
▶ jQuery.fn.init [input#one, prevObject: jQuery.fn.init(4)]
```

- ◆ 배열의 요소 접근의 형태로도 접근 가능

```
if (!$media.prop("paused") && !$media.prop("ended")) {
  // $media.trigger("pause");
  $media[0].pause();
  $(this).html("play");
  window.clearInterval(loop);
}
```

HTML5의 경우 태그 자체가 갖는 기능을
사용할 때는 원래의 객체가 필요

기타 유용한 함수

❖ `$.extend(target, addObject [, addObject, ...])`

- 기존의 객체에 addObject의 속성을 추가 해서 확장 하는 함수

- ◆ target에 이미 addObject의 속성이 있었다면 override함

```
var obj = {name: "andy"};  
  
console.log(obj);  
  
obj.age = 20;  
obj.married=false;  
  
console.log(obj);  
  
$.extend(obj, {  
    addr: "seoul",  
    tel: "010"  
});  
  
console.log(obj);  
  
$.extend(obj, window);
```

고전적 JavaScript의 방식

jQuery의 extend 함수 활용

window 객체의 상속

가

JavaScript - jQuery 활용

조용준
(stgray22@gmail.com)

가

jQuery 활용

❖ jQuery의 목적은 쉽게 사용하기

- 호환성 문제점 해결(크로스 브라우징에 대한 처리 기본 내장)
- DOM 과 관련된 처리
- 일관된 이벤트 연결을 쉽게 구현
- 시각적 효과를 쉽게 구현
- Ajax 애플리케이션을 쉽게 개발
- Attribute API
- Manipulation API
- Event API
- Effective API

가

98

jQuery 활용

❖ 기본적인 함수 사용법

● 대부분 동일한 이름의 함수를 조회와 설정에 사용

단순 계열		key-value의 맵 계열
조회	파라미터 없음 <code>console.log(\$h1.text());</code>	파라미터로 조회할 key 1개 전달 <code>console.log(\$label.css("color"));</code>
설정	파라미터로 설정할 value 1개 전달 <code>\$h1.text("jQuery Features");</code>	파라미터로 설정할 key와 value 두 개 전달 <code>\$label.css("border", "1px dotted black");</code>

● value 항목에는 단순 값, object, function 대응

- ◆ 단순 값은 문자열, 숫자 등 기본 데이터 타입
- ◆ object는 {key:value, key:value} 형태로 여러 쌍을 설정할 수 있음
- ◆ function은

- index, data 파라미터를 통해 기존 값을 받아올 수 있으며 설정할 값을 리턴
- function 내부에서의 this는 event source가 DOM 객체로 전달됨
- function에서 넘겨받은 data는 역시 DOM 객체

jQuery 활용

❖ 브라우저에서 사용되는 JS

● Document Object Model(DOM)

- ◆ HTML 문서의 구조에 관한 객체
- ◆ 태그의 엘리먼트와 앤트리뷰트를 모두 객체로 표현하고 관리

● event 처리

- ◆ 화면에서 발생하는 사용자 이벤트와 상호작용 처리
- ◆ callback function을 활용하는 구조

● Browser Object Model(BOM)

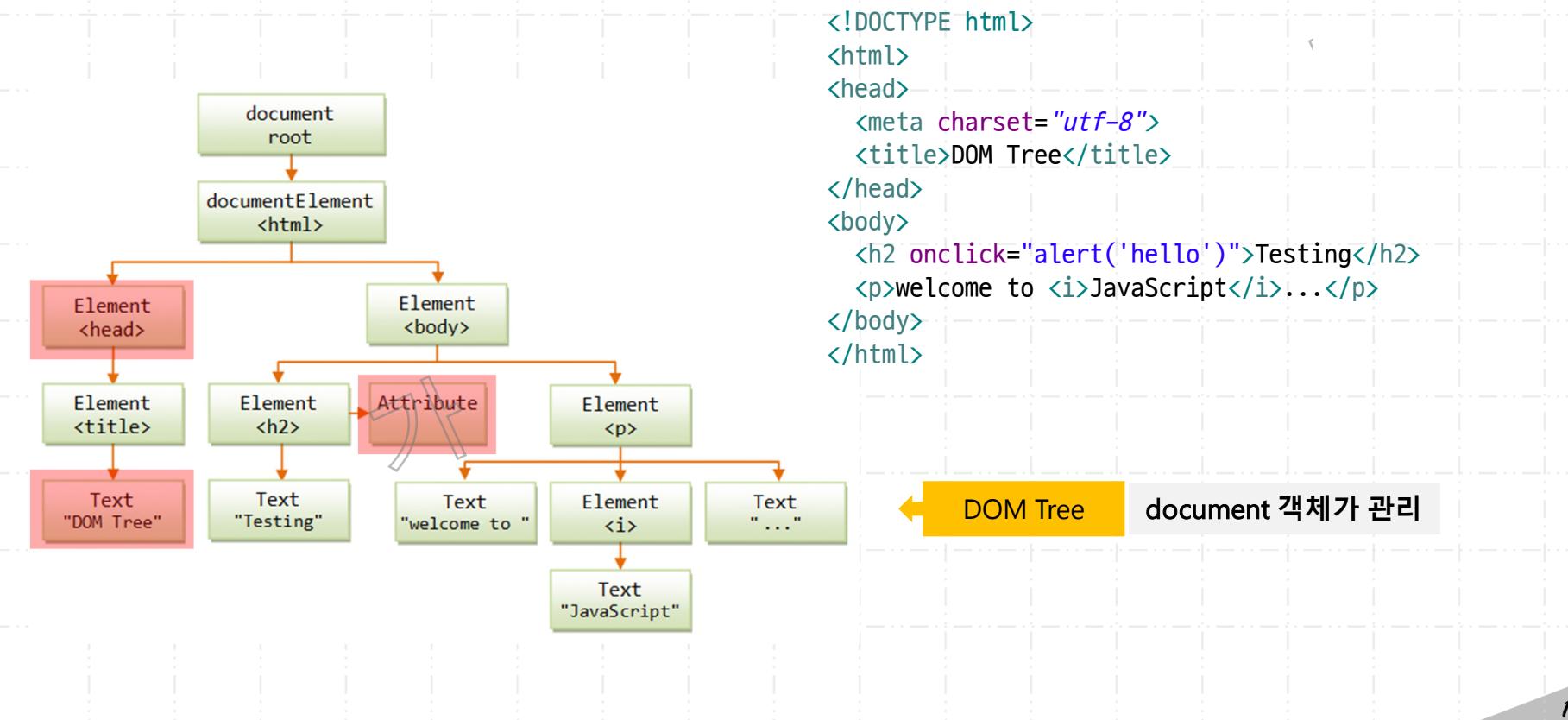
- ◆ 브라우저에 계층적으로 내장된 객체
- ◆ window, screen, location, history, navigator

100

DOM

❖ DOM(Document Object Model)

- 문서를 구성하고 있는 요소들에 대한 function과 property를 갖는 문서 객체
 - ◆ 문서를 구성하는 모든 것(element, attribute, text)을 node로 표현
 - ◆ 문서 객체의 선택 → 제어, 구조변경 및 CSS 적용 등



101

DOM

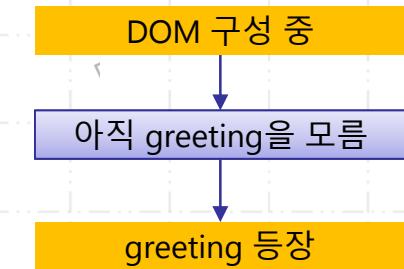
❖ DOM

● JavaScript에서 DOM 활용을 위한 문서 구조

◆ DOM 객체는 브라우저가 요소들을 읽어 들인 후 생성 가능

- 자바스크립트에서 DOM을 제어하기 위해서는 선언 위치가 중요함

```
<html>
  <head>
    <script>
      alert(document.getElementById("greeting").innerHTML);
    </script>
  </head>
  <body>  <p id="greeting">welcome to </p> </body>
</html>
```



◆ 또는 onload 이벤트 후 실행되도록 함

```
<script>
  window.onload=function(){
    alert(document.getElementById("greeting").innerHTML);
  }
</script>
```

DOM

❖ Vanilla 스크립트에서의 DOM 처리 API

● 요소의 선택

종류	활용예	설명
id 선택자	<code>document.getElementById("id")</code>	아이디를 이용해 요소 선택
tag 선택자	<code>document.getElememtsByTagName("tag_name")</code>	태그 이름에 해당하는 요소들을 유사배열로 리턴
폼 요소 선택	<code>document.formName.inputName</code>	formName 내부의 inputName 요소 선택
css 선택자 적용	<code>document.querySelector("CSS_선택자")</code>	CSS 선택자 바로 사용 가능
	<code>document.querySelectorAll("CSS 선택자")</code>	배열 형태로 리턴

가

DOM

❖ Vanilla 스크립트에서의 DOM 처리 API

● 요소의 활용

속성	설명
innerHTML	node의 HTML 내용을 문자열로 리턴
attributes	엘리먼트 노드의 attribute node들
style	CSS style요소에 접근(style.color='red')
value	form에서 input 요소의 값 리턴
nodeName	노드의 이름(tagName과 같은 값)
nodeType	노드의 타입으로 Element=1, Attribute=2, Text=3의 값을 가짐

◆ **innerHTML, attributes, style, value**는 쌍방향으로 조회 및 할당 가능

```
let greeting = document.getElementById("greeting");

console.log(greeting.innerHTML);
greeting.innerHTML = "Hello JavaScript Dom";

console.log(greeting.style);
```

104

조용준(stgray22@gmail.com)

DOM 처리 지원

❖ Attributes API

● HTML 태그의 속성을 제어하는 함수 제공

- ◆ 문서의 태그 명을 제외하고는 모두 속성

```

```

● text([newValue]), html([newValue])

- ◆ 요소의 text 또는 html 을 반환하거나 새로운 값으로 대체

- text는 html 요소를 태그로 반영하지 못하고 단순 문자열로만 처리

```
var $h1 = $("h1");
console.log($h1.text());
$h1.text("jQuery Features");
```

조회

value 설정

```
var $li = $("li");
$li.each(function(idx, data){
    console.log($(data).html());
});
```

each()를 통한 접근

순수 DOM 객체를 다시 jQuery 객체로 변경

```
$li.html(function(idx, data) {
    return "<b>" + idx + " : " + data + "</b>";
});
```

function 설정의 형태

105

DOM 처리 지원

❖ Attribute API

● val([newValue])

◆ <input>, <select>, <textarea>등 <form>과 관련된 태그에서 value 조회 또는 설정

- radio, checkbox, select 등은 일반적으로 :checked, :selected 등과 같이 사용

◆ 활용 예

```
$("#checkedValue").val("newValue");
```

```
var radio = $(":radio:checked");
var value = radio.val();
```

가

106

DOM 처리 지원

❖ Attribute API

● css(prop [, value]), css(obj)

- ◆ 선택한 요소가 가지는 css의 prop 속성을 반환하거나 설정
- ◆ obj 객체를 이용해 여러 속성을 동시에 설정 가능
 - CSS 속성값 중 '-'가 포함된 항목은 반드시 문자열로 처리해야 함

◆ 활용 예

```
var $h1 = $("h1");
console.log($h1.css("color"));

$h1.css({
  color : "red",
  background : "skyblue",
  "text-decoration" : "underline"
});
```

object 설정의 형태



DOM 처리 지원

❖ Attribute API

● attr(name [, value]), attr(obj)

- ◆ 지정된 속성명에 해당하는 속성 값의 반환 또는 속성 설정
- ◆ 활용 예

```
var $link = $("a");
console.log($link.attr("href"));
$link.attr("href", "http://jquery.com");
```

● removeAttr(name)

- ◆ 지정된 속성명에 해당하는 속성을 제거
- ◆ 활용 예

```
$linkremoveAttr("title");
```

DOM 처리 지원

❖ Attribute API

● prop()

◆ 속성을 다룬다는 측면에서 attr과 용법은 동일하지만 용도가 다름

attr()	prop()
HTML의 속성(attribute)을 취급	JavaScript의 프로퍼티(property)를 취급
속성은 HTML Element에 있는 정보 → 고정적	프로퍼티는 JavaScript에서 사용하는 정보 (HTML의 정보와 일치 또는 불일치) → 동적

```
<input type="radio" name="gender" value="male" id="male" checked>
```

```
<a href="ex02_class.html" title="jquery">
```

```
$a = $("a");
console.log($a.attr("href"));
console.log($a.prop("href"));

$radio = $(":radio");
console.log($radio.attr("checked"));
console.log($radio.prop("checked"));
```

ex02_class.html

<http://localhost:9090/jQuery/ch07/>

checked
true

check 해지 시

checked
false

109

DOM 처리 지원

❖ Attribute API

● addClass(className)

◆ 선택된 요소에 className에 해당하는 class를 추가하는 함수

- 미리 템플릿처럼 다양한 class를 작성해 놓은 후 이벤트 상황에 따라 class 적용

◆ 활용 예

```
var $h1 = $("h1");
$h1.addClass("blueClass");
```

● removeClass([className])

◆ 선택된 요소에서 className에 해당하는 class를 제거하는 함수

- className 생략 시 모든 class 삭제

● toggleClass(className)

◆ 선택된 요소에 className에 해당하는 class가 있으면 삭제, 없으면 추가하는 함수

● hasClass(className)

◆ 선택된 요소에 className 속성 존재 여부를 true / false로 리턴하는 함수

● 위 기능들은 attr("class")를 가지고 조절할 수 있지만

◆ 너무 빈번히 사용되므로 전용의 함수들을 가짐

DOM 처리 지원

❖ Manipulation API

- DOM 요소의 추가, 삭제, 수정, 복사 등과 관련된 API
 - ◆ 동적인 HTML 화면을 손쉽게 구현 가능
- 새로운 DOM 요소 생성
 - ◆ \$의 파라미터로 생성하려는 요소 입력
 - 아직 부모가 없는 상태이므로 다른 부모 태그에 추가해야 화면에 반영됨
 - ◆ 활용 예

```
var $h1 = $("<h1>");
```

가

11

DOM 처리 지원

❖ Manipulation API

- 형제 레벨의 요소 추가
- 선택요소.before(content)
 - ◆ 선택된 요소 앞에 content를 끼워 넣음
- 선택요소.after(content)
 - ◆ 선택된 요소 뒤에 content를 끼워 넣음
- content.insertBefore(선택요소)
 - ◆ content를 선택된 요소 앞에 끼워 넣음
- content.insertAfter(선택요소)
 - ◆ content를 선택된 요소 뒤에 끼워 넣음
- 기존 요소를 content로 이용하는 경우 기존 요소가 move
- 활용 예

```
var $h1 = $("h1");
$h1.before("<hr>");

$("<h3>앞에 끼워넣기</h3>").insertBefore($h1);
```

선택된 요소 기준

새로운 요소 기준

```
<hr>
<h3>앞에 끼워넣기</h3>
<h1>jQuery 특징</h1>
```

DOM 처리 지원

❖ Manipulation API

- 자식 레벨의 요소 추가
- 선택요소.append(content)
 - ◆ 선택된 요소의 막내 자식으로 content를 추가함
- 선택요소.prepend(content)
 - ◆ 선택된 요소의 첫째 자식으로 content를 추가함
- content.appendTo(선택요소)
 - ◆ content를 선택된 요소의 막내 자식으로 추가함
- content.prependTo(선택요소)
 - ◆ content를 선택된 요소의 첫째 자식으로 추가함
- 기존 요소를 content로 이용하는 경우 기존 요소가 move
- 활용 예

```
var $h1 = $("h1");
$h1.append("<h3>append</h3>");

$("<h3>appendTo</h3>").appendTo($h1);
```

선택된 요소 기준

새로운 요소 기준

The diagram illustrates the movement of an element between two `<h1>` tags. It shows the original structure with an `<h1>` tag containing the text "jQuery 특징". Inside this tag, there is another `<h1>` tag with the text "append" and a third `<h1>` tag with the text "appendTo". A large blue arrow points from the original position of the "append" tag to the position after the "jQuery 특징" text. To the right of the diagram, four colored boxes explain the steps:

- before: The original state where the "append" tag is a child of the inner `<h1>` tag.
- prepend: The state after using `prepend`, where the "append" tag is now the first child of the inner `<h1>` tag.
- append: The state after using `appendTo`, where the "appendTo" tag is now a child of the inner `<h1>` tag.
- after: The final state where the "append" tag has moved to the position immediately after the "jQuery 특징" text.

113

DOM 처리 지원

❖ Manipulation API

● wrap(content)

- ◆ 선택된 요소를 content로 감쌈
- ◆ 기존에 있는 요소를 이용해서 감쌀 경우 기존의 요소가 복제돼서 사용됨
- ◆ target이 현재 DOM tree에 있는 경우만 가능 – clone 결과물은 불가

● wrapAll(content)

- ◆ 선택한 요소들을 개별적으로 감싸지 않고 한꺼번에 감쌈
- ◆ 대상이 흩어져있는 경우 첫 번째 타겟으로 모두 이동 후 감쌈

● wrapInner(content)

- ◆ 선택한 요소의 자식 요소를 각각 감쌈

● unwrap()

- ◆ 선택된 요소를 감싸고 있는 부모를 제거함

● 활용 예

```
var $target = $("a");
$target.wrap("<div class='accent'>");
```

```
<div class="accent">
  <a href="https://jquery.com" title="jquery">관련 사이트 : jquery.com</a>
</div>
```

DOM 처리 지원

❖ Manipulation API

● target.replaceWith(content)

- ◆ target을 content로 대체함

- ◆ 만약 content가 기존 요소라면 기존 요소가 target 위치로 이동함

● content.replaceAll(target)

- ◆ replaceWith와 동일한 기능이며 단지 content와 target의 위치가 다름

● 활용 예

```
var $target = $("h1");
$target.replaceWith("<h2>javascript 라이브러리 jQuery");
```

● target.clone([true|false]);

- ◆ 선택된 요소를 복제

- 파라미터는 하위 요소까지 복제하는 옵션으로 default는 false

◆ 활용 예

```
var link1 = $("a").eq(0);
var link2 = $("a").eq(1).clone();
var ul = $("ul");
ul.append(link1); // 이동
ul.append(link2); // 복제
```

DOM 처리 지원

❖ Manipulation API

● empty()

- ◆ 선택된 요소의 하위 내용을 모두 삭제해서 빈(empty) 태그로 만듦

● remove([selector])

- ◆ 선택된 요소를 삭제함
- ◆ selector가 지정된 경우 해당 selector에 적합한 경우만 삭제(2차 필터)

● detach()

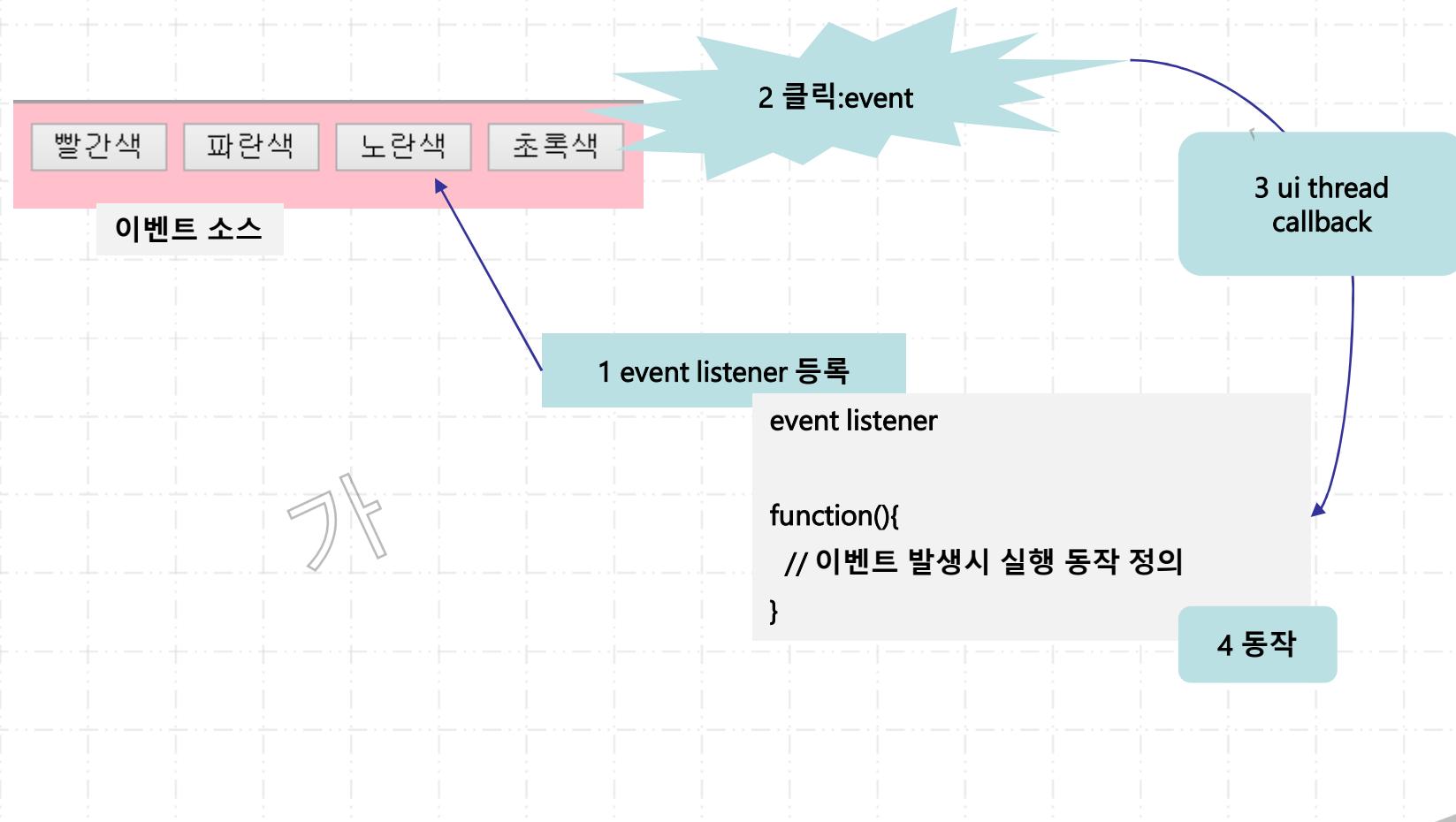
- ◆ 선택된 요소를 잘라내서 리턴함
- ◆ remove와 달리 잘라낸 내용을 저장 후 다시 붙이기 가능

가

이벤트

❖ 이벤트

- html 페이지에서 발생하는 모든 클라이언트의 행동
- 이벤트 발생시 작성한 자바 스크립트 함수가 동작



이벤트

❖ Vanilla 스크립트에서의 이벤트 처리

● 이벤트의 종류

	종류	설명
마우스 이벤트	mouseover	마우스가 지정한 요소 위로 올라갔을 때
	mouseout	마우스가 지정한 요소를 벗어났을 때 발생
	mousemove	마우스가 지정한 요소 영역에서 움직일 때 발생
	click / dblclick	마우스로 지정한 요소를 클릭했을 때 발생 / 더블클릭 했을 때 발생
키보드 이벤트	keypress	지정한 요소에서 키보드가 눌렸을 때 발생(shift + A로 하나의 이벤트 발생)
	keydown	지정한 요소에서 키보드가 눌렸을 때 발생(shift 한번, A 한번 두 번의 이벤트 발생)
	keyup	지정한 요소에서 눌린 키보드를 떼었을 때 발생
기타 이벤트	focus / blur	지정한 요소에 포커스가 갔을 때 / 포커스를 잃었을 때
	change	지정한 요소의 value 속성값이 바뀌고 포커스가 이동했을 때 발생
	load / unload	지정한 요소의 하위 요소를 모두 로딩했을 때. 처음은 물론 history back도 적용 / 문서를 닫거나 다른 문서로 이동했을 때
	submit / reset	폼 요소를 서버로 전송할 때 / 폼 요소의 취소 버튼을 클릭할 때
	resize	지정된 요소의 크기가 변경되었을 때
	error	문서 객체 로딩 동안 문제가 발생했을 때

이벤트

❖ Vanilla 스크립트에서의 이벤트 처리

● 요소에 직접 이벤트 등록(DOM 0)

```
<button id="btn1" onclick="changeColor('pink')">빨간색</button>

function changeColor(color, obj) {
    let bodyTag = document.querySelector("body");
    bodyTag.style.backgroundColor = color;
}
```

◆ 가장 간편한 방법이지만 유지 보수가 어려움

● DOM을 이용한 이벤트 등록(DOM 1)

```
let btn2 = document.getElementById("btn2");

btn2.onclick = function() {
    changeColor("skyblue");
};
```

● DOM level2의 표준 이벤트 모델

```
let btn3 = document.getElementById("btn3");

btn3.addEventListener("click", function() {
    changeColor("yellow");
});
```

이벤트 처리 지원

❖ jQuery 이벤트

- 일반적인 java script의 모든 이벤트 처리 가능
- 추가적인 jQuery 이벤트 제공

❖ 주요 이벤트

window event	mouse event	focus event	key event	form, input event
load	click	focus	keydown	change
ready	dblclick	focusin	keyup	select
unload	mouseout	focusout		submit
resize	mouseover	blur		reset
scroll	hover			input
error	mousedown			
	mouseup			
	mouseenter			
	mouseleave			
	mousemove			

120

이벤트 처리 지원

❖ 이벤트 연결

● event 이름으로 등록

- ◆ `$(document).ready(function(e){ })`;

● on()

- ◆ 현재 DOM 요소에 등록 : `$(selector).on(event_name, [data,] event_handler)`;

- ◆ data는 JSON형태이며 handler에 전달되는 event 객체의 data 를 통해서 접근 가능

```
$("#one").on("click", function() {  
    $eventzone.text("축 당첨");  
});
```

```
$("#one").on("click", {msg : "축 당첨"},writeHandler);  
  
function writeHandler(e) {  
    console.log(e.data.msg);  
}
```

● one(): 1회성 이벤트 등록

- ◆ `$(selector).one(event_name, [data,], event_handler)`;

이벤트 처리 지원

❖ 이벤트 제거

- **off()** 이용

- ◆ **on**으로 등록된 이벤트 제거
- ◆ 현재 DOM 요소의 이벤트 제거: **\$(selector).off(event_name [, event_handler]);**

가

122

이벤트 처리 지원

❖ 동일한 이벤트 소스에 여러 가지 이벤트 리스너 등록 및 삭제

- 객체를 통해 이벤트 종류와 콜백 등록

```
$eventzone.on({  
    mouseenter : function() {},  
    mouseleave : function() {},  
    mousemove : function() {}  
});
```

- off() 함수의 return은 다시 jQuery 객체

```
$eventzone.off("mouseenter").off("mouseleave")  
  
$eventzone.off("mouseenter mouseleave");
```

가

123

이벤트 처리 지원

❖ 장래의 DOM 요소에 등록과 제거

- 장래의 요소는 처음 스크립트 실행 시점에 DOM에 나오지 않은 요소

- ◆ 기본적인 on, off의 이벤트 적용 대상이 아님

- 특정 selector(상위 요소) 기준

- ◆ `$(selector).on(event_name, selector, [data,] event_handler);`

- 처음 selector는 적용할 범위
 - 두 번째 selector는 실제 적용할 요소

- ◆ `$(selector).one(event_name, selector, [data,] event_handler);`

- ◆ `$(selector).off(event_name, selector [, event_handler]);`

```
$(“body”).on(“click”, “button”, function() {});
```

- document 기준

- ◆ `$(document).on(event_name, selector, [data,] event_handler);`

- ◆ `$(document).one(event_name, selector, [data,] event_handler);`

- ◆ `$(document).off(event_name, selector [, event_handler]);`

```
$(document).on(“click”, “button”, function() {});
```

이벤트 처리 지원

❖ 이벤트 강제 발생

- 이벤트 발생 시 실행될 함수나 on과 연결된 이벤트 핸들러를 강제적으로 실행
- `$(selector).trigger(event_name)`
- `$(selector).trigger(event_name, data);`

가

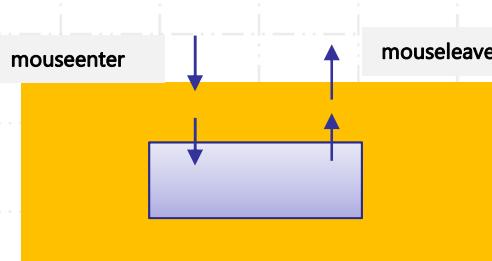
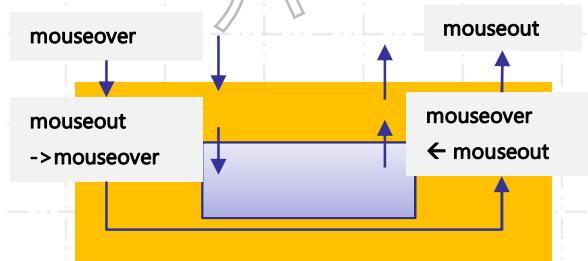
125

이벤트 처리 지원

❖ mouse 이벤트

● 마우스 사용으로 발생하는 이벤트

이벤트 명	설명
click	선택한 요소를 클릭했을 경우
dblclick	선택한 요소를 더블 클릭했을 경우
mousedown	마우스 버튼을 눌렀을 때
mouseup	마우스 버튼을 눌렀다 떼었을 때
mouseenter	선택한 요소 범위 내에 마우스가 들어갔을 때
mouseleave	선택한 요소 범위에서 마우스가 벗어났을 때
mouseover	선택한 요소 위에 마우스가 위치할 때
mouseout	선택한 요소가 보이는 영역에서 마우스가 벗어났을 때
mousemove	마우스가 움직일 때
hover	mouseenter+ mouseleave



126

이벤트 처리 지원

❖ 이벤트 객체의 활용

- 이벤트 객체의 레퍼런스를 통해 이벤트의 상세 내용(마우스 클릭 좌표 등) 파악
- 이벤트 객체의 생성
 - ◆ call back 메서드에 파라미터 선언
 - ◆ runtime에 브라우저가 호출 시 객체 생성 후 할당
- 이벤트 종류에 따라 사용할 수 있는 데이터가 다름

종류	설명
마우스이벤트	offsetX, offsetY 현재 요소를 기준으로 마우스 이벤트가 발생한 x, y 좌표
	clientX, clientY 브라우저 기준으로 마우스 이벤트가 발생한 x, y 좌표(스크롤 바 이동 너비 무시)
	pageX, pageY 브라우저 기준으로 마우스 이벤트가 발생한 x, y 좌표(스크롤 바 이동 너비 포함)
	screenX, screenY 화면 모니터 기준으로 이벤트가 발생한 x, y 좌표
키보드이벤트	keyCode 눌린 키의 유니코드 값
	altKey, ctrlKey, shiftKey 각각 alt, ctrl, shift 키가 눌렸으면 true
전체	target 이벤트가 발생한 요소(this와 동일)

이벤트 처리 지원

❖ keyboard 이벤트

● 키보드 사용에서 발생하는 이벤트

이벤트 명	설명
keydown	선택한 요소에서 키보드가 눌렸을 때 발생하는 이벤트로 문자키를 포함한 키 코드 반환 모든 키에 대해 발생
keyup	선택한 요소에서 키보드에서 손을 떼었을 때 발생

```
$("#user_id").on("keydown", function(e) {  
    console.log("key down : "+e.keyCode);  
    printRemained(this);  
});  
  
$("#user_pass").on("keyup", function(e) {  
    console.log("key up : "+e.keyCode);  
    printRemained(this);  
});  
  
function printRemained(x){  
    var length = $(x).val().length;  
    $("#remained").text(3-length);  
}
```

keydown은 아직 입력 완료 상태가 아님!

keyCode로 어떤 한글이 입력되었는지는 파악되지 않음
: 모두 229

이벤트 처리 지원

❖ 이벤트 listener에서의 this

● 실행문 내에서의 this

◆ 이벤트가 발생한 이벤트 소스를 나타냄

```
$("#ssn1").on("keyup", function() {  
    if ($(this).val().length === 6) {  
        $("#ssn2").focus();  
    }  
});
```

가

이벤트 처리 지원

❖ window 이벤트

● ready(function)

- ◆ HTML의 모든 DOM 요소들이 로딩될 때 호출
- ◆ window의 load 이벤트와의 차이점

`window.onload`

외부 리소스 및 이미지를 포함한 모든 요소 로딩 완료

- 이미지가 크거나 로딩이 느릴 때 JS 반응도 느림
- 이미지의 실제 크기 등 정보 활용 가능

여러 개를 지정하면 마지막 하나만 수행

`$(document).ready()`

외부 리소스 및 이미지는 상관 없이 DOM 요소만 로딩

- 빠른 반응
- 이미지의 실제 크기 등 정보 활용 불가

여러 번 등록해도 순차적으로 수행

가

130

조용준(stgray22@gmail.com)

이벤트 처리 지원

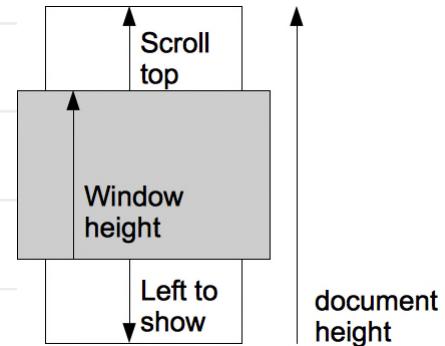
❖ window 이벤트

● document 또는 window 객체에서 발생되는 이벤트

소속	이벤트 명	설명
frame 요소	load	이미지 또는 프레임 요소에 연동된 소스가 로딩이 완료된 후 이벤트 발생
document	ready	지정한 HTML 문서 객체가 로딩이 완료된 후 이벤트 발생
window	resize	윈도우의 크기를 변화시켰을 때 발생
window	scroll	페이지를 스크롤 할 때 발생
window	error	페이지에 에러가 있을 때 발생 - 오류 내용은 이벤트 객체의 originalEvent를 통해서 확인 가능

● scroll 이벤트를 통한 무한 스크롤 구현

```
var scrollHeight = $(window).scrollTop() + $(window).height();
var documentHeight = $(document).height();
// 20은 오차 보정용 숫자
if(scrollHeight + 20 >= documentHeight){
    $("body").append($("#if").clone(true));
}
```



즉 문서(document height)가 작아지면 append 해줌

이벤트 처리 지원

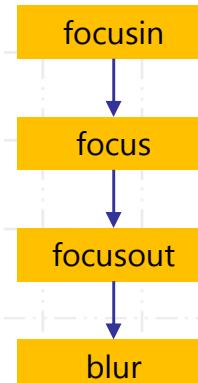
❖ focus 이벤트

- 마우스로 <a> 또는 input 요소를 클릭했을 때 발생
 - ◆ 마우스 없이 실행했을 경우는 tab 키로 요소를 순환할 경우 발생

이벤트 명	설명
focus	선택한 요소에 포커스가 있을 경우 이벤트를 발생시키거나 선택한 요소에 강제로 포커스 생성
focusin	선택한 요소에 포커스를 얻기 직전에 발생
focusout	선택한 요소에서 포커스를 잃기 직전에 발생
blur	선택한 요소에서 포커스가 이동될 경우 이벤트를 발생시키거나 강제로 포커스를 삭제

```
$("#user_id").on({
  focus:function(){
    $(this).css("background", "orange");
  },
  blur:function(){
    $(this).css("background", "white");
  }
});

$("#focusB").on("click", function(){
  $("#user_id").focus();
});
```



이벤트 처리 지원

❖ focus 이벤트

● 키보드 접근성을 위한 이벤트 등록

- ◆ 마우스 이벤트만을 등록했을 경우 키보드만을 사용하는 사용자는 매우 불편
- ◆ focus 이벤트를 통해 최소한의 편의성 보장
- ◆ <a>, <input>, <select>, <textarea> 등 사용자의 조작이 들어가는 태그

마우스 이벤트	마우스 이벤트에 대응하는 키보드 이벤트
mouseover or mouseenter	focus
mouseout or mouseleave	blur

◆ on method 적용 시 여러 이벤트 이름에 binding

```
$("#eventB").on("mouseleave blur", function(){
    $(this).css("background", "white");
});

$("#eventB").on({
    "mouseenter focus":function(){
        $(this).css("background", "orange");
    },
    "mouseleave blur":function(){
        $(this).css("background", "white");
    }
});
```

가

133

이벤트 처리 지원

❖ form-input 이벤트

● form 또는 input 요소에서 발생하는 이벤트

이벤트 명	설명
change	선택한 입력 양식의 값이 변경되고 포커스가 다른 요소로 이동 되었을 경우 발생
submit	submit 버튼을 클릭할 때
reset	reset 버튼을 클릭할 때

```
$("#my_form").on({
  submit:function(e){
    var userId = $("#user_id").val();
    console.log(userId);
    if(userId.length==0){
      alert("아이디는 필수 입력입니다.");
      $("#user_id").focus();
      //e.preventDefault();
      return false;
    }
  },
  reset:function(){
    alert("폼을 리셋합니다.");
  }
});
```



이벤트 처리 지원

❖ 이벤트의 진행 제어

● event.preventDefault()

- ◆ element가 갖는 기본 액션을 막음
- ◆ 주로 validation 실패 시 form 전송 방지에 사용됨
- ◆ 또는 이벤트 handler 내부에서 return false 처리

● event.stopPropagation()

- ◆ 이벤트가 여러 요소에 겹쳐서 발생 시 다른 요소에 이벤트의 전파를 막음



135

시각적 효과 지원

❖jQuery가 제공하는 시각적 효과 메서드

● 내부적으로 css 처리를 jQuery 가 진행함

종류	설명
.hide([duration] [,easing] [,callback])	선택한 요소를 숨김
.show([duration] [,easing] [,callback])	선택한 요소를 노출시킴
.toggle([duration] [,easing] [,callback])	선택한 요소를 hide() 과 show() 를 토글 함
.slideDown([duration] [,callback])	선택한 요소를 밑으로 펼쳐지며 노출시킴
.slideUp([duration] [,callback])	선택한 요소를 위로 접으며 숨김
.slideToggle([duration] [,callback])	선택한 요소를 slideDown()과 slideUp()을 토글
.fadeIn([duration] [,easing] [,callback])	선택한 요소를 투명도를 조절하며 나타남
.fadeOut([duration] [,easing] [,callback])	선택한 요소를 투명도를 조절하며 숨김
.fadeToggle([duration] [,easing] [,callback])	선택한 요소를 fadeIn()과 fadeOut() 을 토글
.fadeTo(duration, opacity [,easing] [,callback])	선택한 요소를 지정한 투명도까지 숨김

136

조용준(stgray22@gmail.com)

시각적 효과 지원

❖jQuery가 제공하는 시각적 효과 메서드

● parameter

- ◆ duration: 효과가 진행할 속도로 ms 단위의 숫자 또는 slow | normal | fast
- ◆ easing : animation의 easing 속성으로 별도의 jquery-ui 플러그인이 없으면 linear 또는 swing 만 지원(<http://api.jqueryui.com/easings/>)
 - CDN 방식으로 스크립트 로딩 후 \$.easing 객체에서 easing 정보 조회

```
<script src="https://code.jquery.com/ui/1.12.0/jquery-ui.js"></script>
// easing 속성 설정
function setupEasing() {
    $("#easings").empty();
    $.each($.easing, function(key) {
        $("#easings").append("<option>" + key + "</option>");
    });
}
```

◆ callback : 효과를 완료한 후 실행할 function

```
$("#btnHide").on("click", function() {
    target.hide(1000 * 10, "swing", function() {
        console.log("click");
        alert("이제 안보이죠?");
    });
});
```

시각적 효과 지원

❖ 사용자 지정 효과

● animate 메서드 활용

- ◆ `$(selector).animate(options [, duration] [, easing] [, callback]);`
- ◆ options에는 animation의 속성이 객체 형태로 저장

- height, width, top, bottom, left, right, margin, padding, opacity, scrollTop

```
$document.ready(function() {
    $("div").each(function(index, item) {
        $(item).animate({
            height : "-=10",
            left : Math.random() * 500,
            top : Math.random() * 500
        }, 1000 * 10, "swing", function() {
            console.log("이동 완료!");
        });
    });
});

// 아이템이 클릭되었을 때 특정 위치로 스크롤
$(document).on("click", "#mytable .link", function(){
    let targetNum = $(this).attr("data-to");
    let position = $("#mytable").find("#"+targetNum).offset().top; // offset: document 내에서의 절대 좌표
    $("html, body").animate({
        scrollTop: position
    }, 250); /* 250 is animation speed */
});
```

138

시각적 효과 지원

❖ 애니메이션 제어를 위한 메서드

종류	설명
stop()	현재 진행중인 효과를 정지시킴
delay()	일정 시간 동안 기다렸다가 효과를 진행함
queue()	큐에 대기중인 효과의 목록을 리턴하거나 추가시킴
clearQueue()	실행중인 효과는 빼고 큐에 대기중인 이벤트의 목록을 지움
dequeue()	실행중인 효과를 포함해서 큐에 대기중인 이벤트의 목록을 지움
finish()	진행중인 효과를 강제로 완료 시점으로 이동시킨 후 종료

가

139

BOM

가

조용준
(stgray22@gmail.com)

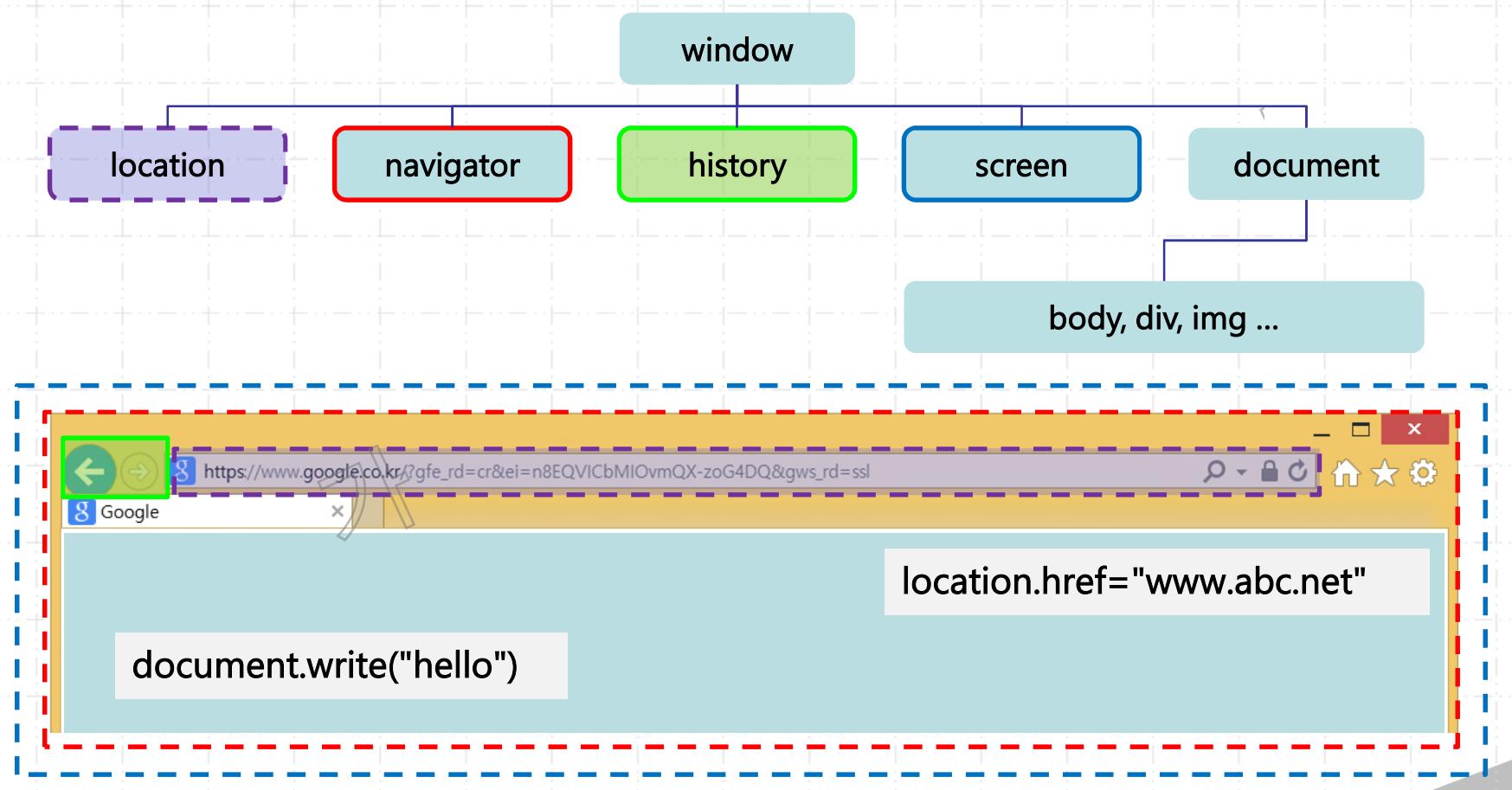


BOM

❖ BOM(Browser Object Model)

● 브라우저에 내장된 객체

◆ window - 브라우저 객체의 최 상위 객체



141

조용준(stgray22@gmail.com)

BOM

❖ window 객체

● 브라우저 객체의 최 상위 객체

종류	설명
<pre>let child = window.open(strUrl, strWindowName, [strWindowFeatures]);</pre>	strUrl 사이트를 내부적으로 strWindowName이라는 이름, 옵션은 strWindowFeatures를 사용해서 열고 child 변수에 할당함 toolbar=no, top=0, left=0, width=400, height=400
window.close();	특정 창을 닫을 때(child와 같은 window 타입의 변수 이용)
window.alert(message);	message를 출력하는 경고창을 띄울 때
window.prompt(text, value);	text의 질의에 대한 응답 창을 띄울 때. value는 기본 값
window.confirm(message);	message에 대한 확인/취소 창을 띄울 때
<pre>let intervalID = window.setInterval(func, delay[, param1, param2, ...]);</pre>	주기적인 delay를 두고 func를 실행함. 이때 func에는 param1, param2와 같은 파라미터 전달 가능
window.clearInterval(intervalID)	setInterval로 설정된 반복 작업을 중지함
<pre>let timeoutID = window.setTimeout(func, [delay, param1, param2, ...]);</pre>	delay(기본은 0) 이후 func를 실행함. 이때 func에는 param1, param2와 같이 파라미터 전달 가능
window.clearTimeout(timeoutID)	setTimeout으로 설정된 작업을 중지함

142

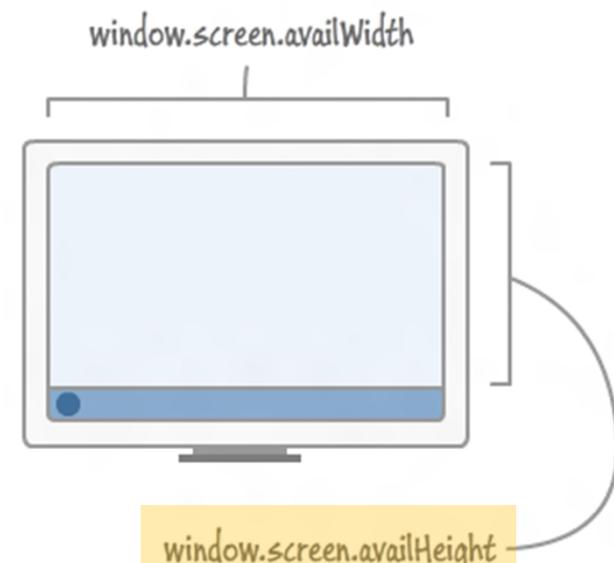
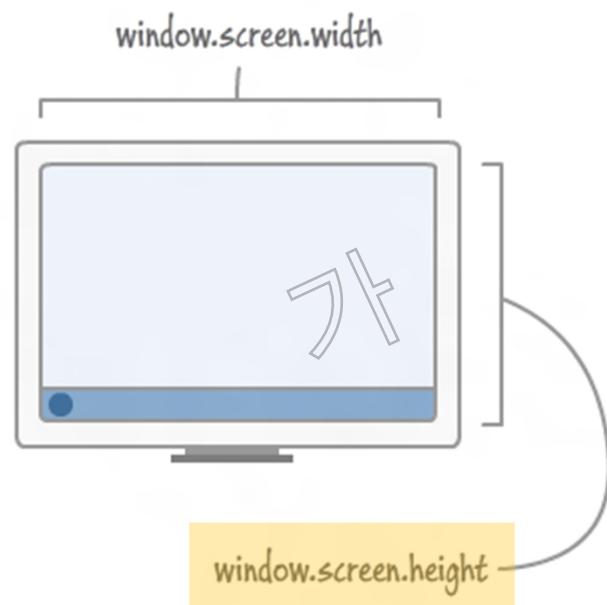
조용준(stgray22@gmail.com)

BOM

❖ screen 객체

● 사용자의 모니터 정보를 제공하는 객체

속성	내용	속성	내용
screen.width	화면의 가로	screen.height	화면의 세로
screen.availWidth	작업 표시줄을 제외한 화면의 가로	screen.availHeight	작업표시줄을 제외한 화면의 세로
screen.colorDepth	모니터의 표현 가능한 컬러 bit		



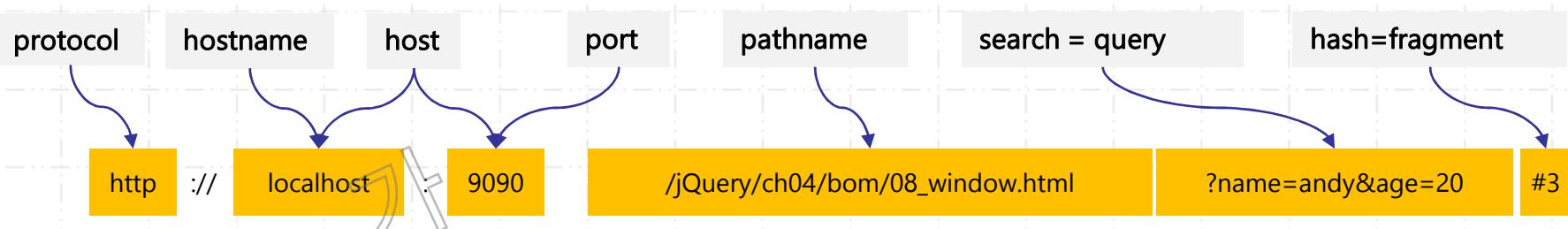
143

BOM

❖ location 객체

- 사용자 브라우저 주소창의 url에 대한 정보와 새로 고침 기능을 제공하는 객체

속성	내용	속성	내용
location.href	주소 영역에 주소 설정 또는 url 반환	location.replace(url)	Url로 현재 페이지를 이동
location.hostname	URL의 hostname 설정 또는 반환	location.host	URL의 호스트 이름과 포트 반환
location.port	URL의 포트 번호 반환	location.protocol	URL의 프로토콜 반환
location.search	URL의 쿼리 스트링 반환	location.reload()	새로 고침



◆ hash 기호

- 몇 번째라는 뜻

◆ reload() : 페이지 새로 고침

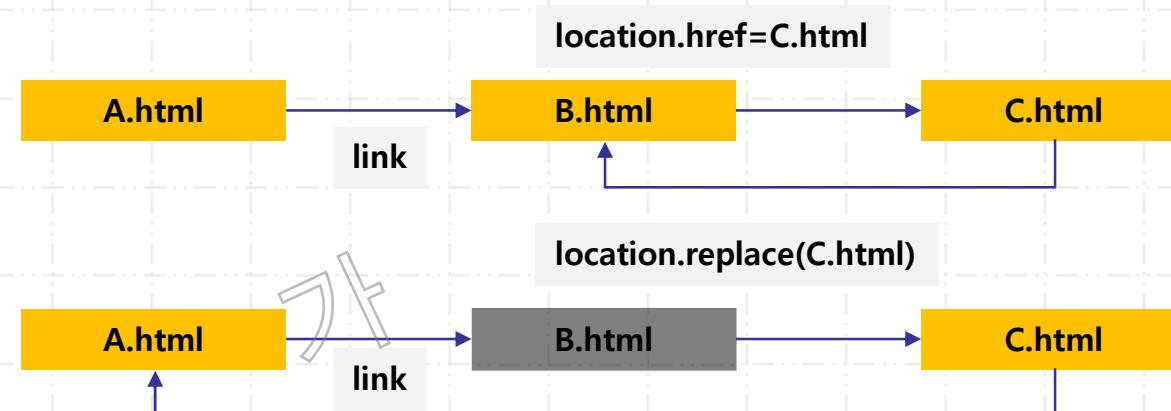
144

BOM

❖ location 객체

● location.href vs location.replace()

	location.href	location.replace()
기능	새로운 페이지로의 이동	새로운 페이지로 대체(덮어 씀)
형태	속성	기능
히스토리 기록 여부	기록됨	기록되지 않음



- href는 일반적인 페이지 이동 시 이용
- replace는 이전 페이지로의 접근의 금지가 필요한 경우(단계별 회원 가입 등)

145

BOM

❖ history 객체

- 사용자가 방문한 곳에 대한 목록을 제공하여 뒤로 가기 기능 활용 가능

속성	내용	속성	내용
history.back()	이전 페이지로 이동	history.forward()	다음 방문한 페이지로 이동
history.go(숫자)	-2 → 2단 계 전의 페이지	history.length	방문 기록에 저장된 목록의 개수

가

BOM

❖ navigator 객체

● 브라우저와 운영체제의 정보 제공

속성	내용	속성	내용
navigator.appCodeName	브라우저 코드명	navigator.appName	브라우저 이름
navigator.appVersion	브라우저 버전	navigator.language	브라우저 사용 언어
navigator.product	사용 엔진 이름	navigator.platform	사용자 OS
navigator.userAgent	운영체제 정보		

- 접속PC정보: [Windows64,Win64][Chrome,66.0.3359.139] Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36
- 수동설치 후에는 반드시 새로고침을 하거나 다시 접속하시기 바랍니다.
- 설치완료 메시지가 반복적으로 나오는 경우는 브라우저 종료 및 해당프로그램 삭제 후 재설치 하시기 바랍니다.

BOM

❖ geolocation api 활용

- BOM 객체 중 하나인 navigator 객체로부터 geolocation 객체 획득

- ◆ navigator.geolocation

- ◆ geolocation 제공 함수

- getCurrentPosition(successCallback, errorCallback, options)
요청 시 1회 위치 정보 요청
 - watchPosition(successCallback, errorCallback, options)
위치가 변경되면 계속해서 새로운 위치 정보 확인
 - clearWatch(id)
watchPosition의 리턴값을 id로 저장 후 이를 인자로 모니터링을 중단함

- ◆ 위치 정보의 종류(successCallback의 파라미터로 전달됨)

속성	설명	속성	설명
coords.latitude	위도	coords.longitude	경도
coords.accuracy	실제와의 가능 오차(m)	timestamp	위치 정보를 얻은 시각

148

조용준(stgray22@gmail.com)

❖ geolocation api 활용

● options 객체

◆ 위치 정보를 얻기 위한 옵션 지정

속성	설명
enableHighAccuracy	<ul style="list-style-type: none">정확도가 높은 위치 정보 요청 (default false)옵션 사용 시 시간이나 소모 전력이 높아지므로 주의 필요
timeout	<ul style="list-style-type: none">위치 정보 확인에 대한 시간 제한 설정(ms)기본 값은 infinity로 시간 제한 없음
maximumAge	<ul style="list-style-type: none">위치 정보의 유효 기간 설정 – 캐싱된 위치 정보의 유효 기간 (ms)기존에 가져왔던 값이 maximumAge 보다 오래된 값이면 새로 값을 요청기본 값은 언제나 0으로 즉시 요청함

가

BOM

❖ geolocation api 활용

● watchPosition

- ◆ 네비게이션 처럼 사용자의 위치 이동을 감지해야 할 경우 필요
- ◆ 함수의 사용법은 `getCurrentPosition`과 동일
- ◆ 차이점은 `clearWatch`가 실행되기 전까지 위치가 변경되면 정보 갱신

```
$("#stop").on("click", function() {
    navigator.geolocation.clearWatch(watchid);
});

$("#getLocation").on("click", function() {
    watchid = navigator.geolocation.watchPosition(function(p) {
        // do something
    })
})
```

가

150

Ajax

가

조용준
(stgray22@gmail.com)



Ajax

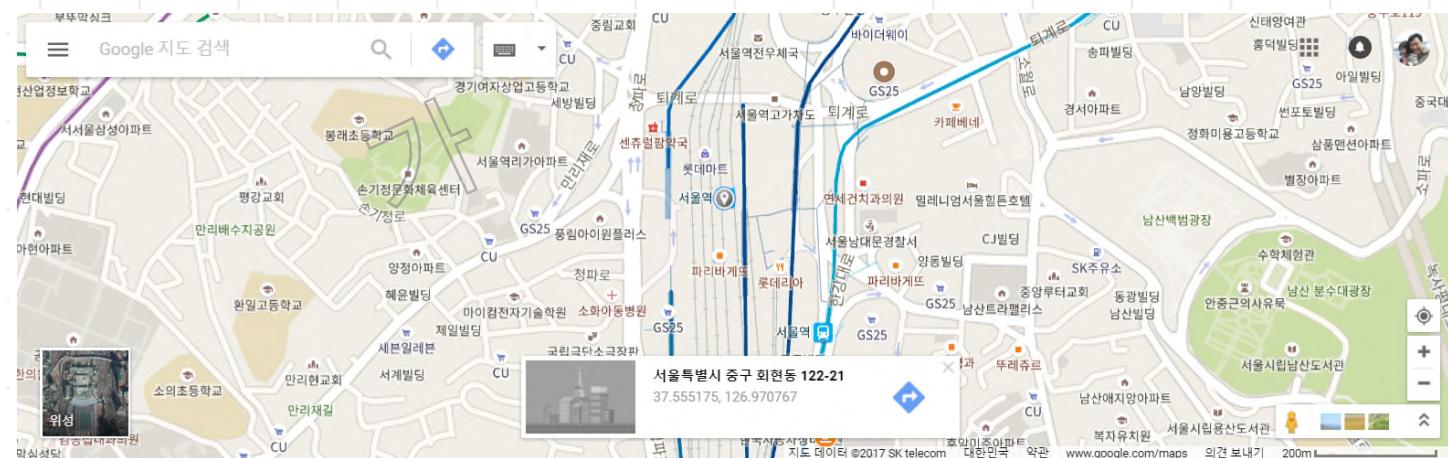
❖ Ajax?

● Asynchronous JavaScript and XML

- ◆ 비동기로 처리되는 JavaScript와 XML (초창기는 XML을 데이터 교환 수단으로 사용함)
- ◆ 동기 : 서버로 요청을 보내고 응답을 받아서 처리하는 방식
- ◆ 비동기 : 응답 상태와 상관 없이 다음 동작 수행

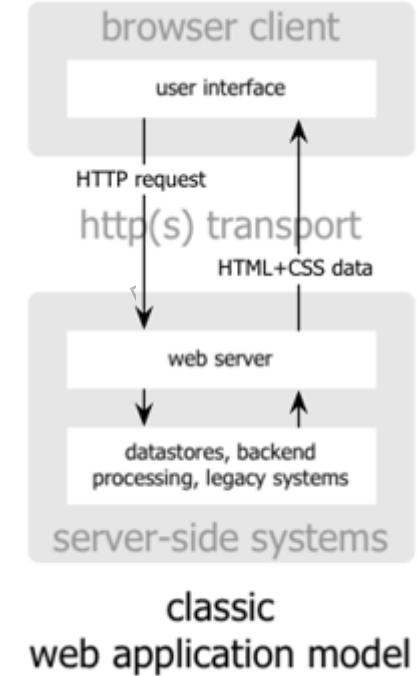
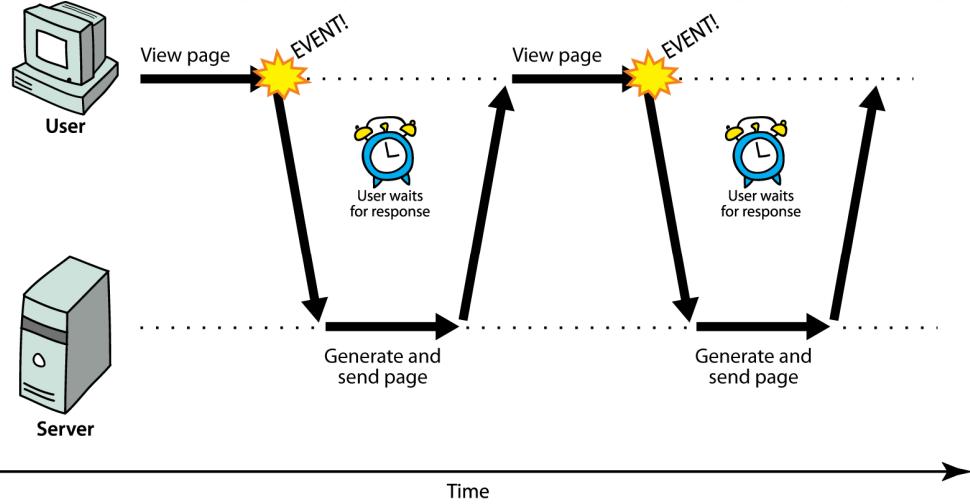
● 주 사용처

- ◆ 화면 전환 없이 클라이언트와 서버간 XML, JSON, HTML 등 정보 교환
 - 댓글을 달 때 페이지 전환 없이 바로 작성
 - 구글 실시간 검색, 구글 맵 등



Ajax

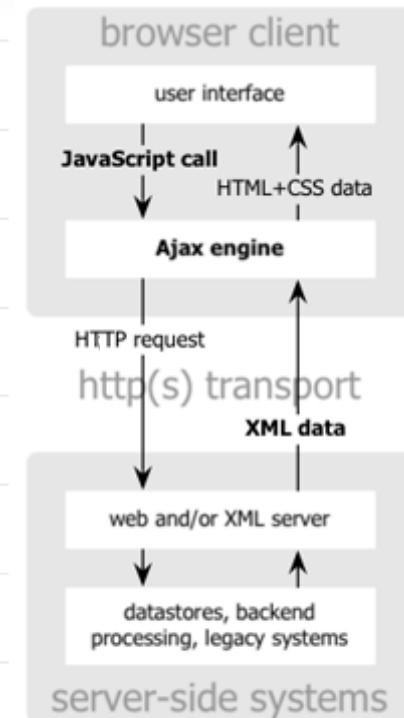
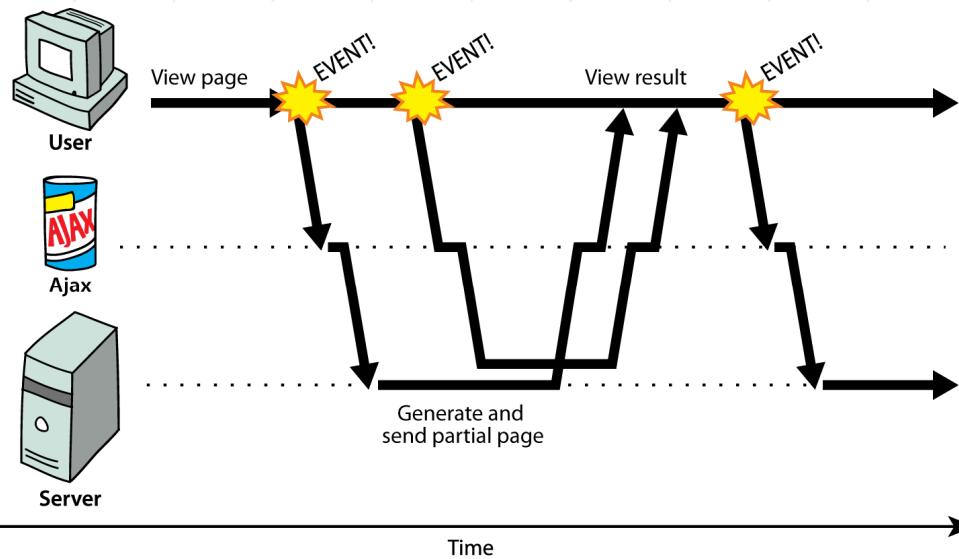
❖ Synchronous



- 품을 채우고 제출 하면 웹 서버로 요청 전달
- 서버에서 요청에 따라 새로운 웹 페이지 생성 → html 형태로 회신
- 동일한 화면이더라도 전체를 새로 그린다.
- 요청을 날린 후 응답이 오기 전까지 클라이언트는 대기 상태에 빠짐

Ajax

❖ Asynchronous

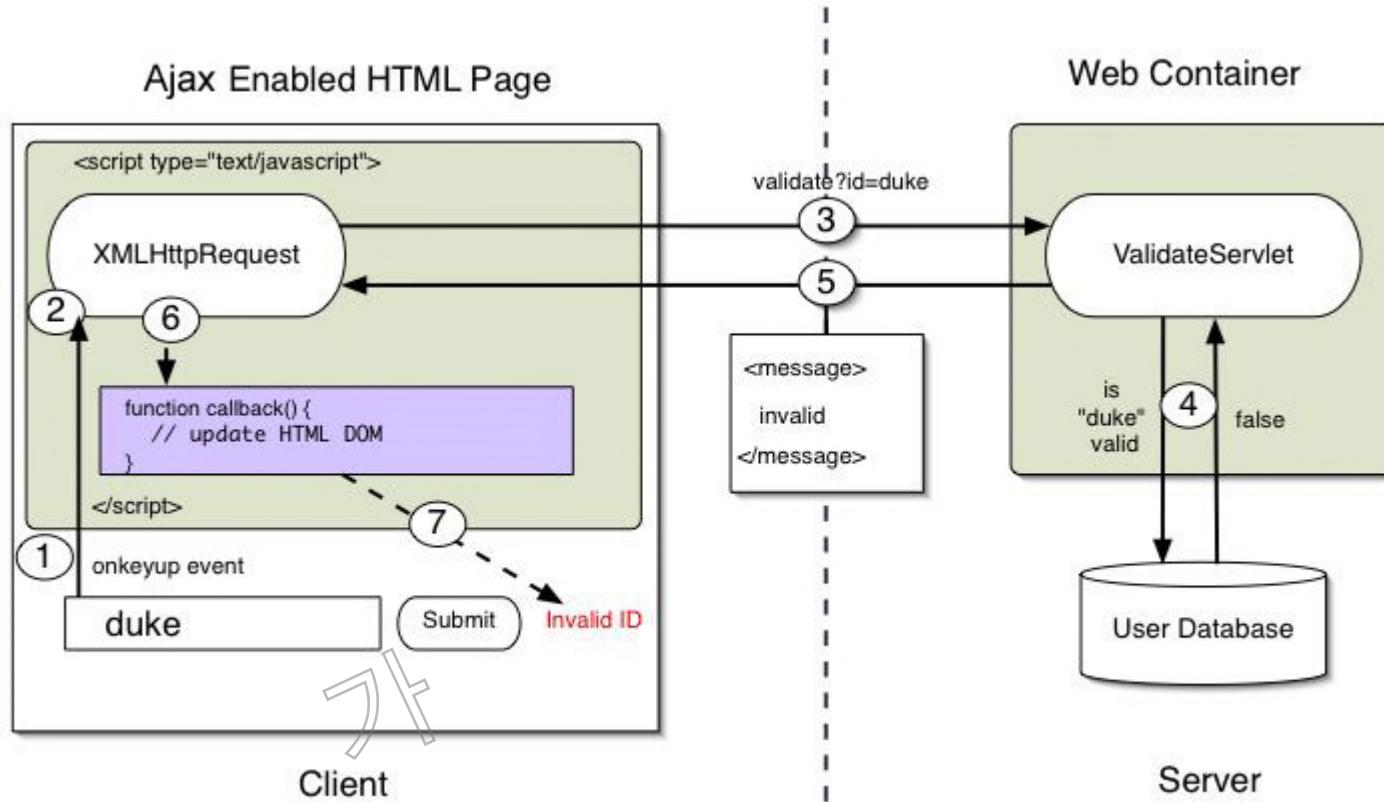


- 필요한 데이터만 XMLHttpRequest 형태로 전달
- 클라이언트에게 필요한 데이터 조각만 회신
- 전송되는 데이터의 양이 줄어들어 효율적
- 요청을 기다리지 않고 계속해서 다른 작업 수행 가능

Ajax
web application model

Ajax

❖ Asynchronous



155

조용준(stgray22@gmail.com)

Ajax

❖ XMLHttpRequest의 일반 메서드와 속성

● 요청과 관련된 메서드

메서드/속성	설명
XMLHttpRequest()	생성자
open(method, url, async)	method : 전송 방식 get post url : 요청을 처리할 스크립트의 위치 async : 비동기 여부 – 언제나 true <code>xhr.open("GET", "/xxservlet?name=andy&age=20", true);</code> <code>xhr.open("POST", "/xxservlet", true);</code>
send(data)	실제로 요청하는 메서드 - get 방식은 data를 URL에 연결해서 전송 - post의 경우 send에 파라미터로 전송
abort()	전송을 취소시키는 메서드

가

● 응답과 관련된 속성

메서드/속성	설명
responseText	텍스트 형태로 요청에 대한 응답 리턴
responseXML	XML 문서 형태로 요청에 대한 응답 리턴

Ajax

❖ Ajax level2

● 단계적 응답 이벤트

이벤트	설명
loadstart	요청을 시작할 때 발생
progress	데이터를 전송하거나 로딩할 때 주기적(약 50ms)으로 발생 e.lengthComputable : 량이 체크 가능한지 파악, e.loaded : 로딩 량, e.total : 전체 량
error	요청 처리 중 에러가 일어날 때 발생
load	요청이 성공적으로 완료됐을 때 발생
timeout	지정된 시간 동안 요청을 완료하지 못했을 때 발생
loadend	이벤트가 성공/실패로 종료했을 때 발생
abort	실행을 중지시킴



157

Ajax

❖ JavaScript를 이용한 ajax 처리

```
$("#get").on("click", function() {
  let xhr = new XMLHttpRequest();
  xhr.open("get", "../../events?num1=100&num2=200", true);
  xhr.timeout = 1000 * 6;
  $(xhr).on({
    loadstart : function() {
      console.log("load start");
    },
    progress : function() {
      console.log("progress");
    },
    error : function() {
      console.log("error");
    },
    timeout : function() {
      console.log("timeout");
    },
    abort : function() {
      console.log("abort");
    },
    load : function() {
      console.log(xhr.responseText);
    },
    loadend : function() {
      console.log("loadend");
    }
  });
  xhr.send();
});
```

Ajax

❖ Ajax를 위한 jQuery method

메서드	설명
\$.ajax()	\$.post(), \$.get(), \$.getJSON()을 하나로 합쳐 놓은 가장 대표적 메서드
\$.get()	데이터를 get 방식으로 서버에 전송 후 응답을 받을 때 사용
\$.post()	데이터를 post 방식으로 서버에 전송 후 응답을 받을 때 사용
\$.getJSON()	데이터를 get 방식으로 서버에 전송 후 JSON형식으로 응답을 받을 때 사용
\$.getScript()	Ajax를 이용하여 외부 자바 스크립트를 호출 할 때 사용
serialize()	폼에 값을 전송할 때 name=key&name=key의 쿼리 스트링 형태로 변환해서 전송하는 메서드
serializeArray()	폼의 값을 전송할 때 JSON의 배열 형태로 변환해서 전송하는 메서드 [{key: value}, {key: value}]

용도에 따라
세분화

가

159

Ajax

❖ Ajax 메서드

● \$.ajax(settings):

◆ settings: JSON 형태로 필요한 property 설정

property	타입	설명
url	String	request를 전달할 서버의 url
type	String	get 또는 post로 기본값은 get
data	object, String, Array	서버로 전송할 데이터 object: { key:value, key:value} Array: [{ key:value, key:value}, { key:value, key:value}] String: URL 인코딩된 값
contentType	String or Boolean	서버로 전송할 데이터의 content-type 기본 값은 application/x-www-form-urlencoded; charset=UTF-8 false일 경우 어떤 contentType도 설정하지 않음
dataType	String	응답 받을 데이터의 타입 으로 xml, json, script, html, text 생략 시 MIME 타입에 맞춰 자동 설정
timeout	number	만료 시간 설정
processData	Boolean	data로 서버로 전송하는 값을 queryString 형태로 변환할 것인지 설정 기본은 true 이며 파일 업로드 등으로 불필요할 경우 false 설정
xhr	Function	ajax에서 사용되는 XMLHttpRequest를 리턴 받는 function XMLHttpRequest를 override 할 때 이용
headers	JSON	서버로 전송될 request header를 JSON 객체 형태로 작성

160

Ajax

❖ Ajax 메서드

● \$.ajax(settings):

◆ 관련 call back

property	타입	설명
beforeSend	function(xhr, settings)	서버로 요청을 전송하기 직전에 실행될 call back function
success	function(data, status, xhr)	전송이 성공했을 경우 호출될 call back function
error	function(xhr, status, error)	전송이 실패했을 경우 호출될 call back function. xhr에 오류 정보 포함
complete	function(xhr, textStatus)	작업 완료 시 호출될 call back function textStatus는 "success", "error", "abort" 등 문자열



Ajax

❖ Ajax 메서드

● \$.ajax() 처리 예시

```
$("#jsonget").on("click", function() {
    $.ajax({
        url:"info.json",
        success:function(data, status, xhr){
            for(let item of data.movie){
                $("#dataarea").append("<li>" + item.title + "</li>");
            }
        },
        error: function(xhr, status){
            console.log("에러 발생", xhr.status);
            console.log(status);
        }
    });
});
```

가

이미 JSON 데이터

```
{"movie": [
    {"title":"Terminator", "actor":"아놀드"},
    {"title":"관상", "actor":"송강호"}
]}
```

Ajax

❖ Ajax 메서드

● xml data 처리

```
<?xml version="1.0" encoding="UTF-8"?>
<movies>
  <movie>
    <title>terminator</title>
    <actor>아놀드</actor>
  </movie>
  <movie>
    <title>Rambo</title>
    <actor>실베스타</actor>
  </movie>
</movies>
```

가

```
$.ajax({
  url:"info.xml",
  success:function(data){
    var titles =$(data).find("title");
    if(titles.length > 0){
      titles.each(function(index, data){
        var title = $(data).text();
        printData(title);
      });
    }
  }
});
```

문서 로딩 후 DOM을 이용

Ajax

❖ Ajax 메서드

● 일반적인 get과 post의 처리

```
setInterval(function(){
  $.ajax({
    url: "../AJaxServlet",
    success: function(data){
      $("#reply").append($("<li>" + data + "</li>"));
    },
    type: "get"
  });
}, 1000*10);

$("#sendB").on("click", function() {
  $.ajax({
    type: "post",
    url: "../AJaxServlet",
    success: function(data){
      $("#reply").append($("<li>" + data + "</li>"));
    },
    data: {
      message: $("#message").val()
    }
  });
});
```

?message=내용의 query string으로 변경 후 전송

Ajax

❖ 파라미터 전달

● 일반 파라미터 형태로 전달하려는 경우

◆ 단순 객체, \$("#myForm").serialize() 또는 \$("#myForm").serializeArray() 함수 이용

- 단순 객체:
- serialize(): mid=admin&pass=1
- serializeArray():

```
data : {
    "mid" : $("#mid").val(),
    "pass" : $("#pass").val()
},
(2) [{...}, {...}] ⓘ
▶ 0: {name: "mid", value: "admin"}
▶ 1: {name: "pass", value: "1"}
```

◆ 데이터를 처리하는 서버에서는 개별 파라미터로 처리

```
Map<String, String[]> params = request.getParameterMap();
System.out.println("파라미터 목록-----");
for(String key: params.keySet()) {
    System.out.println("parameter 확인: "+key+" : "+Arrays.toString(params.get(key)));
}
```

파라미터 목록-----

parameter 확인: mid : [admin]
parameter 확인: pass : [1]

Ajax

❖ 파라미터 전달

● 클라이언트에서의 JSON 형태 자료 전달

```
$("#myForm").on("submit", function(e) {  
    e.preventDefault();  
    let data = {  
        "mid" : $("#mid").val(),  
        "pass" : $("#pass").val()  
    };  
    $.ajax({  
        url : "../../AJaxJSonServlet",  
        type : "post",  
        data : JSON.stringify(data),  
        contentType : "application/json",  
        success : function(responseTxt) {  
            dropHere.html(responseTxt.message);  
        },  
        error : function(xhr, statusTxt, errThrown) {  
            alert("요청 실패" + errThrown + ", " + statusTxt);  
        }  
    });  
});
```

전송할 데이터 객체 생성

data를 직렬화해서 보내는데 json 타입이다.!!

- application/x-www-form-urlencoded가 default

가

Ajax

❖ 파라미터 전달

● 서버단에서의 JSON 활용

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ... {  
    request.setCharacterEncoding("utf-8");  
    Reader reader = new BufferedReader( new InputStreamReader(request.getInputStream(), "utf-8"));  
    Gson gson = new Gson();  
    Person person = gson.fromJson(reader, Person.class);  
    System.out.println("person: "+person);
```

```
Map<String, String> result = new HashMap<>();  
if (person.getMid().equals("admin")) {  
    result.put("message", "OK");  
} else {  
    result.put("message", "Fail");  
}
```

클라이언트에서 자료 수신 형태

```
response.setContentType("application/json;charset=UTF-8");  
response.getWriter().write(gson.toJson(result));  
}
```

클라이언트에게 자료 전달 형태

```
class Person {  
    private String mid;  
    private String pass;  
  
    // getter / setter  
}
```

167

Ajax

❖ 공공 DB의 활용

- <https://www.data.go.kr>

◆ 공공 성격의 데이터들을 xml 또는 json 형태로 서비스하고 있음

한국관광공사 DB 활용 예시

❖ 상세정보는 공통정보, 소개정보, 반복정보, 이미지정보의 4개 API로 제공됩니다.

관광정보 목록



공통정보 조회



소개정보/반복정보 조회



대표이미지, 주소, 좌표, 전화, 개인정보 등을 제공합니다.

이미지정보 조회



각 관광타입의 휴무일, 안내정보 등의 소개정보를 제공합니다.
1:N구조의 반복적인 상세정보를 제공합니다.

Data는 공사에 존재하며 쿼리
를 통해 사용하는 형태

Ajax

❖ 공공 DB의 활용

● <https://www.data.go.kr>

◆ 공공 성격의 데이터들을 xml 또는 json 형태로 서비스하고 있음

- 한국관광공사 > 국내관광정보 서비스 > 국문관광정보서비스 > 지역기반 관광정보 조회

1) REST방식의 URL 요청 예시

응답 표준은 XML이며, JSON을 요청할 경우 "&_type=json" 을 추가하여 요청합니다.

- Json 요청 :

`http://api.visitkorea.or.kr/openapi/service/rest/KorService/areaCode?ServiceKey=ServiceKey& numOfRows=10&pageNo=1&_type=json`

나) 요청 메시지 (Request Parameter)

항목명	항목 설명	기본값	필수 항목	상세 내용
numOfRows	한 페이지 결과 수	10		한 페이지 결과 수
pageNo	페이지 번호	1		현재 페이지 번호
arrange	정렬 구분	A		(A=제목순, B=조회순, C=수정일순, D=생성일순)

query string 형태로 요청하면

다) 응답 메시지 (Response Message)

항목명	항목 설명	상세 내용
resultCode	결과코드	응답 결과코드
resultMsg	결과메시지	응답 결과메시지
numOfRows	한 페이지 결과 수	한 페이지 결과 수
pageNo	페이지 번호	현재 페이지 번호
totalCount	전체 결과 수	전체 결과 수
addr1	주소	주소(예, 서울 종구 다동)를 응답
addr2	상세주소	상세주소

JSON(xml) 형태로 리턴

Ajax

❖ 공공 DB의 활용

```
$ajax({
    url : myUrl,
    dataType : "json",
    success : function(responseTxt, statusTxt, xhr) {
        var obj = responseTxt;
        console.log(obj);
        var itemarr = responseTxt["response"]["body"]["items"]["item"];
        var itemarr = responseTxt.response.body.items.item;
        $.each(itemarr, function(index, item) {
            dropHere.append("주소1: " + item.addr1 +
                            " , 주소2: " + item.addr2 + " : <img src='" + item.firstimage + "'><br>")
        });
    },
    error : function(xhr, statusTxt, errThrown) {
        alert(errThrown);
    }
});
```

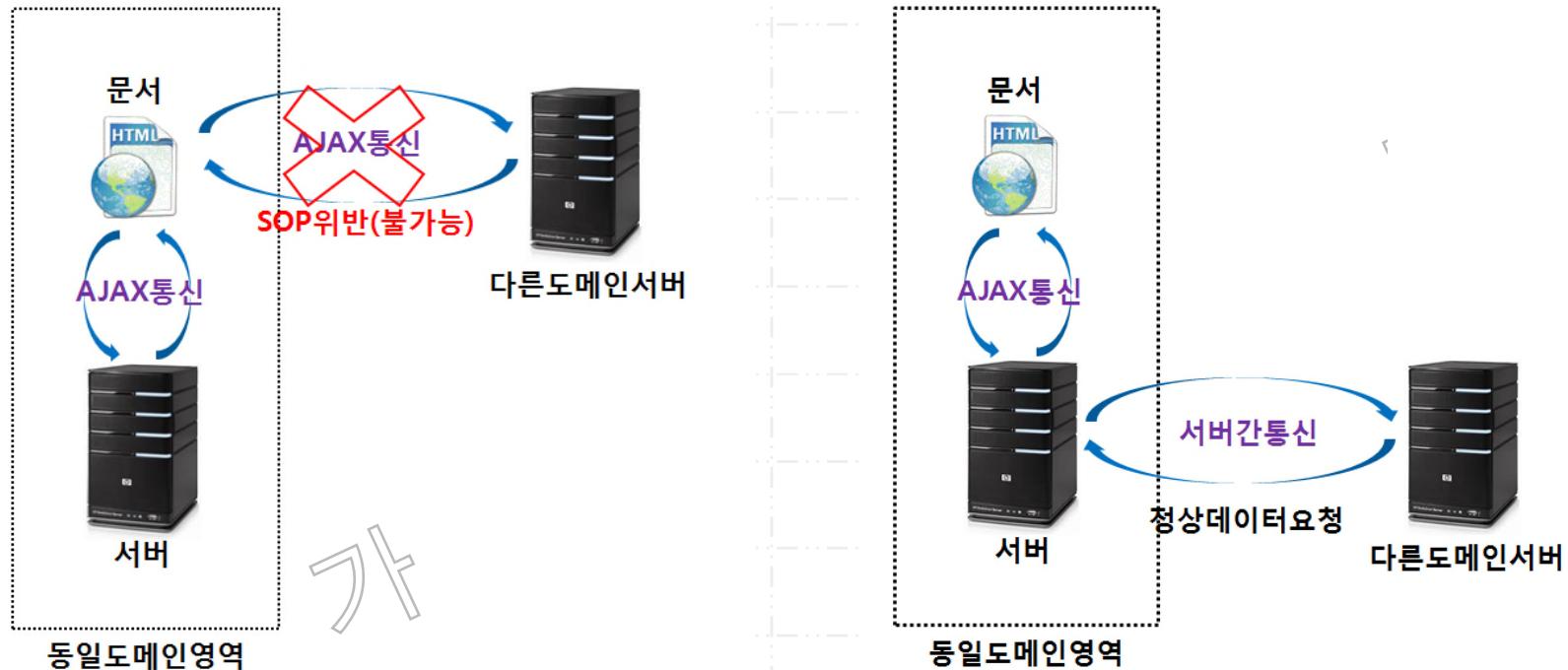
가

Ajax

❖ cross-domain request 처리

● SOP(Same origin Policy: 동일 근원 정책)

- ◆ JavaScript 단에서 Ajax 사용 시 사용 문서와 동일한 도메인으로만 데이터 전송 허용



- ◆ 서버에서 별도로 허용하지 않으면 직접적으로는 사용 불가

```
response.setHeader("Access-Control-Allow-Origin", "http://eshome.com:8081");
```

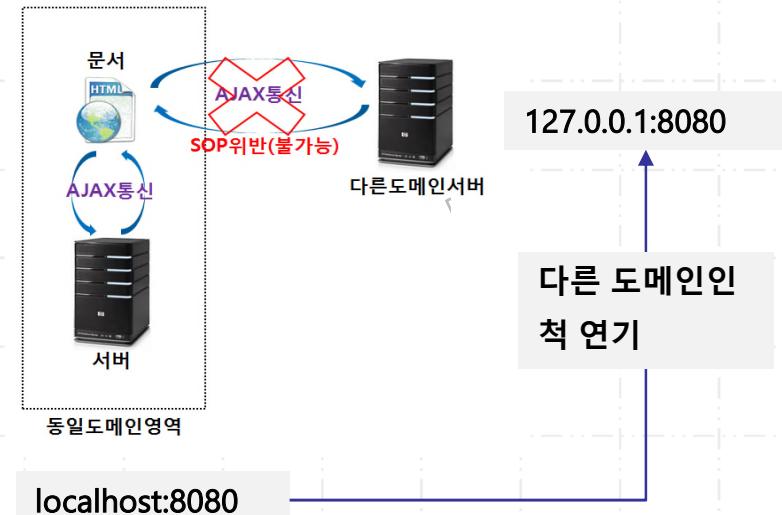
```
response.setHeader("Access-Control-Allow-Origin", "*");
```

Ajax

❖ cross-domain request 처리

● 문제 상황

```
$("#getData").click(function() {  
    dropHere.html("");  
    $.ajax({  
        url : "http://127.0.0.1:8080/jQuery/AjaxJsonServlet",  
        success : function(data) {  
            dropHere.html("");  
            $.each(data, function(index, item) {  
                dropHere.append(index + " : " + item + "<br>");  
            });  
        }  
    });  
});
```



XMLHttpRequest cannot load <http://127.0.0.1:8080/jQuery/AjaxJsonServlet>. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:8080' is therefore not allowed access.

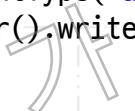
Ajax

❖ cross-domain request 처리

● solution 1 : CORS(Cross Origin Resource Sharing) 기법

◆ 서버에서 클라이언트의 접속 허용

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setHeader("Access-Control-Allow-Origin", "*");
    request.setCharacterEncoding("utf-8");
    String formData = request.getParameter("formData");
    Gson gson = new Gson();
    Person person = gson.fromJson(formData, Person.class);
    Map<String, String> message = new HashMap<>();
    if (person.getMid().equals("admin")) {
        message.put("message", "OK");
    } else {
        message.put("message", "Fail");
    }
    response.setContentType("application/json;charset=UTF-8");
    response.getWriter().write(gson.toJson(message));
}
```



모든 클라이언트의 요청을 수용

◆ 서버의 코드를 수정할 수 없다면 사용 불가

Ajax

❖ cross-domain request 처리

● solution 2: consumer server proxy

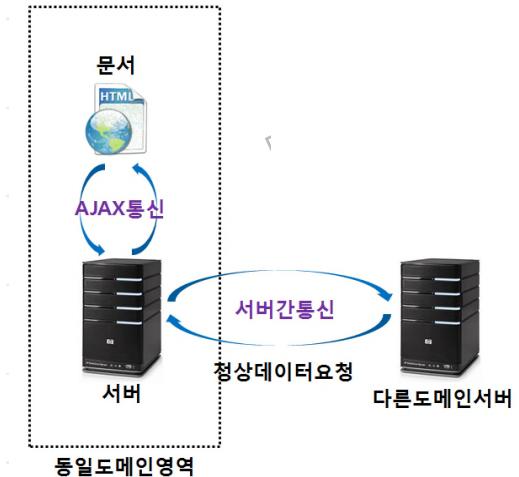
◆ 내부에서 Servlet이 받아서 요청

```
$("#getData").click(function() {  
    dropHere.html("");  
    $.ajax({  
        data : {  
            getUrl : "http://127.0.0.1:8080/jQuery/AJAXJsonServlet"  
        },  
        url : "../../crossDomainServlet",  
        success : function(data) {  
            dropHere.html("");  
            $.each(data, function(index, item) {  
                dropHere.append(index + " : " + item + "<br>");  
            });  
        }  
    });  
});
```

가

ajax service를 하는 곳

동일 서버의 종계 servlet



◆ 서버 프로그램의 복잡도 증가

File Upload

❖ FormData 객체

● HTML5에서 추가된 객체로 multipart/form-data 형태로 자료 전달

- ◆ ajax level2에서 파일을 업로드하기 위해 디자인됨

● 파라미터 설정

- ◆ 객체 생성시 <form> 요소를 넘겨줘서 자동 설정

- file 요소를 제외한 모든 요소들이 (key, value)의 쌍으로 저장됨

```
var formElement = document.querySelector("form");
var formData = new FormData(formElement);
```

- ◆ append(name, value)로 개별 데이터 추가

```
var formData = new FormData();
formData.append("id", $("#id").val());
formData.append("pass", $("#pass").val());
```

- ◆ 파일 정보 추가

- input type이 file인 요소에 등록된 객체의 name 값 추가

```
formData.append("fileName", document.getElementById("file").files[0].name);
```

- ◆ 전송

- send 메서드의 파라미터로 전송

```
xhr.send(formData);
```

File Upload

❖ XMLHttpRequestUpload

- XMLHttpRequest의 upload 속성으로 리턴되며 기본 사용법은 XHR과 동일

- ◆ 업로드 상황을 모니터링할 수 있는 progress, loadstart 이벤트 추가 가능

```
var xmlupload = xhr.upload;

xmlupload.addEventListener("loadstart", function(e) {
  progress.style.display = "inline";
});
xmlupload.addEventListener("loadend", function(e) {
  progress.style.display = "none";
});
xmlupload.addEventListener("progress", function(e) {
  progress.max = e.total;
  progress.value = e.loaded;
});
```

```
<form id="uploadForm">
  <input type="text" id="desc" name="desc">
  <input type="file" id="file" name="file">
  <input type="submit" value="upload">
  <progress id="progress"></progress>
  <div id="message"></div>
</form>
```

```
<style>
  #progress {
    display: none;
  }
</style>
```

File Upload

❖ Servlet 처리

- FormData를 처리하기 위해 서블릿은 @MultipartConfig 설정 필요

- ◆ 기본 서블릿은 application/x-www-form-urlencoded 만 처리 가능

```
@MultipartConfig  
@WebServlet("/upload")  
public class UploadServlet extends HttpServlet {  
  
    protected void doPost(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        req.setCharacterEncoding("UTF-8");  
        res.setCharacterEncoding("UTF-8");  
        String filename = req.getParameter("fileName");  
        String desc = req.getParameter("desc");  
  
        String path = this.getServletContext().getRealPath("/");  
        Part file = req.getPart("file");  
        byte[] buffer = new byte[256];  
        try (InputStream filecontent = file.getInputStream();  
             FileOutputStream output = new FileOutputStream(new File(path+"/"+filename))) {  
            int len;  
            while ((len = filecontent.read(buffer)) > 0) {  
                output.write(buffer, 0, len);  
            }  
        }  
        res.getWriter().write(filename + " 업로드 성공");  
    }  
}
```

File Upload

❖jQuery에서의 upload 처리

```
$("#uploadForm").on("submit", function(e) {  
    e.preventDefault();  
    var formData = new FormData(this);  
    formData.append("fileName", document.getElementById("file").files[0].name);  
    $.ajax({  
        xhr: function(){  
            let xhr = new XMLHttpRequest();  
            $(xhr.upload).on({  
                loadstart : function(e) { $("#progress").css("display", "inline");},  
                loadend :   function(e) { $("#progress").css("display", "none"); },  
                progress : function(e) {  
                    $("#progress").attr("max",e.total);  
                    $("#progress").val(e.loaded);  
                }  
            });  
            return xhr;  
        },  
        url : "../upload",  
        type : "post",  
        data : formData,  
        processData : false,  
        contentType : false,  
        success : function(data) {  
            $("#message").text(data);  
        }  
    });  
});
```

xhr 설정

업로드 설정

178