

웹앱 과정 #2

JavaScript 기본

2019.03

박재성

목차

1. 언어 소개

2. 문법 설명

3. 언어 특징

4. DOM

5. BOM

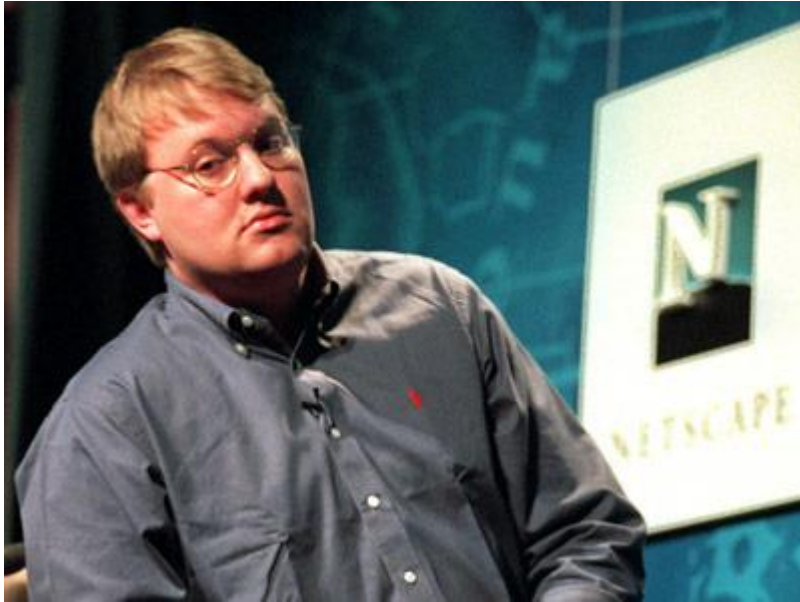
6. Event

7. Ajax

8. 실습

언어 소개

JavaScript History#1



1995년 12월 4일 (Netscape 2.0) 발표

Netscape 개발자이자 CEO 였던, Marc Andreessen은 당시 웹은 정적이었기 때문에, DOM 인터렉션을 통해 다이내믹한 웹을 만들고자 하는 비전을 가지고 있었음

- 당시 Java와 Java Applets 큰 인기, 그러나 비 개발자에겐 덩치가 크고 복잡
- 작고 단순한 작업을 대상, 비 개발자(ex 디자이너)도 쉽게 접근 가능한 언어 필요

JavaScript History#2

전문가 초빙(Brendan Eich)

- Lisp 파생언어인 Scheme 개발을 계획
- 10일만에 JavaScript 개발
- 초기 이름은 여러 차례 변경

Mocha → LiveScript → JavaScript



JavaScript History#3



**NETSCAPE AND SUN ANNOUNCE JAVAScript, THE OPEN, CROSS-
PLATFORM OBJECT SCRIPTING LANGUAGE FOR ENTERPRISE NETWORKS
AND THE INTERNET**

**28 INDUSTRY-LEADING COMPANIES TO ENDORSE JAVAScript AS A COMPLEMENT TO JAVA
FOR EASY ONLINE APPLICATION DEVELOPMENT**

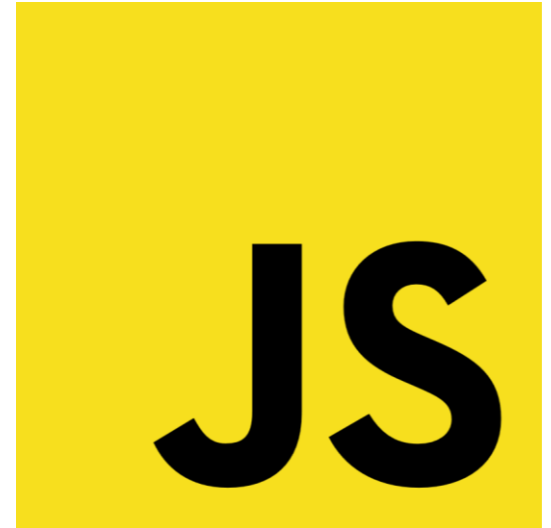
"Java"라는 이름이 포함된 이유?

- JavaScript는 Java의 보조자 역할 (C/C++ and Visual Basic 같은)
- 기존 언어 채택할 수도 있었지만, 'Java'스럽게 보여야 한다는 내부적 의견
- 'Java'를 돋보이게 하는 마케팅적 측면(Netscape과 Sun간의 협정) 압력

JavaScript 표준화

1996/11 표준화 작업 시작

- 상표권 이슈로 'Java'를 사용하지 못함
- 표준화 진행을 위해 ECMA
(European Computer Manufacturers Association)
협회를 통해 진행 → ECMAScript
- 표준을 위한 식별자 'ECMA-262' 부여
- 담당 위원회는 TC39



spec이 표준에 포함되기 위해선 총 5단계의 승인/리뷰 과정을 거쳐야 한다.
Stage 0 ~ 4 단계: <https://tc39.github.io/process-document/>

JavaScript 릴리스 Timeline

Ver/Edition	Date	Description
ES1	1997.06	Netscape 2.0
ES2	1998.08	ISO와 ECMA간의 표준 불일치로 인한 버전
ES3	1999.12	
ES4	-	Dropped - 대용량 스케일 vs 심플함 유지(MS/Yahoo) 8년간 논의, 결국 합의도출 실패
ES5 (v3.1)	2009.12	ES Harmony - 신규 스펙 들을 통칭
ES5.1	2011.06	
ES6	2015.06	moves to annuals release ES2015/ES6, ES2016/ES7, ...

실습 파일 다운로드

→ 다운로드 주소는 수업에서 공지

문법

변수

1. **var (variable) 키워드를 통해 변수 선언**

2. **데이터 타입을 지정하지 않음**

3. **전역 변수**

```
var str = "1";  
var num = 1;
```

4. **지역 변수**

```
function test() {  
    var local = "1";  
}
```

5. **var을 선언하지 않으면 속성으로 지정**

```
function test() {  
    pro = 1;  
}
```

→ 파일 : javascript/01_var.html

타입#1

- undefined
- null
- Boolean
- String
- Number
- Object

타입#2

var foo; *// undefined*

var foo = null; *// null*

var foo = false; *// Boolean*

var foo = "hello"; *// String*

var foo = 123; *// Number*

var foo = { bar : 123 }; *// Object*

타입#3

```
var foo = [1, 2, "abc"];  
→ foo[1]; // 2
```

```
var foo = {  
  val1 : "string",  
  "delete" : 123,  
  val3 : function() { ... },  
  val4 : ["a", "b", "c"]  
};  
→ foo.val1; // "string"  
→ foo["delete"]; // 123  
→ foo.val3();  
→ foo.val4[2]; // "c"
```

→ 파일 : javascript/02_var.html

함수

- **Function 생성자에 정의된 sum 변수에 할당**
 - `var sum = new Function("x", "y", "return x + y");`
 - this가 항상 window
- **sum으로 명명된 함수의 함수선언**
 - `function sum(x,y) { return x + y; }`
 - 초기에 생성
- **sum변수에 할당된 함수 함수 표현식**
 - `var sum = function(x,y) { return x + y; }`
 - 해당 구문이 실행될 때 생성.
- **익명함수 생성**
 - `function (x,y) { return x + y; }`

→ 파일 : javascript/03_function.html

Concatenation

- `var name = "홍길동";`
- `var name2 = "김철수";`

→ `name + name2`
결과 : "홍길동김철수"

→ `name + "과 " + name2`
결과 : "홍길동과 김철수"

→ `name + 123`
결과 : ?

다이얼로그(메세지) 박스

- 경고 출력
`alert("string");`
- 사용자로부터 값을 입력받을 때
`prompt("문자열" [, default_value]);`

→ 파일 : javascript/04_messagebox.html

call by value / call by reference

call by value	call by reference
primitive value - String - number - boolean	- Object - Array - Function

```
var arr = [1, 3];  
var val = 4;
```

```
function add(a, b) {  
    b += 1;  
    return a.push(b);  
}  
add(arr, val);  
val; // 4  
arr; // [1, 3, 5]
```

→ 파일 : javascript/05_call-by.html
 javascript/06_call-by.html

연산자

- Unary

`delete, typeof, ++, --, +, -, !`

- Multiplicative

`*, /, %`

- Relational

`<, >, <=, >=, instanceof, in`

- Logical

`&&, ||`

- Conditional

`(? :)`

- Equality

`==, !=, ===, !==`

- Assignment

`=, *=, /=, %=, +=, -=`

→ 파일 : javascript/07_operator.html

분기/반복 문

- **if**

```
if( a == 1) {  
    //...  
} else if(a == 2) {  
    //...  
} else {  
    //...  
}
```
- **Iteration**

```
while(a == 1) {  
    //...  
}  
  
for(var i = 0; i < 10; i++) {  
    //...  
}  
  
for( var i in obj) {  
    //...  
}
```

기타 문

- **with**

```
with(Math) {  
    // 전달되는 객체의 스코프로 확장한다.  
    console.log(PI); // Math.PI  
    console.log(sin(5)); // Math.sin(5)  
}
```

- **try/catch**

```
try {  
    //...  
} catch(e) {  
    //...  
} finally {  
    //...  
}
```

→ 파일 : javascript/08_statement.html

언어 특징

prototype based programming#1

class based programming

- class(정의)와 instance(실행)가 구분(분류/관계 초점)
- instance는 클래스의 설계와 행동에 따름
- instance는 수정할 수 없음
- class 상속은 상하 구조가 있음

prototype based programming

- class == instance(실행에 초점)
- prototype을 복제
- 언제든지 수정 가능
- prototype 상속은 상하 구조가 아니라 위임방식

prototype based programming#2

```
function Parent(name) {  
    this.name = name;  
}
```

```
Parent.prototype.getName = function() {  
    return this.name;  
}
```

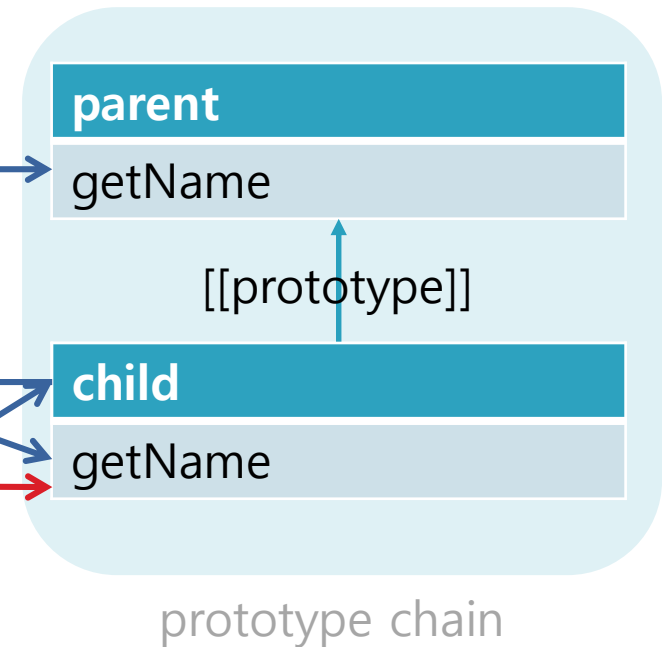
```
function Child(name) {  
    this.name = name;  
    this.getName = function() {  
        return "child = " + this.name;  
    }  
}
```

```
Child.prototype = new Parent();  
Child.prototype.constructor = Child;
```

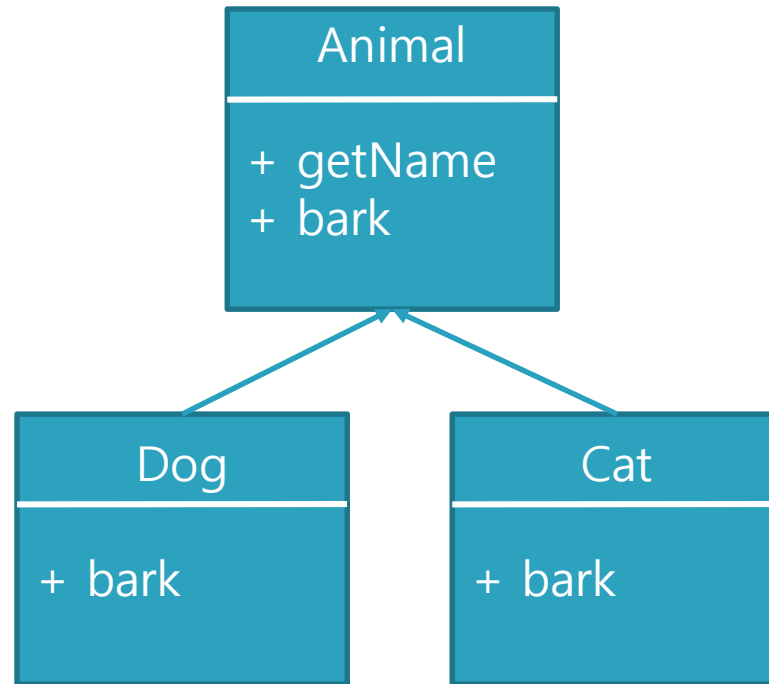

prototype based programming#3

```
var child = new Child("test");  
child.getName(); // "child = test"
```

```
delete child.getName;  
child.getName(); // "test"
```



prototype based programming#4



→ 파일 : javascript/09_oop.html

this#1

- 실행한 주체를 가리킨다.
- 실행될 때 결정되며 중간에 변경할 수 없다.
- 실행방법에 따라 결정된다.

```
this.a = 10; //global.a === window.a  
alert(a);
```

```
b = 20;  
alert(this.b); //global.b
```

```
var c = 30;  
alert(this.c); //global.c
```

→ 파일 : javascript/10_this1.html

this#2

• 어떻게 실행했는지에 따라 this 키워드는 결정

```
var foo = {  
  x: 10  
};
```

```
bar.test();  
foo.test = bar.test;  
foo.test();
```

```
var bar = {  
  x: 20,  
  test: function() {  
    alert(this === bar);  
    alert(this.x);  
  }  
};
```

```
// this에 값을 할당할 수 없기 때문에 아래 코드는 오류가 발생된다.  
this = foo;  
alert(this.x);
```

```
};
```

→ 파일 : javascript/11_this2.html

this#3

```
function foo() {  
    alert(this);  
}
```

```
foo();  
alert(foo === foo.prototype.constructor);  
foo.prototype.constructor();
```

→ 파일 : javascript/12_this3.html

this#4

```
var foo = {  
  bar: function() {  
    alert(this);  
    alert(this === foo);  
  }  
};  
  
foo.bar();  
var exampleFunc = foo.bar;  
alert(exampleFunc === foo.bar);  
exampleFunc();
```

→ 파일 : javascript/13_this4.html

this#5

```
function A() {  
    alert(this);  
    this.x = 10;  
}
```

```
var a = new A();  
alert(a.x);
```

→ 파일 : javascript/14_this5.html

this#6

```
setTimeout(function() {  
    console.log(this);  
}, 10);
```

```
document.getElementById("id").addEventListener("click", function() {  
    console.log(this);  
});
```

→ 파일 : javascript/15_this6.html

this#7

- **this**를 바꾸는 함수.
- *func.call*([[scope]][, param[,param...]]);
- *func.apply*([[scope]][,[param,...,param]]);

```
var b = 10;
```

```
function a(c) {  
    alert(this.b);  
    alert(c);  
}
```

```
a(20); // this === global, this.b == 10, c == 20
```

```
a.call({b: 20}, 30); // this === {b: 20}, this.b == 20, c == 30
```

```
a.apply({b: 30}, [40]); // this === {b: 30}, this.b == 30, c == 40
```

closure#1

- 코드 블록에 데이터가 저장된 상태
→ 코드가 쓰여진 상황에 대한 유효범위를 갖는 코드블록
- 함수를 파라미터나 반환 값으로 사용하게 되면
값이 저장되며, 함수 생성과 동시에 closure가
생성
- JavaScript는 static scope

Static(lexical) Scope : 어휘적 유효범위

```
var z = 10;  
function foo() {  
    console.log(z);  
}
```

foo(); // 10 – with using both static and dynamic scope

```
(function () {  
    var z = 20;  
    foo(); // 10 – with static scope, 20 – with dynamic scope  
})();
```


```
(function (funArg) {  
    var z = 30;  
    funArg(); // 10 – with static scope, 30 – with dynamic scope  
})(foo);
```

→ 파일 : javascript/16_closure1.html

closure#2

```
var data = [];  
for (var k = 0; k < 3; k++) {  
    data[k] = function () {  
        alert(k);  
    };  
}
```

```
data[0](); //3  
data[1](); //3  
data[2](); //3
```

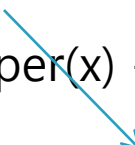


Data	k
function	3
function	3
function	3

→ 파일 : javascript/17_closure2.html

closure#3

```
var data = [];  
for (var k = 0; k < 3; k++) {  
    data[k] = (function _helper(x) {  
        return function () {  
            alert(x);  
        };  
    })(k);  
}
```



Data	k
function	0
function	1
function	2

```
data[0](); // 0  
data[1](); // 1  
data[2](); // 2
```

→ 파일 : javascript/18_closure3.html

DOM(Document Object Model)

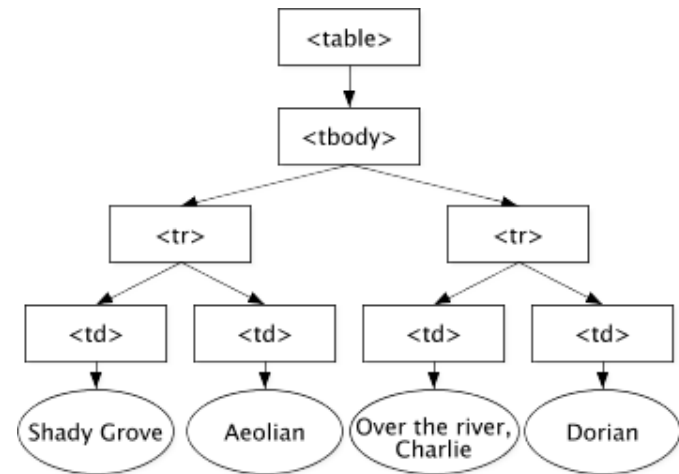
DOM?

- 문서 객체 모델은 객체 지향 모델로써 **구조화된 문서를 표현하는 형식**
- DOM은 **플랫폼/언어 중립적**으로 구조화된 문서를 표현하는 W3C의 공식 표준이다.
- DOM은 W3C 가 표준화한 **다수의 API 기반**이 된다.
- DOM은 동적으로 **문서의 내용, 구조, 스타일**에 **접근하고 변경**하는 수단

DOM?

HTML

```
<table>  
  <tbody>  
    <tr>  
      <td>Shady Grove</td>  
      <td>Aeolian</td>  
    </tr>  
    <tr>  
      <td>Over the River, Charlie</td>  
      <td>Dorian</td>  
    </tr>  
  </tbody>  
</table>
```



Element search

`document.getElementById("id");`

- 아이디로 엘리먼트를 찾기.

`document.querySelector("css selector");`

- css 선택자로 엘리먼트를 찾기.

`document.querySelectorAll("css selector");`

- css 선택자로 다수의 엘리먼트 찾기

→ 파일 : dom/01_search.html

attribute

엘리먼트의 속성 제어.

ex)

```
var ele = document.getElementById("some_id");  
ele.getAttribute("id"); //some_id  
ele.setAttribute("id","some_id2"); //some_id2  
ele.removeAttribute("id"); //삭제
```

→ 파일 : dom/02_attr.html

style

엘리먼트의 스타일 제어

ex)

```
var ele = document.getElementById("some_id");
```

```
ele.style.width; //100px - 인라인 style속성의 값
```

```
ele.style.width = "200px"; // 200px - 인라인 style속성으로 설정
```

```
ele.style.backgroundColor = "red"; // 배경색을 빨간색으로 설정
```

```
//적용된 style의 값
```

```
document.defaultView.getComputedStyle(ele,null).getPropertyValue("width");
```

```
//다량의 스타일을 설정
```

```
ele.style.cssText = "border: 1px solid red; width:100px";
```

→ 파일 : dom/03_style.html

class

엘리먼트의 클래스 제어

ex)

```
var ele = document.getElementById("some_id");
```

```
ele.className = "test";
```

```
ele.className = ele.className.replace(/test/, "");
```

```
// html5
```

```
ele.classList.add("test");
```

```
ele.classList.remove("test");
```

```
ele.classList.contains("test");
```

→ 파일 : dom/04_class.html

DOM create

DOM을 생성

```
var ele = document.createElement("div");  
ele.id = "some";
```

DOM append

DOM을 추가

```
var parent = document.getElementById("parent");
```

//기본

```
var child = document.createElement("div");  
ele.id = "child";  
parent.appendChild(child);
```

//innerHTML

```
parent.innerHTML = "<div id='child'> </div>";
```

//insertAdjacentHTML

```
parent.insertAdjacentHTML("beforeend", "<div id='child'> </div>");
```

→ 파일 : dom/05_create-append.html

DOM remove

DOM을 삭제

```
var parent = document.getElementById("parent");
```

```
//기본
```

```
var child = document.getElementById("child");  
parent.removeChild(child);
```

```
//innerHTML
```

```
parent.innerHTML = "";
```

→ 파일 : dom/06_remove.html

BOM (Browser Object Model)

BOM (Browser Object Model)?

- 브라우저와 관련된 객체
- 표준이 없고 브라우저 벤더사마다 알아서 구현
- DOM 0이란 표현도 함

Window

최상위 객체로 브라우저가 표시하고 있는 윈도우 문서를 의미한다.

```
window;
```

navigator

브라우저 관련 정보

`navigator.userAgent`; // 브라우저 사용자 에이전트 정보
`navigator.plugins`; // 브라우저의 설치된 플러그인 정보

location, history

location

현재 문서의 location 정보를 갖는 객체

```
window.location = "http://www.naver.com";  
location.pathname;  
location.hostname;
```

history

브라우저가 탐색한 history 정보를 갖는 객체

```
history.back(); // 뒤로 이동 - history.go(-1)  
history.go(1);  // 앞으로 이동
```

Timer

타이머

```
var timerID = setTimeout(listener, milliseconds);  
clearTimeout(timerID);
```

// 한번 호출

```
var timerID = setTimeout(function(){ alert("call");}, 100);  
clearTimeout(timerID);
```

// 주기적으로 호출

```
var timerID = setInterval(listener, milliseconds);  
clearInterval(timerID);
```

Timer의 이해

자바스크립트는 단일 Thread

- Thread는 Queue형식으로 관리
- 이벤트, 비동기 callback(timer callback, ajax callback)도 Queue로 관리함

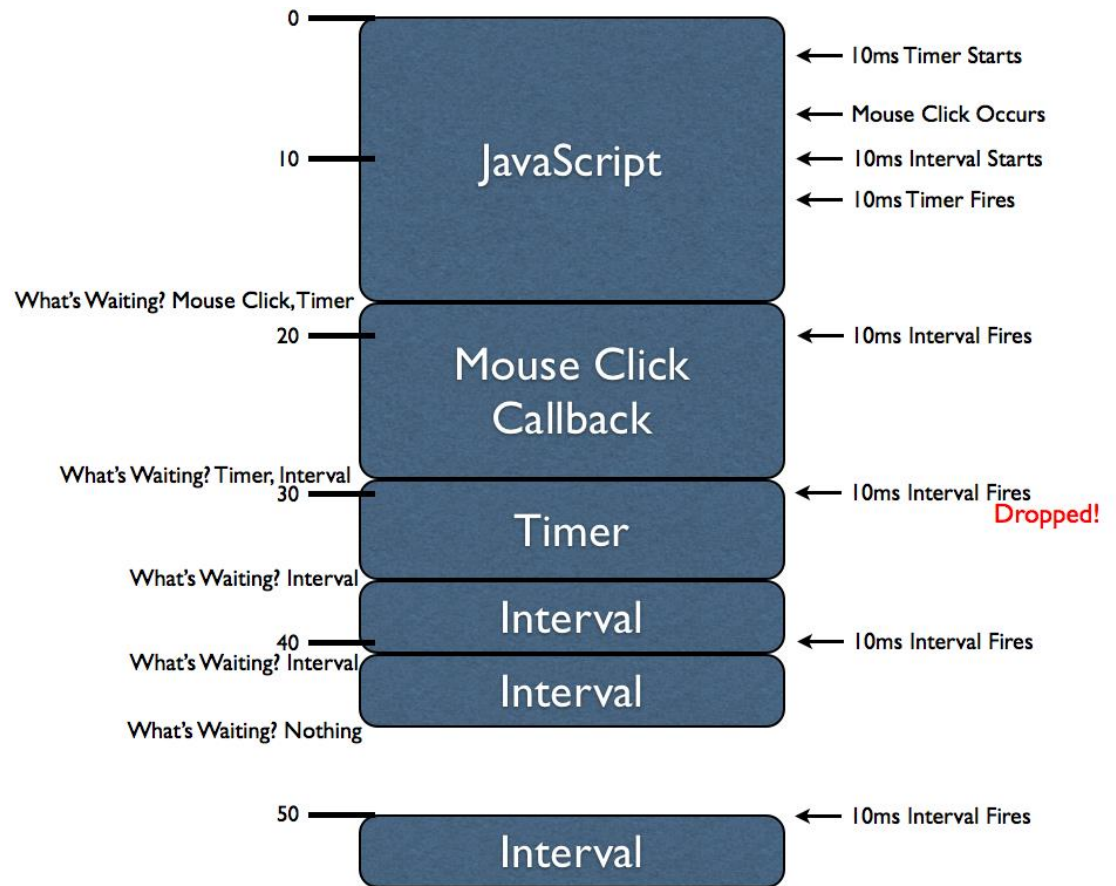
Timer의 이해

```
test();
```

```
function test() {
  setTimeout(function() {
    alert("timer");
  }, 10);
```

// mouse click

```
setInterval(function() {
  alert("interval");
}, 10);
}
```



실습#1

1. Animation 객체에는 start, change메서드가 있다.
2. start메서드를 호출하면 setInterval을 이용하여 change메서드를 10ms마다 호출한다.
 1. 10회를 호출하면 멈춘다.

→ 파일 : bom/01_timer.html

실습#2

test라는 아이디를 가진 엘리먼트에
timer를 이용하여 opacity값이 1부터 0까지
0.01씩 감소하여 스크립트를 작성하시오.
100회를 실행하면 멈추도록 함

조건 :

- timer를 이용할 것(css3 사용하면 안됨.)
- Pseudocode을 기반으로 할 것.
- Animation.run을 누르면 실행할 것.

→ 파일 : bom/02_timer.html

Event

Event?

이벤트 리스너 등록과
이벤트 흐름을 묘사하고
사용자 인터페이스 제어와
Document 의 변이(Mutation)를
알리기 위함

종류#1

- User Interface Event Types
 - DOMActivate, load, unload, abort, error, select, resize, scroll
- Focus Event Types
 - blur, DOMFocusIn, DOMFocusOut, focus, focusin, focusout
- Mouse Event Types
 - click, dblclick, mousedown, mouseenter, mouseleave, mousemove, mouseover, mouseout, mouseup

종류#2

- Wheel Event Types
 - wheel
- Keyboard Event Types
 - keydown, keypress, keyup
- Composition Event Types
 - compositionstart, compositionupdate, compositionend

종류#3

- Mutation Event Types
 - DOMAttrModified, DOMCharacterDataModified, DOMNodeInserted, DOMNodeInsertedIntoDocument, DOMNodeRemove, DOMNodeRemovedFromDocument, DOMSubtreeModified
- Mutation Name Event Types
 - DOMAttributeNameChanged, DOMElementNameChanged
- CSS3 Event Types
 - transitionend, animationstart, animationend, animationiteration

→ 파일 : Event/01_drag.html

사용 방법

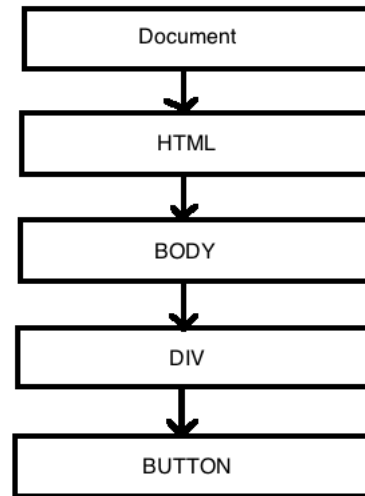
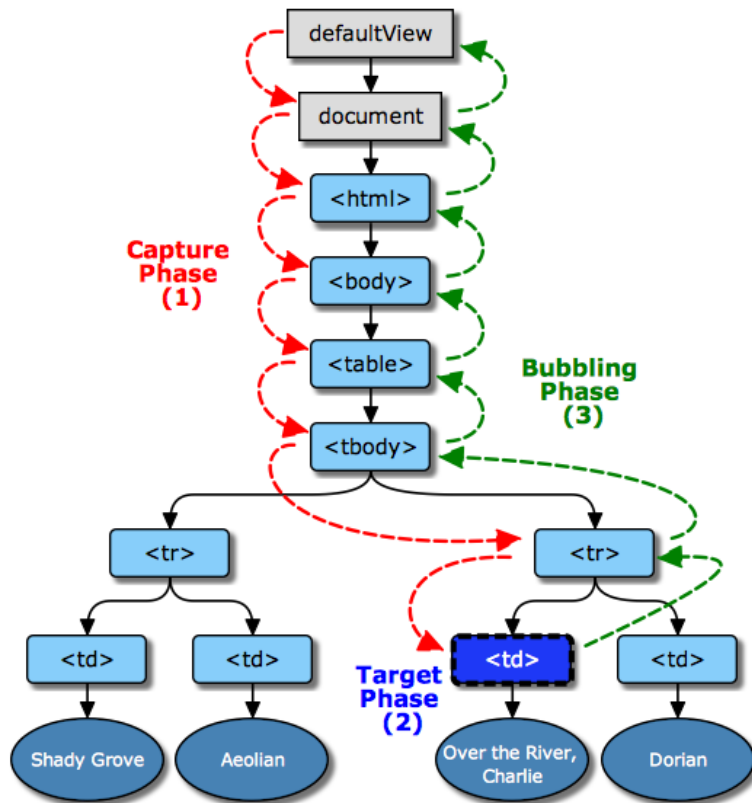
element.addEventListener(type, listener, useCapture);

- 이벤트 추가

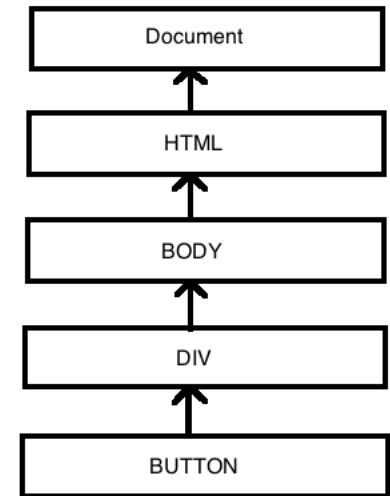
element.removeEventListener(type, listener, useCapture);

- 이벤트 해제

버블링/캡처링



Event Capturing



Event Bubbling

버블링/캡처링

```
<div id="hello" style="width:100px;height:100px;background-color:red;">
  <div id="world" style="width:30px;height:30px;background-color:blue;">
    </div>
  </div>
<script type="text/javascript">
  var hello = document.getElementById("hello");
  var world = document.getElementById("world");

  hello.addEventListener("click",function() { alert("hello"); }, true);
  world.addEventListener("click",function() { alert("world"); }, false);
</script>
```

→ 파일 : Event/02_bubbling.html

preventDefault / stopPropagation

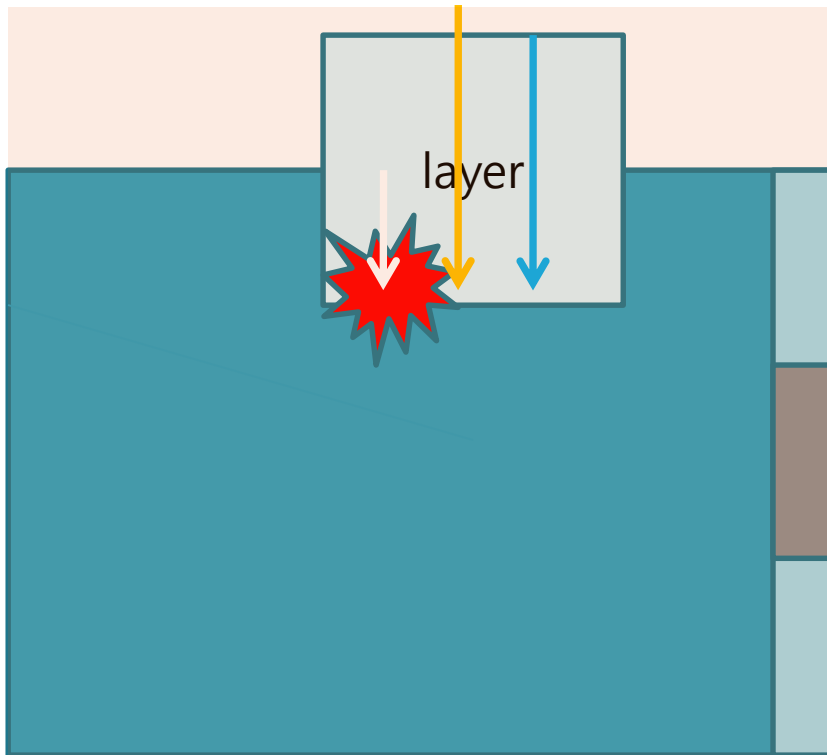
- **e.preventDefault()**
 - 기본 기능을 막을 때 사용.
- **e.stopPropagation()**
 - 버블링을 막을 때 사용.

→ 파일 : Event/03_preventDefault.html

Event 객체

- **이벤트가 발생하면 event객체가 함수로 전달**
 - 하지만 브라우저 마다 속성이나 전달 방식의 차이가 있음.
 - IE8이하는 window.event속성에 기타 브라우저는 함수의 첫 번째 인자로
- **이벤트를 발생한 엘리먼트**
 - IE8이하는 e.srcElement, 기타 브라우저 e.target
- **이벤트가 정의된 엘리먼트**
 - e.currentTarget
- **키코드**
 - ie는 e.keyCode, 기타 which(최신 브라우저는 모두 지원)
- **버블링, 기본 기능 정지**
 - IE8 이하 e.returnValue = false; e.cancelBubble = true;
 - 기타 브라우저 e.stopPropagation(), e.preventDefault();

Event 객체



속성	설명
offsetX	화면에서 x좌표
offsetY	화면에서 y좌표(흰색)
pageX	page에서 x좌표
pageY	page에서 y좌표(오렌지 색)
clientX	엘리먼트 기준 x좌표
clientY	엘리먼트 기준 y좌표(하늘색)

delegete

엘리먼트에 이벤트를 직접 할당하는 것이 아니라 **상위 엘리먼트에 위임**.

delegate

```
<ul id="parent">  
  <li class="odd">1 </li>  
  <li>2 </li>  
  <li class="odd">3 </li>  
  <li>4 </li>  
</ul>
```


```
var aOdd = document.querySelectorAll(".odd");  
  
for(var i = 0, l = aOdd.length; i < l ;i++) {  
  aOdd[i].addEventListener("click",function() {  
    alert("click");  
  });  
}
```

→ 파일 : Event/04_delegate.html

delegete

```
<ul id="parent">  
  <li class="odd">1 </li>  
  <li>2</li>  
  <li class="odd">3</li>  
  <li>4</li>  
</ul>
```

```
var ul = document.getElementById("parent");  
ul.addEventListener("click",function(e) {  
  if(e.target.classList.contains("odd")) {  
    alert("click");  
  }  
});
```



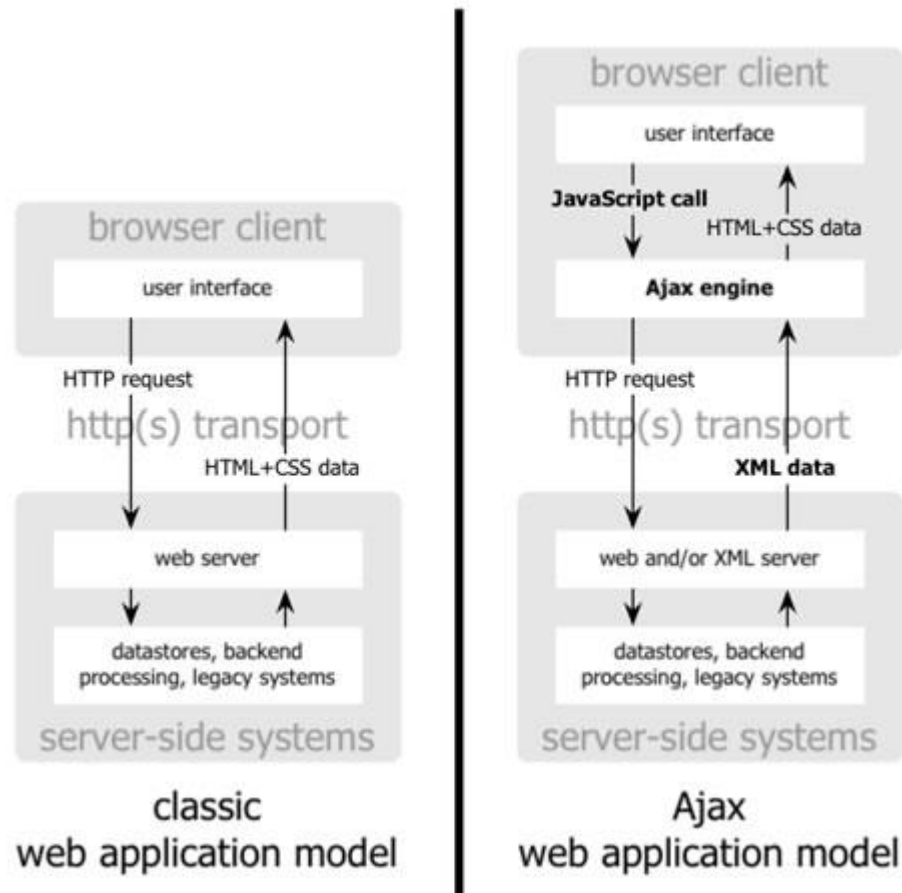
→ 파일 : Event/05_delegate-add.html

delegete

1. 메모리 누수 걱정을 덜 할 수 있다.
2. 보다 적은 돔과 함수를 사용한다.
3. 돔이 변경되었을 때 이벤트를 핸들링의 걱정이 없다.

Ajax

Ajax?



종류

- **동일 도메인**
 - **XHR (XMLHttpRequest)**
 - 기본적으로 사용되는 AJAX
- **다른 도메인**
 - **JSONP(JSON with Padding)**
 - 스트립트 태그를 삽입하여 SOP(Same Origin Policy) 해결

직접 구현

```
var req = new XMLHttpRequest();

req.addEventListener("load", function() {
    console.log(req.responseText);
}, false);

req.addEventListener("error", function() {
    console.log("fail");
}, false);

req.open('POST', 'user/get-name');
req.setRequestHeader('X-Test', 'one');
req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded; charset=utf-8");
req.send("key=value");
```

→ 파일 : Ajax/01_ajax.html

CORS : Cross Origin Resource Sharing (client)

```
var req = new XMLHttpRequest();

req.addEventListener("load", function() {
    console.log(req.responseText);
}, false);

req.addEventListener("error", function() {
    console.log("fail");
}, false);

req.open('POST', 'user/get-name');
req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded; charset=utf-8");
req.setRequestHeader('X-Test', 'one');
req.withCredentials = "true"; // 쿠키 설정
req.send("key=value");
```

CORS : Cross Origin Resource Sharing (server)

```
app.post("/cors", function(req, res) {  
  res.set({  
    "Access-Control-Allow-Origin": "http://localhost:3000",  
    "Access-Control-Allow-Headers": "X-Test",  
    "Access-Control-Allow-Credentials": true  
  });  
  
  res.send(req.body);  
});
```

→ 파일 : Ajax/02_cors.html

CORS: express.js middleware

Express.js 미들웨어를 설치해 간편하게 처리할 수도 있다.

```
$ npm install cors --save
```

```
var cors = require("cors");

// 기본 옵션을 사용
app.use(cors());

// 또는 상세 옵션을 설정
app.use(cors({
  origin: "http://localhost:3000",
  credentials: true,
  optionsSuccessStatus: 200 // for some legacy browsers
}));

app.all("/cors2", function (req, res, next) {
  res.json({
    msg: "CORS가 성공적으로 호출됨!"
  });
});
```

실습

TODO

할 일을 입력하세요. 추가

- ☐ dfsdfsdfsdf
- ☐ 11111111111
- ☐ sdfsdfsdf
- ☐ sadfafsafa
- ☐ sadfdasdfasf

1. select를 complete이 0인 데이터만 가져오기

→ 파일: quiz/todo/todo-server.js

TODO

할 일을 입력하세요.

추가

- ☐ dfsdfsdfsdf
- ☐ 11111111111
- ☐ sdfsdfsdf
- ☐ sadfafsafa
- ☐ sadfdasdfasf

1. todo가 없는 상태에서 추가 버튼 누르면 alert실행
2. 등록을 Ajax로 변경
3. Ajax와 기본 모드 모두 동작하게 함

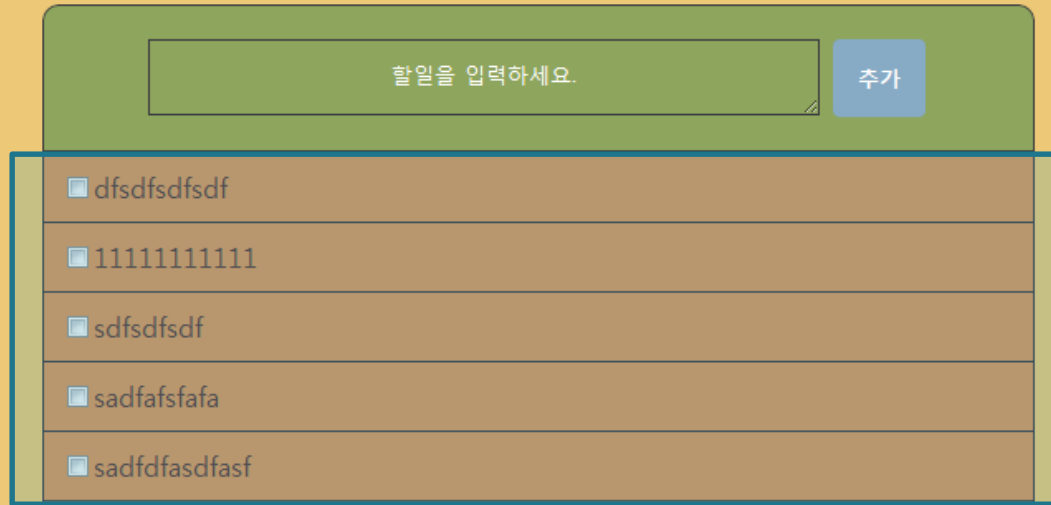
TODO

할 일을 입력하세요. 추가

- ☐ dfsdfsdfsdf
- ☐ 11111111111
- ☐ sdfsdfsdf
- ☐ sadfafsafa
- ☐ sadfdasdfasf

1. checkbox를 클릭하면
Ajax로 complete을 1로 변경
2. 해당 엘리먼트 display : none;

TODO



할 일을 입력하세요. 추가

- ☐ dfsdfsdfsdf
- ☐ 11111111111
- ☐ sdfsdfsdf
- ☐ sadfafsafa
- ☐ sadfdasdfasf

1. `display:none;`한 엘리먼트에 `transition`을 이용하여 `fade-out`처리
2. `webkitTransitionEnd` 이벤트를 이용하여 완료가 되면 `display:none;`

고맙습니다.
