생능출판

HTML 표준화 문서를 기반으로 하는 지침서

속이 깊은 HTML5 & CSS3

김명진 지음

14강

HTML API-1

오프라인 웹, 파일접근, 웹 스토리지

학습 목표

HTML5에서는 새롭고 다양한 API들을 추가하였다. 오프라인 상태에서도 웹에 있는 자원을 사용할 수 있도록 해주는 오프라인 웹 애플리케이션 API와 함께 데스크탑 PC의 로컬 파일의 데이터를 직접 스크립트로 접근할 수 있도록 해주는 API기능이 추가되었다. 그리고 웹 스토리지 관련 API를 추가하여 반복적으로 요청되는 데이터를 직접 클라이언트 측 브라우저에 효과적으로 저장하고 관리하도록 하고 있다. 이번 장에서는 HTML5에 새롭게 추가된 이러한 기능들에 대하여 살펴보도록 하겠다.

Section

1. 오프라인 웹 애플리케이션 API
2. 파일 API
3. 웹 스토리지

오프라인 웹 애플리케이션

표준화 문서	HTML 5 - http://www.w3.org/html/wg/drafts/html/CR/
표준화 단계	Last Call Working Draft(Editor's Draft) (2014-06-27)
문서 위치	5.7 Offline Web Application

소개

- 오프라인 웹 애플리케이션이란 말 그대로 오프라인 상태에서도 사용이 가능하다는 것을 의미
  - 일정한 제약 사항이 있을 수 있지만, 온라인 지원이 안 되는 곳(비행기 안에서 업무를 볼 수도 있고, 여행지에서 기록을 할 수도 있는 등)에서도 웹에 접근하여 사용이 가능하다.
  - 파일들을 클라이언트에서 캐시로 저장하여 사용하므로 웹 문서의 접근이 빠르다.
  - 지속적인 네트워크의 접속이 줄어들기 때문에 모바일 기기에서는 효과적인 전원 관리가 가능하다.
  - 브라우저는 오직 자원이 변경된 경우에만 다운로드를 시도

소개

파일	코드

clock.html	<!DOCTYPE HTML>
	<head>
	  <title>Clock</title>
	  <script src="clock.js"></script>
	  <link rel="stylesheet" href="clock.css">
	</head>
	<body>
	  The time is: <output id="clock"></output>
	</body>
	</html>

clock.css	output { font: 2em sans-serif; }

clock.js	setInterval(function () {
	  document.getElementById('clock').value = new Date();
	}, 1000);

clock.html	<!DOCTYPE HTML>
	<html manifest="clock.appcache">
	<head>
	  <title>Clock</title>
	  <script src="clock.js"></script>
	  <link rel="stylesheet" href="clock.css">
	</head>
	<body>
	  The time is: <output id="clock"></output>
	</body>
	</html>
		clock.appcache
		CACHE MANIFEST
		clock.html
		clock.css
		clock.js
		clock.appcache

매니페스트 파일

- 오프라인에서도 웹 사이트 이용이 가능하도록 하기 위한 캐시에 저장할 자원의 목록(여러 문서와 파일들) 파일
  - 매니페스트 파일에는 어떠한 파일을 캐시로 저장해야 하는지를 지정하며 서버로부터 웹 문서들과 함께 다운로드 되는 파일이다.

캐시 매니페스트 문법

- 매니페스트는 text/cache-manifest MIME 타입을 이용하여 제공되어야 한다.
- 명세서에서 정의한대로 MIME 타입을 이용하여 제공된 자원 모두는 애플리케이션 캐시 매니페스트의 문법을 따라야 한다.

설명문	0개 이상의 U+0020 공백과 "tab"(U+0009) 문자, 이 뒤에 나오는 "#"(U+0023) 문자 하나, 이 뒤에 나오는 "LF"(U+000D)와 "CR"(U+000D) 문자를 제외한 0개 이상의 문자로 구성된다. 예) **# 캐시할 파일은 다음과 같습니다.**
섹션 헤더	현재 섹션을 변경한다. ✓ **CACHE:** 명시적인 섹션으로 전환된다. ✓ **FALLBACK:** 대체 섹션으로 전환된다. ✓ **NETWORK:** 온라인 화이트리스트 섹션으로 전환된다.
현재 섹션에서 사용되는 데이터	데이터 줄에 적용해야 하는 포맷은 현재 섹션에 달려 있다. ➢ 현재 섹션이 명시적인 섹션일 때, 데이터 줄은 0개 이상의 공백과 탭 문자, 매니페스트가 아닌 자원을 식별하는 유효한 URL, 0개 이상의 공백과 탭 문자로 구성된다. ➢ 현재 섹션이 대체 섹션일 때, 데이터 줄은 0개 이상의 공백과 탭 문자, 매니페스트가 아닌 자원을 식별하는 유효한 URL, 0개 이상의 공백과 탭 문자, 매니페스트가 아닌 자원을 식별하는 다른 유효한 URL, 0개 이상의 공백과 탭 문자로 구성된다. ➢ 현재 섹션이 온라인 화이트리스트 섹션일 때, 데이터 줄은 0개 이상의 공백과 탭 문자와 단일 "*" 문자나 매니페스트가 아닌 자원을 식별하는 유효한 URL, 0개 이상의 공백과 탭 문자로 구성된다.

매니페스트 파일

캐시 매니페스트 문법 – 섹션

섹션	설명
CACHE:	기본 값을 나타내는 부분으로, 클라이언트에서 캐시가 되어야 할 파일들을 지정하는 부분 이곳에 지정된 파일들은 오프라인에서도 해당 파일에 접근이 가능하다.
NETWORK:	반드시 온라인 상태에서만 접근할 수 있는 파일들을 지정하는 부분으로, CACHE 섹션과는 반대 문서에서 서버와 다시 통신하기 위해 사용하는 모든 URL에 대해서 URL 모두 이 섹션에 자동으로 들어가게 하려면, "*" 문자를 그 URL중 하나로 명시할 수 있다. 온라인 화이트리스트에 작성된 네임스페이스와 네임스페이스는 접두어가 일치해야 한다.
FALLBACK:	대체되는 자원을 지정하는 부분 클라이언트에서 서버로 문서를 요청했는데, 해당 문서가 존재하지 않을 때 대신 표시할 자원을 지정하는 것이다. 다른 섹션이 각 파일들의 이름을 한 줄에 하나씩 지정하는 반면에, 이 섹션은 좌우 쌍으로 2개 지정해야 한다.

매니페스트 파일

캐시 매니페스트 문법 – 예

방법1	방법2
CACHE MANIFEST images/sound-icon.png images/background.png **NETWORK:** comm.cgi **FALLBACK:** signup.html offline.html **CACHE:** style/default.css	**CACHE MANIFEST** **NETWORK:** comm.cgi **FALLBACK:** signup.html offline.html **CACHE:** style/default.css images/sound-icon.png images/background.png

- 오프라인 애플리케이션 캐시 매니페스트에서 다음과 같이 상대 경로나 절대 경로 URL을 사용할 수 있다.

CACHE MANIFEST /main/home /main/app.js /settings/home	**CACHE MANIFEST** http://img.example.com/logo.png http://img.example.com/check.png http://img.example.com/cross.png

매니페스트 파일

캐시 매니페스트 작성하기

- 서버의 mime.types 파일에 다음과 같이 지정(manifest 확장자)

```
text/cache-manifest        manifest
```

- manifest 속성을 이용하여 매니페스트 파일을 다음과 같이 지정

```
<!DOCTYPE html>
<html manifest = "매니페스트 파일 이름" >
....
</html>
```

clock.html

```
<!DOCTYPE HTML>
<html manifest="clock.appcache">
<head>
    <title>Clock</title>
    <script src="clock.js"></script>
    <link rel="stylesheet" href="clock.css">
</head>
<body>
    <p>The time is: <output id="clock"></output></p>
</body>
</html>
```

clock.appcache

```
CACHE MANIFEST
clock.html
clock.css
clock.js
clock.appcache
```

이벤트 처리

- 매니페스트가 업데이트되고 유지되도록 여러 이벤트가 ApplicationCache 객체에서 발생하여 사용자에게 캐시 업데이트의 상태에 대하여 적절하게 알릴 수 있다.

이벤트 이름 (인터페이스)	이벤트 발생 조건	다음 이벤트
checking (Event)	브라우저에서 업데이트할 것이 있는지 확인하거나, 처음으로 매니페스틀 다운로드 하려고 할 때.	noupdate, downloading, obsolete, error
noupdate (Event)	매니페스트가 변경된 적이 없다.	이벤트 발생 순서에서 가장 마지막으로 발생
downloading (Event)	브라우저에서 업데이트할 것을 찾아서 가져오고 있거나, 패니페스트에 기록된 자원을 처음으로 다운로드 하려고 할 때.	progress, error, cached, updateready
progress (ProgressEvent)	패니페스트에 기록된 자원을 다운로드 하려고 할 때.	progress, error, cached, updateready
cached (Event)	매니페스트에 기록된 자원이 다운로드 되어 있으며, 애플리케이션은 이제 캐시에 저장된 상태일 때	이벤트 발생 순서에서 가장 마지막으로 발생
updateready (Event)	매니페스트에 기록된 자원이 새롭게 다운로드 되었을 때.	이벤트 발생 순서에서 가장 마지막으로 발생
obsolete (Event)	매니페스트가 404나 410 문서로 찾을 수 없는 파일이 되어 애플리케이션 캐시가 제거될 때.	이벤트 발생 순서에서 가장 마지막으로 발생
error (Event)	매니페스트가 404나 410 문서이기 때문에 애플리케이션을 캐시에 저장하려는 시도가 취소 될 때.	이벤트 발생 순서에서 가장 마지막으로 발생
	매니페스트가 변경되지 않았지만, 이 매니페스트를 참조하는 문서가 적절히 다운로드 되지 않을 때	
	매니페스트에 기록된 자원을 가져오는 동안 치명적인 오류가 발생할 때	
	캐시 업데이트가 이루어지고 있는 동안 매니페스트가 변경될 때	브라우저에서 그 파일들을 다시 가져오려고 곧 시도할 것이다.

이벤트 처리

```
window.applicationCache.onchecking = function(e) { ... }
window.applicationCache.onnoupdate = function(e) { ... }
window.applicationCache.ondownloading = function(e) { ... }
...
```

```
function handleCacheEvent(e) { ... }
window.applicationCache.addEventListener('checking', handleCacheEvent, false);
window.applicationCache.addEventListener('noupdate', handleCacheEvent, false);
window.applicationCache.addEventListener('downloading', handleCacheEvent, false);
...
```

속성과 메서드

cache = window.applicationCache	(창에서) Window의 활성화된 문서에 적용되는 ApplicationCache 객체를 반환한다.
cache = self.applicationCache	(공유한 워커에서) 현재 공유하는 워커에 적용되는 ApplicationCache 객체를 반환한다.

cache.status		애플리케이션 캐시의 현재 상태를 아래에 정의된 상수로 반환한다.
UNCACHED	0	캐시를 하지 않는다.
IDLE	1	최신의 캐시를 이용중인 상태이다.
CHECKING	2	캐시의 업데이트를 체크중인 상태이다.
DOWNLOADING	3	캐시를 업데이트중인 상태이다.
UPDATEREADY	4	최신의 캐시를 이용할 수 있는 상태이다.
OBSOLETE	5	오류가 발생하여 캐시가 되지 않은 상태이다.

이벤트 처리

속성과 메서드

***cache*.abort()**	애플리케이션 캐시 다운로드 프로세스를 취소한다.
***cache*.update()**	애플리케이션 캐시 다운로드 프로세스를 호출한다.
***cache*.swapCache()**	가장 최근의 애플리케이션 캐시로 전환한다.

```
var appCache = window.applicationCache;
switch (appCache.status) {
  case appCache.UNCACHED: break;          //캐시를 하지 않을 경우에 대한 처리를 한다.
  case appCache.IDLE: break;              //최신의 캐시 이용 상태일 경우에 대한 처리를 한다.
  case appCache.CHECKING: break;          //캐시의 업데이트 체크중일 경우에 대한 처리를 한다.
  case appCache.DOWNLOADING: break;       //캐시의 업데이트중일 경우에 대한 처리를 한다.
  case appCache.UPDATEREADY: break;       //최신 캐시를 이용할 수 있는 상태에 대한 처리를 한다.
  case appCache.OBSOLETE: break;          //캐시가 되지 않은 상태에 대한 처리를 한다.
};
```

➢ **브라우저 상태를 체크하기 위한 브라우저 객체의 onLine 속성과 예제**

***window*.navigator.onLine**	브라우저가 온라인이면 true를 반환하고 그렇지 않으면 false를 반환한다.

```
<script>
  function updateIndicator() {
        document.getElementById('indicator').textContent = navigator.onLine ? '온라인' : '오프라인';
  }
</script>
```

```
<body onload="updateIndicator()" ononline="updateIndicator()" onoffline="updateIndicator()">
    <p>현재 네트워크의 상태는: <span id="indicator">(알려져 있지 않습니다)</span></p>
</body>
```

파일 API

표준화 문서	File API - http://www.w3.org/TR/FileAPI/
표준화 단계	W3C Last Call Working Draft(Editors Draft), (2014-06-30)

소개

- 파일 API는 PC에 있는 파일의 데이터를 직접 스크립트로 접근할 수 있도록 지원한다.
  - 이전 표준화 문서의 파일 API에서는 파일을 읽을 수만 있고 클라이언트에서 직접 편집할 수 없었다.
  - 파일의 표현을 변경할 수는 있었지만, 이러한 편집 내용을 저장하려면 파일 및 편집 내용을 서버에 업로드를 하고 서버에서 모든 작업을 처리한 이후에 변경된 파일을 다운로드 해야만 했다.
  - 최근 표준화 문서에는 블롭(Blob) 인터페이스를 통해서 파일을 생성할 수 있도록 하고 있다.

블롭(Blob) 인터페이스

- 파일(File) 인터페이스가 일반적으로 기본 파일 시스템으로부터 얻어진 파일 데이터를 나타내는 것이라면, 블롭(Blob) 인터페이스는 변경되지 않는 원시 데이터를 나타낸다.
- Blob(Binary Large Object)은 본래 커다란 파일을 부르는 단어로써, 비디오 같은 멀티미디어 데이터를 객체로 다루기 위해서 주로 사용된다.
- 블롭(Blob) 객체는 바이트 시퀀스에 대한 비동기 액세스를 제공한다. 이는 FileReader 객체를 통해서 웹 애플리케이션 내에서 사용할 수 있도록 하기 위한 것이다.
- 블롭을 구성하는 방법은 Blob 생성자를 호출하는 방법과 다른 블롭 데이터 서브 세트를 포함하는 블롭을 생성하기 위한 slice() 메서드를 사용하는 방법이 있다.

블롭(Blob) 인터페이스

Constructor(**sequence<(ArrayBuffer or ArrayBufferView or Blob or DOMString)> blobParts**, optional **BlobPropertyBag options**)	
Interface	**Blob**
속성	**size, type**
메서드	**slice(start, end, contentType), close()**

매개 변수	설 명
blobParts	Blob 객체에 들어가는 데이터 객체의 배열을 나타낸다. blobParts는 지정하는 순서에 상관없이 ArrayBuffer, ArrayBufferView, Blob, 그리고 임의의 문자열이 될 수 있다.
options	새로운 Blob 객체에 대한 속성을 제공하는 BlobPropertyBag 객체를 나타낸다. BlobPropertyBag 객체는 Blob 객체의 MIME 타입을 나타내는(문자열) type 속성을 가지고 있다.

사용 예제

ECMAScript

```
var a = new Blob(); //새로운 Blob 객체를 만든다.
var buffer = new ArrayBuffer(1024); //1024바이트 크기의 ArrayBuffer를 생성한다.
var shorts = new Uint16Array(buffer, 512, 128); //buffer에 기초한 ArrayBufferView 객체를 생성한다.
var bytes = new Uint8Array(buffer, shorts.byteOffset + shorts.byteLength);
var b = new Blob( ["문자열1" + "문자열2"], { type: "text/plain;charset=UTF-8" } );
var c = new Blob( [b, shorts] );          //Blob 객체 b를 이용한다.
var a = new Blob( [b, c, bytes] );        //Blob 객체 b, c, 그리고 ArrayBufferView 객체를 이용.
var d = new Blob( [buffer, b, c, bytes] ); //Blob 객체 b, c, 그리고 ArrayBuffer, ArrayBufferView 객체를 이용.
```

블롭(Blob) 인터페이스

속성과 메서드

***Blob*.size**	(읽기전용) Blob 객체에 포함되어 있는 데이터의 크기를 바이트 단위로 반환한다.
***Blob*.type**	(읽기전용) Blob 객체에 포함되어 있는 데이터의 MIME 타입을 나타내는 문자열을 반환한다.
***Blob*.slice()**	소스 Blob의 지정된 바이트 범위(start에서 end까지)의 데이터를 포함하는 새로운 Blob 객체를 반환한다. 선택적으로 start, end 매개 변수를 지정할 수 있으며, 선택적으로 콘텐츠 타입(contentType)을 지정하여 파일 데이터를 HTTP를 통해서 받았을 때 데이터를 제한하는데 사용할 수 있다.
***Blob*.close()**	이 메소드를 호출하면, 원래의 Blob 객체를 영구적으로 제거한다.

사용 예제

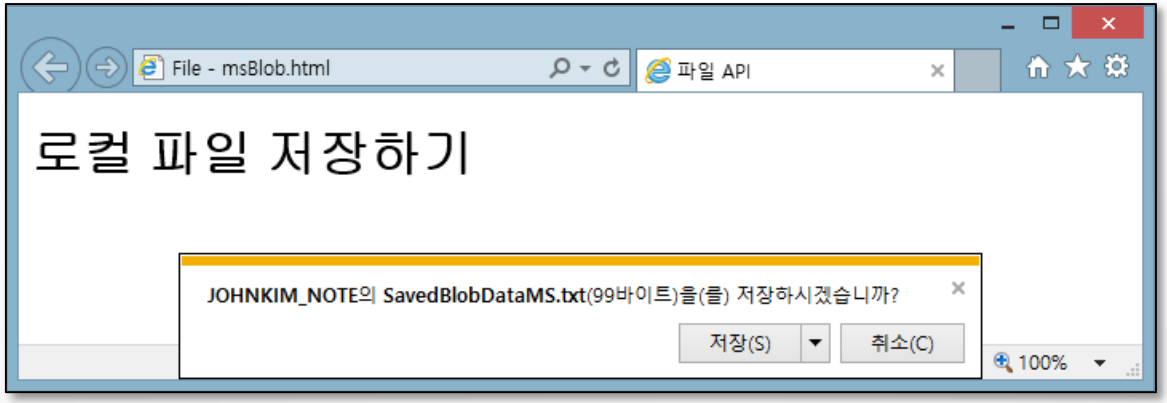
ECMAScript

```
var file = document.getElementById('file').files[0];
if(file) {
    var fileClone1 = file.slice();          //파일의 복사본을 만든다. 아래 방법과 동일하다.
    var fileClone2 = file.slice(0, file.size); //파일 크기만큼의 복사본을 만든다. 위 방법과 동일하다.
    //파일의 중간에서 시작하는 남은 부분(Chunk)의 1/2 조각 파일
    var fileChunkFromEnd = file.slice(-(Math.round(file.size/2)));
    //파일의 처음부터 시작하는 1/2 부분의 조각 파일
    var fileChunkFromStart = file.slice(0, Math.round(file.size/2));
    //파일의 처음부터 마지막 150바이트 이전까지의 조각 파일
    var fileNoMetadata = file.slice(0, -150, "application/experimental");
}
```

블롭(Blob) 인터페이스

예제 File_msBlob.html

```
01 <script>
02 if (typeof Blob !== "undefined") { demoBlobs(); } //Blob이 사용가능한지 점검한다.
03 function demoBlobs(){
04     var blob1 = new Blob(   //첫 번째 블롭 객체를 생성한다.
05             ["블롭을 통해서 로컬 시스템에 파일로 ", "이렇게 내용을 저장할 수 있습니다.₩r₩n"],
06             { type: "text/plain", endings: "native" } );
07     //두 번째 블롭은 첫 번째 블록 배열에 추가한다.
08     var blob2 = new Blob([blob1, "그리고 이렇게 기본 블롭에 데이터를 추가할 수도 있습니다."]);
09     window.navigator.msSaveBlob(blob1, 'SavedBlobDataMS.txt');        //첫 번째 블롭(blob1) 저장
10     window.navigator.msSaveOrOpenBlob(blob2, 'SavedBlobDataMS.txt'); //두 번째 블롭(blob2) 저장
11 }
12 </script>
```



파일(File) 인터페이스

- 파일(File) 인터페이스는 다음과 같이 Blob 기반으로 되어 있고 Blob의 기능을 상속받고 확장한다.
  - File 객체는 Blob 객체가 되며, 파일의 바이트 시퀀스에 대한 접근은 FileReader 객체를 통해서 이루어진다.
  - Blob을 상속받기 때문에 Blob의 속성과 메서드를 사용할 수 있다.

Constructor(Blob fileBits, [EnsureUTF16] DOMString fileName)	
Interface	File : Blob {

속성과 메서드

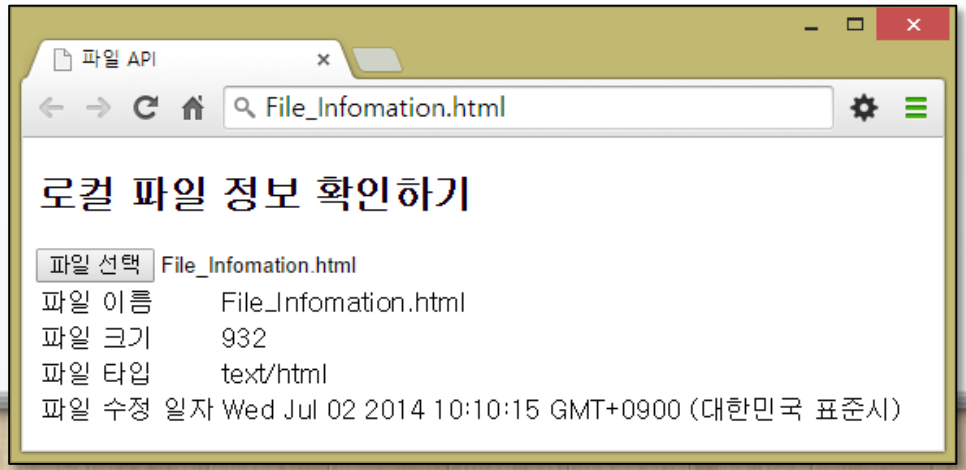
File.name	(읽기 전용) File 객체가 참조하는 파일을 이름을 UTF-16 문자열로 반환한다. 파일에 대한 경로는 포함되지 않는다.
File.lastModifiedDate	(읽기 전용) File 객체가 참조하는 파일을 마지막으로 수정한 날짜(Date)를 반환한다.

파일(File) 인터페이스

예제		File_Information.html

Script	01
	02
	03
	04
	05
	06
	07
	08
	09
	10
	11
	
	12
	13
		document.addEventListener("DOMContentLoaded", function() {
		  var **input** = document.getElementById('**file**');
		  var table = document.getElementById("table");
		
		  input.addEventListener("**change**", function(event) {
		  var ***file*** = event.target.**files[0]**;
		  if (!***file***) { return; }
		  table.innerHTML = "<tr><td> 파일 이름 </td><td>" + ***file.name*** + "</td></tr>";
		  table.innerHTML += "<tr><td> 파일 크기 </td><td>" + ***file.size*** + "</td></tr>";
		  table.innerHTML += "<tr><td> 파일 타입 </td><td>" + ***file.type*** + "</td></tr>";
		  table.innerHTML += "<tr><td> 파일 수정 일자 </td><td>" + ***file.lastModifiedDate*** + "</td></tr>";
		  }, false);
		}, false);

HTML	01 02	<input id="file" type="file" /> <table id="table"></table>



파일 리더(File Reader) 인터페이스

- 파일 리더(File Reader) 인터페이스는 메모리에 File 객체 또는 Blob 객체를 읽기 위한 방법을 제공한다.
  - 진행 이벤트와 이벤트 핸들러 속성을 사용하여 해당 파일이나 블롭 데이터에 비동기로 접근하도록 해준다.

Interface	FileReader : EventTarget
속성	readyState, result, error
메서드	readAsArrayBuffer(blob), readAsText(blob, label), readAsDataURL(blob)
상수값	EMPTY = 0, LOADING = 1, DONE = 2;
이벤트 핸들러	onloadstart, onprogress, onload, onabort, onerror, onloadend

이벤트 핸들러

이벤트 이름	이벤트 발생 시점
onloadstart	데이터를 읽기 시작했을 때 발생한다.
onprogress	blob을 읽는(그리고 복호화(decoding)) 도중에 연속으로 발생한다.
onabort	데이터 읽기가 중단되었을 때 발생한다. abort() 메서드 호출에 의해서도 발생한다.
onerror	데이터 읽기가 실패했을 때 발생한다.
onload	데이터 읽기를 성공적으로 완료했을 때 발생한다.
onloadend	데이터 읽기 요구가(성공이든 실패든 관계 없이) 완료했을 때 발생한다.

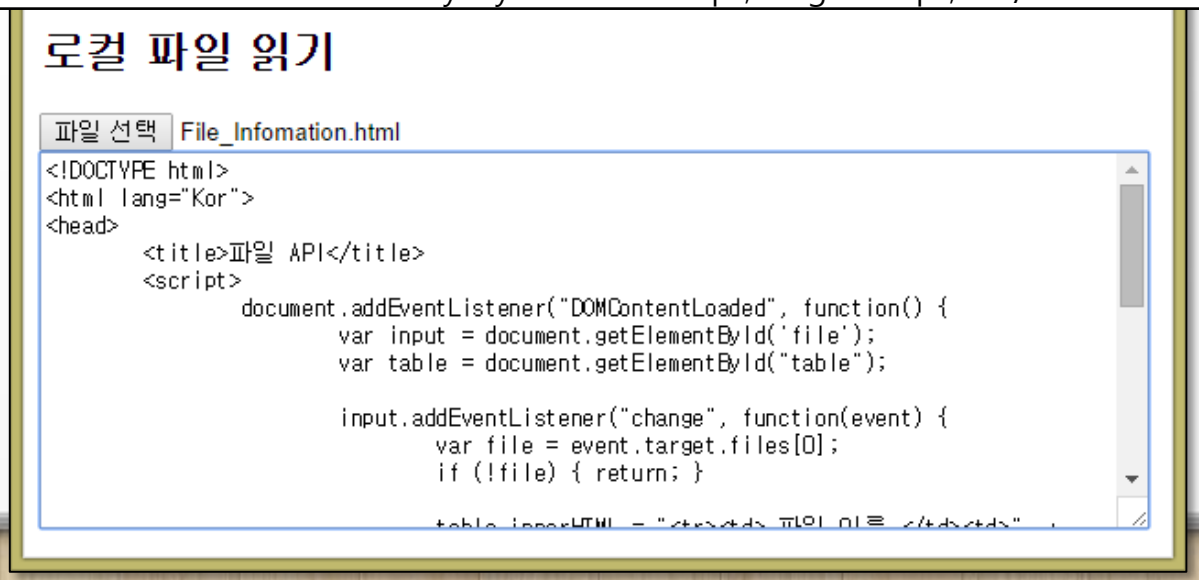
파일 리더(File Reader) 인터페이스

속성과 메서드

FileReader.readyState	(읽기 전용) FileReader 객체의 상태를 가리키는 다음의 상수 값을 반환한다. ✓ EMPTY – File 객체가 생성되었지만, 데이터가 아직 로딩되지 않은 상태 ✓ LOADING – 데이터가 현재 로딩되고 있는 상태 ✓ DONE – 전체 읽기 요구가 완료된 상태
FileReader.result	(읽기 전용) 파일의 읽기 명령이 완료되었을 때의 유효한 콘텐츠를 반환한다.
FileReader.error	(읽기 전용) 파일을 읽는 도중에 발생하는 오류를 반환한다.
FileReader.abort()	파일 읽기를 중단한다.
FileReader.readAsArrayBuffer()	지정된 Blob 객체(또는 File 객체)가 나타내는 파일의 데이터를 읽고 그 데이터를 ArrayBuffer 객체로 변환하여 result 속성에 저장한다.
FileReader.readAsText()	지정된 Blob 객체(또는 File 객체)가 나타내는 파일의 데이터를 읽고 그 데이터를 UTF-8 텍스트로 변환하여 result 속성에 저장한다. 데이터가 EUC-KR이라고 해도 UTF-8로 변환된다. 두 번째 인수 label이 지정되면, 지정된 문자 인코딩 방식으로 변환된다.
FileReader.readAsDataURL()	지정된 Blob 객체(또는 File 객체)가 나타내는 파일의 데이터를 읽고 그 데이터를 Data URL 형식의 문자열로 변환하여 result 속성에 저장한다.

파일 리더(File Reader) 인터페이스

예제		File_Data.html
Script	01	document.addEventListener("DOMContentLoaded", function() {
	02	var **input** = document.getElementById('file');
	03	var view = document.getElementById("content");
	04	
	05	**input**.addEventListener("change", function(event) {
	06	**var file = event.target.files[0];**
	07	if (!file) { return; }
	08	**var reader = new FileReader();** //FileReader 객세를 생성
	09	**reader.readAsText(file, "utf-8");** //데이터를 읽는다.
	10	**reader.onload** = function() { //파일 데이터를 성공적으로 읽기 완료되면
	11	**view.textContent = reader.result;** //텍스트 영역에 내용을 표시한다.
	12	};
	13	}, false);
	14	}, false);
HTML	01	<input id="file" type="file" />
	02	<textarea id="content" readonly style="width:500px; height:200px;"></textarea>



URL 인터페이스

- URL을 특정(img 또는 video 등) 요소의 src 속성에 직접 지정할 수 있다.
  URL 인터페이스는 이런 고유 URL를 생성하거나 삭제하기 위한 메서드들을 제공한다.

Interface	URL
메서드	createObjectURL(blob), createFor(blob), revokeObjectURL(url)

속성과 메서드

URL.createObjectURL()	지정된 Blob 및 File 객체를 지정하면 고유한 URL을 반환한다.
URL.revokeObjectURL()	지정된 Blob URL 문자열을 지정하면 Blob URL 저장소에서 삭제한다. ***myURL*** = window.URL.**createObjectURL**(myBlob); window.URL.**revokeObjectURL**(***myURL***);
URL.createFor()	고유한 Blob URL을 반환한다.

URL 인터페이스

예제		File_Image.html

Script	01
	02
	03
	04
	05
	06
	07
	08
	09
	10
	11
	12
	13
		document.addEventListener("DOMContentLoaded", function() {
		  var ***input*** = document.getElementById('file');
		
		  input.addEventListener("change", function(event) {
		  var **file = event.target.files[0]**;
		  if (!**file**) { return; }
		  var **url** = **window.URL.createObjectURL(file);**
		  var img = document.createElement("img");
		  img.src = url;
		  img.width = 400; img.height = 300;
		  document.body.appendChild(img);
		  }, false);
		}, false);

HTML	01	<input id="**file**" type="file" />

웹 스토리지

표준화 문서	W3C Recommendation(Editors Draft), (2014-05-14)/html5_canvas_CR/
표준화 단계	W3C Last Call Working Draft(Editors Draft), (2014-06-30)

웹 스토리지 개요

쿠키의 단점 및 웹스토리지의 차이점

용량 제한	쿠키에 담을 수 있는 데이터의 한계는 4KB에 불과하다. 즉, 문서나 이메일 같이 큰 것은 들어갈 수 없다. **웹 스토리지는 도메인당 저장할 수 있는 용량을 5MB로 명세서에서 권장하고 있다.**
네트워크 전송부하 및 보안문제	쿠기의 내용은 서버로 요청이 갈 때마다 HTTP 헤더에 담겨서 전송되기 때문에 불필요한 네트워크의 부하를 많이 발생시키게 된다. 또한 이렇게 전송되는 쿠키 데이터를 네트워크 상에서 참조할 수 있기 때문에 암호화되어 있지 않으면 보안 상의 위험이 뒤따른다. **웹 스토리지는 서버로 요청을 하더라도 HTTP 메시지에는 포함되지 않기 때문에 부하를 줄일 수 있다.**
유효기간 제한	쿠키는 유효 기간을 가지고 있는 특징이 있기 때문에 유효 기간이 지난 값은 삭제되어 접근할 수 없다. **웹 스토리지의 하나인 로컬 스토리지에는 유효 기간이 없기 때문에 사용자가 삭제하기 전까지는 내용이 지워지지 않는다.**
세션 문제	브라우저에서 특정 사이트에 로그인을 하고 작업하다가, 새로운 창을 열어서 다른 아이디로 로그인을 한다면, 이전 창에서 로그인한 아이디는 새로운 창의 아이디로 세션이 바뀌게 된다. 즉, 동일한 사이트라 할지라도 다른 정보를 가지고 작업을 할 수 없다는 뜻이다. **웹 스토리지의 하나인 세션 스토리지는 각 창마다 독립적인 데이터가 저장되기 때문에 이런 문제를 해결할 수 있게 된다.**

사용법	일정 시간 유지될 수 있는 값 하나를 쿠키에 저장하려면 다음과 같은 복잡한 코드가 필요하다. 또한 이렇게 저장되어 있는 값을 읽어오기 위해서는 복잡한 내용을 파싱해서 얻어내야 한다. 이러한 복잡성 때문에 코드를 별도로 작성해서 저장하거나 공개된 코드를 활용해서 사용한다.
	웹 스토리지는 다음과 같이 객체를 다루듯이 값을 할당하고 읽으면 된다.
	`localStorage.name = "myName" //값을 저장할 때`
	`var value = localStorage.name; //값을 읽어올 때`

웹 스토리지 개요

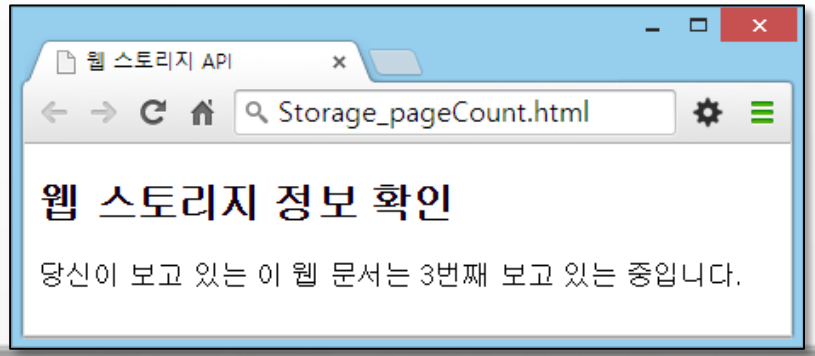
쿠키의 단점 및 웹스토리지의 차이점

데이터	쿠키에는 저장할 수 있는 데이터의 종류가 문자열 형식의 값만 가능하다. 즉, 숫자나 논리값 또는 객체 등을 저장할 수 없기 때문에 해당 값을 사용하기 위해서 저장할 때와 읽을 때 모두 변환을 해야 한다. **웹 스토리지의 경우는 어떤 형식의 데이터 값도 저장이 가능하다.** 예를 들어, 이벤트가 할당되어 이는 웹 문서의 일부를 jQuery등으로 객체화시켜서 웹 스토리지에 저장했다가 사용자가 다음 방문할 때에 서버에 요청하지 않고 웹 스토리지에서 다시 읽어서 웹 문서를 그대로 구성하는 것이 가능하다.

예제		Storage_pageCount.html

Script	01
	02
	03
	04
	05
		<script>
		  if (!localStorage.pageLoadCount) localStorage.pageLoadCount = 0;
		  localStorage.pageLoadCount = parseInt(localStorage.pageLoadCount) + 1;
		  document.getElementById('count').textContent = localStorage.pageLoadCount;
		</script>

HTML	01	<p>당신이 보고 있는 이 웹 문서는 번째 보고 있는 중입니다.



웹 스토리지 인터페이스

Interface	Storage
속성	length
메서드	key(index), getItem(key), setItem(key, value), removeItem(key), clear()

속성과 메서드

length	스토리지에 저장되는 있는 데이터의 수를 반환한다.
key(index)	인덱스(index)를 지정하여 스토리지 저장되어 있는 키 값을 가져온다.
getItem(key)	키(key)에 대응하는 스토리지에 저장되어 있는 값을 가져온다.
setItem(key, value)	키와 데이터(value)를 지정하여 스토리지에 값을 저장한다.
removeItem(key)	키에 대응하는 데이터 값을 삭제한다.
clear()	스토리지에 저장되어 있는 전체 데이터를 삭제한다.

```
if( typeof(Storage) !== "undefined") { // 웹 스토리지 기능을 지원하는 경우
    // 이곳에 세션 및 로컬 스토리지 기능을 구현한다.
} else {
    // 브라우저는 웹 스토리지 기능을 지원하지 않는다
}
```

세션 및 로컬 스토리지

- 세션 스토리지는 도메인마다 별도로 생성이 된다. 그러나, 로컬 스토리지와는 다르게 데이터의 저장 기간이 제한되어 있기 때문에 저장한 데이터를 영구적으로 보관하지 않고 세션이 종료되면 자동 폐기된다.
- 각 세션마다 따로 스토리지가 생성되므로 같은 도메인이라고 해도 다른 윈도우에서 생성이 되면 서로의 스토리지에 접근이 불가능한 것이 특징이다.

Interface	WindowSessionStorage
속성	Storage sessionStorage

Interface	WindowLocalStorage
속성	Storage localStorage

웹 스토리지에 값 할당하기

setItem() 메서드를 사용하는 방법	웹 스토리지에 직접 설정하는 방법
sessionStorage.setItem('key1', 1) sessionStorage.setItem('key2', 2)	sessionStorage.key1 = 1; sessionStorage.key2 = 2;

웹 스토리지에 값 할당하기

setItem() 메서드를 사용하는 방법	웹 스토리지에 직접 설정하는 방법
var key1 = sessionStorage.getItem('key1') var key2 = sessionStorage.getItem('key2')	var key1 = sessionStorage.key1; var key2 = sessionStorage.key2;

세션 및 로컬 스토리지

웹 스토리지에 값 할당하기

값 저장	저당된 값 삭제
sessionStorage.setItem('key1', 1) sessionStorage.setItem('key2', 2)	sessionStorage.removeItem('key1'); //1개의 데이터만 삭제 sessionStorage.clear(); //남아있는 모든 데이터 삭제

예제 Storage_sessionCount.html

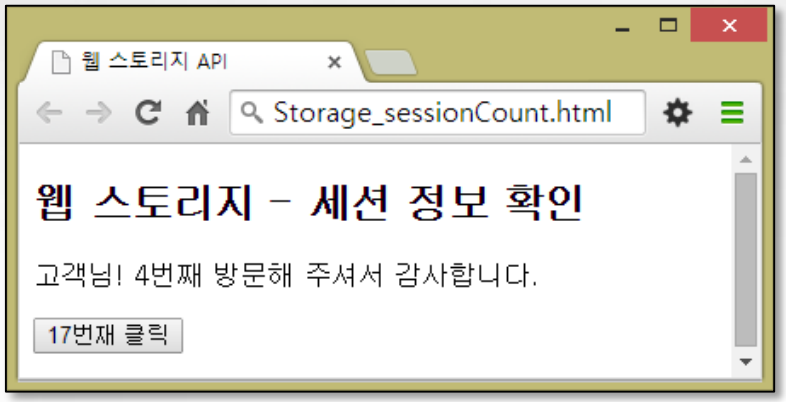
Script

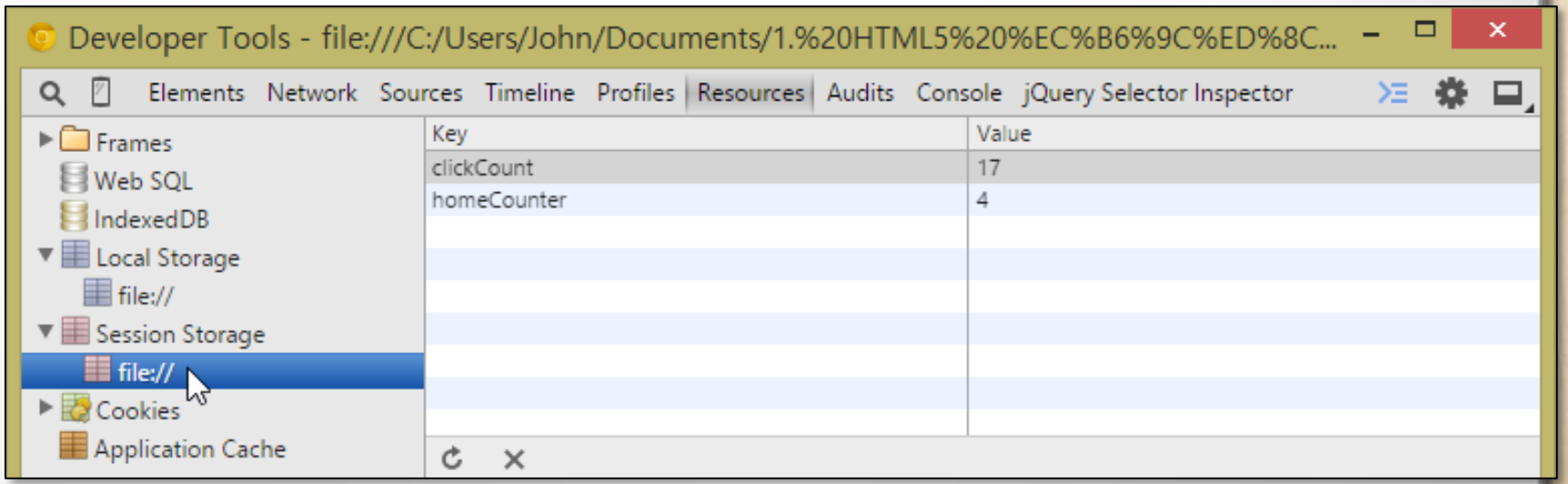
```
01 document.addEventListener("DOMContentLoaded", function() {
02     var count= sessionStorage.getItem("homeCounter");
03     var btn = document.getElementById('btn');
04     if (count== null) {
05             sessionStorage.setItem("homeCounter", 1);
06             document.getElementById(visit).textContent = "첫 ";
07     } else {
08             count++;
09             sessionStorage.setItem("homeCounter", count);
10             document.getElementById(visit).textContent = count;
11     }
12 }, false);
13 function clickCount() {
14     if(sessionStorage.clickCount) {
15             sessionStorage.clickCount = Number(sessionStorage.clickCount) + 1;
16     } else { sessionStorage.clickCount = 1; }
17             btn.textContent = sessionStorage.clickCount + "번째 클릭";
18 }
```

HTML

```
01 <p>고객님! <span id=" visit "></span>번째 방문해 주셔서 감사합니다.</span>
02 <p><button id="btn" onclick="clickCount()" type="button">클릭해 주세요!</button></p>
```

세션 및 로컬 스토리지


웹 스토리지 API
Storage_sessionCount.html
웹 스토리지 - 세션 정보 확인
고객님! 4번째 방문해 주셔서 감사합니다.
17번째 클릭


Developer Tools - file:///C:/Users/John/Documents/1.%20HTML5%20%EC%B6%9C%ED%8C...
Elements Network Sources Timeline Profiles Resources Audits Console jQuery Selector Inspector
Frames
Web SQL
IndexedDB
Local Storage
file://
Session Storage
file://
Cookies
Application Cache
Key
Value
clickCount
17
homeCounter
4

스토리지 이벤트

Interface	StorageEvent : Event
속성	key, oldValue, newValue, url, storageArea

이벤트에서 사용할 수 있는 속성

key	변경된 키를 나타낸다. clear() 메서드가 호출되면 null을 반환한다.
oldValue	변경된 키의 이전 값을 나타내는 것으로, 새로운 키 값이 등록되면 null이 된다.
newValue	변경된 이후의 새로운 값을 나타내는 것으로, 값이 삭제되었다면 null이 된다.
url	키가 변경된 문서의 URL을 나타낸다.
storageArea	영향을 받은 스토리지 객체를 나타낸다.

사용 예제

```
window.addEventListener( "storage", function(event) {
      var key = event.key;
      var newValue = event.newValue;
      var oldValue = event.oldValue;
      var url = event.url;
      var storageArea = event.storageArea;
      //이벤트 핸들 처리.....
} );
```

스토리지 이벤트

실습하기 – Storage_Study01_localEvent.html

Script

```
01  document.addEventListener("DOMContentLoaded", function(event) {
02    var key = document.getElementById("key");
03    var value = document.getElementById("value");
04    var add = document.getElementById("add");
05    var remove = document.getElementById("remove");
06    var clear = document.getElementById("clear");
07    var content = document.getElementById("content");
08    var msg = document.getElementById("event");
09
10    add.addEventListener("click", function(event) { //스토리지에 데이터 저장
11                  if(key.value !== "") {
12                    localStorage.setItem(key.value, value.value);
13                    refreshContents();
14                  }
15    });
16
17    remove.addEventListener("click", function(event) { //스토리지에서 데이터 삭제
18                  if (key.value !== "") {
19                    localStorage.removeItem(key.value);
20                    refreshContents();
21                  }
22    });
23
24    clear.addEventListener("click", function(event) { //스토리지 전체 데이터 삭제
25                  localStorage.clear();
26                  refreshContents();
27    });
```

스토리지 이벤트

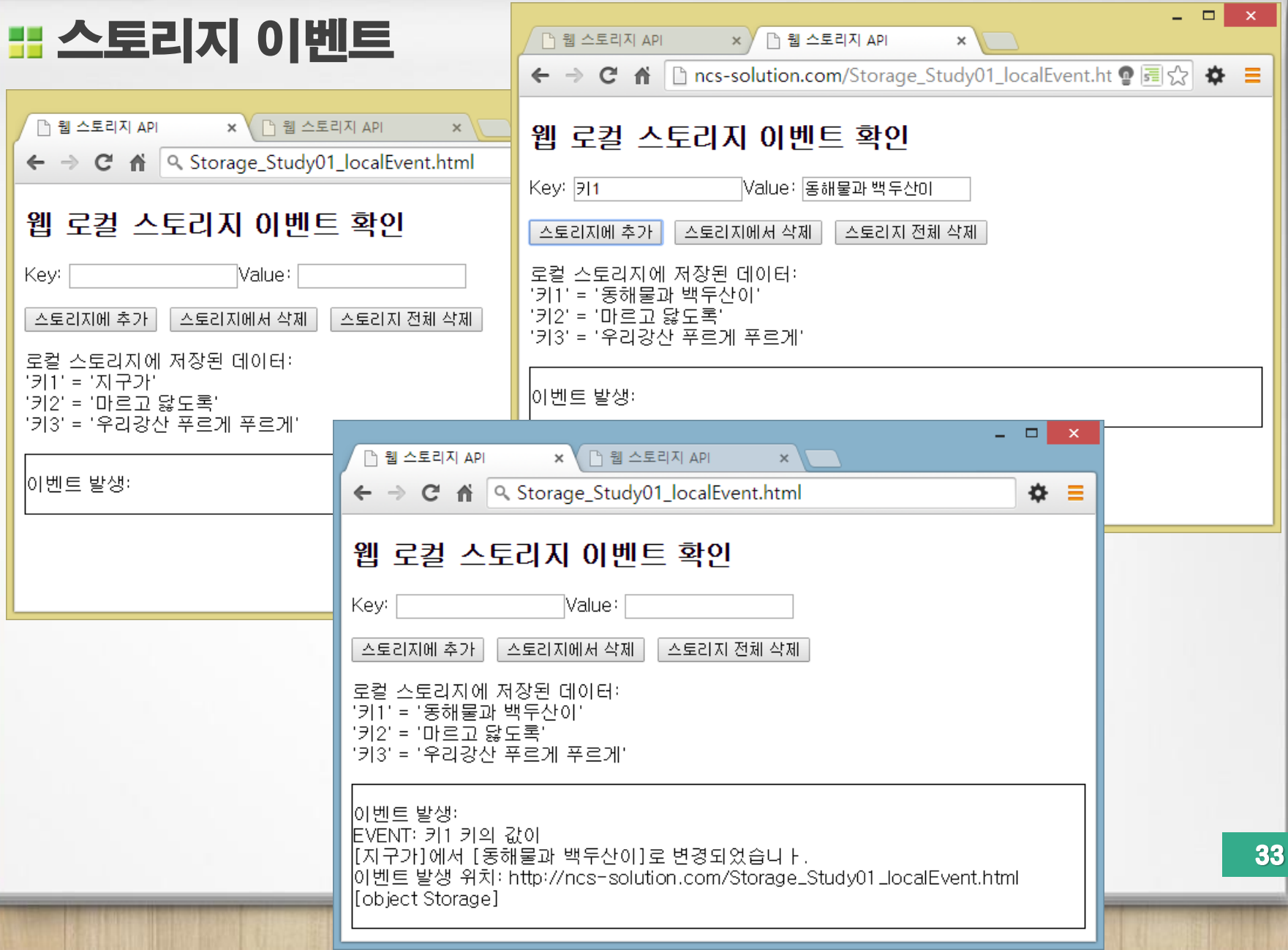
Script

```
29  window.addEventListener("storage", function(event) { //스토리지에 이벤트 등록
30      var key = event.key;
31      var newValue = event.newValue;
32      var oldValue = event.oldValue;
33      var url = event.url;
34      var storageArea = event.storageArea;
35
36      msg.innerHTML ="EVENT: " + key + " 키의 값이 <br>[" + oldValue + "]에서 [" + newValue +
37  "]로 변경되었습니다.<br>" + "이벤트 발생 위치: " + url + "<br>" + storageArea;
38
39      refreshContents();
40  });
41
42  function refreshContents() {
43      var str = "";
44      for ( var i = 0, len = localStorage.length; i < len; i++ ) {
45              var key = localStorage.key(i);
46              var value = localStorage.getItem(key);
47
48              str += "'" + key + "' = '" + value + "'<br />";
49      }
50      content.innerHTML = str;
51      key.value = ""; value.value = "";
52  }
53
54  refreshContents();
55  });
```

HTML

```
01  <p>Key:  <input type="text" id="key" />Value: <input type="text" id="value" /><br /></p>
02  <input type="button" id="add" value="스토리지에 추가" /> 
03  <input type="button" id="remove" value="스토리지에서 삭제" /> 
04  <input type="button" id="clear" value="스토리지 전체 삭제" /><br />
05  <div><p>로컬 스토리지에 저장된 데이터:<br /><span id="content"></span></p><div>
06  <div style="border:1px solid black"><p>이벤트 발생:<br><span id="event"></span></p></div>
```

스토리지 이벤트


웹 스토리지 API
웹 스토리지 API
Storage_Study01_localEvent.html
웹 로컬 스토리지 이벤트 확인
Key:
Value:
스토리지에 추가
스토리지에서 삭제
스토리지 전체 삭제
로컬 스토리지에 저장된 데이터:
'키1' = '지구가'
'키2' = '마르고 닳도록'
'키3' = '우리강산 푸르게 푸르게'
이벤트 발생:
웹 스토리지 API
웹 스토리지 API
ncs-solution.com/Storage_Study01_localEvent.ht
웹 로컬 스토리지 이벤트 확인
Key: 키1
Value: 동해물과 백두산이
스토리지에 추가
스토리지에서 삭제
스토리지 전체 삭제
로컬 스토리지에 저장된 데이터:
'키1' = '동해물과 백두산이'
'키2' = '마르고 닳도록'
'키3' = '우리강산 푸르게 푸르게'
이벤트 발생:
웹 스토리지 API
웹 스토리지 API
Storage_Study01_localEvent.html
웹 로컬 스토리지 이벤트 확인
Key:
Value:
스토리지에 추가
스토리지에서 삭제
스토리지 전체 삭제
로컬 스토리지에 저장된 데이터:
'키1' = '동해물과 백두산이'
'키2' = '마르고 닳도록'
'키3' = '우리강산 푸르게 푸르게'
이벤트 발생:
EVENT: 키1 키의 값이
[지구가]에서 [동해물과 백두산이]로 변경되었습니다.
이벤트 발생 위치: http://ncs-solution.com/Storage_Study01_localEvent.html
[object Storage]

다음 학습

1. 웹 워커
2. 웹 소켓
3. 위치 정보를 위한 Geolocation