



CHAPTER 14.

HTML5 웹 스토리지, 파일 API, 웹 소켓





웹 스토리지

- 웹스토리지(web storage)는 클라이언트 컴퓨터에 데이터를 저장하는 메카니즘
- 웹스토리지는 쿠키보다 안전하고 속도도 빠르다.
- 약 5MB 정도까지 저장이 가능하다.
- 데이터는 키/값(key/value)의 쌍으로 저장





localStorage와 sessionStorage

- localStorage 객체
 - 만료 날짜가 없는 데이터를 저장한다.
 - 도메인이 다르면 서로의 로컬 스토리지에 접근할 수 없음.
- sessionStorage 객체
 - 각 세션(하나의 윈도우)마다 데이터가 별도로 저장
 - 해당 세션이 종료되면 데이터가 사라진다.



localStorage API

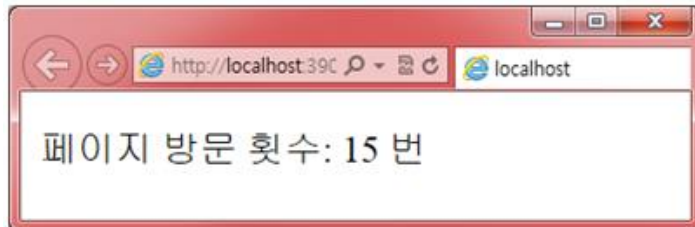
속성/메서드	설명
length	저장된 변수의 개수
setItem(key, value)	key/value를 로컬 스토리지에 저장
getItem(key)	key와 연관된 값을 반환
removeItem(key)	key/value를 로컬 스토리지에서 삭제
clear()	모든 key/value를 삭제

- 아래 3 문장은 동일한 결과
 - `localStorage.setItem("pageLoadCount", 0);`
 - `localStorage["pageLoadCount"] = 0;`
 - `localStorage.pageLoadCount = 0;`

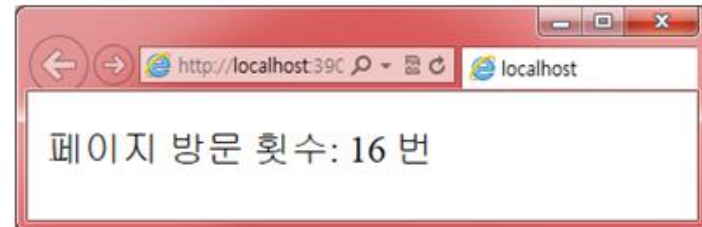


localStorage 예제

```
<!DOCTYPE html>
<html>
<head></head>
<body>
  <p>
    페이지 방문 횟수: <span id="count"> </span>번
  </p>
  <script>
    if (!localStorage.pageLoadCount)
      localStorage.pageLoadCount = 0;
    localStorage.pageLoadCount = parseInt(localStorage.pageLoadCount)
+ 1;
    document.getElementById('count').textContent =
localStorage.pageLoadCount;
  </script>
</body>
</html>
```



->





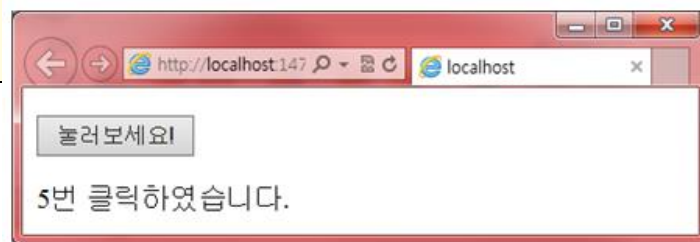
버튼을 클릭한 횟수 저장

```
<!DOCTYPE html>
<html>
<head></head>
<body>
  <p>
    <button onclick="incrementCounter()" type="button">눌러보세요!
  </button>
</p>
<div id="target"></div>
```

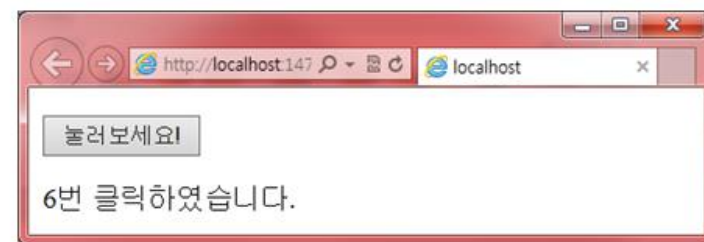


버튼을 클릭한 횟수 저장

```
<script>
function incrementCounter() {
    if (('localStorage' in window) && window['localStorage'] !== null) {
        if (localStorage.count) {
            localStorage.count++;
        }
        else {
            localStorage.count = 1;
        }
        document.getElementById("target").innerHTML =
        localStorage.count + "번 클릭하였습니다.";
    }
    else {
        document.getElementById("target").innerHTML =
        "브라우저가 웹스토리지를 지원하지 않습니다.";
    }
}
</script>
</body>
</html>
```



->





sessionStorage 예제

```
<!DOCTYPE html>
<html>
<head></head>
<body>
  <p>
    <button onclick="incrementCounter()" type="button">눌러보세요!
  </button>
</p>
<div id="target"></div>
```

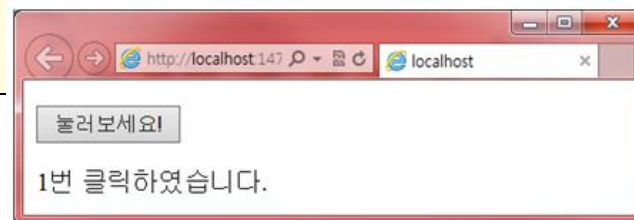
참고:

- **sessionStorage** 객체는 단지 하나의 세션에서만 데이터를 저장한다는 점을 제외하고는 **localStorage** 객체와 동일
- 데이터는 사용자가 브라우저 창을 닫을 때 삭제

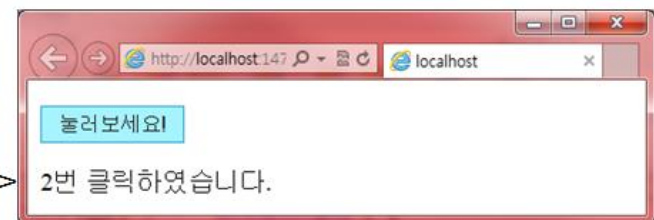


sessionStorage 예제

```
<script>
function incrementCounter() {
  if (('sessionStorage' in window) && window['sessionStorage'] !==
null) {
    if (sessionStorage.count) {
      sessionStorage.count++;
    }
    else {
      sessionStorage.count = 1;
    }
    document.getElementById("target").innerHTML =
sessionStorage.count + "번 클릭하였습니다.";
  }
  else {
    document.getElementById("target").innerHTML =
"브라우저가 웹스토리지를 지원하지 않습니다.";
  }
}
</script>
</body>
</html>
```



->





파일 API

- 파일 API: 웹브라우저가 사용자 컴퓨터에 있는 로컬 파일들을 읽어올 수 있도록 해주는 API
- PC에서 실행되는 일반적인 프로그램처럼 동작(웹 애플리케이션)
- 파일 API의 가장 전형적인 응용 분야는 사용자가 파일을 선택하여서 원격 서버로 전송하는 작업
- 파일 API에서 사용되는 객체는 File, FileReader
 - File 객체는 로컬 파일 시스템에서 얻어지는 파일 데이터를 나타낸다.
 - FileReader 객체는 이벤트 처리를 통하여 파일의 데이터에 접근하는 메소드들을 제공하는 객체이다.



File 객체 주요 속성

속성	설명
<code>name</code>	파일의 이름
<code>size</code>	파일의 크기(단위: 바이트)
<code>type</code>	파일의 타입(MIME type)
<code>lastModifiedDate</code>	최종 변경 날짜



파일 API 예제

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML File API </title>
  <script>
    function readFile() {
      if (!window.File || !window.FileReader) {
        alert('File API가 지원되지 않습니다.');
```

return

```
      }
      var files = document.getElementById('input').files;
      if (!files.length) {
        alert('파일을 선택하십시오!');
```

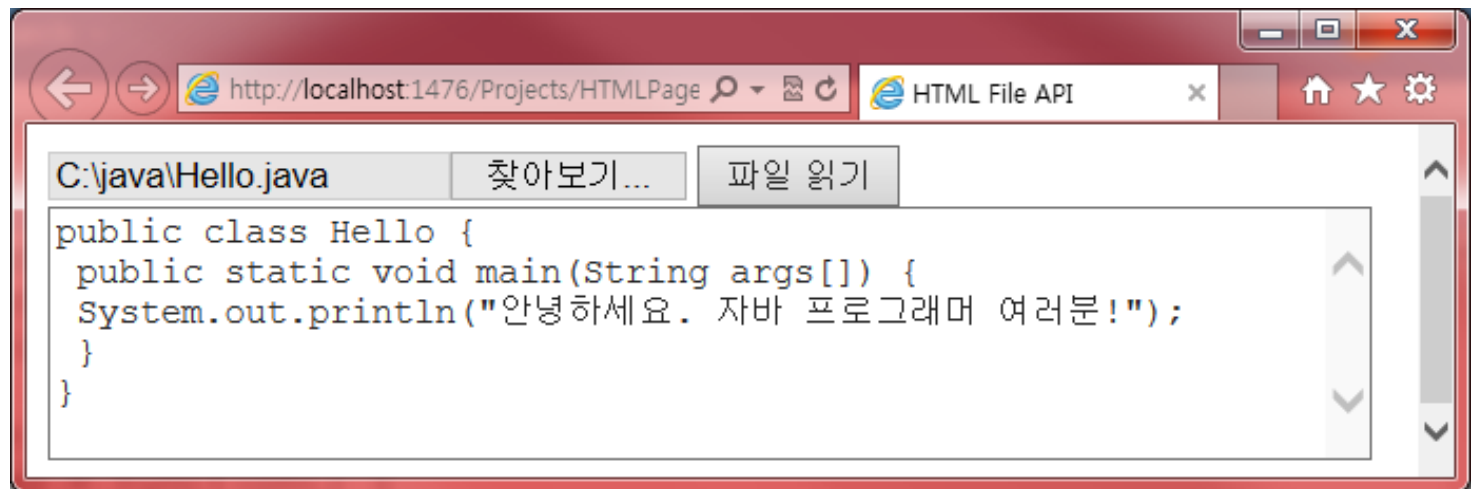
return;

```
      }
      var file = files[0];
      var reader = new FileReader();
      reader.onload = function () {
        document.getElementById('result').value = reader.result;
      };
      reader.readAsText(file, "euc-kr");
    }
  </script>
```



파일 API 예제

```
</head>
<body>
  <input type="file" id="input" name="input">
  <button id="readfile" onclick="readFile()">파일 읽기</button><br />
  <textarea id="result" rows="6" cols="60"> </textarea>
</body>
</html>
```



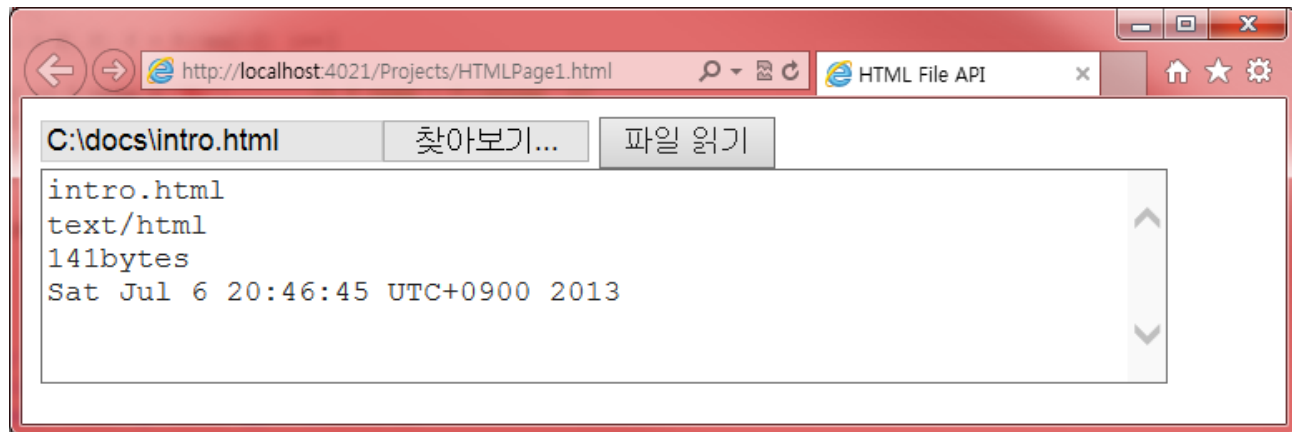


파일 정보 표시 예제

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML File API </title>
  <script>
    function readFile() {
      var files = document.getElementById('input').files;
      output = "";
      for (var i = 0, f; f = files[i]; i++) {
        output += f.name + "\n";          /* f.name - Filename */
        output += f.type + "\n";         /* f.type - File Type */
        output += f.size + "bytes\n"     /* f.size - File Size */
        output += f.lastModifiedDate + "\n"; /* f.lastModifiedDate */
      }
      document.getElementById('result').value = output;
    }
  </script>
</head>
<body>
  <input type="file" id="input" name="input">
  <button id="readfile" onclick="readFile()">파일 읽기</button><br />
  <textarea id="result" rows="6" cols="60"> </textarea>
</body>
</html>
```



실행 결과



웹브라우저로 실행



애플리케이션 캐시

- 애플리케이션이 사용하는 파일들을 클라이언트의 캐시(cache)에 저장
- 애플리케이션 캐시는 다음과 같은 세 가지 장점을 제공한다.
 - 오프 라인 상태일 때도 사용자는 웹 애플리케이션을 사용할 수 있다
 - 캐시된 파일은 더 빨리 로드되어서 그만큼 속도가 빨라진다.
 - 서버 부하가 감소된다.



시계 예제

*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>
  <button onclick="setClock()">시계시작 </button>
  <br />
  <output id="clock"></output>
</body>
</html>
```

clock.css

```
output {
  font: 2em sans-serif
}
```

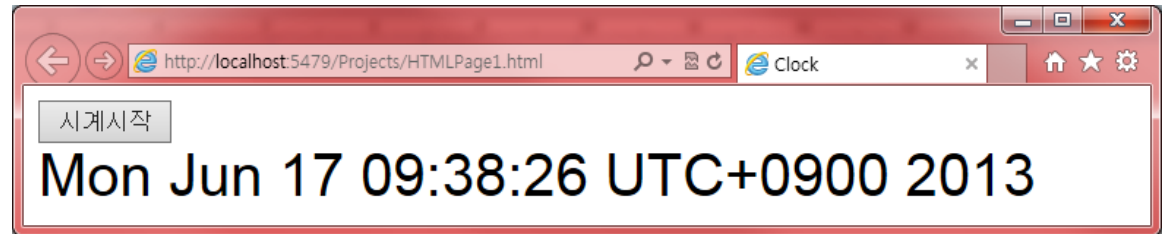
clock.js

```
function setClock() {
  var now = new Date();
  document.getElementById('clock').innerHTML = now;
  setTimeout('setClock()', 1000);
}
```



실행 결과

인터넷 연결이
끊기면 시계는
동작할까요?



웹브라우저로 실행





매니페스트 파일

clock.appcache

CACHE MANIFEST

clock.html

clock.css

clock.js



복잡한 매니페스트 파일

CACHE MANIFEST

2010-06-18:v2

반드시 캐시해야할 파일

CACHE:

index.html

stylesheet.css

images/logo.png

scripts/main.js

사용자가 반드시 온라인이어야 하는 리소스

NETWORK:

login.php

만약 main.jsp 가 접근될 수 없으면 static.html로 서비스한다.

다른 모든 .html파일 대신에 offline.html로 서비스한다.

FALLBACK:

/main.jsp /static.html

*.html /offline.html



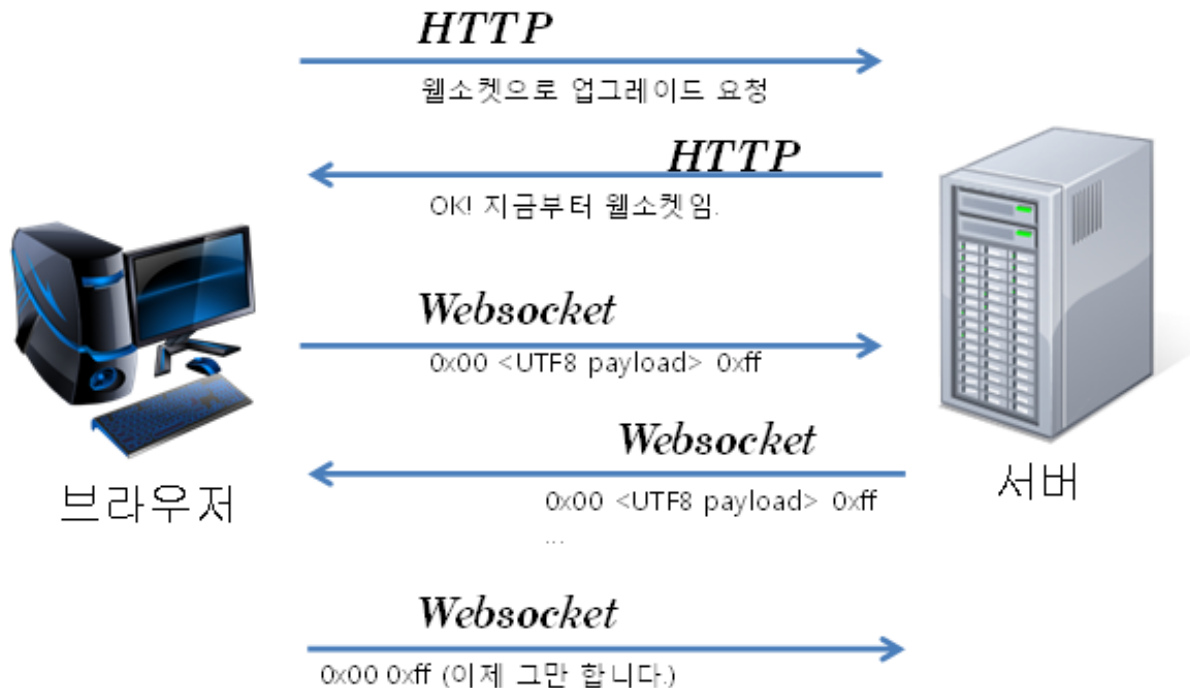
캐시 업데이트

- 일단 애플리케이션이 오프라인이 되면 다음 중 하나가 발생할 때까지 캐시 상태를 유지
 - 사용자가 브라우저의 캐시를 지운다.
 - 매니페스트 파일이 수정된다. (주석 줄의 내용이 변경되어도 됨)
 - 애플리케이션 캐시가 프로그램으로 업데이트된다.
- 참고
 - 일단 파일이 캐시되면, 서버에서 파일이 변경되더라도 브라우저는 캐시된 버전을 계속해서 보여준다. 따라서 캐시를 업데이트하도록 하려면 매니페스트 파일을 변경해야 한다.



웹 소켓

- 웹 소켓(Web Socket)은 웹 애플리케이션을 위한 차세대 양방향 통신 기술
- 애플리케이션은 HTTP의 답답한 구속에서 벗어나서 TCP/IP가 제공하는 모든 기능을 사용할 수 있다(근본적으로 HTTP는 페이지 다운로드만 허용).





소켓이란?

- 소켓은 소프트웨어 수준에서의 통신 접속점으로 네트워크 응용 프로그램들은 소켓을 중심으로 데이터를 송수신한다.
 - 즉 어플리케이션이 네트워크를 접속할 수 있게 하는 인터페이스.
 - 네트워크 어플리케이션이 보낸 데이터는 소켓을 거쳐 운영체제상의 TCP/IP 소프트웨어에 전달됨. 그 후 랜카드를 거쳐 네트워크에 전달된다.
 - 소켓의 구현을 통해 네트워크 계층과 전송 계층의 캡슐화가 가능해짐



- 웹 소켓을 사용할 때 가장 중요한 클래스인 WebSocket
 - `var Socket = new WebSocket(url, [protocol]);`
 - url: 연결하고 싶은 서버의 URL (http: 대신 ws:로 시작)
 - protocol: 웹 소켓 프로토콜
- 웹 소켓의 이벤트

이벤트	설명
onopen	소켓 연결이 확립되면 발생
onclose	연결이 종료되면 발생
onerror	통신에서 오류가 있을 때 발생
onmessage	서버로부터 데이터가 도착하면 발생

- 웹 소켓의 메서드

메서드	설명
<code>send(data)</code>	웹 소켓을 통하여 데이터를 보낸다.
<code>close()</code>	연결을 해제한다.



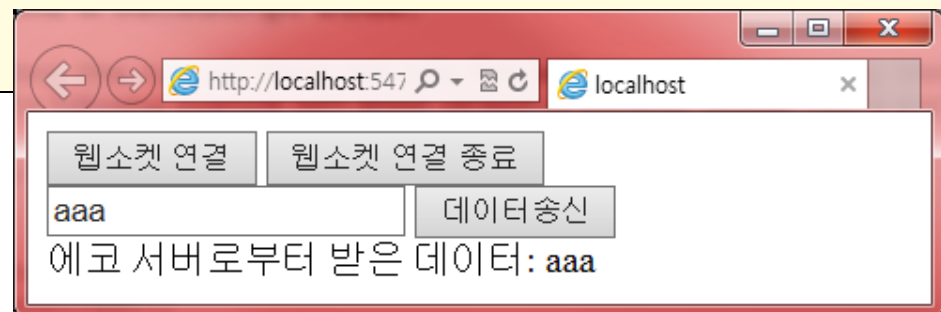
예제

```
<!DOCTYPE HTML>
<html>
<head>
  <script>
    var ws;
    function open() {
      if ("WebSocket" in window) {
        ws = new WebSocket("ws://echo.websocket.org");
        ws.onopen = function () {
          alert("웹소켓 오픈 성공");
        };
        ws.onmessage = function (evt) {
          var msg = evt.data;
          document.getElementById("result").innerHTML = msg;
        };
        ws.onclose = function () {
          alert("웹소켓 연결 해제");
        };
      }
      else {
        alert("웹소켓이 지원되지 않음!");
      }
    }
  }
</script>
</head>
</html>
```



예제

```
function send() {  
    ws.send(document.getElementById("data").value);  
}  
function quit() {  
    ws.close();  
}  
</script>  
</head>  
<body>  
    <button onclick="open()">웹소켓 연결</button>  
    <button onclick="quit()">웹소켓 연결 종료</button><br />  
    <input type="text" id="data" />  
    <button onclick="send()">데이터 송신</button><br />  
    에코 서버로부터 받은 데이터:  
    <output id="result"></output>  
</body>  
</html>
```





Q & A

