14강. HTML API [1] : 오프라인 웹, 파일 접근, 웹 스토리지

1. 오프라인 웹

1.1 오프라인 웹 애플리케이션

- 오프라인 상태에서도 사용이 가능한 애플리케이션이다. 온라인 지원이 안 되는 곳에서도 웹에 접근하여 사용이 가능하다. 즉 파일들을 클라이언트에서 캐시로 저장하여 사용하므로 웹 문서의 접근이 빠르다.

1.2 매니페스트 파일

- 오프라인에서도 웹 사이트 이용이 가능하도록 하기 위해서는 여러 문서와 파일들을 캐시로 저장해야 하는데, 캐시에 저장할 자원의 목록이 매니페스트 파일이다. 어떠한 파일을 캐시로 저장해야 하는지를 지정하며 서버로부터 웹 문서들과 함께 다운로드 되는 파일이다.

CACHE MANIFEST #Cache Section	필수적으로 첫 줄에 포함되어야 하는 내용 설명문부분으로 #로 시작하는 문자열
CACHE: news.html news.js news.css image.jpg	기본 값을 나타내는 부분으로 클라이언트에서 캐시되어야 할 파일들을 지정한다. (오프라인에서도 해당 파일에 접근이 가능하다.)
#Network Section	
NETWORK: new_news.html image_check.jpg	반드시 온라인 상태에서만 접근할 수 있는 파일을 지정한다.
#Fallback Section	
FALLBACK: ./news image_notice.jpg	대체되는 자원을 지정하는 부분이다. (클라이언트에서 서버로 페이지 요청을 하였을 경우 해당 페이지가 존재하지 않을 때 대신 표시할 자원을 지정한다.)

1.3 캐시 매니페스트 작성

- 매니페스트 파일은 클라이언트에서 접근할 수 있어야 하기 때문에 파일의 MIME 타입을 웹 서버의 mime.types 파일에 지정해야 한다.

text/cache-manifest	manifest

- 애플리케이션 문서에서 매니페스트 파일을 다음과 같이 지정한다.

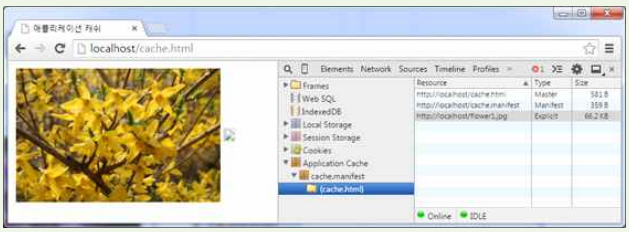
```
<!DOCTYPE html>
<html manifest="매니페스트 파일 이름(예: cache.manifest)">
  ...
</html>
```

- 매니페스트 파일을 지정하면 해당 문서 호출 이후부터 캐시가 동작하게 된다. 그리고 해당 문서는 CACHE 섹션에서 지정하지 않아도 자동으로 캐시가 된다.

1.4 오프라인 접근 예제

cache.html

```
<!DOCTYPE html>
<html manifest="cache.manifest">
<head></head>
<body>  <div>  <table>
    <tr>
      <td><img src="flower1.jpg"></td>
      <td><img src="flower2.jpg"></td>
    </tr>
  </table>  </div></body>
</html>
```



오프라인에서의 실행결과

cache.manifest

```
CACHE MANIFEST
# Version 1.0.0.0

CACHE:
flower1.jpg

NETWORK:
flower2.jpg
```

1.5 이벤트 처리

- 좀더 세세한 동작 제어를 위해서는 이벤트나 캐시의 상태에 따른 처리가 필요하다.
- 매니페스트가 업데이트되고 유지되도록 여러 이벤트가 applicationCache 객체에서 발생하여 사용자에게 캐시 업데이트의 상태에 대해 적절하게 알릴 수 있다. applicationCache 객체에서 status속성 및 update(), swapCache(), abort() 메소드를 사용 할 수 있다.
- 애플리케이션 캐시 상태에 따른 처리로 status 속성을 사용하여 현재의 캐시 상태를 확인할 수 있다.

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

- applicationCache 객체에서 발생하는 이벤트

이벤트	설명
checking	업데이트 여부를 확인하거나,처음으로 매니페스트를 다운로드 하려고 할 때
error	오류로 인하여 업데이트 종료
noupdate	매니페스트가 업데이트되지 않음
downloading	브라우저에서 업데이트할 것을 찾아서 가져오고 있을 때
progress	매니페스트에 기록된 자원을 다운로드 하려고 할 때
updateready	매니페스트에 기록된 자원이 새롭게 다운로드 되었을 때
cached	자원이 다운로드 되어 캐시에 저장된 상태
obsolete	매니페스트를 가져오는 데 실패하여 캐시가 제거될 때

- 이벤트에 따른 처리

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

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

1.6 브라우저의 온라인/ 오프라인 상태 검사

- window.navigator.onLine 속성

```
<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>
</body>
```

2. 파일 접근

2.1 파일 API

- 웹 개발자가 확장 또는 플러그인 없이 안전하게 클라이언트 컴퓨터의 로컬 파일에 접근 가능하다. 읽기전용으로 수정 및 삭제가 불가하다. 사용자가 드래그 앤 드롭한 파일이나 input 요소의 type="file"에서 선택한 파일은 읽기 가능하다.

2.2 파일 인터페이스: 파일 정보 얻기

- 스크립트를 통해서 파일에 접근하기 위해서는 FileList라는 객체를 통해야 한다. 만일 드래그 앤 드롭이라면 드롭 이벤트 객체에서 얻을 수 있는 DataTransfer 객체의 files 속성으로 가져와야 하고, input 요소라면 input 요소 객체의 files 속성에서 가져온다.
- 정보를 알고자 하는 파일의 객체를 구하기 위해서는 선택한 파일의 객체가 저장되어 있는 FileList객체를 구하면 되는데, FileList 객체는 배열형식으로 되어 있기 때문에 인덱스를 지정해서 파일 객체를 가져온다.
- FileList 객체는 드래그 앤 드롭에서는 드롭 되었을 때 얻을 수 있고, input 요소에서는 사용자가 파일을 선택했을 때 change 이벤트를 통해서 얻을 수 있다.
- 파일과 관련된 정보는 name(파일명), type(파일속성), size(파일크기), lastModifiedDate (파일이 마지막으로 수정된 날짜) 속성을 통해 얻을 수 있다.

예제 파일 정보 확인

```
<!DOCTYPE html><html><head>
<script>
function fileinfo(){
 var file = document.getElementById("file").files[0];
var table = document.getElementById("table");
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 날짜 </td><td>" + file.lastModifiedDate
+ "</td></tr>";
  }
</script>
</head>
<body>
  <h2>로컬 파일 정보 확인하기</h2>
  <input id="file" type="file" onchange="fileinfo()">
  <table id="table"></table>
</body>
</html>
```

2.3 파일 리더 인터페이스: 파일 내용 읽기

- 메모리에 File 객체를 읽기 위한 방법으로 FileReader 객체 사용한다.
  1. 객체를 생성한다. var 변수 = new FileReader();
  2. 객체의 메서드를 사용하여 파일 내용을 문자열로 변환하여 저장한다.
  3. 이벤트 처리를 통해 변환하여 저장된 파일의 내용을 반환한다.
- FileReader 객체의 메서드 및 속성은 다음과 같다.

readyState : FileReader 객체의 상태를 반환한다.
result : 읽기가 완료되었을 때의 유효한 콘텐츠(파일 내용)를 반환한다.
error : 읽기 도중에 발생하는 오류를 반환한다.
abort() : 파일 읽기를 중단한다.
readAsArrayBuffer(file) : 파일의 내용을 읽어 ArrayBuffer 객체로 변환하여 저장한다.
readAsText(file, encoding) : 파일의 내용을 읽어 지정한 인코딩 방식(기본: URF-8)의 텍스트로 저장한다.
readAsDataURL(file) : 파일의 내용을 읽어 Data URL 형식의 문자열로 변환하여 저장한다.

- FileReader 객체에서 발생하는 이벤트 핸들러

onloadstart : 데이터를 읽기 시작했을 때 발생한다.
onprogress : 데이터를 읽고 있는 도중에 연속으로 발생한다.
onabort : 데이터 읽기가 중단(abort() 호출)되었을 때 발생한다.
onerror : 데이터 읽기가 실패했을 때 발생한다.
onload : 데이터 읽기를 성공적으로 완료했을 때 발생한다.
onloadend : 데이터 읽기 요구가 완료(성공/실패와는 무관)했을 때 발생한다.

예제 파일 내용 읽기

```
<!DOCTYPE html> <html><head>
<script>
document.addEventListener("DOMContentLoaded", function() {
var input = document.getElementById('file');
var view = document.getElementById("content");
input.addEventListener("change", function(event) {
        var file = event.target.files[0];
        if (!file) { return; }
        var reader = new FileReader();      //FileReader 객체를 생성
        reader.readAsText(file, "utf-8"); //데이터를 읽는다.
        reader.onload = function() { //파일 데이터를 성공적으로 읽기 완료되면
        view.textContent = reader.result; //텍스트 영역에 내용을 표시한다.
        };
         reader.onerror = function(event)  {
           switch(event.target.error.code)  { //FileReader 객체의 오류 값
                  case error.NOT_FOUND_ERR:
                      alert("읽을 파일을 찾지 못하였습니다..");  break;
                   case error.SECURITY_ERR:
                      alert("보안상 안전하지 않습니다..");  break;
                   case error.ABORT_ERR:
                      alert("읽기가 중지되었습니다.");  break;
                   case error.NOT_READABLE_ERR:
                      alert("읽기 권한이 없습니다.");  break;
                   case error.ENCODING_ERR:
                      alert("파일 용량이 상한을 초과하였습니다.");  break;
                      }    }; }, false);
                }, false);
        </script>
</head>
<body>
<h2>로컬 파일 읽기</h2>
<input id="file" type="file" />
<textarea id="content" readonly style="width:600px; height:200px;"></textarea>
</body>
</html>
```

2.4 URL 인터페이스

- 드래그 앤 드롭이나 input 요소에서 가져온 파일에 고유 URL를 생성하거나 삭제하기

위한 메서드들을 제공한다. 생성된 URL을 특정 요소(img, video 등)의 src 속성에 직접 지정 가능하다.

예제

```
<!DOCTYPE html> <html><head>
<script>
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); //고유한 URL을 생성
        var img = document.createElement("img");
        img.src = url
        img.width = 400; img.height = 300;
        document.body.appendChild(img);
    }, false);} , false);
</script></head><body><input id="file" type="file" /></body></html>
```

3. 웹 스토리지

3.1 웹 스토리지

- 기존에는 클라이언트에 간단한 정보를 저장하기 위해서 주로 쿠키(cookie)를 사용하였다. 일반적으로 서버에서는 여러 웹 사이트를 옮겨 다니는 사용자 정보를 알아내기 위해서 쿠키들을 넣어두고 이 값들을 활용한다.
- 웹 스토리지는 클라이언트에 데이터를 저장하기 위한 저장 영역이다.

	쿠키	웹 스토리지
저장 용량	4KB	도메인당 5MB로 명세서에서권장
네트워크 전송 부하 및 보안	서버로 요청이 갈 때마다 HTTP 헤더에 담겨서 전송하여 불필요한 네트워크 부하 및 보안 취약하다.	서버로 요청을 하더라도 HTTP 메시지에는 포함되지 않아 부하를 줄일 수 있다.
유효 기간	유효 기간이 존재하여 자동 삭제된다.	유효 기간이 없어 사용자에 의해 삭제된다.
세션 문제	동일한 사이트라 할지라도 다른 정보를 가지고 작업하는 것이 불가하다.	세션 스토리지는 각 창마다 독립적인 데이터가 저장된다.
사용 방법	복잡	localStorage.name = "myName"; var value = localStorage.name;
데이터 형식	문자열 형식의 값만 저장 가능	어떠한 형식의 데이터 값도 저장 가능

- 웹 스토리지 저장 데이터는 (key, value)로 구성되어 있다. 용도에 따라 로컬 스토리지,

세션 스토리지로 나눌 수 있다.

- 세션 스토리지와 로컬 스토리지의 사용 방법은 동일하고, 의미와 동작의 차이가 있다.

로컬 스토리지 localStorage	세션 스토리지 sessionStorage
• 데이터 저장 기간에 제한 없어 영구적 보관이 가능하다. • 도메인마다 별도의 저장 영역을 생성한다. 도메인마다 생성된 스토리지에 서로 접근 불가하고, 같은 도메인에 속해 있는 웹 페이지들은 모두 접근이 가능하다.	• 데이터 저장 기간이 제한되어 세션이 종료되면 자동 폐기된다. • 각 세션마다 별도의 저장 영역을 생성한다. 같은 도메인이라고 해도 다른 윈도우에서 생성되면 서로의 스토리지에 접근이 불가하다.

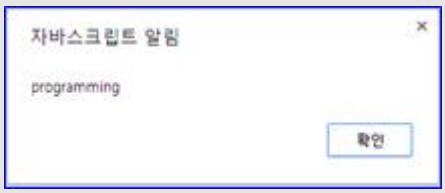
- 다음은 웹 스토리지의 지원여부를 판단할 수 있는 코드이다.

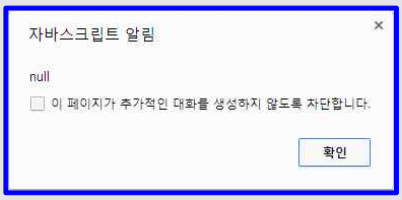
```
if ( typeof(Storage) !== "undefined" ) {
  //이곳에 세션 및 로컬 스토리지 기능을 구현
} else {
  //웹 스토리지 기능을 지원하지 않는 경우
}
```

3.2 데이터 저장, 읽기, 삭제

키: "computer", 값: "programming"		
저장	localStorage.setItem(key, value);	localStorage.setItem("computer", "programming");
	localStorage.key = value	localStorage.computer = "programming";
	localStorage[key] = value	localStorage["computer"] = "programming";
읽기	변수 = localStorage.getItem(key);	var value = localStorage.getItem("computer");
	변수 = localStorage.key	var value = localStorage.computer
	변수 = localStorage[key];	var value = localStorage["computer"];
삭제	localStorage.removeItem(key);	localStorage.removeItem("computer");
	delete localStorage.key	delete localStorage.computer;
	delete localStorage[key];	delete localStorage["computer"];

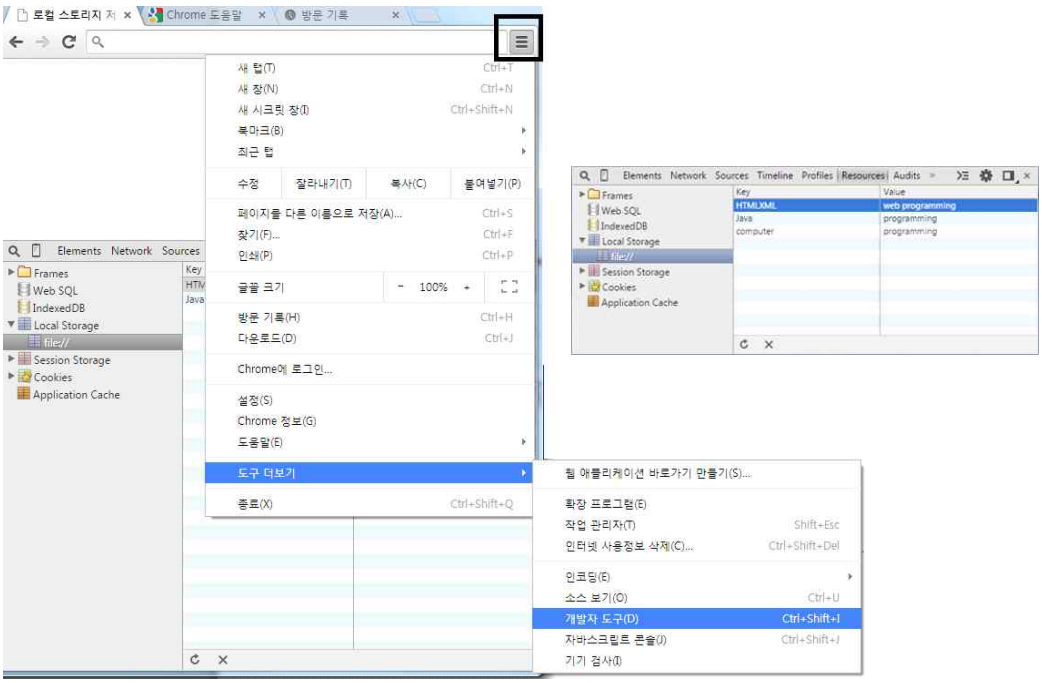
```
window.onload = function()  {
    if ( window.localStorage ) { //로컬 스토리지 지원 여부
        localStorage.setItem("computer", "programming"); // data 저장
        localStorage.Java = "programming";
        localStorage["HTMLXML"] = "web programming";
        var value = localStorage.getItem("computer"); //data 읽기
        alert(value); // programming alert 화면 출력
        localStorage.removeItem("computer"); //data 삭제
        var value = localStorage.getItem("computer");
        alert(value);  //null alert 화면 출력
    }
```





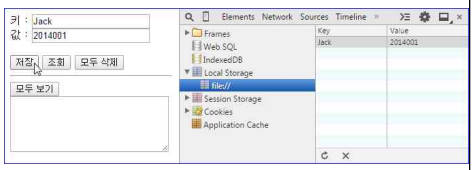
크롬 개발자 도구

맞춤 설정 및 제어 - 도구 더보기 – 개발자 도구 – Resources 를 통해서 localStrorage, Session Storage정보를 확인할 수 있다.



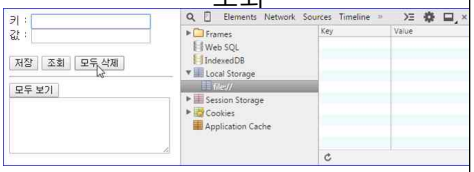
예제 로컬 스토리지

```
<!DOCTYPE html> <html><head>
<script type="text/javascript">
  function setData()    {
  var key = document.getElementById("key");
  var data = document.getElementById("data");
  localStorage.setItem(key.value, data.value);
  alert("데이터 저장이 완료되었습니다.");
}
 function getData()    {
  var key = document.getElementById("key");
  var data = localStorage.getItem(key.value);
  alert("조회한 데이터 값은"+data+ "입니다.");
}
 function removeData()    {
  localStorage.clear();
  alert("모든 데이터를 삭제하였습니다.");
}
 function viewData()    {
  var data = document.getElementById("result");
  data.value = "";
   for(var i = 0; i < localStorage.length; i++) {
      var key = localStorage.key(i);
     data.value += localStorage[key] + ",";}
 }
</script>
</head>
<body>
  키 : <input type="text" id="key"><br>
  값 : <input type="text" id="data"><br><br>
  <input type="button" onclick="setData()" value="저장">
  <input type="button" onclick="getData()" value="조회">
  <input type="button" onclick="removeData()" value="모두 삭제">  <hr>
  <input type="button" onclick="viewData()" value="모두 보기"><br>
  <textarea id="result" cols="30" rows="5"></textarea>
</body>
</html>
```



저장

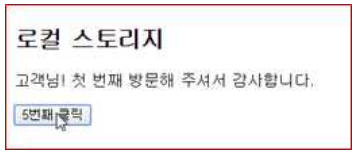
조회



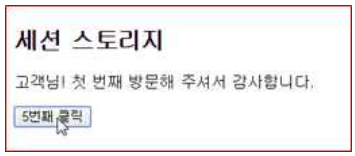
삭제

모두보기 화면

3.3 로컬 스토리지 vs 세션 스토리지



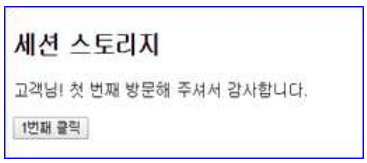
버튼 5번 클릭
첫 번째 방문 / 5번 표시



버튼 5번 클릭
첫 번째 방문 / 5번 표시

브라우저
종료 후
재 실행

버튼 5번 클릭
두 번째 방문/ 10번 표시



버튼 1번 클릭
첫 번째 방문 / 1번 표시

3.4 스토리지 이벤트

- 웹 스토리지 변경에 대해서는 window 객체가 storage 이벤트를 발생시킨다.
- 스토리지 이벤트는 HTTP 프로토콜을 통해서만 확인 가능하다.

("file://실습코드파일.html"와 같은 테스트는 불가하다.)

```
window.addEventListener("storage", function(event) {
  var key = event.key ; //변경된 키를 표시
  var newValue = event.newValue;  // 변경된 후의 새로운 값을 표시
  var oldValue = event.oldValue; // 변경된 키의 이전 값을 표시
  var url = event.url ; //키가 변경된 문서의 URL 표시
  // 이벤트 핸들 처리 ...
}, false );
```