웹 표준에 맞는 HTML5 프로그래밍 강의 노트

제 27회차

HTML5 API 종류 : Web Storage, App Cache, Web Workers, Server-Sent Events

▣ 학습목표

- Web Storage와 App Cache에 대해 설명할 수 있다.
- Web Workers와 Server-Sent Events에 대해 설명할 수 있다.
- Web SQL Database, Web Socket, File API에 대해 설명할 있다.

▣ 학습내용

- Web Storage와 App Cache
- Web Workers와 Server-Sent Events
- Web SQL Database, Web Socket, File API

1. Web Storage와 App Cache

1) Web Storage

- Web Storage : 일종의 클라이언트 사이드 데이터베이스

  - 데이터는 서버가 아닌 각 사용자의 브라우저에 보관
  - 일반 데이터베이스와의 차이점 : Key-value 형식으로 보관 · 갱신 · 호출

- Web Storage의 종류 : Local Storage, Session Storage

- Web Storage와 Cookie

〈Web〉

- 비연결 지향성 → 정보들은 쿠키(Cookie)에 저장됨

〈HTML5〉

- 쿠키의 역할을 하면서 단점을 보완하는 Web Storage 기술을 제공
  - Web Storage는 키/값(key/value)의 쌍으로 저장됨

Web Storage	VS	Cookie
• 약 5MB까지 저장 가능 • 서버로 전송할 필요 없음 • 상대적으로 보안에 안전 • 개수 제한 없음		• 4KB만 저장 가능 • 매번 HTTP 요청 헤더에 붙여 전송 (트래픽 낭비) • 보안에 취약 • 20개 정도로 개수 제한

- Local Storage

  - 만료 날짜가 없는 데이터를 저장함
  - 브라우저를 껐다가 켜도 값이 유지됨
  - 도메인이 다르면 서로의 Local Storage에 접근할 수 없음

1. Web Storage와 App Cache

1) Web Storage

- Local Storage 사용 메소드

속성/메소드	설명
length	저장된 변수의 개수
setItem(key, value)	Key/value를 Local Storage에 저장
getItem(key)	Key와 연관된 값을 반환
removeItem(key)	Key/value를 Local Storage에서 삭제
clear()	모든 key/value를 삭제

<setItem(key, value) 예>

```
localStorage.setItem("name", "Park");
localStorage.name="Park";
localStorage["name"]="Park";
```

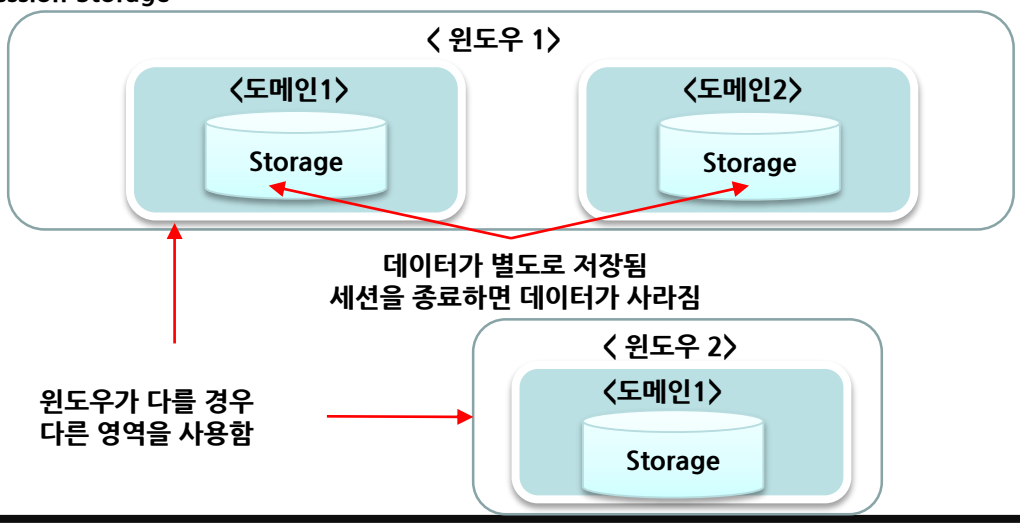
<getItem(key) 예>

```
var name=localStorage.getItem("name");
var name=localStorage.name;
var name=localStorage["name"];
```

<removeItem(key) 예>

```
localStorage.removeItem("name");
delete localStorage.name;
delete localStorage["name"];
```

- Session Storage



1. Web Storage와 App Cache

1) Web Storage

- Session Storage 문법

localStorage → Session Storage

- Local Storage 실습 예제

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Local Storage</title>
  <script>
      function clickCounter(){
          if(typeof(Storage) !== "undefined"){
              if(localStorage.clickcount){
                localStorage.clickcount = Number(localStorage.clickcount)+1;
              }
              else{
                localStorage.clickcount = 1;
              }
              document.getElementById("result").innerHTML = localStorage.clickcount + "
번 클릭했습니다.";
          }
          else{
              document.getElementById("result").innerHTML = "웹 스토리지를 지원하지 않습
니다.";
          }
      }
   </script>
</head>
<body>
   <p> <button onclick="clickCounter()" type="button">Click me! </button></p>
   <div id="result"></div>
</body>
</html>
```

1. Web Storage와 App Cache

2) App Cache

- 오프라인 환경을 고려한 API : Storage, Database

- API들이 오프라인에서 정상적으로 작동하기 위해 필요한 리소스 : CSS, 이미지 자바스크립트

- 오프라인 상태에서도 웹 애플리케이션으로의 접근을 가능케 하는 기능 : Application Cache API

- Manifest : CACHE해야 할 리소스를 줄 바꿈으로 구분하여 나열하기만 한 단순 텍스트 파일

```
CACHE MANIFEST
# 아래가 캐시 해야 할 리소스
index.html
app.js
cache.css
```

- ‘CACHE MANIFEST’ 문자열은 항상 첫 번째 라인에 위치해야 함
- 사이트당 최대 5MB까지 캐시 할 수 있음
- manifest 파일이나 manifest에 명시된 파일의 다운로드가 실패할 경우 브라우저는 가장 최근에 성공적으로 다운로드 한 파일을 그대로 사용하며, 실패 이벤트가 발생함

- Cache Status

- Application Cache의 동작을 세세하게 제어 할 수 있음
- JavaScript로 Application Cache에 액세스
  - 글로벌 객체가 가진 applicationCache 속성 참고
  - ‘window.applicationCache’, 혹은 ‘applicationCache’라 쓰면 됨

〈Status 속성이 가질 수 있는 값〉

상수	정수 값	설명
UNCACHED	0	캐시하지 않음
IDLE	1	최신 캐시를 이용 중
CHECKING	2	업데이트 체크 중
DOWNLOADING	3	업데이트 다운로드 중
UPDATEREADY	4	최신 캐시를 이용할 수 있음
OBSOLETE	5	에러에 의해 캐시 무효화

1. Web Storage와 App Cache

2) App Cache

- Cache Status 예제 소스

```
var sCacheStatus = "Not supported";
if (window.applicationCache)
{
  var oAppCache = window.applicationCache;
  switch ( oAppCache.status )
  {
    case oAppCache.UNCACHED :
      sCacheStatus = “Not cached”;
      break;
    case oAppCache.IDLE :
      sCacheStatus = “Idle”;
      break;
    case oAppCache.CHECKING :
      sCacheStatus = “Checking”;
      break;
    case oAppCache.DOWNLOADING :
      sCacheStatus = “Downloading”;
      break;
    case oAppCache.UPDATEREADY :
      sCacheStatus = "Update ready";
      break;
    case oAppCache.OBSOLETE :
      sCacheStatus = "Obsolete";
      break;}
}
return sCacheStatus;
```

캐시하지 않음

최신 캐시를 이용

업데이트 체크

업데이트 다운로드

최신 캐시 이용 가능

에러에 의해 캐시 무효화

1. Web Storage와 App Cache

2) App Cache

- Cache Event

• 이벤트를 처리하려면 addEventListener()를 이용하여 이벤트 핸들러를 등록함

```
applicationCache.addEventListener("updateready", function(){
                    //updateready 이벤트
}, false);
```

<캐시 이벤트에서 사용되는 상수>

상수	설명
checking	업데이트 체크 중
error	업데이트가 에러로 종료
noupdate	메니페스트가 업데이트되지 않음
downloading	업데이트 다운로드 중
progress	업데이트 다운로드 중
updateready	최신 캐시 얻기 완료. swapCache()를 호출할 수 있음
cached	캐시 완료
obsolete	매니페스트 얻기에 실패하여 캐시를 무효로 함

- Cache(캐시) 이벤트 상수를 활용한 소스코드

```
if (window.applicationCache) {
  var appCache = window.applicationCache;
   appCache.addEventListener('error', appCacheError, false);
   appCache.addEventListener('checking', checkingEvent, false);
   appCache.addEventListener('noupdate', noUpdateEvent, false);
   appCache.addEventListener('downloading', downloadingEvent, false);
   appCache.addEventListener('progress', progressEvent, false);
   appCache.addEventListener('updateready', updateReadyEvent, false);
   appCache.addEventListener('cached', cachedEvent, false);
}
```

2. Web Workers와 Server-Sent Events

1) Web Workers

- Web Workers의 장점 : 브라우저가 OS-레벨의 스레드를 생성

  - 빠른 속도
  - 브라우저에 부담을 주지 않고 백그라운드에서 스크립트 연산을 수행함

- Workers 생성

```
Var myWorker = new Worker('my_worer.js');
myWorker.onmessage = function(event){
  alert("Worker에 의해 실행될 콜백!₩n");
};
```

백그라운드에서 작업할 스크립트를 별도의 파일에 작성하고 새로운 인스턴스에 URI를 기입하여 Workers생성

onmessage 속성에 함수를 대입하여 작업결과를 돌려 받음

- Workers 생성

```
myWorker.terminate()
```

Worker는 남은 작업을 마무리하거나 메모리를 정리한 후 자발적으로 사라짐

2. Web Workers와 Server-Sent Events

1) Web Workers

- Web Workers 실습 예제

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>ex02</title>
</head>

<body>

                <p> Count number: <output id="result"></output></p>
  <button onClick="startWorker()">Start Worker </button>
  <button onClick="stopWorker()">Stop Worker</button>

<script>
        var w;
        function startWorker(){
                if(typeof(Worker) != "undefined"){
                    if(typeof(w) == "undefined") {
                                w = new Worker("myscript.js");
                    }
                    w.onmessage = function(event){
                                document.getElementById("result").innerHTML =
event.data;
                    };
                }
                else{
                    document.getElementById("result").innerHTML = "브라우저가 웹 워커
를 지원하지 않음";
                }
        }
        function stopWorker(){
                w.terminate();
                w = undefined;
        }
  </script>
</body>
</html>
```

2. Web Workers와 Server-Sent Events

2) Server-Sent Events

- Server-Sent Events : HTTP 환경에서 서버로부터의 데이터 푸시를 실제와 비슷하게 지원하기 위한 클라이언트 API와 푸시 데이터를 정의한 사양

> • HTTP에서 구현한 유사 데이터 푸시 기술 : Comet
> → Server-Sent Events는 Comet을 표준화한 하나의 형태

- Server-Sent Events 클라이언트 API를 이용할 경우의 장점

> • 서버로부터 푸시 된 데이터를 수신하여 일반적인 DOM 이벤트처럼 처리함
> • 서버로부터 푸시 될 데이터의 형식은 엄격하게 정해져 있음
> • PHP로 대표되는 서버 사이드 기술을 이용하면 아주 간단히 데이터를 생성할 수 있음

- SSE API의 사용법

```
var myWorker = new Worker('자바스크립트파일URL');
myWorker.onmessage = function(event){
  alert("Worker에 의해 실행된 콜백!₩n");
};
```

〈스크립트 작성법〉

> • Event Source에 이벤트에 스트림 받을 URL을 입력함
> • URL에 대해 정기적으로 GET 요청 수행
> • EventSource에 onmessage 핸들러를 지정함
> • Data 속성에 푸시된 데이터가 저장됨

2. Web Workers와 Server-Sent Events

2) Server-Sent Events

- 서버로부터의 푸시 데이터 형식

- text/event-stream이라는 MME 타입으로 보냄
- 문자 인코딩 : UTF-8로 고정
- 줄 바꿈 코드 : '/r/n', '/n', '/r/' 어느 것이라도 유효함
- ':'(콜론)으로 시작하는 줄 : 주석(무시)
- 빈 줄이나 파일 끝 : 이벤트 구분자가 됨
- 주석이나 빈 줄 이외는 '필드 이름: 필드 값'이라는 형식으로 작성됨

3. Web SQL Database, Web Socket, File API

1) Web SQL Database

- Web SQL Database 특징

- HTML5와 함께 새로 생겨난 개념
- Web Storage 중 하나로 Client에 데이터를 저장
- 풍부한 쿼리 능력을 가진 웹 어플리케이션 개발 지원
- 클라이언트의 저장소에 영구 보존 가능
- 체계적인 데이터 관리

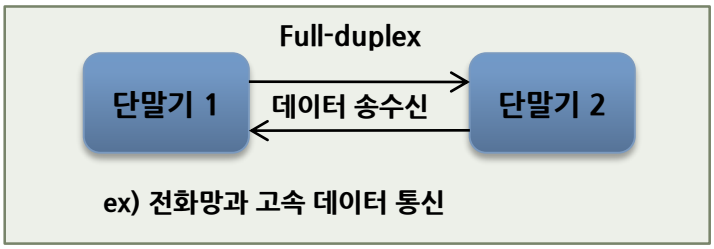
- 비동기와 트랜잭션의 이해

〈Web Database〉

- non-blocking 시스템 방식인 비동기 API 사용
- HTML을 통한 트랜잭션

2) Web Socket

- 웹 서버와 웹 브라우저가 지속적으로 연결된 TCP 라인을 통해 실시간으로 데이터를 주고 받을 수 있도록 함
- Comet의 대안으로 고안
- 웹 어플리케이션이 Full-duplex(전이중 통신) 단일 소켓 연결이 가능
- XHR(XMLHttpRequest)보다 대역폭 소모가 적음



서버와 브라우저 사이에 진정한 양방향 통신 제공

3. Web SQL Database, Web Socket, File API

3) File API

- File API 기능 : 웹 어플리케이션이 로컬 파일에 프로그램적으로 접근할 수 있게 함

- File API 실습 예제

```
<!DOCTYPE html>
<html>
<head>
 <title>File API Demo</title>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
 <h1>File API Demo</h1>
 <h3>파일(들)을 선택하세요.</h3>
 <!-- multiple 속성을 이용하면 파일을 다중으로 업로드 할 수 있음 -->
 <input id="files-upload" type="file" multiple>

 <h3>Uploaded files</h3>
 <ul id="file-list">
  <li class="no-items">(파일이 선택되 않음)</li>
 </ul>
<script>
  var filesUpload = document.getElementById("files-upload"),
      fileList = document.getElementById("file-list");

  function traverseFiles (files) {
   var li,
       file,
       fileInfo;
   fileList.innerHTML = "";

   for (var i=0, il=files.length; i<il; i++) {
    li = document.createElement("li");
    file = files[i];
    fileInfo = "<div><strong>Name:</strong> " + file.name + "</div>";
    fileInfo += "<div><strong>Size:</strong> " + file.size + " bytes</div>";
    fileInfo += "<div><strong>Type:</strong> " + file.type + "</div>";
    li.innerHTML = fileInfo;
    fileList.appendChild(li);
   };
  };
  filesUpload.onchange = function () {
   traverseFiles(this.files);
  };
 </script>
</body>
</html>
```

▣ 정리하기

1. Web Storage와 App Cache

- Web Storage
  - 일종의 클라이언트-사이드 데이터베이스
    → 이 데이터는 서버가 아닌 각 사용자의 브라우저에 보관됨
  - HTML5에서는 쿠키의 역할을 하면서 단점을 보완하는 Web Storage 기술을 제공함
  - 오프라인에서 정상적으로 작동하기 위해서는 CSS, 이미지, 자바스크립트 등과 같은 리소스를 필요로 함
    → 이러한 환경을 궁극적으로 충족시켜 주는 것 : Application Cache API

2. Web Workers와 Server-Sent Events

- Web Workers 장점
  - 빠름
  - 브라우저에 부담을 주지 않고 백그라운드에서 스크립트 연산을 수행함

- Server-Sent Events : HTTP 환경에서 서버로부터의 데이터 푸시를 실제와 비슷하게 지원하기 위한 클라이언트 API와 푸시 데이터를 정의한 사양
  - Event Source에 이벤트에 스트림 받을 URL을 입력함
  - URL에 대해 정기적으로 GET 요청을 수행함

▣ 정리하기

3. Web SQL Database, Web Socket, File API

- Web SQL Database 특징
  - HTML5와 함께 새로 생겨난 개념
  - Web Storage 중 하나로 Client에 데이터를 저장
  - 풍부한 쿼리 능력을 가진 웹 어플리케이션 개발 지원
  - 클라이언트의 저장소에 영구 보존 가능
  - 체계적인 데이터 관리

- Web Socket : 웹 서버와 웹 브라우저가 지속적으로 연결된 TCP 라인을 통해 실시간으로 데이터를 주고 받을 수 있도록 함

- File API 기능 : 웹 어플리케이션이 로컬 파일에 프로그램적으로 접근할 수 있게 함