JavaScript

File API – 김근형 강사

File API

- File API
  - File API 는 PC 에 있는 파일의 데이터를 직접 스크립트로 읽을 수 있게 한다.
  - 단 읽기 전용이므로 파일을 업데이트 하거나 삭제가 불가능하다. 또한 스크립트에서 PC에 있는 모든 파일을 선택할 수는 없다. 단지 사용자가 지정한 파일에만 접근 가능하다.
  - 스크립트에서 이러한 파일에 접근하기 위한 통로는 FileList 객체이다.
  - 드래그 앤 드롭이면 drop 이벤트의 이벤트 객체에서 얻을 수 있는 DataTransfer 객체의 files 속성으로 가져온다.
  - input 요소라면 input 요소 객체의 files 속성에서 가져온다.
  - FileList 객체는 드롭되었을 때 혹은 사용자가 input에서 type=file 의 속성을 가지고 change 이벤트가 발생했을 때 선택한 파일 목록이 저장된 FileList 객체를 가져온다.

File API

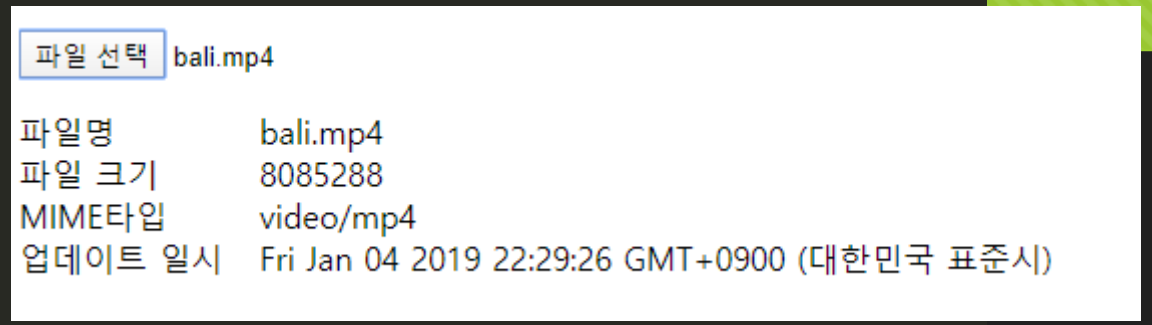
- File 객체
  - 선택된 파일을 나타내는 File 객체에는 파일의 정보가 담겨있다. 파일의 데이터가 저장되는 것이 아니므로 주의

메서드/속성	설명
File.size	파일 크기(바이트)를 반환한다.
File.type	파일의 MIME 형식을 소문자로 변환된 값으로 반환한다. MIME 형식이 불분명하면 빈 문자열을 반환한다.
File.name	파일 이름을 반환한다. 그러나 파일 경로를 포함되지 않는다.
File.lastModifiedDate	파일의 업데이트 시간을 나타내는 문자열을 반환한다. 파일의 업데이트 날짜가 불분명하면 빈 문자열을 반환한다.
File.slice(start,length) File.slice(start, length, contentType)	파일 데이터 중 start 위치에서 length 만큼의 범위의 데이터를 나타내는 Blob 객체를 반환한다. 선택적으로 세 번째 매개변수에 HTTP 응답 해더의 Content-Type 값을 지정할 수 있다. 이것은 파일 데이터를 HTTP를 통해 얻었을 때 데이터를 제한하는 데 사용할 수 있다.

File API

- File 객체 예제

```
<p><input type="file" /></p>
<dl>
    <dt>파일명 </dt>
    <dd id="name">-</dd>
    <dt>파일 크기 </dt>
    <dd id="size">-</dd>
    <dt>MIME타입 </dt>
    <dd id="type">-</dd>
    <dt>업데이트 일시 </dt>
    <dd id="mdate">-</dd>
</dl>
<script type="text/javascript">
    var input = document.querySelector('input[type="file"]');
    input.addEventListener("change", function(event) {
        // FileList 오브젝트
        var files = event.target.files;
        // 선택된 파일을 나타내는 File 오브젝트
        var file = files[0];
        if( ! file ) { return; }
        // 파일의 각종 정보를 표시
        document.querySelector('#name').textContent = file.name;
        document.querySelector('#size').textContent = file.size;
        document.querySelector('#type').textContent = file.type;
        document.querySelector('#mdate').textContent = file.lastModifiedDate;
    }, false);
</script>
```



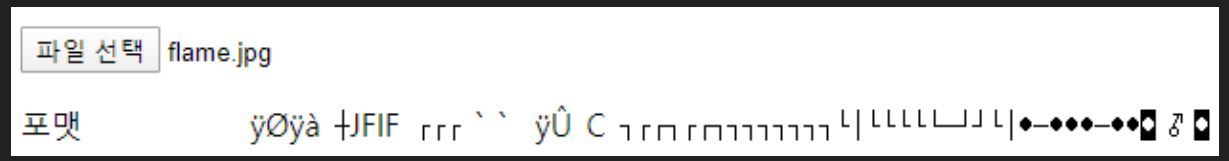
File API

- FileReader
  - File 객체나 Blob 객체를 얻고 나면 다음은 FileReader 객체로 파일의 데이터를 읽는다.

메서드/속성	설명
FileReader = new FileReader()	FileReader 객체를 반환한다.
FileReader.readAsBinaryString(blob)	매개변수에 지정된 Blob 객체 또는 File 객체가 나타내는 파일의 데이터를 읽고 그 데이터를 바이너리로 보존한다.
FileReader.readAsText(blob) FileReader.readAsText(blob, encoding)	매개변수에 지정된 Blob 객체(또는 File 객체)가 나타내는 파일의 데이터를 읽고 그 데이터를 UTF-8 텍스트로 변환한다. 읽어들인 텍스트가 EUC-KR 이여도 UTF-8로 변환된다. 두번째 매개변수에 문자 인코딩을 나타내는 문자열을 지정할 수 있다. 두 번째 매개변수가 지정되면 읽은 데이터는 지정된 문자 인코딩으로 변환된다.
FileReader.readAsDataURL(blob)	매개변수에 지정된 Blob 객체(또는 File 객체)가 나타내는 파일의 데이터를 읽고 그 데이터를 Data URL 형식의 문자열로 반환한다.
FileReader.readAsArrayBuffer(blob)	매개변수에 지정된 Blob 객체(또는 File 객체)가 나타내는 파일의 데이터를 읽고 그 데이터를 ArrayBuffer 객체로 반환한다.
FileReader.abort()	파일 읽기를 중지한다.
FileReader.result	읽어들인 데이터를 반환한다. 반환되는 데이터의 형식은 읽어들인 방식에 따라 다르다.

File API

- FileReader 예제 - 1




```
<p><input type="file" /></p>
<dl>
    <dt>포맷 </dt>
    <dd id="format">-</dd>
</dl>
<script type="text/javascript">
    var input = document.querySelector('input[type="file"]');
    input.addEventListener("change", function(event) {
        // FileList 오브젝트
        var files = event.target.files;
        // 선택된 파일을 나타내는 File 오브젝트
        var file = files[0];
        if( ! file ) { return; }
            // 이미지 파일의 포맷을 판단
            show_image_format(file);
    }, false);
    function show_image_format(file) {
        // FileReader 오브젝트
        var reader = new FileReader();
        // 바이너리 형식으로 파일 데이터 얻기
        reader.readAsBinaryString(file);
        // 파일 데이터 읽기가 성공했을 때의 처리
        reader.onload = function() {
            // 바이너리 데이터
            var bin = reader.result;
            // 앞부분 64바이트 얻기
            var header = bin.slice(0, 64);
            // 앞부분 내용 출력
            document.querySelector('#format').textContent = header;
        };
    }
</script>
```

File API 이벤트 & 로드 상태

- 파일을 읽는 과정에서 발생하는 이벤트
  - 파일 읽기가 완료되면 FileReader 객체에서 load 이벤트가 발생하며 이 과정에서 다양한 이벤트가 발생한다.
  - 파일을 읽는 과정에서 발생하는 이벤트는 아래와 같다.

이벤트	설명
loadstart	파일을 읽기 시작할 때 발생한다.
progress	파일을 읽는 중에 연속적으로 발생한다.
abort	abort() 함수 호출 등의 이유로 파일 읽기가 중지되면 발생한다.
error	파일을 읽다가 오류가 발생하면 발생한다.
load	파일 읽기가 정상적으로 종료되면 발생한다.
loadend	파일 읽기에 대한 처리가 종료되면 발생한다. 파일을 성공적으로 읽었는지 실패하였는지 여부에 상관없이 발생한다.

File API 이벤트 & 로드 상태

- 파일을 읽는 과정에서 발생하는 이벤트
  - 위의 이벤트를 감지하기 위해 이에 해당하는 이벤트 핸들러가 존재한다

이벤트 핸들러	설명
FileReader.onloadstart	loadstart 이벤트가 발생했을 때 실행되는 핸들러를 정의할 수 있다.
FileReader.onprogress	progress 이벤트가 발생했을 때 실행되는 핸들러를 정의할 수 있다.
FileReader.onload	load 이벤트가 발생했을 때 실행되는 핸들러를 정의할 수 있다.
FileReader.onabort	abort 이벤트가 발생했을 때 실행되는 핸들러를 정의할 수 있다.
FileReader.onerrror	error 이벤트가 발생했을 때 실행되는 핸들러를 정의할 수 있다.
FileReader.onloadend	loadend 이벤트가 발생했을 때 실행되는 핸들러를 정의할 수 있다.

File API 이벤트 & 로드 상태

- 파일을 읽는 상태 파악하기
  - FileReader 객체의 readyState 속성에서 그 시점의 파일 읽기 상태를 파악할 수 있다.

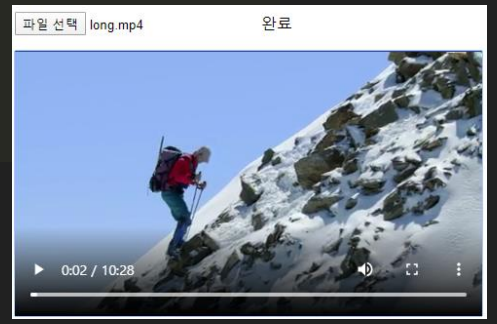
속성	설명
FileReader.readyState	로드 상태를 나타내는 수치를 반환한다. 수치의 의미는 다음과 같다. 0: File 객체가 셍성되었지만 아직 파일 데이터를 읽지 않은 상태(EMPTY) 1: 파일을 읽는 도중(LOADING) 2: 파일의 모든 데이터 읽기가 끝나고 메모리에 저장된 상태, 또는 어떠한 오류가 발생하여 파일의 읽기가 중단된 상태(DONE)
FileReader.EMPTY	항상 0을 반환
FileReader.LOADING	항상 1을 반환
FileReader.DONE	항상 2를 반환

File API 이벤트 & 로드 상태

- 파일을 읽는 상태 파악하기 예제 - 1

```
<p><input type="file"  /><span id="state">대기중 </span></p>
<p><video width="480" height="272" controls="controls"></video></p>
<script type="text/javascript">
    var input = document.querySelector('input[type="file"]');
    input.addEventListener("change", function(event) {
        // FileList 오브젝트
        var files = event.target.files;
        // 선택된 파일을 나타내는 File 오브젝트
        var file = files[0];
        if( ! file ) { return; }
        if( ! file.type.match(/^video\/mp4$/) ) {
            alert("MP4 비디오 파일을 지정해 주세요");
        }
        // 비디오 파일 읽기 시작
        load_file(file);
    }, false);
```

```
function load_file(file) {
    // FileReader 오브젝트
    var reader = new FileReader();
    // Data URL 형식으로 파일 데이터 얻기
    reader.readAsDataURL(file);
    // 파일 읽기의 진행률을 표시
    var span = document.querySelector('#state');
    var handler = window.setInterval( function() {
        if( reader.readyState == reader.EMPTY ) {
            span.textContent = "로드중 ";
        } else if( reader.readyState == reader.LOADING ) {
            span.textContent = "로드중 ";
        } else if( reader.readyState == reader.DONE ) {
            span.textContent = "완료 ";
            // video 요소의 src 속성을 지정
            var video = document.querySelector('video');
            video.src = reader.result;
            video.load();
            // 타이머 해제
            window.clearInterval(handler);
        }
    }, 100);}
```

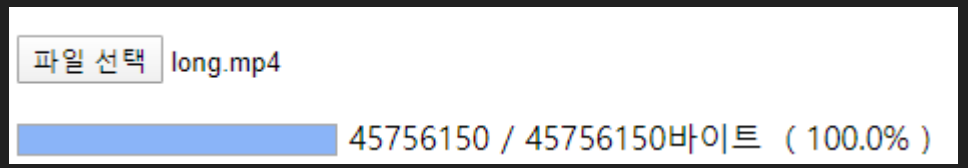


파일 로드 진행률 표시하기

- 실시간으로 파일 읽기 처리의 진행률을 표시하기
  - progress 이벤트를 사용하면 실시간으로 읽기 작업의 진행률을 표시할 수 있다.
  - FileReader 객체에서 발생하는 각종 이벤트의 핸들러는 첫 번째 매개변수에 이벤트 객체가 지정되어 있어 해당 이벤트 객체에는 진행 상황을 나타내는 여러 가지 속성이 있다.
  - 특히 progress 이벤트는 진행 상황을 실시간으로 파악하는 데 필요한 이벤트다.

파일 로드 진행률 표시하기

- 실시간으로 파일 읽기 처리의 진행률을 표시하기 예제




```
<p><input type="file" /></p>
<p>
    <progress value="0" max="100"></progress>
    <span id="loaded">0</span> / <span id="total">0</span>바이트
     (<span id="rate">0</span>%)
</p>
<script type="text/javascript">
    var input = document.querySelector('input[type="file"]');
    input.addEventListener("change", function(event) {
        // FileList 오브젝트,
        var files = event.target.files;
        // 선택된 파일을 나타내는 File 오브젝트
        var file = files[0];
        if( ! file ) { return; }
        // FileReader 오브젝트
        var reader = new FileReader();
        // progress 이벤트 핸들러 지정
        reader.onprogress = show_progress;
        // load 이벤트 핸들러 지정
        reader.onloadend = show_progress;
        // 바이너리 형식으로 파일 데이터 얻기
        reader.readAsBinaryString(file);
    }, false);
    // 파일 로드의 진행 률 표시
    function show_progress(event) {
        document.querySelector('#total').textContent = event.total;
        document.querySelector('#loaded').textContent = event.loaded;
        var rate = ( event.loaded * 100 / event.total ).toFixed(1);
        document.querySelector('#rate').textContent = rate;
        document.querySelector('progress').value = rate;
    }
</script>
```

오류 처리

- 오류의 원인 파악하기
  - PC 상의 파일도 읽기에 실패할 수 있다. 이때 그 원인을 파악할 수 있다.

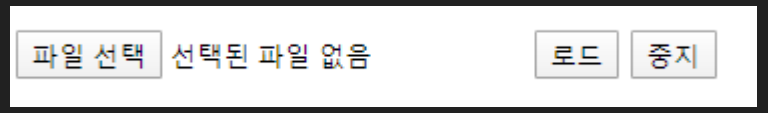
오류 원인 파악 속성	설명
FileReader.error.code	오류의 종류를 나타내는 숫자를 반환한다 1:파일을 찾을 수 없을 때(NOT_FOUND_ERR) 2:웹 어플리케이션에서 사용하기엔 보안상 좋지 않다고 판단했을 때, File에 너무 많은 읽기 호출이 있을 때, 사용자가 파일을 선택한 후 파일의 내용이 바뀌었을 때(SECURITY_ERR) 3:abort() 함수의 호출 등의 이유로 파일 읽기 처리가 중지되었을 때(ABORT_ERR) 4:파일 권한 및 기타 응용 프로그램에서 잠금 처리 등의 이유로 파일을 읽을 수 없었을 때(NOT_READABLE_ERR) 5:Data URL 길이가 URL 길이의 상한을 초과할 때 readAsText() 함수로 불렸을 때는 제외(ENCODING_ERR)
FileReader.error. NOT_FOUND_ERR	항상 1을 반환
FileReader.error. SECURITY_ERR	항상 2을 반환
FileReader.error. ABORT_ERR	항상 3을 반환
FileReader.error. NOT_READABLE_ERR	항상 4을 반환
FileReader.error. ENCODING_ERR	항상 5을 반환

오류 처리

- 오류의 원인 파악하기 예제 - 1

```
<p>
    <input type="file" />
    <button type="button" id="load">로드 </button>
    <button type="button" id="abort">중지 </button>
</p>
<script type="text/javascript">
var input = document.querySelector('input[type="file"]');
input.addEventListener("change", function(event) {
    // FileList 오브젝트
    var files = event.target.files;
    // 선택된 파일을 나타내는 File 오브젝트
    var file = files[0];
    if (!file) { return; }
    // FileReader 오브젝트
    var reader = new FileReader();
    // 로드 버튼에 click 이벤트 리스너 지정
    document.querySelector('#load').addEventListener("click", function() {
        // error 이벤트 핸들러 지정
        reader.onerror = function() {
            show_error(reader);
        };
        // Data URL 형식으로 파일 데이터 얻기
        reader.readAsDataURL(file);
    }, false);
    // 중지 버튼에 click 이벤트 리스너 지정
    document.querySelector('#abort').addEventListener("click", function() {
        abort_load(reader);
    }, false);
}, false);
```

```
// 오류 표시
function show_error(reader) {
    var code = reader.error.code;
    if (code == reader.error.NOT_FOUND_ERR) {
        alert("파일을 발견할 수 없었습니다.");
    } else if (code == reader.error.SECURITY_ERR) {
        alert("보안 오류가 발생 했습니다.");
    } else if (code == reader.error.ABORT_ERR) {
        alert("파일 읽기가 중지 되었습니다.");
    } else if (code == reader.error.NOT_READABLE_ERR) {
        alert("파일 읽기가 막혀 있습니다.");
    } else if (code == reader.error.ENCODING_ERR) {
        alert("파일 용량이 너무 큽니다.");
    }
}
// 로드 중지
function abort_load(reader) {
    //  로드가 시작되지 않았으면 종료
    if (!reader) { return; }
    // 로드 중이 아니라면 종료
    if (reader.readyState == reader.LOADING) { return; }
    // 로드 중지
    reader.abort();
}
</script>
```



파일 URI 생성하기

- 고유 URI 생성하기
  - File API는 드래그 앤 드롭이나 type 속성에 “file” 이 지정된 input 요소에서 가져온 파일에 고유의 URI를 생성할 수 있다.
  - 이 URI를 특정 요소의 src 속성에 직접 지정이 가능하다.
  - 고유 URI 를 생성하기 위한 함수는 아래와 같다.

메서드	설명
window.createObjectURL(blob)	매개변수로 지정한 Blob/File 객체에 고유한 Blob URI 를 생성하고 실제 파일과 연결한다.
window.revokeObjectURL(uri)	매개변수(uri) 로 지정한 Blob URI 를 해제한다.

  - window 객체의 createObjectURL() 함수는 매개변수로 Blob 객체나 File 객체를 지정하면 고유한 URI를 생성한다. 그러나 생성된 URI는 일반적인 URI와 다르다.

파일 URI 생성하기

- 고유 URI 생성하기 예제 - 1

```
<p><input type="file" /></p>
<script type="text/javascript">
document.addEventListener("DOMContentLoaded", function() {
    var input = document.querySelector('input[type="file"]');
    input.addEventListener("change", function(event) {
        // FileList 오브젝트
        var files = event.target.files;
        // 선택된 파일을 나타내는 File 오브젝트
        var file = files[0];
        if (!file) { return; }
        // 파일의 URI를 생성
        var uri = window.webkitURL.createObjectURL(file);
        // img 요소에 지정
        var img = document.createElement("img");
        img.src = uri;
        document.body.appendChild(img);
    }, false);
}, false);
</script>
```


```
▼<body> == $0
  ▶<p>…</p>
  ▶<script type="text/javascript">…</script>
   <img src="blob:null/ccef2483-7ce9-45ac-a61e-a3e3288e1703">
</body>
```

파일 URI 생성하기

- 고유 URI 생성하기 예제 - 2

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style type="text/css">
    dt {
        width: 120px;
        float: left;
    }

    video {
        background-color: black;
    }
    </style>
</head>

<body>
    <p><input type="file" /></p>
    <script type="text/javascript">
    document.addEventListener("DOMContentLoaded", function() {
        var input = document.querySelector('input[type="file"]');
        input.addEventListener("change", function(event) {
            // FileList 오브젝트
            var files = event.target.files;
            // 선택된 파일을 나타내는 File 오브젝트
            var file = files[0];
            if (!file) { return; }
            // 파일의 URI를 생성
```

```
<textarea id="editor" rows="30" cols="80" placeholder="HTML파일을 드랍해 주세요"></textarea>
<script type="text/javascript">
document.addEventListener("DOMContentLoaded", function() {
    /* ------------------------------------------------------
     * ■드랍 측의 스크립트
     * ------------------------------------------------------ */
    // textarea요소
    var textarea = document.querySelector('#editor');
    // textarea 요소에서 발생하는 dragenter, dragover이벤트의
    // 기본 액션을 해제
    textarea.addEventListener("dragenter", cancel_default, false);
    textarea.addEventListener("dragover", cancel_default, false);
    // canvas요소에 drop이벤트의 리스너를 지정
    textarea.addEventListener("drop", drop_file, false);
}, false);
// 기본 액션을 해제
function cancel_default(event) {
    event.preventDefault();
}
// textarea 요소에 파일이 드랍되었을 때의 처리
function drop_file(event) {
    // 기본 액션을 해제
    event.preventDefault();
    // FileList 오브젝트
    var files = event.dataTransfer.files;
    // 드랍된 파일의 File 오브젝트
    var file = files[0];
    if (!file) { return; }
    // 파일의 MIME 타입을 확인
    if (!file.type.match(/^text\//)) { return; }
    // 드랍된 파일의 URI를 생성
    var uri = window.webkitURL.createObjectURL(file);
    // XHR로 파일을 읽어 들여 textarea 요소에 표시
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (this.readyState != 4 || this.status != 200) { return; }
        document.querySelector('#editor').textContent = this.responseText;
    };
    xhr.open("GET", uri);
    xhr.send();
}
```