

05

o5-5. *File*

Web APIs

File 핸들링 제약

개발자가 js로 브라우저의 경계를 벗어나 유저 컴퓨터의 파일에 접근할 수 있게 한다면, 보이지 않게 악성 행동을 할 가능성이 너무 높아진다.

웹은 너무 개방성이 뛰어나다. 링크 하나만으로 여러 앱을 넘나든다.

데스크탑 앱은 그렇지 않다. 어떤 앱이 링크 하나만으로 다른 앱을 실행시키는 게 쉽지 않다.

(유저 보호 차원)

- 브라우저에서 실행되는 프론트 웹 애플리케이션에서 사용자 컴퓨터에 저장되어있는 파일을 핸들링하는 것은 일차적으로 금지되어 있습니다.
file은 web 바깥에 있는 것이다.
- 유저가 브라우저에서 선택한 파일에 한해서만 자바스크립트로 읽어 들일 수 있고 서버에 업로드가 가능합니다.
- 사용자가 선택한 파일이란 이후에 살펴볼 `<input type="file">` 을 통해 유저가 선택한 파일이거나 아니면 유저가 드래그&드랍으로 웹 페이지에 추가시킨 파일을 의미합니다.

브라우저에 파일을 업로드하려면 무조건 post 방식이어야 한다.
get 방식으로 넘기려는 데이터의 양에는 한계가 있다.

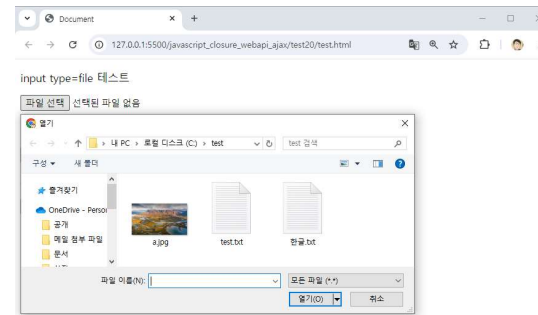
파일 업로드할 때는 멀티 파트 업로드와 같은 독특한 방식을 이용한다. 업로드할 때는 실제 파일 내용만 넘어가선 안 되고, 부수적인 내용까지 업로드 되어야 하기 때문이다.

멀티 파트 업로드는 복잡하기 때문에 분석을 대행해주는 라이브러리를 쓴다.

<input type="file"/>

- 사용자가 애플리케이션에 파일을 추가하기 위한 가장 기본적인 방법은 <input> 태그를 이용하는 방법입니다

<input type="file"/>



multiple 속성

- multiple 속성이 추가되면 사용자가 파일 선택 다이얼로그에서 여러 파일을 선택할 수 있습니다.

accept 속성

- accept 속성을 이용해 사용자가 선택할 파일의 타입을 지정할 수 있습니다

```
<input type="file" accept="image/*"/>  
<input type="file" accept=".txt, .jpg"/>
```

File

- 파일 이용의 기본은 파일명, 파일의 저장시간 혹은 파일 사이즈 정보를 획득하는 것입니다.
- 이를 위해 제공하는 API 가 File 과 FileList 입니다.
- File 은 사용자가 선택한 파일 하나를 표현하며 File 객체 여러 개가 FileList 에 담겨서 이용되게 됩니다.
- 사용자가 선택한 파일 정보 획득은 대부분 `<input type="file"/>` 에서 사용자가 파일을 선택한 순간 이루어지며 이를 위해 change 이벤트를 제공합니다.

File

- 사용자가 파일을 하나 선택하든 여러 개 선택하든 모든 파일은 File 객체로 만들어지면 File 객체들이 `FileList` 에 담겨서 전달됩니다.
- File 객체의 `name`, `size`, `lastModified` 프로퍼티로 파일의 이름, 사이즈, 저장시간을 획득하여 화면에 출력시키고 있습니다.

```
const files = e.target.files;
if (files.length !== 0) {
  for (const file of files) {
    const listItem = document.createElement("li");
    listItem.innerHTML = `file name : ${file.name}, file size :
    ${file.size}, modified : ${new Date(file.lastModified)}~

    resultNode.appendChild(listItem)
  }
}
```

File

파일 내용 읽기

- 파일의 내용을 읽어서 화면에 출력시켜야 한다면 FileReader API 를 사용해야 합니다.
- FileReader 는 파일 내용을 읽기 위해서 제공되는 API 로 다음의 함수를 제공합니다.
- `readAsText()` : 파일의 내용을 텍스트로 읽음
- `readAsDataURL()` : 파일의 내용을 읽어 base64 로 인코딩해 반환
- `readAsArrayBuffer()` : 파일의 내용을 배열 버퍼로 읽음
- `readAsBinaryString()` : 파일의 내용을 바이너리 문자열로 읽음
- 이미지를 읽어 `` 태그를 이용해 화면에 출력할 때는 `readAsDataURL()` 로 읽는 것이 편리합니다.
- `readAsDataURL()` 는 읽은 데이터를 base64로 인코딩하여 반환하는데 이 값을 그대로 `` 태그의 `src` 에 지정하면 이미지가 출력되게 됩니다.
- `readAsArrayBuffer()` 는 버퍼 개념을 적용해 작게 나누어 읽을 때 유용하며 스트림으로 서버에 업로드 할 때 유용합니다.

File

파일 내용 읽기

- FileReader 로 파일을 비동기로 읽기 때문에 읽은 내용은 이벤트를 등록하여 이벤트 콜백함수를 통해 획득해야 합니다. 다음은 FileReader 에서 제공하는 이벤트입니다.
- load : 파일 읽기 작업이 완료된 순간의 이벤트
- error : 에러가 발생한 순간의 이벤트
- abort : abort() 함수가 호출되어 읽기 작업이 취소된 순간의 이벤트
- progress : 파일을 읽는 동안 주기적으로 발생하는 이벤트

```
const reader = new FileReader()
reader.onload = function(e){
  const pNode = document.createElement('p')
  pNode.innerHTML = e.target.result.split("\n").join("<br />")
  liNode.appendChild(pNode)
  resultNode.appendChild(liNode)
}
reader.onerror = function(e){
  const pNode = document.createElement('p')
  pNode.innerHTML = '파일을 읽기에 실패 했습니다.'
  liNode.appendChild(pNode)
  resultNode.appendChild(liNode)
}
reader.readAsText(file)
```

File - FormData

- 파일 업로드를 위해 여러가지 방법이 있지만 많이 사용하는 방법 중 하나가 FormData 로 파일을 담아 Ajax 통신을 통해 업로드 하는 방법입니다.
- FormData 는 자바스크립트 API 이며 서버에 넘겨야 하는 데이터를 키-값 형식으로 표현하기 위해서 제공되는 API 입니다.
- FormData 에 다양한 데이터를 추가할 수 있으며 파일 또한 FormData 에 추가해 서버에 전송할 수 있습니다.
- 이렇게 만들어진 FormData 를 Ajax 통신을 통해 서버에 전송해 주면 됩니다.

File - FormData

```
if(files.length !== 0){  
  const formData = new FormData();  
  for(const file of files) {  
    formData.append('file', file)  
  }  
  formData.append('title', title)  
  
  const resp = await axios.post('http://localhost:8000/upload',  
formData)  
  if (resp.data.status === 200) {  
    alert('upload ok')  
  }  
}
```

File – drag & drop

- 웹 페이지에 드래그 & 드랍으로 파일을 추가해서 업로드 하는 방법을 이용합니다.
- 사용자의 드래그 & 드랍을 위한 태그를 준비한 후 이곳에 드래그 & 드랍 이벤트를 등록하여 이벤트 콜백함 수에서 사용자가 추가한 파일 정보를 추출해 업로드를 진행하면 됩니다.

```
<div  
  id="drop_zone"  
  ondrop="dropHandler(event);"  
  ondragover="dragOverHandler(event);"  
>
```



감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.



윌리엄 셰익스피어
William Shakespeare