Using Fetch

번역이 완료되지 않았습니다. Please help translate this article from English

Fetch API를 이용하면 Request나 Response와 같은 HTTP의 파이프라인을 구성하는 요소를 조작하는것이 가능합니다. 또한 `fetch()` 메서드를 이용하는 것으로 비동기 네트워크 통신을 알기쉽게 기술할 수 있습니다.

이전에 이러한 기능을 `XMLHttpRequest` 에서 제공하고 있었습니다. Fetch는 이러한 API의 대체제로 `Service Workers` 같은 기술로 간단히 이용하는것이 가능합니다. 또한 CORS나 HTTP확장같은 HTTP에 관련한 개념을 모아 정의하고 있습니다.

Fetch의 기본 스펙은 `jQuery.ajax()` 와 기본적으로 두가지가 다르다는 사실에 유념해야합니다.

- `fetch()` 로 부터 반환되는 Promise 객체는 **HTTP error 상태를 reject하지 않습니다.** HTTP Statue Code가 404나 500을 반환하더라도요. 대신 ok 상태가 false인 resolve가 반환되며, 네트워크 장애나 요청이 완료되지 못한 상태에는 reject가 반환됩니다.
- 보통 `fetch` 는 **쿠키를 보내거나 받지 않습니다.** 사이트에서 사용자 세션을 유지 관리해야하는 경우 인증되지 않는 요청이 발생합니다. 쿠키를 전송하기 위해서는 자격증명(credentials) 옵션을 반드시 설정해야 합니다.
  2017년 8월 25일 이후. 기본 자격증명(credentials) 정책이 `same-origin` 으로 변경되었습니다. 파이어폭스는 61.0b13 이후 변경되었습니다.

기본적인 fetch는 누구라도 알기쉽고 간단하게 작성할 수 있습니다. 아래의 코드를 봐주시기 바랍니다.

```
fetch('http://example.com/movies.json')
  .then(function(response) {
    return response.json();
  })
  .then(function(myJson) {
    console.log(JSON.stringify(myJson));
  });
```

네트워크에서 JSON 파일을 가져 와서 콘솔에 인쇄합니다. 간단한 `fetch()` 사용 흐름은 인수 한 개(가져올 자원의 경로)를 가져오고 응답을 포함하는 약속 (`Response` 개체)을 반환하는 것입니다.

이것은 단순한 HTTP Response이며, 실제 JSON이 아닙니다. response 객체로부터 사진을 가져오기 위해서는 `json()` 메서드를 사용할 필요가 있습니다. (`Body`의 믹스인 (역주:php의 트레이드와 같은것입니다.)으로 정의되어, 이것은 `Request` 객체와 `Response` 객체의 쌍방에 구현되어 있습니다.

> 노트: http Request와 http Response의 Body mixin은 Body 컨텐츠를 다른 mine 타입으로 사용하는 비슷한 메서드를 제공하고 있습니다. 상세한 내용은 Body 섹션을 참고해 주시기 바랍니다.

Fetch Reqeust는 검색된 리소스로부터의 지시가 아닌 CSP의 `connect-src`의 디렉티브 (directive)에 의해 제어됩니다.

리퀘스트의 옵션 적용

`fetch()` 메서드에 두번째 파라미터를 적용하는것도 가능합니다. `init` 오브젝트는 다른 여러 세팅을 컨트롤 할 수 있게 해줍니다.

모든 설정 가능한 옵션의 상세 설명은 `fetch()`를 참고해주시기 바랍니다.

```
// Example POST method implementation:

postData('http://example.com/answer', {answer: 42})
  .then(data => console.log(JSON.stringify(data))) // JSON-string from `re
  .catch(error => console.error(error));

function postData(url = '', data = {}) {
  // Default options are marked with *
    return fetch(url, {
        method: 'POST', // *GET, POST, PUT, DELETE, etc.
        mode: 'cors', // no-cors, cors, *same-origin
        cache: 'no-cache', // *default, no-cache, reload, force-cache, on
        credentials: 'same-origin', // include, *same-origin, omit
        headers: {
            'Content-Type': 'application/json',
            // 'Content-Type': 'application/x-www-form-urlencoded',
        },
```

```
        redirect: 'follow', // manual, *follow, error
        referrer: 'no-referrer', // no-referrer, *client
        body: JSON.stringify(data), // body data type must match "Content-
    })
    .then(response => response.json()); // parses JSON response into nativ
}
```

자격 증명(credentials)이 포함된 Request 요청

자격 증명이 포함된 인증서를 보내도록 하려면 `fetch()` 메서드에 `credentials: 'include'` 를 추가하도록 합니다. 이것은 cross-origin 요청에서도 사용됩니다.

```
fetch('https://example.com', {
  credentials: 'include'
})
```

요청하려는 URL이 호출 스크립트와 동일한 origin을 가지고 있을때만 자격증명을 전송하려면 `credentials: 'same-origin'` 를 추가해 주시기 바랍니다.

```
// The calling script is on the origin 'https://example.com'

fetch('https://example.com', {
  credentials: 'same-origin'
})
```

브라우저의 보안을 유지하는것 대신 자격증명을 포함하지 않는것을 원한다면 `credentials: 'omit'` 를 작성해 주시기 바랍니다.

```
fetch('https://example.com', {
  credentials: 'omit'
})
```

Uploading JSON data

POST프로토콜로 JSON인코딩된 데이터를 보내기 위해 `fetch()` 를 사용합니다.

```
var url = 'https://example.com/profile';
var data = {username: 'example'};

fetch(url, {
  method: 'POST', // or 'PUT'
  body: JSON.stringify(data), // data can be `string` or {object}!
  headers:{
    'Content-Type': 'application/json'
  }
}).then(res => res.json())
.then(response => console.log('Success:', JSON.stringify(response)))
.catch(error => console.error('Error:', error));
```

Uploading a file

`<input type="file" />` input엘리먼트, `FormData()`, `fetch()`를 사용하여 파일을 업로드 할 수 있습니다.

```
var formData = new FormData();
var fileField = document.querySelector('input[type="file"]');

formData.append('username', 'abc123');
formData.append('avatar', fileField.files[0]);

fetch('https://example.com/profile/avatar', {
  method: 'PUT',
  body: formData
})
.then(response => response.json())
.catch(error => console.error('Error:', error))
.then(response => console.log('Success:', JSON.stringify(response)));
```

Uploading multiple files

`<input type="file" multiple />` input엘리먼트와`FormData()`, `fetch()`를 사용하여 파일 다중업로드를 할 수 있습니다.

```
var formData = new FormData();
var photos = document.querySelector('input[type="file"][multiple]');

formData.append('title', 'My Vegas Vacation');
for (var i = 0; i < photos.files.length; i++) {
  formData.append('photos', photos.files[i]);
}

fetch('https://example.com/posts', {
  method: 'POST',
  body: formData
})
.then(response => response.json())
.then(response => console.log('Success:', JSON.stringify(response)))
.catch(error => console.error('Error:', error));
```

문서 파일을 열단위로 처리하기

응답하는 곳에서 읽는 덩어리들은 줄 단위로 깔끔하게 떨어지지 않으며 문자열이 아니라 Uint8Arrys(8비트 부호 없는 정수)입니다. 만약 텍스트 파일은 fetch하고 이것을 줄 단위로 처리하고자 한다면, 이런 복잡함을 다루는 것은 사용자에게 달려있습니다. 아래의 예시는 line iterator를 생성하여 처리하는 한가지 방법을 보여주고 있습니다. (간단하게 하기위해, 텍스트는 UTF-8이라고 가정하며, fetch errors를 다루지 않는다고 합시다).

```
async function* makeTextFileLineIterator(fileURL) {
  const utf8Decoder = new TextDecoder("utf-8");
  let response = await fetch(fileURL);
  let reader = response.body.getReader();
  let {value: chunk, done: readerDone} = await reader.read();
  chunk = chunk ? utf8Decoder.decode(chunk) : "";

  let re = /\n|\r|\r\n/gm;
  let startIndex = 0;
  let result;

  for (;;) {
    let result = re.exec(chunk);
    if (!result) {
      if (readerDone) {
        break;
```

```
      }
      let remainder = chunk.substr(startIndex);
      ({value: chunk, done: readerDone} = await reader.read());
      chunk = remainder + (chunk ? utf8Decoder.decode(chunk) : "");
      startIndex = re.lastIndex = 0;
      continue;
    }
    yield chunk.substring(startIndex, result.index);
    startIndex = re.lastIndex;
  }
  if (startIndex < chunk.length) {
    // last line didn't end in a newline char
    yield chunk.substr(startIndex);
  }
}

for await (let line of makeTextFileLineIterator(urlOfFile)) {
  processLine(line);
}
```

fetch의 성공 여부를 체크

한가지 예를 들자면, 네트워크 error가 발생했을때 또는 CORS 가 서버단에서 잘못 설정되어있다면 `fetch()` promise 객체는 `TypeError` 메시지와 함께 반려 할것입니다. 비록 이 현상은 보통 허가 이슈나 그와 비슷한 것을 의미할지라도, 404 는 네트워크 error를 구성하지는 않습니다. 성공적인 `fetch()` 를 체크하는 정확한 방법은 promise 객체가 해결되었는지를 체크하는 것을 포함합니다. 그리고 `Response.ok` property 가 "true"의 값을 가지고 있는지 확인하는 것입니다. 코드는 아래와 같이 구현될겁니다:

```
fetch('flowers.jpg').then(function(response) {
  if(response.ok) {
    return response.blob();
  }
  throw new Error('Network response was not ok.');
}).then(function(myBlob) {
  var objectURL = URL.createObjectURL(myBlob);
  myImage.src = objectURL;
}).catch(function(error) {
  console.log('There has been a problem with your fetch operation: ', erro
});
```

request 객체를 fetch로 전달

`fetch()`를 사용해 요청한 리소스의 경로를 전달하는것 대신`Request()` 생성자를 사용해 Request 객체를 작성하여 `fetch()` 메서드를 인수로 전달하는것도 가능합니다.

```
var myHeaders = new Headers();

var myInit = { method: 'GET',
               headers: myHeaders,
               mode: 'cors',
               cache: 'default' };

var myRequest = new Request('flowers.jpg', myInit);

fetch(myRequest).then(function(response) {
  return response.blob();
}).then(function(myBlob) {
  var objectURL = URL.createObjectURL(myBlob);
  myImage.src = objectURL;
});
```

`fetch()` 메서드의 인수와 똑같은 인수를 `Request()` 객체에 전달하여 적용하는것이 가능합니다. 또한 Request 객체의 클론을 생성하기 위해 이미 존재하는 Request 객체를 전달하는것도 가능합니다.

```
1 | var anotherRequest = new Request(myRequest,myInit);
```

이것은 Request와 Resposne의 Body를 하나만 사용하고 있으므로 사용성이 높습니다.필요하면 `init` 객체를 변화시켜 Response나 Request를 재사용할 수 있도록 복사합니다. The copy must be made before the body is read, and reading the body in the copy will also mark it as read in the original request.

> 노트: `clone()` 메서드를 사용해 Request 객체의 클론을 생성할 수 있습니다. 다른 카피 메서드와 약간 다른 의미가 있습니다. 이전 요청의 body가 이미 읽어들여진 경우 전자는 실패하지만 `clone()` 메서드는 실패하지 않습니다. 이 기능은 Response와 동일합니다.

Headers

`Headers` 인터페이스에서 `Headers()` 생성자를 사용해 헤더 객체를 생성할 수 있습니다. 헤더 객체는 Key와 Value로 이루어진 간단한 multi-map입니다.

```
var content = "Hello World";
var myHeaders = new Headers();
myHeaders.append("Content-Type", "text/plain");
myHeaders.append("Content-Length", content.length.toString());
myHeaders.append("X-Custom-Header", "ProcessThisImmediately");
```

똑같이 배열을 전달하거나 객체 리터럴을 생성자에 전달하는것으로 생성할 수 있습니다.

```
myHeaders = new Headers({
  "Content-Type": "text/plain",
  "Content-Length": content.length.toString(),
  "X-Custom-Header": "ProcessThisImmediately",
});
```

다음과 같은 코드로 헤더의 내용을 들여다 볼 수 있습니다.

```
console.log(myHeaders.has("Content-Type")); // true
console.log(myHeaders.has("Set-Cookie")); // false
myHeaders.set("Content-Type", "text/html");
myHeaders.append("X-Custom-Header", "AnotherValue");

console.log(myHeaders.get("Content-Length")); // 11
console.log(myHeaders.getAll("X-Custom-Header")); // ["ProcessThisIm

myHeaders.delete("X-Custom-Header");
console.log(myHeaders.getAll("X-Custom-Header")); // [ ]
```

이러한 몇몇개의 조작법은 `ServiceWorkers`에서밖에 도움되지 않지만 헤더를 조작하기 위해서 보다 나은 API를 제공하고 있습니다.

모든 Header 메서드는 유효한 HTTP 헤더가 전달되지 않았을 경우 TypeError을 반환합니다. immutable Guard(다음 섹션 참고)가 설정되어 있는 경우에도 TypeError를 반환합니다. TypeError

를 반환하지 않고 실패하는 경우도 있습니다. 다음 예를 참고해주시기 바랍니다.

```
var myResponse = Response.error();
try {
  myResponse.headers.set("Origin", "http://mybank.com");
} catch(e) {
  console.log("은행이 아니잖아요!!");
}
```

헤더의 좋은 사용법으로 처리하기 전에 컨텐츠 타입으로 올바른가의 여부를 판별하는 방법이 있습니다. 예를 들어,

```
fetch(myRequest).then(function(response) {
    var contentType = response.headers.get('content-type');
    if(contentType && contentType.includes('application/json')) {
      return response.json();
    }
    throw new TypeError("Oops, we haven't got JSON!");
  })
  .then(function(json) { /* process your JSON further */ })
  .catch(function(error) { console.log(error); });
```

가드

헤더는 리퀘스트를 송신할 수 있으며 리스폰스를 수신할 수 있습니다. 어떤 정보를 수정할 수 있게 하기 위해, 혹은 수정하기 위해 여러 종류의 제어가 가능합니다. 헤더는 guard 프로퍼티는 이것을 가능하게 합니다. 가드는 Request나 Response에 포함되지 않지만 헤더 객체에서 조작 가능한 여러 메서드들의 사용 가능 여부에 영향을 미칩니다.

가드의 설정값은 다음과 같습니다.

- `none`: 기본치
- `request`: (`Request.headers`)에서 얻은 헤더 객체에 대한 가드
- `request-no-cors`: `Request.mode` `no-cors`에 생성된 (`Request.headers`)에서 사용할 수 있는 값만 헤더에 확보함
- `response`: (`Response.headers`) Response에서 얻은 객체애 대한 가드
- `immutable`: 대개 ServiceWorker에서 사용됨. 헤더의 설정을 읽기 전용으로 바꿈.

> 메모: `request`에서 가드된 헤더의`Content-Length` 헤더는 추가나 변경할 수 없는 가능성이 있습니다. 마찬가지로 리스폰스 헤더에 `Set-Cookie`를 삽입하는것은 불가능합니다.ServiceWorker는 동기 Reponse를 추출하여 쿠키를 설정합니다.

Response 객체

위에서 본 바와 같이 `Response` 인스턴스들은 fetch() promise가 resolve됐을때 반환됩니다.

아래는 어떤 리스폰스 객체라도 공통으로 사용되는 리스폰스 프로퍼티입니다.

- `Response.status` — HTTP Status의 정수치, 기본값 200
- `Response.statusText` — HTTP Status 코드의 메서드와 일치하는 문자열, 기본값은 "OK"
- `Response.ok` 상술한 프로퍼티에서 사용한 HTTP Status 코드가 200에서 299중 하나임을 체크하는 값, `Boolean`를 반환

Response 객체는 개발자의 손에 의해 동적으로 만드는것이 가능합니다. 이 방법은 `ServiceWorkers`내에서 활약할 때가 많습니다. 예를들어 Request를 획득했을 때 `respondWith()` 메서드에 의해 커스텀된 리스폰스를 반환하는 경우가 있습니다.

```
var myBody = new Blob();

addEventListener('fetch', function(event) {
  event.respondWith(
    new Response(myBody, {
      headers: { "Content-Type" : "text/plain" }
    })
  );
});
```

`Response()` 생성자는 파라미터로써 두개의 객체를 전달하는것이 가능합니다.첫번째는 Response Body, 두번째는 초기화 객체(`Request()`의 클론을 생성하는 방법과 비슷합니다.) 입니다.

> 付記: 정적 메서드 `error()`는 단순한 에러 Response를 반환합니다. `redirect()` 메서드 또한 지정한 URL에 리다이렉트할 Response를 반환합니다. 이것들은 Service Workers에서

> 만 사용되고 있습니다.

Body

Request, Response 둘다 Body를 가지고 있습니다. body는 아래에서 기술한 타입들 중 하나의 인스턴스입니다.

- `ArrayBuffer`
- `ArrayBufferView` (Uint8Array같은 TypedArray)
- `Blob`/File
- 문자열
- `URLSearchParams`
- `FormData`

`Body` 믹스인은 `Request`나 `Response`에 구현되어, 콘텐츠를 추출하기 위해 아래의 메서드가 정의되어 있습니다. 이러한 메서드들은 전부 최종적으로 요청으로 반환된 값을 내장하고 있는 promise를 반환합니다.

- `arrayBuffer()`
- `blob()`
- `json()`
- `text()`
- `formData()`

이것들은 비 텍스트 데이터를 XHR보다 훨씬 간단하게 사용하는것을 도와줍니다.

Request 바디는 body 파라미터를 전달하는 것으로 설정할 수 있습니다.

```
var form = new FormData(document.getElementById('login-form'));
fetch("/login", {
  method: "POST",
  body: form
})
```

Both request and response (and by extension the `fetch()` function), will try to intelligently determine the content type. A request will also automatically set a `Content-Type` header if none is set in the dictionary.

Feature detection(특징 추출)

Fetch API support는 `Headers`, `Request`, `Response` or `fetch()` on the `Window` or `Worker` 로 존재여부를 확인함으로써 추출할 수 있습니다. 예를 들어:

```
if (window.fetch) {
    // run my fetch request here
} else {
    // do something with XMLHttpRequest?
```

Polyfill

Fetch를 지원하지 않는 브라우저를 위해 미지원 브라우저를 위한 Fetch Polyfill이 지원되고 있습니다.

Specifications

Specification	Status	Comment
Fetch	LS Living Standard	Initial definition

Browser compatibility

Update compatibility data on GitHub

`fetch`	
Chrome	42
Edge	14
Firefox	39
IE	No
Opera	29
Safari	10.1
WebView Android	42
Chrome Android	42
Firefox Android	39
Opera Android	29
Safari iOS	10.3
Samsung Internet Android	4.0

Support for blob: and data:	
Chrome	48
Edge	79
Firefox	?
IE	No
Opera	?
Safari	?
WebView Android	43
Chrome Android	48
Firefox Android	?
Opera Android	?
Safari iOS	?
Samsung Internet Android	5.0

referrerPolicy	
Chrome	52

Edge	79
Firefox	52
IE	No
Opera	39
Safari	11.1
WebView Android	52
Chrome Android	52
Firefox Android	52
Opera Android	41
Safari iOS	No
Samsung Internet Android	6.0

signal	
Chrome	66
Edge	16
Firefox	57
IE	No
Opera	53
Safari	11.1
WebView Android	66
Chrome Android	66
Firefox Android	57
Opera Android	47
Safari iOS	11.3
Samsung Internet Android	9.0

Streaming response body	
Chrome	43
Edge	14
Firefox	Yes

IE	No
Opera	29
Safari	10.1
WebView Android	43
Chrome Android	43
Firefox Android	No
Opera Android	No
Safari iOS	10.3
Samsung Internet Android	4.0

What are we missing?

Full support

No support

Compatibility unknown

Experimental. Expect behavior to change in the future.

See implementation notes.

User must explicitly enable this feature.

관련항목

- ServiceWorker API
- HTTP 액세스 제어 (CORS)
- HTTP
- Fetch polyfill
- Fetch examples on Github

마지막 변경일: 2020년 6월 25일, by MDN contributors

관련 주제

Fetch API

- Guides
  - Using Fetch
  - Fetch basic concepts
  - Cross-global fetch usage [Translate]
- Interfaces
  - `Body`
  - `Headers`
  - `Request`
  - `Response`
- Methods
  - `WindowOrWorkerGlobalScope.fetch()`