CORS

다른 도메인으로부터 리소스가 요청될 경우 해당 리소스는 **cross-origin HTTP 요청** 에 의해 요청된다. 하지만 대부분의 브라우저들은 보안 상의 이유로 스크립트에서의 cross-origin HTTP 요청을 제한한다. 이것을 `Same-Origin-Policy(동일 근원 정책)`이라고 한다. 요청을 보내기 위해서는 요청을 보내고자 하는 대상과 프로토콜도 같아야 하고, 포트도 같아야 함을 의미한다.*(이 때, 서브 도메인 네임은 상관없다.)*

이러한 문제를 해결하기 위해 과거에는 flash 를 proxy 로 두고 타 도메인간 통신을 했다. 하지만 모바일 운영체제의 등장으로 flash 로는 힘들어졌다. (iOS 는 전혀 플래시를 지원하지 않는다.) 대체제로 나온 기술이 `JSONP(JSON-padding)` 이다. jQuery v.1.2 이상부터 `jsonp` 형태가 지원되 ajax 를 호출할 때 타 도메인간 호출이 가능해졌다. `JSONP` 에는 타 도메인간 자원을 공유할 수 있는 몇 가지 태그가 존재한다. 예를들어 `img`, `iframe`, `anchor`, `script`, `link` 등이 존재한다.

여기서 `CORS` 는 타 도메인 간에 자원을 공유할 수 있게 해주는 것이다. `Cross-Origin Resource Sharing` 표준은 웹 브라우저가 사용하는 정보를 읽을 수 있도록 허가된 **출처 집합**을 서버에게 알려주도록 허용하는 특정 HTTP 헤더를 추가함으로써 동작한다.

HTTP Header	Description
Access-Control-Allow-Origin	접근 가능한 `url` 설정
Access-Control-Allow-Credentials	접근 가능한 `쿠키` 설정
Access-Control-Allow-Headers	접근 가능한 `헤더` 설정
Access-Control-Allow-Methods	접근 가능한 `http method` 설정

Preflight Request

실제 요청을 보내도 안전한지 판단하기 위해 preflight 요청을 먼저 보내는 방법을 말한다. 즉, `Preflight Request` 는 실제 요청 전에 인증 헤더를 전송하여 서버의 허용 여부를 미리 체크하는 테스트 요청이다. 이 요청으로 트래픽이 증가할 수 있는데 서버의 헤더 설정으로 캐쉬가 가능하다. 서버 측에서는 브라우저가 해당 도메인에서 CORS 를 허용하는지 알아보기 위해 preflight 요청을 보내는데 이에 대한 처리가 필요하다. preflight 요청은 HTTP 의 `OPTIONS` 메서드를 사용하며 `Access-Control-Request-*` 형태의 헤더로 전송한다.

이는 브라우저가 강제하며 HTTP `OPTION` 요청 메서드를 이용해 서버로부터 지원 중인 메서드들을 내려받은 뒤, 서버에서 `approval(승인)` 시에 실제 HTTP 요청 메서드를 이용해 실제 요청을 전송하는 것이다.

Reference

- MDN - HTTP 접근 제어 CORS
- Cross-Origin-Resource-Sharing 에 대해서
- 구루비 - CORS 에 대해서

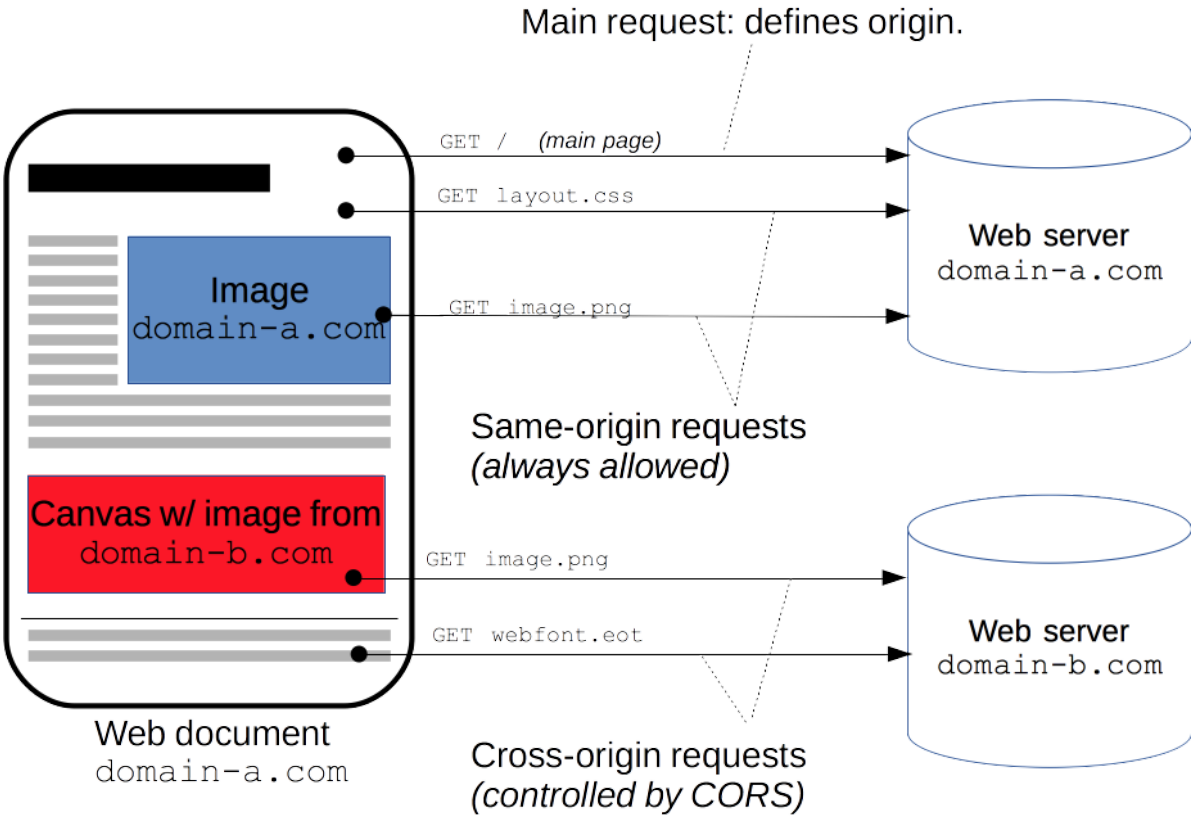
MDN - HTTP 접근 제어 CORS

이에 대한 응답으로 서버는 Access-Control-Allow-Origin 헤더를 다시 보냅니다.

교차 출처 리소스 공유(Cross-Origin Resource Sharing, CORS)는 추가 HTTP 헤더를 사용하여, 한 출처에서 실행 중인 웹 애플리케이션이 다른 출처의 선택한 자원에 접근할 수 있는 권한을 부여하도록 브라우저에 알려주는 체제입니다. 웹 애플리케이션은 리소스가 자신의 출처(도메인, 프로토콜, 포트)와 다를 때 교차 출처 HTTP 요청을 실행합니다.

교차 출처 요청의 예시: `https://domain-a.com`의 프론트 엔드 JavaScript 코드가 `XMLHttpRequest`를 사용하여 `https://domain-b.com/data.json`을 요청하는 경우.

보안 상의 이유로, 브라우저는 스크립트에서 시작한 교차 출처 HTTP 요청을 제한합니다. 예를 들어, `XMLHttpRequest`와 Fetch API는 동일 출처 정책을 따릅니다. 즉, 이 API를 사용하는 웹 애플리케이션은 자신의 출처와 동일한 리소스만 불러올 수 있으며, 다른 출처의 리소스를 불러오려면 그 출처에서 올바른 CORS 헤더를 포함한 응답을 반환해야 합니다.



CORS 체제는 브라우저와 서버 간의 안전한 교차 출처 요청 및 데이터 전송을 지원합니다. 최신 브라우저는 `XMLHttpRequest` 또는 Fetch와 같은 API에서 CORS를 사용하여 교차 출처 HTTP 요청의 위험을 완화합니다.

이 글은 누가 읽어야 하나요?

모든 사람이요, 진짜로.

명확히 말하자면, 이 글은 **웹 관리자**, **서버 개발자** 그리고 **프론트엔드 개발자**를 위한 것입니다. 최신 브라우저는 헤더와 정책 집행을 포함한 클라이언트 측 교차 출처 공유를 처리합니다. 그러나 CORS 표준에 맞춘다는 것은 서버에서도 새로운 요청과 응답 헤더를 처리해야 한다는 것입니다. 서버 개발자에게는 (PHP 코드 조각과 함께 하는) 서버 관점의 교차 출처 공유를 다루고 있는 다른 글로 보충하면 도움이 될 것입니다.

어떤 요청이 CORS를 사용하나요?

교차 출처 공유 표준은 다음과 같은 경우에 사이트간 HTTP 요청을 허용합니다.

- 위에서 논의한 바와 같이, `XMLHttpRequest`와 Fetch API 호출.

- 웹 폰트(CSS 내 `@font-face`에서 교차 도메인 폰트 사용 시), so that servers can deploy TrueType fonts that can only be cross-site loaded and used by web sites that are permitted to do so.
- WebGL 텍스쳐.
- `drawImage()`를 사용해 캔버스에 그린 이미지/비디오 프레임.
- 이미지로부터 추출하는 CSS Shapes.

이 글은 교차 출처 리소스 공유에 대한 일반적인 논의이며 필요한 HTTP 헤더에 대한 내용도 포함하고 있습니다.

기능적 개요

교차 출처 리소스 공유 표준은 웹 브라우저에서 해당 정보를 읽는 것이 허용된 출처를 서버에서 설명할 수 있는 새로운 HTTP 헤더를 추가함으로써 동작합니다. 추가적으로, 서버 데이터에 부수 효과(side effect)를 일으킬 수 있는 HTTP 요청 메서드(`GET`을 제외한 HTTP 메서드)에 대해, CORS 명세는 브라우저가 요청을 `OPTIONS` 메서드로 "프리플라이트"(preflight, 사전 전달)하여 지원하는 메서드를 요청하고, 서버의 "허가"가 떨어지면 실제 요청을 보내도록 요구하고 있습니다. 또한 서버는 클라이언트에게 요청에 "인증정보"(쿠키, HTTP 인증)를 함께 보내야 한다고 알려줄 수도 있습니다.

CORS 실패는 오류의 원인이지만, 보안상의 이유로 JavaScript에서는 오류의 상세 정보에 접근할 수 없으며, 알 수 있는 것은 오류가 발생했다는 사실 뿐입니다. 정확히 어떤 것이 실패했는지 알아내려면 브라우저의 콘솔을 봐야 합니다.

이후 항목에서는 시나리오와 함께, 사용한 HTTP 헤더의 상세 내용을 다룹니다.

접근 제어 시나리오 예제

교차 출처 리소스 공유가 동작하는 방식을 보여주는 세 가지 시나리오를 제시하겠습니다. 모든 예제는 지원하는 브라우저에서 교차 출처 요청을 생성할 수 있는 `XMLHttpRequest`를 사용합니다.

서버 관점의 교차 출처 리소스 공유에 대한 논의는 (PHP 코드와 함께 하는) 서버 사이드 접근 제어 (CORS) 문서에서 확인할 수 있습니다.

단순 요청(Simple requests)

일부요청은 CORS preflight 를 트리거하지 않습니다. Fetch 명세(CORS를 정의한)는 이 용어를 사용하지 않지만, 이 기사에서는 "simple requests"라고 하겠습니다. "simple requests"는 **다음 조건을 모두 충족하는 요청입니다:**

- 다음 중 하나의 메서드
  - `GET`
  - `HEAD`
  - `POST`
- 유저 에이전트가 자동으로 설정 한 헤더 (예를들어,

  `Connection`

  ,

  `User-Agent`

  ,
  Fetch 명세에서 “forbidden header name”으로 정의한 헤더

)외에, 수동으로 설정할 수 있는 헤더는 오직

Fetch 명세에서 “CORS-safelisted request-header”로 정의한 헤더

뿐입니다.

  - `Accept`
  - `Accept-Language`
  - `Content-Language`
  - `Content-Type` (아래의 추가 요구 사항에 유의하세요.)
  - `DPR`
  - `Downlink`
  - `Save-Data`
  - `Viewport-Width`
  - `Width`
- `Content-Type`

  헤더는 다음의 값들만 허용됩니다.

  - `application/x-www-form-urlencoded`
  - `multipart/form-data`
  - `text/plain`
- 요청에 사용된 `XMLHttpRequestUpload` 객체에는 이벤트 리스너가 등록되어 있지 않습니다. 이들은 `XMLHttpRequest.upload` 프로퍼티를 사용하여 접근합니다..
- 요청에 `ReadableStream` 객체가 사용되지 않습니다.

참고: 이는 웹 컨텐츠가 이미 발행할 수 있는 것과 동일한 종류의 cross-site 요청입니다. 서버가 적절한 헤더를 전송하지 않으면 요청자에게 응답 데이터가 공개되지 않습니다. 따라서 cross-site 요청 위조를 방지하는 사이트는 HTTP 접근 제어를 두려워 할 만한 부분이 없습니다.

주의: WebKit Nightly 와 Safari Technology Preview 는 `Accept`, `Accept-Language`, `Content-Language` 헤더에서 허용되는 값에 대한 추가 제약이 있습니다. 이러한 헤더 중 하나에 ”nonstandard” 값이 존재하면, WebKit/Safari 는 더이상 요청을 “simple request”로 간주하지 않습니다. 다음 Webkit 버그 외에 WebKit/Safari 가 “nonstandard” 으로 간주하는 값은 문서화되어 있지 않습니다.

- Require preflight for non-standard CORS-safelisted request headers Accept, Accept-Language, and Content-Language
- Allow commas in Accept, Accept-Language, and Content-Language request headers for simple CORS
- Switch to a blacklist model for restricted Accept headers in simple CORS requests

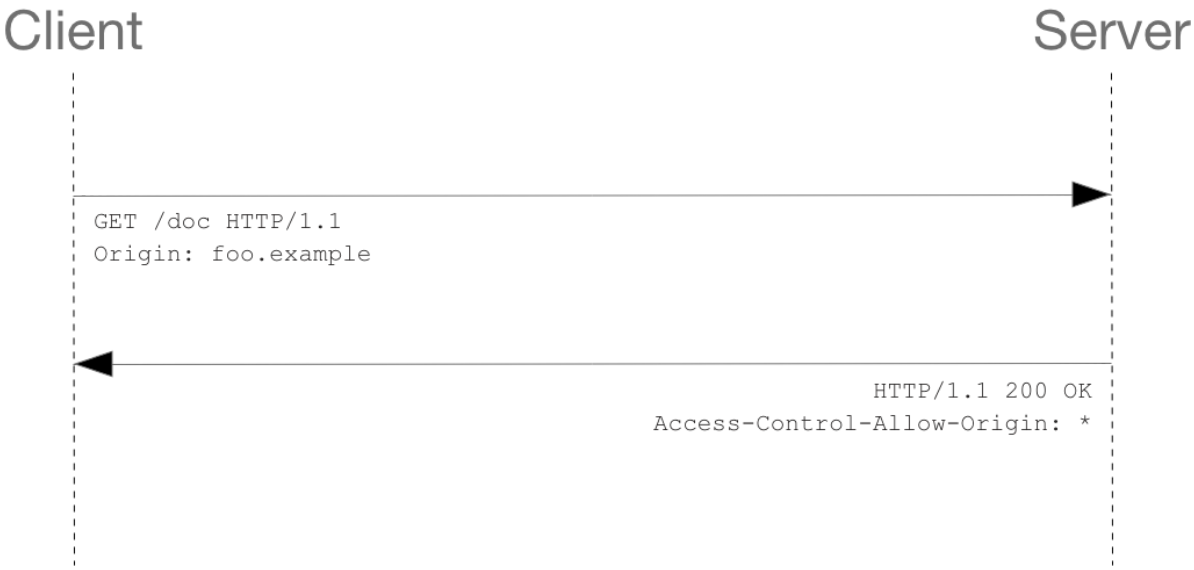
이 부분은 명세가 아니기 때문에 다른 브라우저에는 이러한 추가 제한 사항이 없습니다.

예를들어, `https://foo.example` 의 웹 컨텐츠가 `https://bar.other` 도메인의 컨텐츠를 호출하길 원합니다. `foo.example`에 배포된 자바스크립트에는 아래와 같은 코드가 사용될 수 있습니다.

```
const xhr = new XMLHttpRequest();
const url = 'https://bar.other/resources/public-data/';

xhr.open('GET', url);
xhr.onreadystatechange = someHandler;
xhr.send();
```

클라이언트와 서버간에 간단한 통신을 하고, CORS 헤더를 사용하여 권한을 처리합니다.



이 경우 브라우저가 서버로 전송하는 내용을 살펴보고, 서버의 응답을 확인합니다.

```
GET /resources/public-data/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101
Firefox/71.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive
Origin: https://foo.example
```

요청 헤더의 `Origin`을 보면, `https://foo.example`로부터 요청이 왔다는 것을 알 수 있습니다.

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 00:23:53 GMT
Server: Apache/2
Access-Control-Allow-Origin: *
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/xml

[…XML Data…]
```

서버는 이에 대한 응답으로 `Access-Control-Allow-Origin` 헤더를 다시 전송합니다. 가장 간단한 접근 제어 프로토콜은 `Origin` 헤더와 `Access-Control-Allow-Origin` 을 사용하는 것입니다. 이 경우 서버는 `Access-Control-Allow-Origin: *`, 으로 응답해야 하며, 이는 **모든** 도메인에서 접근할 수 있음을 의미합니다. `https://bar.other` 의 리소스 소유자가 *오직* `https://foo.example` 의 요청만 리소스에 대한 접근을 허용하려는 경우 다음을 전송합니다.

```
Access-Control-Allow-Origin: https://foo.example
```

이제 `https://foo.example` 이외의 도메인은 corss-site 방식으로 리소스에 접근할 수 없습니다. 리소스에 대한 접근을 허용하려면, `Access-Control-Allow-Origin` 헤더에는 요청의 `Origin` 헤더에서 전송된 값이 포함되어야 합니다.

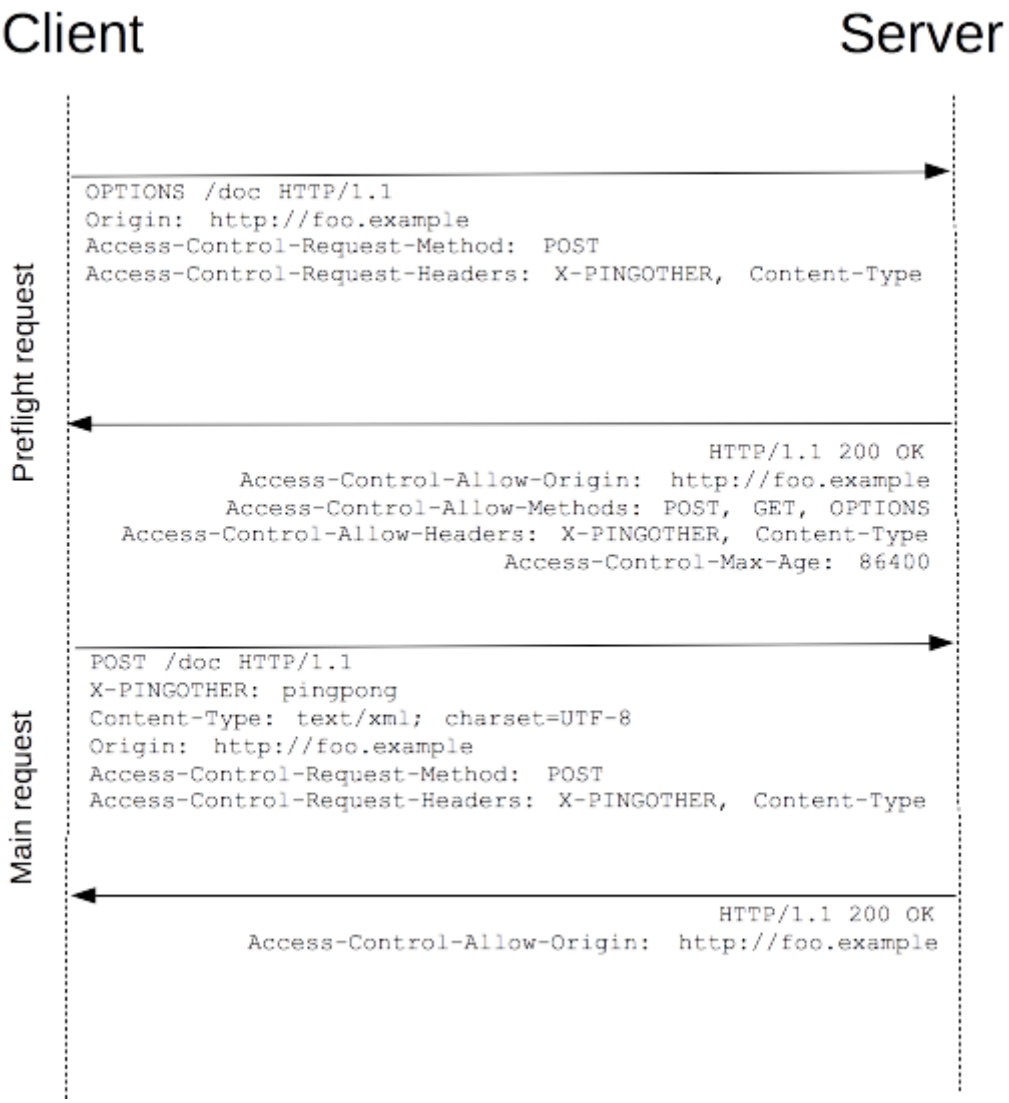
프리플라이트 요청

"preflighted" request는 위에서 논의한 "simple requests" 와는 달리, 먼저 `OPTIONS` 메서드를 통해 다른 도메인의 리소스로 HTTP 요청을 보내 실제 요청이 전송하기에 안전한지 확인합니다. Cross-site 요청은 유저 데이터에 영향을 줄 수 있기 때문에 이와같이 미리 전송(preflighted)합니다.

다음은 preflighted 할 요청의 예제입니다.

```
const xhr = new XMLHttpRequest();
xhr.open('POST', 'https://bar.other/resources/post-here/');
xhr.setRequestHeader('Ping-Other', 'pingpong');
xhr.setRequestHeader('Content-Type', 'application/xml');
xhr.onreadystatechange = handler;
xhr.send('<person><name>Arun</name></person>');
```

위의 예제는 `POST` 요청과 함께 함께 보낼 XML body를 만듭니다. 또한 비표준 HTTP `Ping-Other` 요청 헤더가 설정됩니다. 이러한 헤더는 HTTP/1.1의 일부가 아니지만 일반적으로 웹 응용 프로그램에 유용합니다. Content-Type 이 `application/xml` 이고, 사용자 정의 헤더가 설정되었기 때문에 이 요청은 preflighted 처리됩니다.



(참고: 아래 설명 된 것처럼 실제 `POST` 요청에는 `Access-Control-Request-*` 헤더가 포함되지 않습니다. `OPTIONS` 요청에만 필요합니다.)

클라이언트와 서버간의 완전한 통신을 살펴보겠습니다. 첫 번째 통신은 *preflight request/response*입니다.

```
OPTIONS /resources/post-here/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101
Firefox/71.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive
Origin: http://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-Type


HTTP/1.1 204 No Content
Date: Mon, 01 Dec 2008 01:15:39 GMT
Server: Apache/2
Access-Control-Allow-Origin: https://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
Vary: Accept-Encoding, Origin
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
```

preflight request가 완료되면 실제 요청을 전송합니다.

```
POST /resources/post-here/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101
Firefox/71.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive
X-PINGOTHER: pingpong
Content-Type: text/xml; charset=UTF-8
Referer: https://foo.example/examples/preflightInvocation.html
Content-Length: 55
Origin: https://foo.example
Pragma: no-cache
Cache-Control: no-cache

<person><name>Arun</name></person>


HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:15:40 GMT
Server: Apache/2
Access-Control-Allow-Origin: https://foo.example
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 235
Keep-Alive: timeout=2, max=99
Connection: Keep-Alive
Content-Type: text/plain
```

```
[Some GZIP'd payload]
```

첫 번째 예제의 1 - 10 행은 `OPTIONS` 메서드를 사용한 preflight request를 나타냅니다. 브라우저는 위의 자바스크립트 코드 스니펫이 사용중인 요청 파라미터를 기반으로 전송해야 합니다. 그렇게 해야 서버가 실제 요청 파라미터로 요청을 보낼 수 있는지 여부에 응답할 수 있습니다. OPTIONS는 서버에서 추가 정보를 판별하는데 사용하는 HTTP/1.1 메서드입니다. 또한 safe 메서드이기 때문에, 리소스를 변경하는 데 사용할 수 없습니다. OPTIONS 요청과 함께 두 개의 다른 요청 헤더가 전송됩니다. (10, 11행)

```
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-Type
```

`Access-Control-Request-Method` 헤더는 preflight request의 일부로, 실제 요청을 전송할 때 `POST` 메서드로 전송된다는 것을 알려줍니다. `Access-Control-Request-Headers` 헤더는 실제 요청을 전송 할 때 `X-PINGOTHER` 와 `Content-Type` 사용자 정의 헤더와 함께 전송된다는 것을 서버에 알려줍니다. 이제 서버는 이러한 상황에서 요청을 수락할지 결정할 수 있습니다.

위의 13 - 22 행은 서버가 요청 메서드와 (`POST`) 요청 헤더를 (`X-PINGOTHER`) 받을 수 있음을 나타내는 응답입니다. 특히 16 - 19행을 살펴보겠습니다.

```
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
```

서버는 `Access-Control-Allow-Methods` 로 응답하고 `POST` 와 `GET` 이 리소스를 쿼리하는데 유용한 메서드라고 가르쳐줍니다. 이 헤더는 `Allow` 응답 헤더와 유사하지만, 접근 제어 컨텍스트 내에서 엄격하게 사용됩니다.

또한 `Access-Control-Allow-Headers` 의 값을 "`X-PINGOTHER, Content-Type`" 으로 전송하여 실제 요청에 헤더를 사용할 수 있음을 확인합니다. `Access-Control-Allow-Methods` 와 마찬가지로 `Access-Control-Allow-Headers` 는 쉼표로 구분된 허용 가능한 헤더 목록입니다.

마지막으로 `Access-Control-Max-Age` 는 다른 preflight request를 보내지 않고, preflight request에 대한 응답을 캐시할 수 있는 시간(초)을 제공합니다. 위의 코드는 86400 초(24시간) 입니다. 각 브라우저의 최대 캐싱 시간 은 `Access-Control-Max-Age` 가 클수록 우선순위가 높습니다.

Preflighted requests 와 리다이렉트

모든 브라우저가 preflighted request 후 리다이렉트를 지원하지는 않습니다. preflighted request 후 리다이렉트가 발생하면 일부 브라우저는 다음과 같은 오류 메시지를 띄웁니다.

> 요청이 'https://example.com/foo'로 리다이렉트 되었으며, preflight가 필요한 cross-origin 요청은 허용되지 않습니다.

> 요청에 preflight가 필요합니다. preflight는 cross-origin 리다이렉트를 허용하지 않습니다.

CORS 프로토콜은 본래 그 동작(리다이렉트)이 필요했지만, 이후 더 이상 필요하지 않도록 변경되었습니다. 그러나 모든 브라우저가 변경 사항을 구현하지는 않았기 때문에, 본래의 필요한 동작은 여전히 나타납니다.

브라우저가 명세를 따라잡을 때 까지 다음 중 하나 혹은 둘 다를 수행하여 이 제한을 해결할 수 있습니다.

- preflight 리다이렉트를 방지하기 위해 서버측 동작을 변경
- preflight를 발생시키지 않는 simple request 가 되도록 요청을 변경

이것이 가능하지 않은 경우 다른 방법도 있습니다.

1. Fetch API를 통해 `Response.url` 이나 `XMLHttpRequest.responseURL` 를 사용하여 simple request 를 작성합니다. 이 simple request를 이용하여 실제 preflighted request가 끝나는 URL을 판별하세요.
2. 첫 번째 단계에서 `Response.url` 혹은 `XMLHttpRequest.responseURL` 로부터 얻은 URL을 사용하여 또 다른 요청(실제 요청)을 만듭니다.

그러나 요청에 `Authorization` 헤더가 있기 때문에 preflight를 트리거하는 요청일 경우에, 위의 단계를 사용하여 제한을 제거할 수 없습니다. 또한 요청이 있는 서버를 제어하지 않으면 문제를 해결할 수 없습니다.

인증정보를 포함한 요청

`XMLHttpRequest` 혹은 Fetch 를 사용할 때 CORS 에 의해 드러나는 가장 흥미로운 기능은 "credentialed" requests 입니다. credentialed requests는 HTTP cookies 와 HTTP Authentication 정보를 인식합니다. 기본적으로 cross-site `XMLHttpRequest` 나 Fetch 호출에서 브라우저는 자격 증명을 보내지 **않습니다.** `XMLHttpRequest` 객체나 `Request` 생성자가 호출될 때 특정 플래그를 설정해야 합니다.

이 예제에서 원래 `http://foo.example` 에서 불러온 컨텐츠는 쿠키를 설정하는 `http://bar.other` 리소스에 simple GET request를 작성합니다. foo.example의 내용은 다음과 같은 자바스크립트를 포함할 수 있습니다.

```
const invocation = new XMLHttpRequest();
const url = 'http://bar.other/resources/credentialed-content/';

function callOtherDomain() {
  if (invocation) {
    invocation.open('GET', url, true);
    invocation.withCredentials = true;
    invocation.onreadystatechange = handler;
    invocation.send();
  }
}
```

7행은 쿠키와 함께 호출하기위한 `XMLHttpRequest` 의 플래그를 보여줍니다. 이 플래그는 `withCredentials` 라고 불리며 부울 값을 갖습니다. 기본적으로 호출은 쿠키 없이 이루어집니다. 이것은 simple `GET` request이기 때문에 preflighted 되지 않습니다. 그러나 브라우저는 `Access-Control-Allow-Credentials` : `true` 헤더가 없는 응답을 **거부합니다**. 따라서 호출된 웹 컨텐츠에 응답을 제공하지 **않습니다**.

클라이언트와 서버간의 통신 예제는 다음과 같습니다.

```
GET /resources/credentialed-content/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101
Firefox/71.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive
Referer: http://foo.example/examples/credential.html
Origin: http://foo.example
Cookie: pageAccess=2


HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:34:52 GMT
Server: Apache/2
Access-Control-Allow-Origin: https://foo.example
Access-Control-Allow-Credentials: true
Cache-Control: no-cache
Pragma: no-cache
Set-Cookie: pageAccess=3; expires=Wed, 31-Dec-2008 01:34:53 GMT
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 106
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/plain


[text/plain payload]
```

10행에는 `http://bar.other`의 컨텐츠를 대상으로 하는 쿠키가 포함되어 있습니다. 하지만 17행의 `Access-Control-Allow-Credentials` : `true` 로 응답하지 않으면, 응답은 무시되고 웹 컨텐츠는 제공되지 않습니다.

자격증명 요청 및 와일드카드(Credentialed requests and wildcards)

credentialed request 에 응답할 때 서버는 `Access-Control-Allow-Origin` 헤더 "`*`" 와일드카드를 사용하는 대신에 **반드시** 에 값을 지정해야 합니다.

위 예제의 요청 헤더에 `Cookie` 헤더가 포함되어 있기 때문에 `Access-Control-Allow-Origin` 헤더의 값이 "*"인 경우 요청이 실패합니다. 위 요청은 `Access-Control-Allow-Origin` 헤더가 "`*`" 와일드 카드가 아니라 "`http://foo.example`" 본래 주소이기 때문에 자격증명 인식 컨텐츠는 웹 호출 컨텐츠로 리턴됩니다.

위 예제의 `Set-Cookie` 응답 헤더는 추가 쿠키를 설정합니다. 실패한 경우 사용한 API에 따라 예외가 발생합니다.

Third-party cookies

CORS 응답에 설정된 쿠키에는 일반적인 third-party cookie 정책이 적용됩니다. 위의 예제는 `foo.example`에서 페이지를 불러지만 20행의 쿠키는 `bar.other` 가 전송합니다. 때문에 사용자의 브라우저 설정이 모든 third-party cookies를 거부하도록 되어 있다면, 이 쿠키는 저장되지 않습니다.

HTTP 응답 헤더

이 섹션에서는 Cross-Origin 리소스 공유 명세에 정의된 대로 서버가 접근 제어 요청을 위해 보내는 HTTP 응답 헤더가 나열되어 있습니다. The previous section gives an overview of these in action.

Access-Control-Allow-Origin

리턴된 리소스에는 다음 구문과 함께 하나의 `Access-Control-Allow-Origin` 헤더가 있을 수 있습니다.

```
Access-Control-Allow-Origin: <origin> | *
```

`Access-Control-Allow-Origin` 은 단일 출처를 지정하여 브라우저가 해당 출처가 리소스에 접근하도록 허용합니다. 또는 자격 증명이 **없는** 요청의 경우 "`*`" 와일드 카드는 브라우저의 origin에 상관없이 모든 리소스에 접근하도록 허용합니다.

예를들어 `https://mozilla.org` 의 코드가 리소스에 접근 할 수 있도록 하려면 다음과 같이 지정할 수 있습니다.

```
Access-Control-Allow-Origin: https://mozilla.org
```

서버가 "`*`" 와일드카드 대신에 하나의 origin을 지정하는 경우, 서버는 `Vary` 응답 헤더에 `Origin` 을 포함해야 합니다. 이 origin은 화이트 리스트의 일부로 요청 orgin에 따라 동적으로 변경될 수 있습니다. 서버 응답이 `Origin` 요청 헤더에 따라 다르다는것을 클라이언트에 알려줍니다.

Access-Control-Expose-Headers

`Access-Control-Expose-Headers` 헤더를 사용하면 브라우저가 접근할 수 있는 헤더를 서버의 화이트 리스트에 추가할 수 있습니다.

```
Access-Control-Expose-Headers: <header-name>[, <header-name>]*
```

예를들면 다음과 같습니다.

```
Access-Control-Expose-Headers: X-My-Custom-Header, X-Another-Custom-Header
```

`X-My-Custom-Header` 와 `X-Another-Custom-Header` 헤더가 브라우저에 드러납니다.

Access-Control-Max-Age

`Access-Control-Max-Age` 헤더는 preflight request 요청 결과를 캐시할 수 있는 시간을 나타냅니다. preflight request 예제는 위를 참조하세요.

```
Access-Control-Max-Age: <delta-seconds>
```

`delta-seconds` 파라미터는 결과를 캐시할 수 있는 시간(초)를 나타냅니다.

Access-Control-Allow-Credentials

`Access-Control-Allow-Credentials` 헤더는 `credentials` 플래그가 true일 때 요청에 대한 응답을 표시할 수 있는지를 나타냅니다. preflight request에 대한 응답의 일부로 사용하는 경우, credentials을 사용하여 실제 요청을 수행할 수 있는지를 나타냅니다. simple `GET` requests는 preflighted되지 않으므로 credentials이 있는 리소스를 요청하면, 이 헤더가 리소스와 함께 반환되지 않습니다. 이 헤더가 없으

면 브라우저에서 응답을 무시하고 웹 컨텐츠로 반환되지 않는다는 점을 주의하세요.

```
Access-Control-Allow-Credentials: true
```

Credentialed requests 은 위에 설명되어 있습니다.

Access-Control-Allow-Methods

`Access-Control-Allow-Methods` 헤더는 리소스에 접근할 때 허용되는 메서드를 지정합니다. 이 헤더는 preflight request에 대한 응답으로 사용됩니다. 요청이 preflighted 되는 조건은 위에 설명되어 있습니다.

```
Access-Control-Allow-Methods: <method>[, <method>]*
```

이 헤더를 브라우저로 전송하는 예제를 포함하여 preflight request 의 예제는, 위에 나와 있습니다.

Access-Control-Allow-Headers

preflight request 에 대한 응답으로 `Access-Control-Allow-Headers` 헤더가 사용됩니다. 실제 요청시 사용할 수 있는 HTTP 헤더를 나타냅니다.

```
Access-Control-Allow-Headers: <header-name>[, <header-name>]*
```

HTTP 요청 헤더

이 섹션에는 cross-origin 공유 기능을 사용하기 위해 클라이언트가 HTTP 요청을 발행할 때 사용할 수 있는 헤더가 나열되어 있습니다. 이 헤더는 서버를 호출할 때 설정됩니다. cross-site `XMLHttpRequest` 기능을 사용하는 개발자는 프로그래밍 방식으로 cross-origin 공유 요청 헤더를 설정할 필요가 없습니다.

Origin

`Origin` 헤더는 cross-site 접근 요청 또는 preflight request의 출처를 나타냅니다.

```
Origin: <origin>
```

origin 은 요청이 시작된 서버를 나타내는 URI 입니다. 경로 정보는 포함하지 않고, 오직 서버 이름만 포함합니다.

참고: `origin` 값은 `null` 또는 URI 가 올 수 있습니다.

접근 제어 요청에는 **항상** `Origin` 헤더가 전송됩니다.

Access-Control-Request-Method

`Access-Control-Request-Method` 헤더는 실제 요청에서 어떤 HTTP 메서드를 사용할지 서버에게 알려주기 위해, preflight request 할 때에 사용됩니다.

```
Access-Control-Request-Method: <method>
```

이 사용법의 예제는 위에서 찾을 수 있습니다.

Access-Control-Request-Headers

`Access-Control-Request-Headers` 헤더는 실제 요청에서 어떤 HTTP 헤더를 사용할지 서버에게 알려주기 위해, preflight request 할 때에 사용됩니다.

```
Access-Control-Request-Headers: <field-name>[, <field-name>]*
```

이 사용법의 예제는 위에서 찾을 수 있습니다.

명세

Specification	Status	Comment
Fetch The definition of 'CORS' in that specification.	Living Standard	New definition; supplants W3C CORS specification.

브라우저 호환성

Report problems with this compatibility data on GitHub

	desktop						mobile					
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox Android	Opera Android	iOS Safari	Samsung Internet
`Access-Control-Allow-Origin`	Full support4	Full support12	Full support3.5	Full support10	Full support12	Full support4	Full support2	Full supportYes	Full support4	Full support12	Full support3.2	Full supportYes

Legend

Cross-Origin-Resource-Sharing 에 대해서

Jul 21, 2015 in Network

개요

HTTP 요청은 기본적으로 Cross-Site HTTP Requests가 가능하다.

다시 말하면, `<img>` 태그로 다른 도메인의 이미지 파일을 가져오거나, `<link>` 태그로 다른 도메인의 CSS를 가져오거나, `<script>` 태그로 다른 도메인의 JavaScript 라이브러리를 가져오는 것이 모두 가능하다.

하지만 `<script></script>`로 둘러싸여 있는 **스크립트**에서 생성된 Cross-Site HTTP Requests는 **Same Origin Policy**를 적용 받기 때문에 Cross-Site HTTP Requests가 불가능하다.

AJAX가 널리 사용되면서 `<script></script>`로 둘러싸여 있는 스크립트에서 생성되는 `XMLHttpRequest`에 대해서도 Cross-Site HTTP Requests가 가능해야 한다는 요구가 늘어나자 W3C에서 CORS라는 이름의 권고안이 나오게 되었다.

CORS 요청의 종류

CORS 요청은 Simple/Preflight, Credential/Non-Credential의 조합으로 4가지가 존재한다.

브라우저가 요청 내용을 분석하여 4가지 방식 중 해당하는 방식으로 서버에 요청을 날리므로, 프로그래머가 목적에 맞는 방식을 선택하고 그 조건에 맞게 코딩해야 한다.

Simple Request

아래의 3가지 조건을 모두 만족하면 Simple Request

- GET, HEAD, POST 중의 한 가지 방식을 사용해야 한다.
- POST 방식일 경우 Content-type이 아래 셋 중의 하나여야 한다.
    - application/x-www-form-urlencoded
    - multipart/form-data
    - text/plain
- 커스텀 헤더를 전송하지 말아야 한다.

Simple Request는 서버에 1번 요청하고, 서버도 1번 회신하는 것으로 처리가 종료된다.

```
Simple RequestGET /resources/public-data/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre)
Gecko/20081130 Minefield/3.1b3pre
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7
Connection: keep-alive
Referer: http://foo.example/examples/access-control/simpleXSInvocation.html
Origin: http://foo.example


HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 00:23:53 GMT
Server: Apache/2.0.61
Access-Control-Allow-Origin: *
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/xml

[XML Data]
```

Preflight Request

Simple Request 조건에 해당하지 않으면 브라우저는 Preflight Request 방식으로 요청한다.

따라서, Preflight Request는

- GET, HEAD, POST 외의 다른 방식으로도 요청을 보낼 수 있고,
- application/xml 처럼 다른 Content-type으로 요청을 보낼 수도 있으며,
- 커스텀 헤더도 사용할 수 있다.

이름에서 짐작할 수 있듯, Preflight Request는 예비 요청과 본 요청으로 나뉘어 전송된다.

먼저 서버에 예비 요청(Preflight Request)를 보내고 서버는 예비 요청에 대해 응답하고,
그 다음에 본 요청(Actual Request)을 서버에 보내고, 서버도 본 요청에 응답한다.

하지만, 예비 요청과 본 요청에 대한 서버단의 응답을 프로그래머가 프로그램 내에서 구분하여 처리하는 것은 아니다.
프로그래머가 `Access-Control-` 계열의 Response Header만 적절히 정해주면,
OPTIONS 요청으로 오는 예비 요청과 GET, POST, HEAD, PUT, DELETE 등으로 오는 본 요청의 처리는 서버가 알아서 처리한다.

아래는 Preflight Requests로 오가는 HEADER를 보여준다.

다시 강조하지만, 아래 내용에서 프로그래머가 OPTIONS 요청의 처리 로직과 POST 요청의 처리 로직을 구분하여 구현하는 것이 아니다.

```
Preflight Request and Actual RequestOPTIONS /resources/post-here/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre)
Gecko/20081130 Minefield/3.1b3pre
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7
Connection: keep-alive
Origin: http://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER


HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:15:39 GMT
Server: Apache/2.0.61 (Unix)
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER
Access-Control-Max-Age: 1728000
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 0
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/plain

POST /resources/post-here/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre)
Gecko/20081130 Minefield/3.1b3pre
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7
Connection: keep-alive
X-PINGOTHER: pingpong
Content-Type: text/xml; charset=UTF-8
Referer: http://foo.example/examples/preflightInvocation.html
Content-Length: 55
Origin: http://foo.example
Pragma: no-cache
Cache-Control: no-cache

<?xml version="1.0"?><person><name>Arun</name></person>
```

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:15:40 GMT
Server: Apache/2.0.61 (Unix)
Access-Control-Allow-Origin: http://foo.example
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 235
Keep-Alive: timeout=2, max=99
Connection: Keep-Alive
Content-Type: text/plain

[Some GZIP'd payload]
```

Request with Credential

HTTP Cookie와 HTTP Authentication 정보를 인식할 수 있게 해주는 요청

```
Simple Credential Requestvar invocation = new XMLHttpRequest();
var url = 'http://bar.other/resources/credentialed-content/';

function callOtherDomain(){

  if(invocation) {
    invocation.open('GET', url, true);
    invocation.withCredentials = true;
    invocation.onreadystatechange = handler;
    invocation.send();
  }
  ...
```

요청 시 `xhr.withCredentials = true`를 지정해서 Credential 요청을 보낼 수 있고,
서버는 Response Header에 반드시 `Access-Control-Allow-Credentials: true`를 포함해야 하고,
`Access-Control-Allow-Origin` 헤더의 값에는 `\*`가 오면 안되고 `http://foo.origin`과 같은 구체적인 도메인이 와야 한다.

```
Server Response Header to Simple Request with CredentialHTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:34:52 GMT
Server: Apache/2.0.61 (Unix) PHP/4.4.7 mod_ssl/2.0.61 OpenSSL/0.9.7e
mod_fastcgi/2.4.2 DAV/2 SVN/1.4.2
X-Powered-By: PHP/5.2.6
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Credentials: true
Cache-Control: no-cache
Pragma: no-cache
Set-Cookie: pageAccess=3; expires=Wed, 31-Dec-2008 01:34:53 GMT
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 106
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/plain


[text/plain payload]
```

Request without Credential

CORS 요청은 기본적으로 Non-Credential 요청이므로, `xhr.withCredentials = true`를 지정하지 않으면 Non-Credential 요청이다.

CORS 관련 HTTP Response Headers

서버에서 CORS 요청을 처리할 때 지정하는 헤더

Access-Control-Allow-Origin

Access-Control-Allow-Origin 헤더의 값으로 지정된 도메인으로부터의 요청만 서버의 리소스에 접근할 수 있게 한다.

```
Response HeaderAccess-Control-Allow-Origin: <origin> | *
```

`<origin>`에는 요청 도메인의 URI를 지정한다.
모든 도메인으로부터의 서버 리소스 접근을 허용하려면 `*`를 지정한다. Request with Credential의 경우에는 `*`를 사용할 수 없다.

Access-Control-Expose-Headers

기본적으로 브라우저에게 노출이 되지 않지만, 브라우저 측에서 접근할 수 있게 허용해주는 헤더를 지정한다.

기본적으로 브라우저에게 노출이 되는 HTTP Response Header는 아래의 6가지 밖에 없다.

- Cache-Control
- Content-Language
- Content-Type
- Expires
- Last-Modified
- Pragma

다음과 같이 `Access-Control-Expose-Headers`를 Response Header에 지정하여 회신하면 브라우저 측에서 커스텀 헤더를 포함하여, 기본적으로는 접근할 수 없었던 Content-Length 헤더 정보도 알 수 있게 된다.

```
Response HeaderAccess-Control-Expose-Headers: Content-Length, X-My-Custom-Header,
X-Another-Custom-Header
```

Access-Control-Max-Age

Preflight Request의 결과가 캐쉬에 얼마나 오래동안 남아있는지를 나타낸다.

```
Response HeaderAccess-Control-Max-Age: <delta-seconds>
```

Access-Control-Allow-Credentials

Request with Credential 방식이 사용될 수 있는지를 지정한다.

```
Response HeaderAccess-Control-Allow-Credentials: true | false
```

예비 요청에 대한 응답에 `Access-Control-Allow-Credentials: false`를 포함하면, 본 요청은 Request with Credential을 보낼 수 없다.

Simple Request에 `withCredentials = true`가 지정되어 있는데, Response Header에 `Access-Control-Allow-Credentials: true`가 명시되어 있지 않다면, 그 Response는 브라우저에 의해 무시된다.

Access-Control-Allow-Methods

예비 요청에 대한 Response Header에 사용되며, 서버의 리소스에 접근할 수 있는 HTTP Method 방식을 지정한다.

```
Response HeaderAccess-Control-Allow-Methods: <method>[, <method>]*
```

Access-Control-Allow-Headers

예비 요청에 대한 Response Header에 사용되며, 본 요청에서 사용할 수 있는 HTTP Header를 지정한다.

```
Response HeaderAccess-Control-Allow-Headers: <field-name>[, <field-name>]*
```

CORS 관련 HTTP Request Headers

클라이언트가 서버에 CORS 요청을 보낼 때 사용하는 헤더로, 브라우저가 자동으로 지정하며, XMLHttpRequest를 사용하는 프로그래머가 직접 지정해 줄 필요 없다.

Origin

Cross-site 요청을 날리는 요청 도메인 URI을 나타내며, access control이 적용되는 모든 요청에 `Origin` 헤더는 반드시 포함된다.

```
Request HeaderOrigin: <origin>
```

`<origin>`은 서버 이름(포트 포함)만 포함되며 경로 정보는 포함되지 않는다.

`<origin>`은 공백일 수도 있는데, 소스가 data URL일 경우에 유용하다.

Access-Control-Request-Method

예비 요청을 보낼 때 포함되어, 본 요청에서 어떤 HTTP Method를 사용할 지 서버에게 알려준다.

```
Request HeaderAccess-Control-Request-Method: <method>
```

Access-Control-Request-Headers

예비 요청을 보낼 때 포함되어, 본 요청에서 어떤 HTTP Header를 사용할 지 서버에게 알려준다.

```
Request HeaderAccess-Control-Request-Headers: <field-name>[, <field-name>]*
```

XDomainRequest

`XDomainRequest`(XDR)는 W3C 표준이 아니며, IE 8, 9에서 비동기 CORS 통신을 위해 Microsoft에서 만든 객체다.

- XDR은 `setRequestHeader`가 없다.
- XDR과 XHR을 구분하려면 `obj.contentType`을 사용한다.(XHR에는 이게 없음)
- XDR은 http와 https 프로토콜만 가능

결론

- CORS를 쓰면 AJAX로도 Same Origin Policy의 제약을 넘어 다른 도메인의 자원을 사용할 수 있다.
- CORS를 사용하려면
    - 클라이언트에서 `Access-Control-**` 류의 HTTP Header를 서버에 보내야 하고,
    - 서버도 `Access-Control-**` 류의 HTTP Header를 클라이언트에 회신하게 되어 있어야 한다.