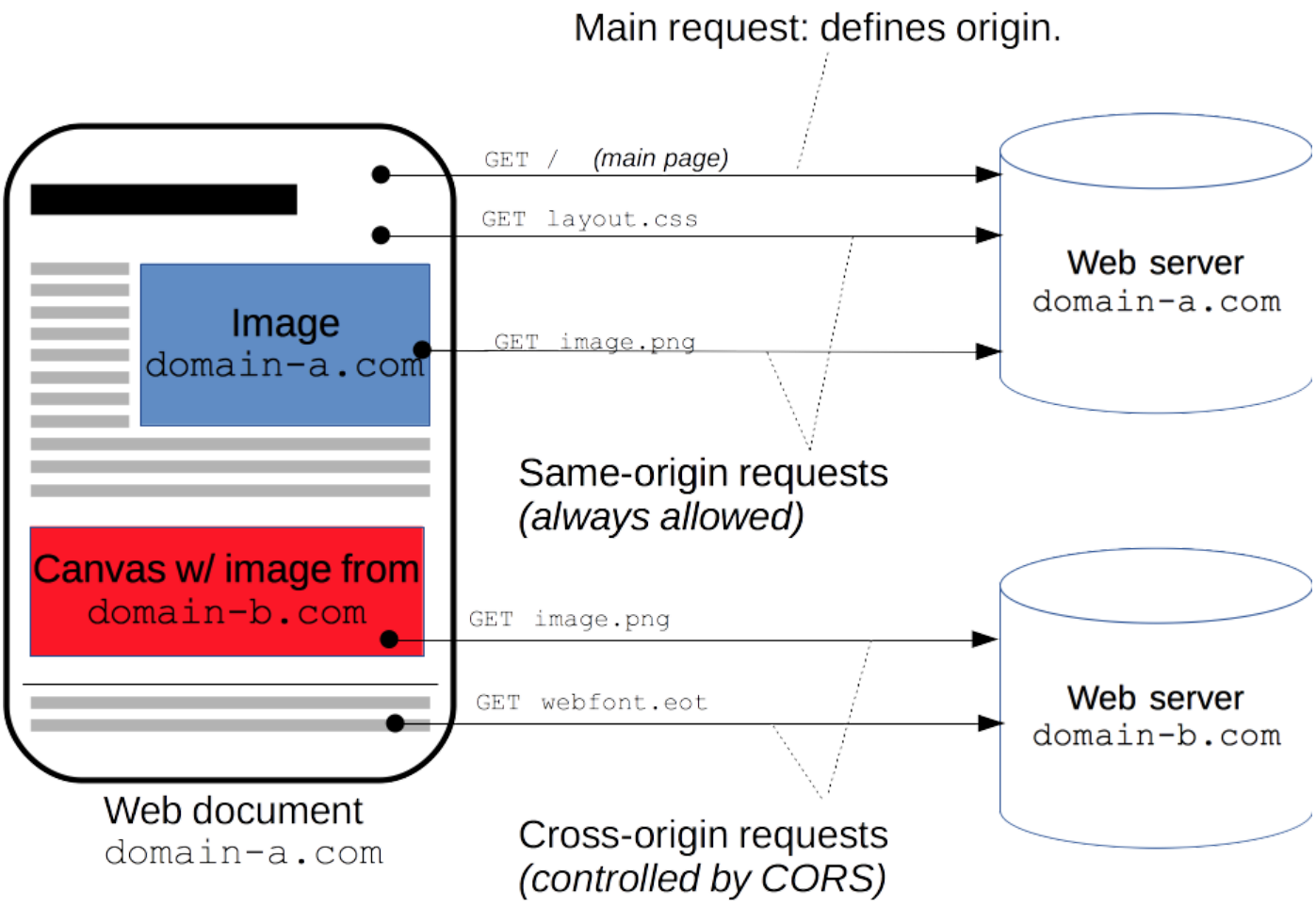
Cette page a été traduite à partir de l'anglais par la communauté. Vous pouvez également contribuer en rejoignant la communauté francophone sur MDN Web Docs.

Cross-origin resource sharing (CORS)

Le « *Cross-origin resource sharing* » (CORS) ou « partage des ressources entre origines multiples » (en français, moins usité) est un mécanisme qui consiste à ajouter des en-têtes HTTP afin de permettre à un agent utilisateur d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant. Un agent utilisateur réalise une requête HTTP **multi-origine (*cross-origin*)** lorsqu'il demande une ressource provenant d'un domaine, d'un protocole ou d'un port différent de ceux utilisés pour la page courante.

Prenons un exemple de requête multi-origine : une page HTML est servie depuis `http://domaine-a.com` contient un élément `<img> src` ciblant `http://domaine-b.com/image.jpg`. Aujourd'hui, de nombreuses pages web chargent leurs ressources (feuilles CSS, images, scripts) à partir de domaines séparés (par exemple des CDN (*Content Delivery Network* en anglais ou « Réseau de diffusion de contenu »).

Pour des raisons de sécurité, les requêtes HTTP multi-origine émises depuis les scripts sont restreintes. Ainsi, `XMLHttpRequest` et l'API Fetch respectent la règle d'origine unique. Cela signifie qu'une application web qui utilise ces API peut uniquement émettre des requêtes vers la même origine que celle à partir de laquelle l'application a été chargée, sauf si des en-têtes CORS sont utilisés.



Le CORS permet de prendre en charge des requêtes multi-origines sécurisées et des transferts de données entre des navigateurs et des serveurs web. Les navigateurs récents utilisent le CORS dans une API contenante comme `XMLHttpRequest` ou `Fetch` pour aider à réduire les risques de requêtes HTTP multi-origines.

À qui est destiné cet article ?

Cet article est destiné à toutes et à tous.

Il pourra notamment servir aux administrateurs web, aux développeurs côté serveur ainsi qu'aux développeurs côté client. Les navigateurs récents permettent de gérer les règles de partage multi

développeurs côté client. Les navigateurs récents permettent de gérer les règles de partage multi-origine côté client grâce à certaines règles et en-têtes mais cela implique également que des serveurs puissent gérer ces requêtes et réponses. Aussi, pour compléter le spectre concerné, nous vous invitons à lire d'autres articles complétant le point de vue « serveur » (par exemple cet article utilisant des fragments de code PHP (en-US)).

Quelles requêtes utilisent le CORS ?

Le standard CORS est utilisé afin de permettre les requêtes multi-origines pour :

- L'utilisation des API `XMLHttpRequest` ou Fetch
- Les polices web (pour récupérer des polices provenant d'autres origines lorsqu'on utilise `@font-face` en CSS),
  afin que les serveurs puissent déployer des polices TrueType uniquement chargées en *cross-site* et utilisées par les sites web qui l'autorisent
- Les textures WebGL
- Les *frames* (images ou vidéo) dessinées sur un canevas avec `drawImage`
- Les feuilles de style (pour les accès CSSOM)
- Les scripts (pour les exceptions non silencieuses (*unmuted exceptions*)).

Cet article propose un aperçu général de *Cross-Origin Resource Sharing* ainsi qu'un aperçu des en-têtes HTTP nécessaires.

Aperçu fonctionnel

Le standard CORS fonctionne grâce à l'ajout de nouveaux en-têtes HTTP qui permettent aux serveurs de décrire un ensemble d'origines autorisées pour lire l'information depuis un navigateur web. De plus, pour les méthodes de requêtes HTTP qui entraînent des effets de bord sur les données côté serveur (notamment pour les méthodes en dehors de `GET` ou pour les méthodes `POST` utilisées avec certains types MIME), la spécification indique que les navigateurs doivent effectuer une requête préliminaire (« *preflight request* ») et demander au serveur les méthodes prises en charges via une requête utilisant la méthode `OPTIONS` puis, après approbation du serveur, envoyer la vraie requête. Les serveurs peuvent également indiquer aux clients s'il est nécessaire de fournir des informations d'authentification (que ce soit des cookies ou des données d'authentification HTTP) avec les requêtes.

Les sections qui suivent évoquent les différents scénarios relatifs au CORS ainsi qu'un aperçu des en-têtes HTTP utilisés.

Exemples de scénarios pour le contrôle d'accès

Voyons ici trois scénarios qui illustrent le fonctionnement du CORS. Tous ces exemples utilisent l'objet `XMLHttpRequest` qui peut être utilisé afin de faire des requêtes entre différents sites (dans les navigateurs qui prennent en charge cette fonctionnalité).

Les fragments de code JavaScript (ainsi que les instances serveurs qui gèrent ces requêtes) se trouvent sur http://arunranga.com/examples/access-control/ et fonctionnent pour les navigateurs qui prennent en charge `XMLHttpRequest` dans un contexte multi-site.

Un aperçu « côté serveur » des fonctionnalités CORS se trouve dans l'article Contrôle d'accès côté serveur (en-US).

Requêtes simples

Certaines requêtes ne nécessitent pas de requête CORS préliminaire. Dans le reste de cet article, ce sont ce que nous appellerons des requêtes « simples » (bien que la spécification Fetch (qui définit le CORS) n'utilise pas ce terme). Une requête simple est une requête qui respecte les conditions suivantes :

- Les seules méthodes autorisées sont :
  - `GET`
  - `HEAD`
  - `POST`
- En dehors des en-têtes paramétrés automatiquement par l'agent utilisateur (tels que `Connection`, User-Agent (en-US) ou tout autre en-tête dont le nom fait partie de la spécification Fetch comme « nom d'en-tête interdit »), les seuls en-têtes qui peuvent être paramétrés manuellement sont, selon la spécification :
  - `Accept`
  - `Accept-Language`

  - `Content-Language`
  - `Content-Type` (cf. les contraintes supplémentaires ci-après)
- Les seules valeurs autorisées pour l'en-tête `Content-Type` sont :
  - `application/x-www-form-urlencoded`
  - `multipart/form-data`
  - `text/plain`
- Aucun gestionnaire d'évènement n'est enregistré sur aucun des objets `XMLHttpRequestUpload` utilisés pour la requête, on y accède via la propriété `XMLHttpRequest.upload` (en-US).
- Aucun objet `ReadableStream` (en-US) n'est utilisé dans la requête.

> **Note :** Cela correspond aux classes de requêtes généralement produites par du contenu web. Aucune donnée de réponse n'est envoyée au client qui a lancé la requête sauf si le serveur envoie un en-tête approprié. Aussi, les sites qui empêchent les requêtes étrangères falsifiées ne craignent rien de nouveau.

> **Note :** WebKit Nightly et Safari Technology Preview ajoutent des restrictions supplémentaires pour les valeurs autorisées des en-têtes `Accept`, `Accept-Language` et `Content-Language`. Si l'un de ces en-têtes a une valeur non-standard, WebKit/Safari considère que la requête ne correspond pas à une requête simple. Les valeurs considérées comme non-standard par WebKit/Safari ne sont pas documentées en dehors de ces bugs WebKit :
> *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*
> et *Switch to a blacklist model for restricted Accept headers in simple CORS requests* .
> Aucun autre navigateur n'implémente ces restrictions supplémentaires, car elles ne font pas partie de la spécification.

Si, par exemple, on a un contenu web situé sous le domaine `http://toto.example` qui souhaite invoquer du contenu situé sous le domaine `http://truc.autre`, on pourrait utiliser du code JavaScript semblable à ce qui suit sur `toto.example` :

```
var invocation = new XMLHttpRequest();
var url = 'http://truc.autre/resources/public-data/';

function callOtherDomain() {

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

Cela entraînera un échange simple entre le client et le serveur laissant aux en-têtes CORS le soin de gérer les privilèges d'accès :

Voyons dans le détail ce que le navigateur envoie au serveur et quelle sera sa réponse :

```
GET /resources/public-data/ HTTP/1.1
Host: truc.autre
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.
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://toto.example/exemples/access-control/simpleXSInvocation.
Origin: http://toto.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]
```

Les lignes 1 à 10 correspondent aux en-têtes envoyés. L'en-tête qui nous intéresse particulièrement ici est `Origin`, situé à la ligne 10 : on y voit que l'invocation provient du domaine `http://toto.example`.

Les lignes 13 à 22 détaillent la réponse HTTP du serveur situé sous le domaine `http://truc.autre`. Dans la réponse, le serveur renvoie un en-tête `Access-Control-Allow-Origin` (visible à la ligne 16). On voit ici les en-têtes `Origin` et `Access-Control-Allow-Origin` pour un contrôle d'accès dans sa forme la plus simple. Ici, le serveur répond avec `Access-Control-Allow-Origin: *` ce qui signifie que la ressource peut être demandée par n'importe quel domaine. Si les propriétés de la ressource située sous `http://truc.autre` souhaitaient restreindre l'accès à la ressource à l'origine `http://toto.example`, ils auraient renvoyé :

```
Access-Control-Allow-Origin: http://toto.example
```

On notera que, dans ce cas, aucun autre domaine que `http://toto.example` (tel qu'identifié par l'en-tête `Origin`) ne pourra accéder à la ressource. L'en-tête `Access-Control-Allow-Origin` devrait contenir la valeur qui a été envoyée dans l'en-tête `Origin` de la requête.

Requêtes nécessitant une requête préliminaire

À la différence des requêtes simples, les requêtes préliminaires envoient d'abord une requête HTTP avec la méthode `OPTIONS` vers la ressource de l'autre domaine afin de déterminer quelle requête peut être envoyée de façon sécurisée. Les requêtes entre différents sites peuvent notamment utiliser ce mécanisme de vérification préliminaire lorsque des données utilisateurs sont impliquées.

impliquees.

Une requête devra être précédée d'une requête préliminaire si **une** des conditions suivantes est respectée :

- La requête utilise une des méthodes suivantes :
  - `PUT`
  - `DELETE`
  - `CONNECT`
  - `OPTIONS`
  - `TRACE`
  - `PATCH`
- **Ou si**, en dehors des en-têtes automatiquement paramétrés par l'agent utilisateur (comme `Connection`, User-Agent (en-US) ou tout autre en-tête dont le nom est réservé dans la spécification), la requête inclut tout autre en-tête que ceux définis sur la liste blanche :
  - `Accept`
  - `Accept-Language`
  - `Content-Language`
  - `Content-Type` (cf. les contraintes supplémentaires ci-après)
  - `Last-Event-ID`
  - `DPR`
  - `Save-Data`
  - `Viewport-Width`
  - `Width`
- **Ou si** l'en-tête `Content-Type` possède une valeur autre que :
  - `application/x-www-form-urlencoded`
  - `multipart/form-data`
  - `text/plain`
- **Ou si** un ou plusieurs gestionnaires d'évènements sont enregistrés sur l'objet

`XMLHttpRequestUpload` utilisé dans la requête.

- **Ou si** un objet `ReadableStream` (en-US) est utilisé dans la requête.

> **Note :** WebKit Nightly et Safari Technology Preview ajoutent des restrictions supplémentaires pour les valeurs autorisées des en-têtes `Accept`, `Accept-Language` et `Content-Language`. Si l'un de ces en-têtes a une valeur non-standard, WebKit/Safari
>
> considère que la requête ne correspond pas à une requête simple. Les valeurs considérées comme non-standard par WebKit/Safari ne sont pas documentées en dehors de ces bugs WebKit :
> *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*
> et *Switch to a blacklist model for restricted Accept headers in simple CORS requests* .
> Aucun autre navigateur n'implémente ces restrictions supplémentaires, car elles ne font pas partie de la spécification.

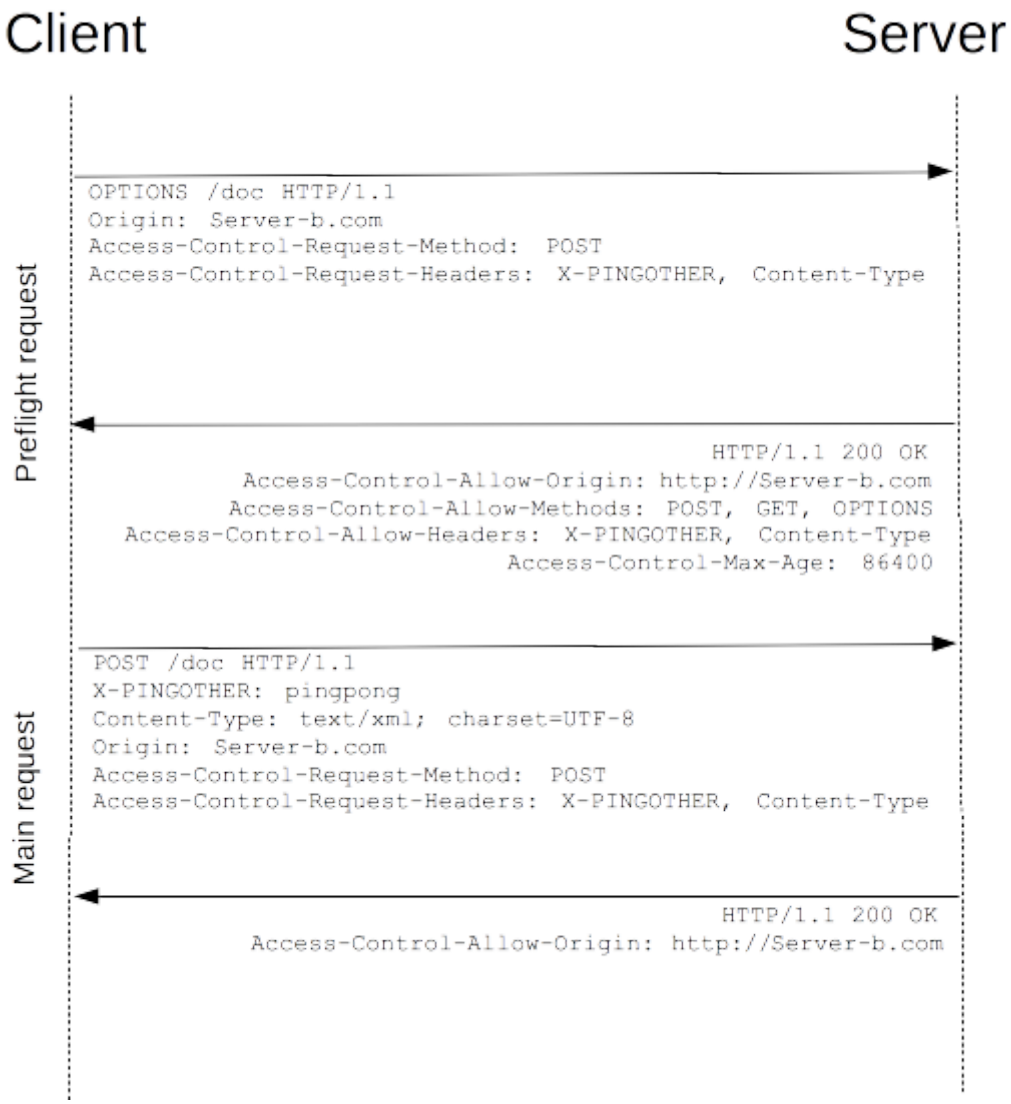
Voici un exemple d'une requête qui devra être précédée d'une requête préliminaire :

```
var invocation = new XMLHttpRequest();
var url = 'http://truc.autre/resources/post-here/';
var body = '<?xml version="1.0"?><personne><nom>Toto</nom></personne>';

function callOtherDomain(){
  if(invocation)
    {
      invocation.open('POST', url, true);
      invocation.setRequestHeader('X-PINGOTHER', 'pingpong');
      invocation.setRequestHeader('Content-Type', 'application/xml');
      invocation.onreadystatechange = handler;
      invocation.send(body);
    }
}

......
```

Dans le fragment de code ci-avant, à la ligne 3, on crée un corps XML envoyé avec la requête `POST` ligne 8. Sur la ligne 9, on voit également un en-tête de requête HTTP non standard : `X-PINGOTHER: pingpong`. De tels en-têtes ne sont pas décrits par le protocole HTTP/1.1 mais peuvent être utilisés par les applications web. La requête utilisant un en-tête `Content-Type` qui vaut `application/xml` et un en-tête spécifique, il est nécessaire d'envoyer au préalable une requête préliminaire.



> **Note :** Comme décrit après, la vraie requête POST n'inclut pas les en-têtes `Access-Control-Request-*` qui sont uniquement nécessaires pour la requête OPTIONS.

Voyons ce qui se passe entre le client et le serveur. Le premier échange est la requête/réponse préliminaire :

```
OPTIONS /resources/post-here/ HTTP/1.1
Host: truc.autre
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.
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://toto.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-Type


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://toto.example
Access-Control-Allow-Methods: POST, GET
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 0
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/plain
```

Une fois que la requête préliminaire est effectuée, la requête principale est envoyée :

```
POST /resources/post-here/ HTTP/1.1
Host: truc.autre
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.
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://toto.example/exemples/preflightInvocation.html
Content-Length: 55
Origin: http://toto.example
Pragma: no-cache
```

```
Cache-Control: no-cache

<?xml version="1.0"?><personne><nom>Toto</nom></personne>


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://toto.example
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 235
Keep-Alive: timeout=2, max=99
Connection: Keep-Alive
Content-Type: text/plain

[Une charge utile GZIPée]
```

Entre les lignes 1 à 12 qui précèdent, on voit la requête préliminaire avec la méthode `OPTIONS`. Le navigateur détermine qu'il est nécessaire d'envoyer cela à cause des paramètres de la requête fournie par le code JavaScript. De cette façon le serveur peut répondre si la requête principale est acceptable et avec quels paramètres. OPTIONS est une méthode HTTP/1.1 qui est utilisée afin de déterminer de plus amples informations à propos du serveur. La méthode OPTIONS est une méthode « sûre » (*safe*) et ne change aucune ressource. On notera, qu'avec la requête OPTIONS, deux autres en-têtes sont envoyés (cf. lignes 10 et 11) :

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

L'en-tête Access-Control-Request-Method (en-US) indique au serveur, pendant la requête préliminaire, que la requête principale sera envoyée avec la méthode `POST`. L'en-tête `Access-Control-Request-Headers` indique au serveur que la requête principale sera envoyée avec un en-tête `X-PINGOTHER` et un en-tête `Content-Type` spécifique. Le serveur peut alors déterminer s'il souhaite accepter une telle requête.

Dans les lignes 14 à 26 qui suivent, on voit la réponse renvoyée par le serveur qui indique que la méthode de la requête (`POST`) ainsi que ses en-têtes (`X-PINGOTHER`) sont acceptables. Voici ce qu'on peut notamment lire entre les lignes 17 et 20 :

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

Le serveur répond avec un en-tête `Access-Control-Allow-Methods` et indique que les méthodes `POST` et `GET` sont acceptables pour manipuler la ressource visée. On notera que cet

en-tête est semblable à l'en-tête de réponse `Allow`, toutefois, `Access-Control-Allow-Methods` est uniquement utilisé dans le cadre du contrôle d'accès.

Le serveur envoie également l'en-tête `Access-Control-Allow-Headers` avec une valeur "`X-PINGOTHER, Content-Type`" qui confirme que les en-têtes souhaités sont autorisés pour la requête principale. Comme `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers` est une liste d'en-têtes acceptables séparés par des virgules.

Enfin, l'en-tête Access-Control-Max-Age (en-US) indique avec une valeur exprimée en secondes, la durée pendant laquelle cette réponse préliminaire peut être mise en cache avant la prochaine requête préliminaire. Ici, la réponse est 86400 secondes, ce qui correspond à 24 heures. On notera ici que chaque navigateur possède un maximum interne (en-US) qui a la priorité lorsque `Access-Control-Max-Age` lui est supérieur.

Requêtes préliminaires et redirection

À l'heure actuelle, la plupart des navigateurs ne prennent pas en charge les redirections pour les requêtes préliminaires. Si une redirection se produit pour une requête préliminaire, la plupart des navigateurs émettront un message d'erreur semblables à ceux-ci.

La requête a été redirigée vers 'https://example.com/toto', ce qui n'est pas autorisé pour les requêtes multi-origines qui doivent être précédées d'une requête préliminaire. (*The request was redirected to 'https://example.com/toto', which is disallowed for cross-origin requests that require preflight.*)

Il est nécessaire d'effectuer une requête préliminaire pour cette requête, or, ceci n'est pas

autorisé pour suivre les redirections multi-origines. (*Request requires preflight, which is disallowed to follow cross-origin redirect.*)

Le protocole CORS demandait initialement ce comportement. Toutefois, il a été modifié et ces erreurs ne sont plus nécessaires . Toutefois, la plupart des navigateurs n'ont pas encore implémenté cette modification et conservent alors le comportement conçu initialement.

En attendant que les navigateurs comblent ce manque, il est possible de contourner cette limitation en utilisant l'une de ces deux méthodes :

- Modifier le comportement côté serveur afin d'éviter la requête préliminaire ou la redirection (dans le cas où vous contrôler le serveur sur lequel la requête est effectuée)
- Modifier la requête afin que ce soit une requête simple qui ne nécessite pas de requête préliminaire.

S'il n'est pas possible d'appliquer ces changements, on peut également :

1. Effectuer une requête simple (avec `Response.url (en-US)` si on utilise l'API Fetch ou `XHR.responseURL (en-US)` si on utilise XHR) afin de déterminer l'URL à laquelle aboutirait la requête avec requête préliminaire.
2. Effectuer la requête initialement souhaitée avec l'URL *réelle* obtenue à la première étape.

Toutefois, si la requête déclenche une requête préliminaire suite à l'absence de l'en-tête `Authorization`, on ne pourra pas utiliser cette méthode de contournement et il sera nécessaire d'avoir accès au serveur pour contourner le problème.

Requêtes avec informations d'authentification

Une des fonctionnalités intéressante mise en avant par le CORS (via `XMLHttpRequest` ou Fetch) est la possibilité d'effectuer des requêtes authentifiées reconnaissant les cookies HTTP et les informations d'authentification HTTP. Par défaut, lorsqu'on réalise des appels `XMLHttpRequest` ou Fetch entre différents sites, les navigateurs n'enverront pas les informations d'authentification. Pour cela, il est nécessaire d'utiliser une option spécifique avec le constructeur `XMLHttpRequest` ou `Request` lorsqu'on l'appelle

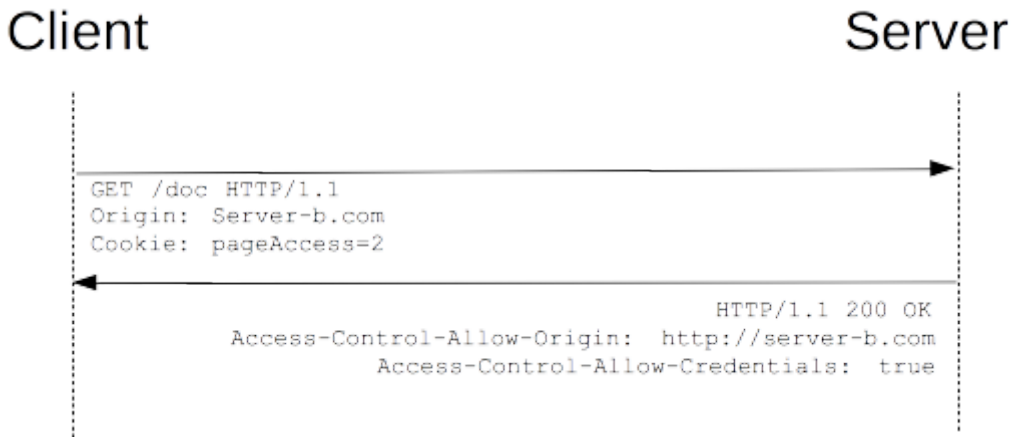
constructeur `XMLHttpRequest` ou `Request` lorsqu'on l'appelle.

Dans cet exemple, le contenu chargé depuis `http://toto.example` effectue une requête GET simple vers une ressource située sous `http://truc.autre` qui définit des *cookies*. Voici un exemple de code JavaScript qui pourrait se trouver sur `toto.example` :

```
var invocation = new XMLHttpRequest();
var url = 'http://truc.autre/resources/credentialed-content/';

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

À la ligne 7, on voit que l'option `withCredentials`, du constructeur `XMLHttpRequest`, doit être activée pour que l'appel utilise les *cookies*. Par défaut, l'appel sera réalisé sans les *cookies*. Cette requête étant une simple requête `GET`, il n'est pas nécessaire d'avoir une requête préliminaire. Cependant, le navigateur rejettera tout réponse qui ne possède pas l'en-tête Access-Control-Allow-Credentials (en-US) : `true` et la réponse correspondante ne sera pas disponible pour le contenu web qui l'a demandée.



Voici un exemple d'échange entre le client et le serveur :

```
GET /resources/access-control-with-credentials/ HTTP/1.1
Host: truc.autre
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.
```

```
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://toto.example/exemples/credential.html
Origin: http://toto.example
Cookie: pageAccess=2


HTTP/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
X-Powered-By: PHP/5.2.6
Access-Control-Allow-Origin: http://toto.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]
```

Bien que la ligne 11 contienne le *cookie* pour le contenu sous `http://truc.autre`, si `truc.autre` n'avait pas répondu avec Access-Control-Allow-Credentials (en-US) `: true` (cf. ligne 19), la réponse aurait été ignorée et n'aurait pas pu être consommée par le contenu web.

Requêtes authentifiées et jokers (*wildcards*)

Lorsqu'il répond à une requête authentifiée, le serveur **doit** indiquer une origine via la valeur de l'en-tête `Access-Control-Allow-Origin`, il ne doit pas utiliser le joker " `*` ".

Avec la requête précédente, on voit la présence d'un en-tête `Cookie` mais la requête échouerait si la valeur de l'en-tête de réponse `Access-Control-Allow-Origin` était " `*` ". Ici, ce n'est pas le cas : `Access-Control-Allow-Origin` vaut " `http://toto.example` " et le contenu récupéré par la requête est alors envoyé au contenu web.

Dans cet exemple, on notera également que l'en-tête de réponse `Set-Cookie` définit un autre *cookie*. En cas d'échec, une exception (dépendant de l'API utilisée) sera levée.

Cookies tiers

On notera que les *cookies* provenant de réponses CORS sont également sujets aux règles qui s'appliquent aux *cookies* tiers. Dans l'exemple précédent, la page est chargée depuis `toto.example` et, à la ligne 22, le *cookie* est envoyé par `truc.autre`. Aussi, ce *cookie* n'aurait pas été enregistré si l'utilisateur avait paramétré son navigateur pour rejeter les *cookies* tiers.

En-têtes de réponse HTTP

Dans cette section, on liste les en-têtes de réponse HTTP qui sont renvoyés par le serveur pour le contrôle d'accès, tels que définis par la spécification *Cross-Origin Resource Sharing*. La section précédente illustre, avec des exemples concrets, leur fonctionnement.

`Access-Control-Allow-Origin`

Une ressource de réponse peut avoir un en-tête `Access-Control-Allow-Origin` avec la syntaxe suivante :

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

Le paramètre `origin` définit un URI qui peut accéder à la ressource. Le navigateur doit respecter cette contrainte. Pour les requêtes qui n'impliquent pas d'informations d'authentification, le serveur pourra indiquer un joker (" `*` ") qui permet à n'importe quelle origine d'accéder à la ressource.

Si on souhaite, par exemple, autoriser `http://mozilla.org` à accéder à la ressource, on pourra répondre avec :

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

Si le serveur indique une origine spécifique plutôt que " `*` ", il pourra également inclure la valeur `Origin` dans l'en-tête de réponse `Vary` pour indiquer au client que la réponse du serveur variera selon la valeur de l'en-tête de requête `Origin`.

Access-Control-Expose-Headers

L'en-tête Access-Control-Expose-Headers (en-US) fournit une liste blanche des en-têtes auxquels les navigateurs peuvent accéder. Ainsi :

```
Access-Control-Expose-Headers: X-Mon-En-tete-Specifique, X-Un-Autre-
```

Cela permettra que les en-têtes `X-Mon-En-tete-Specifique` et `X-Un-Autre-En-tete` soient utilisés par le navigateur.

Access-Control-Max-Age

L'en-tête Access-Control-Max-Age (en-US) indique la durée pendant laquelle le résultat de la requête préliminaire peut être mis en cache (voir les exemples ci-avant pour des requêtes impliquant des requêtes préliminaires).

```
Access-Control-Max-Age: <delta-en-secondes>
```

Le paramètre `delta-en-seconds` indique le nombre de secondes pendant lesquelles les résultats peuvent être mis en cache.

Access-Control-Allow-Credentials

L'en-tête Access-Control-Allow-Credentials (en-US) indique si la réponse à la requête doit être exposée lorsque l'option `credentials` vaut `true`. Lorsque cet en-tête est utilisé dans une réponse préliminaire, cela indique si la requête principale peut ou non être effectuée avec des informations d'authentification. On notera que les requêtes `GET` sont des requêtes simples et si une requête est effectuée, avec des informations d'authentification pour une ressource, et que cet en-tête n'est pas renvoyé, la réponse sera ignorée par le navigateur et sa charge ne pourra pas être consommée par le contenu web.

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

Voir les scénarios ci-avant pour des exemples.

Access-Control-Allow-Methods

L'en-tête `Access-Control-Allow-Methods` indique la ou les méthodes qui sont autorisées pour accéder à la ressoure. Cet en-tête est utilisé dans la réponse à la requête préliminaire (voir ci-avant les conditions dans lesquelles une requête préliminaire est nécessaire).

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

Voir un exemple ci-avant pour l'utilisation de cet en-tête.

`Access-Control-Allow-Headers`

L'en-tête Access-Control-Allow-Headers (en-US) est utilisé dans une réponse à une requête préliminaire afin d'indiquer les en-têtes HTTP qui peuvent être utilisés lorsque la requête principale est envoyée.

```
Access-Control-Allow-Headers: <nom-champ>[, <nom-champ>]*
```

En-têtes de requête HTTP

Dans cette section, nous allons décrire les en-têtes que les clients peuvent utiliser lors de l'envoi de requêtes HTTP afin d'exploiter les fonctionnalités du CORS. Ces en-têtes sont souvent automatiquement renseignés lors d'appels aux serveurs. Les développeurs qui utilisent `XMLHttpRequest` pour les requêtes multi-origines n'ont pas besoin de paramétrer ces en-têtes dans le code JavaScript.

`Origin`

L'en-tête `Origin` indique l'origine de la requête (principale ou préliminaire) pour l'accès multi-origine.

```
Origin: <origine>
```

L'origine est un URI qui indique le serveur à partir duquel la requête a été initiée. Elle n'inclut aucune information relative au chemin mais contient uniquement le nom du serveur.

> **Note :** `origine` peut être une chaîne vide (ce qui s'avère notamment utile lorsque la source est une URL de donnée).

Pour chaque requête avec contrôle d'accès, l'en-tête `Origin` sera **toujours** envoyé.

`Access-Control-Request-Method`

L'en-tête Access-Control-Request-Method (en-US) est utilisé lorsqu'on émet une requête préliminaire afin de savoir quelle méthode HTTP pourra être utilisée avec la requête principale.

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

Voir ci-avant pour des exemples d'utilisation de cet en-tête.

`Access-Control-Request-Headers`

L'en-tête `Access-Control-Request-Headers` est utilisé lorsqu'on émet une requête préliminaire afin de communiquer au serveur les en-têtes HTTP qui seront utilisés avec la requête principale.

```
Access-Control-Request-Headers: <nom-champ>[, <nom-champ>]*
```

Voir ci-avant pour des exemples d'utilisation de cet en-tête.

Spécifications

Spécification	État	Commentaires
Fetch La définition de 'CORS' dans cette spécification.	Standard évolutif	Nouvelle définition, remplace la spécification W3C pour le CORS .

Compatibilité des navigateurs

Report problems with this compatibility data on GitHub

`Access-Control-Allow-Origin`	
Chrome	4
Edge	12

Firefox	3.5
Internet Explorer	10
Opera	12
Safari	4
WebView Android	2
Chrome Android	Yes
Firefox for Android	4
Opera Android	12
Safari on iOS	3.2
Samsung Internet	Yes

 Full support

Notes de compatibilité

- Internet Explorer 8 et 9 exposent les fonctionnalités relatives au CORS via l'objet `XDomainRequest`. L'implémentation complète est disponible à partir d'IE 10.
- Bien que Firefox 3.5 ait introduit la prise en charge des requêtes `XMLHttpRequest` entre différents sites et des polices web, certaines requêtes étaient limitées jusqu'à des versions ultérieures. Plus précisément, Firefox 7 permet les requêtes multi-origines pour les textures WebGL et Firefox 9 permet la récupération d'images dessinées sur un canevas via `drawImage`.

Voir aussi

- Exemples de codes utilisant `XMLHttpRequest` et le CORS (en anglais)
- Exemples de code côté client et côté serveur utilisant le CORS (en anglais)
- Le CORS vu côté serveur (PHP, etc.) (en-US)
- `XMLHttpRequest`

- L'API Fetch
- Utiliser le CORS - HTML5 Rocks (en anglais)

  Une réponse Stack Overflow pour répondre aux problèmes fréquemment posés par le
- CORS (en anglais)

  :
  - Comment éviter les requêtes préliminaires
  - Comment utiliser un proxy CORS pour contourner *No Access-Control-Allow-Origin header*
  - Comment corriger *Access-Control-Allow-Origin header must not be the wildcard*

Last modified: 28 août 2021, by MDN contributors