CORS

Cross-Origin Resource Sharing

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

¿Qué es CORS?

El *Intercambio de Recursos de Origen Cruzado* (Cross-origin resource sharing - CORS) es una especificación de W3C que define como el navegador y el servidor deben comunicarse cuando se accede a recursos de orígenes cruzados.

W3C Recommendation: https://www.w3.org/TR/cors/

CORS hace uso de intercambio de **encabezados HTTP adicionales**, para que tanto el navegador como el servidor tengan el conocimiento suficiente para determinar si un requerimiento o respuesta debería ser exitoso o no.

No debería utilizarse CORS como la única manera de securizar recursos, más bien se utiliza para dar directivas a los navegadores sobre el acceso a recursos de orígenes cruzados.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

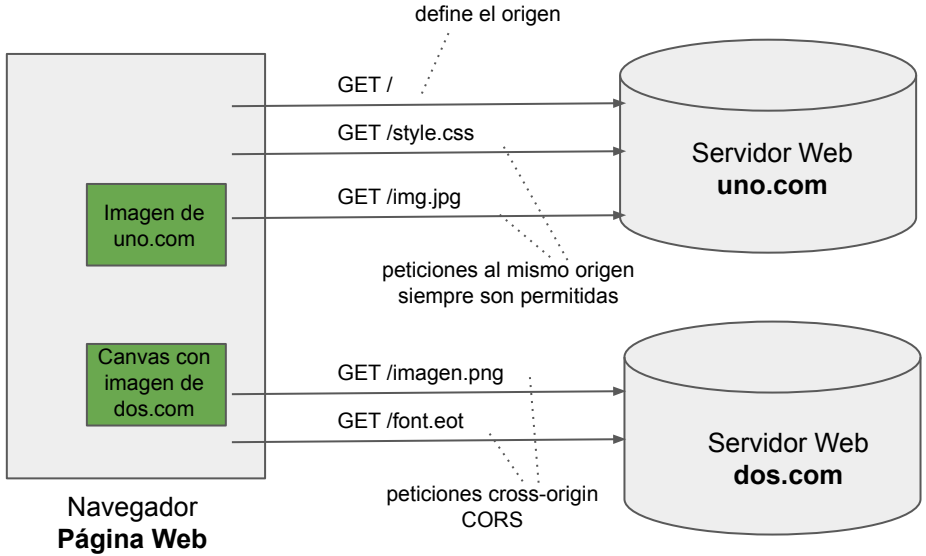
Soporte CORS

CORS es soportado por los siguientes navegadores:

- Chrome 3+
- Firefox 3.5+
- Opera 12+
- Safari 4+
- Internet Explorer 8+

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

Peticiones Cross-origin



Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

Control de acceso en CORS

CORS diferencia los intercambios de orígenes cruzados en dos tipos:

1. Simple requests
2. *Pre-flight* requests

1. Simple requests son las peticiones:
    1. HEAD
    2. GET
    3. POST con Content-Type de tipo:
        - application/x-www-form-urlencoded
        - multipart/form-data
        - text/plain

2. Pre-flight requests: son el resto de las peticiones, por ejemplo PUT DELETE o POST con Content-Type distinto a los mencionados

Los encabezados tienen que ser simples tambíen:

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

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

Simple requests en CORS

- La petición HTTP incluye un encabezado llamado **Origin** indicando el origen del código del cliente.
- El servidor evalúa en base al origen especificado si da curso o no a la petición.
- En caso positivo, el servidor responde con el recurso solicitado y un encabezado **Access-Control-Allow-Origin** indicando el origen permitido. Opcionalmente puede especificar "*" para indicar que todos los orígenes son permitidos.
- Si el encabezado Access-Control-Allow-Origin no se encuentra en la respuesta o si no concuerda con el origen de la petición el navegador deniega la petición.

Nota: la presencia del encabezado Origin no significa que la petición es de tipo cross-origin, por ejemplo Chrome y Safari incluyen el encabezado Origin en las peticiones POST/PUT/DELETE del mismo origen. Por otro lado TODAS las peticiones de tipo cross-origin contienen este encabezado.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

Simple requests en CORS

Ejemplo de un simple request en CORS

Petición desde navegador Web

```
GET /greeting/ HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_5)
AppleWebKit/536.30.1 (KHTML, like Gecko) Version/6.0.5 Safari/536.30.1
Accept: application/json, text/plain, */*
Referer: http://foo.client.com/
Origin: http://foo.client.com
```

Respuesta del servidor Web

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Wed, 20 Nov 2013 19:36:00 GMT
Server: Apache-Coyote/1.1
Content-Length: 35
Connection: keep-alive
Access-Control-Allow-Origin: http://foo.client.com

[response payload]
```

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

Pre-flight requests en CORS

Para el caso de utilizar los verbos PUT DELETE o por ejemplo POST con Content-Type: application/json, CORS envía previamente otra petición llamada *“pre-flight”* para asegurarse que la petición original es segura de enviarse.

La respuesta *pre-flight* puede ser cacheada para no tener que realizarla antes de cada petición del cliente.

Las peticiones pre-flight utilizan el verbo **OPTIONS** de HTTP.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

Pre-flight requests en CORS

Ejemplo de *"pre-flight"* request en CORS

Petición desde navegador Web

```
OPTIONS /cors HTTP/1.1
Origin: http://api.bob.com
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: X-Custom-Header
Host: api.alice.com
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```

La petición Pre-flight pregunta previamente al servidor si podría enviar la petición PUT y utilizar el encabezado personalizado X-Custom-Header.

Respuesta del servidor Web

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: 0
Connection: keep-alive
Access-Control-Allow-Origin: http://api.bob.com
Access-Control-Allow-Methods: GET, POST, PUT
Access-Control-Allow-Headers: X-Custom-Header
Access-Control-Max-Age: 86400
Content-Type: text/html; charset=utf-8
```

El servidor permite al origen http://api.bob.com acceder mediante los métodos GET, POST PUT y el encabezado X-Custom-Header (el servidor puede agregar una lista de todos los permitidos). La respuesta *pre-flight* es buena para 86400 segundos (1 día), luego tendrá que volver a consultar.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

Pre-flight requests en CORS

Ejemplo de *"pre-flight"* request en CORS

Petición desde navegador Web

```
PUT /cors HTTP/1.1
Origin: http://api.bob.com
Host: api.alice.com
X-Custom-Header: value
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```

El navegador envía la petición de tipo PUT del código del cliente e incluye el encabezado personalizado.

Respuesta del servidor Web

```
HTTP/1.1 204 No Content
Server: Apache-Coyote/1.1
Connection: keep-alive
Access-Control-Allow-Origin: http://api.bob.com
Content-Type: text/html; charset=utf-8
```

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

Implementación de CORS con Java

Es posible llevar a cabo una implementación de CORS en Java a través de distintos métodos:

- Escribiendo y adecuando los métodos de nuestra aplicación para que atiendan las peticiones de tipo OPTIONS de CORS

```
@OPTIONS
@Path("{path : .*}")
public Response options() {
    return Response.ok("")
            .header("Access-Control-Allow-Origin", "*")
            .header("Access-Control-Allow-Headers", "origin, content-type, accept, authorization")
            .header("Access-Control-Allow-Credentials", "true")
            .header("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE, OPTIONS, HEAD")
            .header("Access-Control-Max-Age", "1209600")
            .build();
}
```

- Escribiendo un filtro en Java que intercepte las peticiones y conteste agregando los encabezados propios de CORS
- Escribiendo una solución que utilice las facilidades que provea el framework con el que desarrollamos nuestro REST (x ej, un filtro de Spring MVC)
- Utilizar un filtro desarrollado por terceros que implemente CORS

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

Filtro de Tomcat para CORS

CORS Filter de Tomcat: https://tomcat.apache.org/tomcat-8.0-doc/config/filter.html#CORS_Filter
Es posible utilizarlo agregando en el archivo web.xml de nuestro proyecto la configuración del filtro, por ejemplo:

```
<filter>
        <filter-name>CorsFilter</filter-name>
        <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
        <init-param>
                <param-name>cors.allowed.origins</param-name>
                <param-value>*</param-value>
        </init-param>
        <init-param>
                <param-name>cors.allowed.methods</param-name>
                <param-value>GET,POST,HEAD,OPTIONS,PUT</param-value>
        </init-param>
        <init-param>
                <param-name>cors.allowed.headers</param-name>
        <param-value>Content-Type,X-Requested-With,accept,Origin,Access-Control-Request-Method,Access-Control-Request-Headers,Authorization</param-value>
        </init-param>
        <init-param>
                <param-name>cors.exposed.headers</param-name>
                <param-value>Access-Control-Allow-Origin,Access-Control-Allow-Credentials,Authorization</param-value>
        </init-param>
        <init-param>
                <param-name>cors.support.credentials</param-name>
                <param-value>true</param-value>
        </init-param>
        <init-param>
                <param-name>cors.preflight.maxage</param-name>
                <param-value>10</param-value>
        </init-param>
</filter>
<filter-mapping>
        <filter-name>CorsFilter</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
<filter>
```

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional

CORS con Spring

Ejemplo @CrossOrigin en la clase y en un método

```
@CrossOrigin(maxAge = 3600)
@RestController
@RequestMapping("/account")
public class AccountController {

    @CrossOrigin("http://example.com")
    @RequestMapping(method = RequestMethod.GET, "/{id}")
    public Account retrieve(@PathVariable Long id) {
        // ...
    }

    @RequestMapping(method = RequestMethod.DELETE, path = "/{id}")
    public void remove(@PathVariable Long id) {
        // ...
    }
}
```

Si utilizamos `@CrossOrigin` sin parámetros, por defecto tiene:

- Todos los orígenes son permitidos
- Los métodos HTTP permitidos son los especificados en *@RequestMapping*
- El tiempo de respuesta cacheada (*maxAge*) es de 30 minutos.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional