# CROSS-ORIGIN RESOURCE SHARING (CORS)

## Introdução

//TO DO

## Definição

O Cross-Origin Resource Sharing é a especificação que define um modelo, robusto e padronizado, de para compartilhamento de recursos entre origens dissemelhantes. Este modelo de compartilhamento compreende o consumo dos recursos compartilhados através de chamadas assíncronas e também provê mecanismos de segurança que permitem a manutenção da segurança propiciada pelo Same-Origin Policy [W3C - CORS, 2011].

O CORS consiste na adição de alguns cabeçalhos na conversação HTTP entre cliente e servidor, além da alteração do objeto XMLHttpRequest disponibilizado pelos navegadores web.

## Requisitos

O mecanismo Cross-Origin Resource Sharing teve a sua modelagem influenciada por premissas e requisitos dos mais diversos aspectos arquiteturais. Abaixo são enumerados os requisitos desta especificação:

1. Não deve introduzir possibilidades de ataques para servidores que são protegidos apenas por um firewall;
2. Não deve ser possível realizar operações criativas ou destrutivas, ou seja, qualquer operação diferente de GET, HEAD e OPTIONS, sem antes realizar uma verificação de autorização;
3. Deve tentar prevenir, na medida do possível, ataques de força bruta distribuídos com a finalidade de obtenção de contas de usuário para servidores externos;
4. Não deve permitir o carregamento e exposição de conteúdos de servidores terceiros sem explícito consentimento destes servidores, uma vez que estes conteúdos podem conter informações sensíveis;
5. Não deve ser necessário que administradores de sistemas façam alterações para manter o atual nível de segurança;
6. Deve ser possível configurar políticas de autorização distintas entre domínios distintos para diferentes recursos hospedados em uma mesma origem;
7. Deve ser possível disponibilizar conteúdos de todos os tipos. Desta forma, deve ser possível transmitir conteúdos de qualquer “Content-Type” se o servidor assim permitir;
8. Deve ser possível definir servidores específicos ou um conjunto de servidores para fazer o consumo de determinado recurso;
9. Não deve ser necessário que o servidor filtre o corpo das requisições para poder negar ou permitir o acesso das origens solicitantes;
10. Não deve ser necessário alterar a interface do objeto XMLHttpRequest para ser possível realizar requisições cross-domain;
11. Deve ser possível realizar requisições com os mais diversos verbos HTTP;
12. Deve ser compatível com os mecanismos de autenticação HTTP e gerenciamento de sessão mais usados;
13. Deve ser claro para o provedor de recursos quando o acesso é concedido e quando é negado;

## O objeto XMLHttpRequest nível 2

XMLHttpRequest nível 2 é o resultado da adequação do objeto XMLHttpRequest as modificações estipuladas pela especificação CORS e da inclusão de respostas progressivas.

Com relação a conformidade do XMLHttpRequest com a especificação CORS, foram adicionados mecanismos para trabalhar novos cabeçalhos HTTP. Com isso, requisições HTTP cross-domain realizadas pelo componente XMLHttpRequest passam a enviar o cabeçalho Origin. Este cabeçalho, por questões de segurança, é protegido pelos navegadores de forma que não pode ser alterado por scripts ou códigos de aplicação. Seu conteúdo é similar ao do cabeçalho Referer, porém, não contém uma URL completa, e sim, somente a URL base da aplicação cliente. Como o Referer pode conter dados sensíveis, muitas vezes é omitido pelo navegador, o que tornou a criação do cabeçalho Origin necessária.

A outra modificação importante no XMLHttpRequest foi adição da funcionalidade de respostas progressivas. Nas versões anteriores, havia apenas um evento “readystatechange”, implementado com muita inconsistência pelos diversos navegadores web. Nesta sua nova versão, o XMLHttpRequest possui seis eventos para a monitoração do progresso de eventos, são eles: “loadstart”, “progress”, “abort”, “error”, “load” e “loadend”.

## Como funciona?

Os navegadores modernos, como o Firefox 7, Chrome 14 e Safari 5, utilizam o objeto XMLHttpRequest como um container da API que implementa as funcionalidades de um “cliente HTTP”. O Internet Explorer, nas versões 8 e 9, implementa parte desta API através do objeto XDomainRequest, que é similar ao XMLHttpRequest, e também funciona como um container. Vale ressaltar que, uma boa alternativa ao uso nativo destes objetos é a biblioteca JavaScript JQuery 1.5.1, ou superior, que encapsula a lógica de criação dos objetos XMLHttpRequest e XDomainRequest de acordo com o navegador utilizado e fornece uma interface amigável para utilização.

Uma requisição cross-domain, que utilize o XMLHttpRequest nível 2 e os métodos HTTP “GET” ou “POST simples”, funciona da seguinte forma:

1. O cliente envia um cabeçalho chamado “Origin”, que contém a origem da requisição, ou seja, protocolo, domínio e porta da página solicitante;
2. O servidor, adequadamente preparado para receber esta requisição, aceita ou não a origem informada, e em caso positivo, responde com o cabeçalho “Access-Control-Allow-Origin”, que indica os domínios com permissão de acesso ao recurso, ou o valor "\*" (asterisco) se for um recurso público.

O padrão CORS trabalha adicionando novos cabeçalhos HTTP, nas requisições do cliente e nas respostas do servidor, que permitem aos servidores efetuarem um processo de “Handshake” , como podemos observar na **Error! Reference source not found.**



Figura 1: Fluxo de requisições GET e POST simples.

A especificação CORS obriga que requisições que não utilizem os verbos GET ou POST, ou mesmo requisições que utilizem POST para enviar dados com tipo de conteúdo diferente de “application/x-www-form-urlencoded”, “multipart/form-data”, ou “text/plain”, ou mesmo que possuam algum cabeçalho customizado façam um “pré-voo”, que em inglês é conhecido como “preflight”. Esta determinação serve para mitigar possíveis ataques contra os recursos disponibilizados, evitando assim que dados sensíveis possam ser alterados facilmente.

O pré-voo é uma requisição, com o verbo HTTP OPTIONS e com o cabeçalho Origin informando a origem solicitante, que acontece antes da requisição original. Com isto, o navegador descobre se o recurso solicitado está preparado para receber requisições de outras origens e se a origem solicitante tem acesso ao recurso desejado. Após este processo, caso a validação seja bem sucedida, o navegador dispara a requisição original.

O mecanismo de pré-voo é transparente para o desenvolvedor web pois ele acontece automaticamente, sendo o navegador responsável por este trabalho.

Para ilustrar este mecanismo, a Listagem 1: Chamada AJAX cross-domain para um Web Service apresenta um código JavaScript carregado na página web http://consumer.com.br acessando um Web Service disponibilizado em http://provider.com.br.

**01. function** callOtherDomain(){

**02. var** url = 'http://provider.com.br/clientes/adiciona';

**03. var** invocation = **new** XMLHttpRequest();

**04.**  invocation.open('POST', url, **true**);

**05.**  invocation.setRequestHeader('Content-Type', 'application/xml');

**06.**  invocation.send('nome=PauloVitor');

**07.** }

Listagem 1: Chamada AJAX cross-domain para um Web Service.

Em seguida, na Figura 2: Cabeçalhos trafegados no mecanismo de pré-voo., são apresentados os cabeçalhos HTTP trafegados durante um pré-voo. É importante ressaltar os cabeçalhos “Access-Control-Allow-Origin” e “Access-Control-Allow-Methods” na resposta do servidor. O primeiro indica quais são as origens que podem acessar o recurso e o segundo indica quais são os verbos HTTP permitidos.



Figura 2: Cabeçalhos trafegados no mecanismo de pré-voo.

Uma vez que a origem solicitante recebe a resposta do mecanismo de pré-voo, o navegador avalia se a origem está habilitada a acessar o recurso solicitado e, também, se o verbo HTTP, com o qual a requisição original deve ser realizada, encontra-se na lista de métodos permitidos. Caso esta validação seja bem sucedida, a requisição original é enviada pelo navegador, como podemos observar na Figura 3: Execução da requisição original após pré-voo.. Caso contrario, o navegador aborta o procedimento, não realizando a requisição original.

******

Figura 3: Execução da requisição original após pré-voo.

## O uso de credenciais

O termo credenciais proposto pela especificação representa os cookies, autenticação HTTP e certificados SSL do lado do cliente. Esta recurso, sem dúvida, é um dos recursos mais interessantes da especificação CORS. Por padrão, este recurso é desabilitado e o XMLHttpRequest ignora as credenciais. Para habilitar esta opção, uma flag deve ser definida no objeto XMLHttpRequest após sua inicialização. É importante destacar que o uso de credenciais ainda não é suportado pelo objeto XDomainRequest utilizado pelo Internet Explorer 8 e 9.

O trecho de código apresentado na Listagem 2: Forma de utilizar o objeto XMLHttpRequest para requisições credenciadas. mostra, na linha 5, como habilitar a flag “withCredentials” do objeto XMLHttpRequest para realizar requisições com cookie.

Listagem 2: Forma de utilizar o objeto XMLHttpRequest para requisições credenciadas.

**01. function** callOtherDomain(){

**02. var** url = 'http://provider.com.br/clientes/adiciona';

**03. var** invocation = **new** XMLHttpRequest();

**04.**  invocation.open('POST', url, **true**);

**05.**  invocation.withCredentials = **true**;

**06.**  invocation.setRequestHeader('Content-Type', 'application/xml');

**07.**  invocation.send('nome=PauloVitor');

**08.** }

Uma vez que o cliente, através da flag “withCredentials”, defina sua intenção em usar requisições credenciadas cabe ao servidor informar se determinado recurso aceita trabalhar este tipo de requisição. Mais especificamente, para que o navegador tenha acesso a estas informações, as respostas do servidor devem conter o cabeçalho “Access-Control-Allow-Credentials” com o valor “true”. Por padrão, esta opção é definida como “false”, inibindo o tráfego de cookies e inviabilizando o consumo de serviços autenticados.

****

Figura 4: Cabeçalhos trafegados em uma requisição credenciada.

A Figura 4: Cabeçalhos trafegados em uma requisição credenciada. demonstra os cabeçalhos trafegados, entre cliente e servidor, com base no código ilustrado na Listagem 2: Forma de utilizar o objeto XMLHttpRequest para requisições credenciadas..

## Impacto arquitetural

Todas as mudanças advindas com a implementação desta nova forma de compartilhamento de recursos permite cria inúmeras possibilidades arquiteturais no desenvolvimento de aplicações web.

Por exemplo, é possível se desenvolver aplicações que utilizam serviços hospedados em inúmeras origens. Pode-se hospedar uma aplicação web que utiliza conteúdo estático e outra apenas para receber as requisições AJAX, sendo que toda a comunicação passa a ser feita via XMLHttpRequest. Esta possibilidade amplia as opções de implantação de uma aplicação web.

Como podemos observar na Figura 5: Diferença entre agregação no lado servidor podemos realizar a agregação de conteúdo de diferentes fontes no lado cliente, mais especificamente no navegador do usuário. Podemos ainda, caso o servidor de destino permita, acessar conteúdos protegidos utilizando as credenciais do usuário, provendo uma experiência personalizada para este usuário. A agregação de conteúdo no lado servidor, por outro lado, afunila todo o conteúdo disponibilizado por diversos fornecedores em apenas uma infra-estrutura física, muitas vezes causando gargalos.

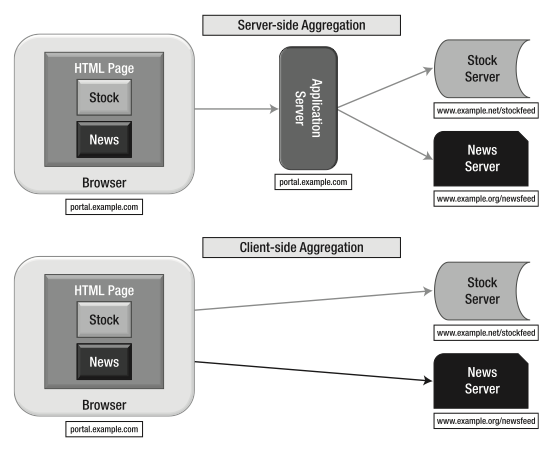


Figura 5: Diferença entre agregação no lado servidor e cliente [LUBBERS; ALBERS; SALIM, 2010].

Uma vez que passamos a agregar conteúdos, espalhados pela internet, no navegador do usuário final de uma aplicação web estamos implicitamente distribuindo grande parte do processamento desta aplicação. Essa distribuição de processamento computacional propicia altos níveis de escalabilidade.

## Suporte dos navegadores atuais

Atualmente, grande parte dos navegadores modernos possuem suporte integral a especificação CORS, como podemos aferir através da Figura 6: Matriz de compatibilidade dos navegadores web com o padrão CORS [DEVERIA, 2011]..

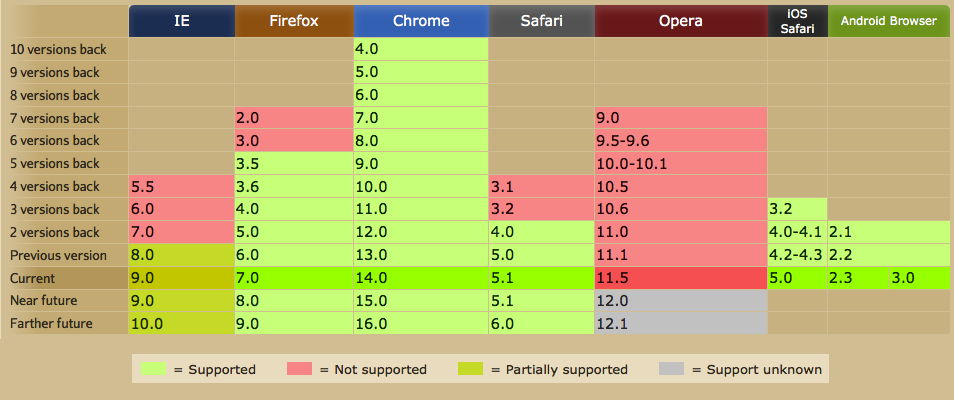


Figura 6: Matriz de compatibilidade dos navegadores web com o padrão CORS [DEVERIA, 2011].

No contexto dos computadores de mesa, mais conhecidos como Desktop, os navegadores do Google, Mozilla e Apple, respectivamente, Chrome, Firefox e Safari são absolutamente aderentes ao CORS. O navegador Opera, atualmente na versão 11.51, não possui suporte, porém o assunto já é tratado com grande importância nos fóruns de sua comunidade e, acredita-se que, em sua próxima versão este navegador já compreenderá o novo mecanismo de compartilhamento. A Microsoft, mantenedora do Internet Explorer, implementa parcialmente o comportamento definido pela especificação, não suportando requisições credenciadas e o mecanismo de pré-voo. Outro ponto a ser considerado com relação a implementação da Microsoft é sua opção por não seguir a convenção, adotada pelos demais navegadores, de adicionar os novos comportamentos definidos pelo CORS no objeto XMLHttpRequest já existente, tendo criado um novo objeto denominado XDomainRequest.

No cenário do mundo mobile, praticamente todo o trefego de informações realizado é feito através dos navegadores iOS Safari, Android Browser, Opera Mini e Opera Mobile, sendo que, destes, apenas os dois últimos ainda não possuem suporte ao CORS, porém, como são mantidos pela mesma empresa do navegador Opera, suas aderências são consideradas eminentes.