Engineer

CORS

別のオリジンのページに応答を読ませてよいかを、サーバーがブラウザーへ伝える仕組み

どのようなものか

CORSはCross-Origin Resource Sharingの略で、別のウェブサイトのページが応答を読んでもよいと、サーバーからブラウザーへ伝えるための規則です。ブラウザーは、オリジンを分けて扱います。オリジンは、スキーム、ホスト、ポートの組み合わせです。そのため、`https://app.example.com` 、 `https://api.example.com` 、 `http://app.example.com` は、それぞれ異なる三つのオリジンになります。一つのオリジンで動くスクリプトは、相手の許可がなければ、ほかのオリジンの応答を読めません。CORSは、その許可を伝える方法です。サーバーが応答に加える少数のHTTPヘッダー群で、ほとんどは `Access-Control-` で始まります。ブラウザーは、呼び出したスクリプトへ何かを渡す前に、これらを確認します。

プロトコルは、ブラウザーによるリソース取得を定め、随時更新されるWHATWGのFetch Standardにあります。その理由も明記されています。ページは、別のオリジンのデータを使い、HTMLフォームで送れるものより多様なリクエストを送れるべきですが、それは相手が明示的に許可した場合だけです。この明示的な許可を求める仕組みがなければ、公開ウェブのページが企業のファイアウォールの内側のサーバーを読んだり、利用者のCookieを付けて別のサイトの利用者の非公開データを読んだりできてしまいます。

中心となるやり取りは短いものです。スクリプトが別のオリジンのリソースを要求すると、ブラウザーは、ページ自身のオリジンを示す `Origin` ヘッダーを加えます。サーバーは `Access-Control-Allow-Origin` で答え、値には、そのオリジンを正確に記すか、誰でもよいことを示す `*` を使います。一致すれば、スクリプトは応答を受け取ります。一致しなければ、ネットワークエラーが渡され、本文は一切受け取れません。

送信自体の前に許可が必要なリクエストもあります。 `GET` 、 `HEAD` 、 `POST` で、少数の一般的なヘッダーだけを持ち、本文が `application/x-www-form-urlencoded` 、 `multipart/form-data` 、 `text/plain` のいずれかなら、そのまま送ります。HTMLフ

ォームでも、どのサイトからでも既に同じものを送れるからです。それを超える `PUT` 、 `DELETE` 、JSONの本文、 `Authorization` ヘッダーなどには、プリフライトが先行します。 `OPTIONS` リクエストを送り、 `Access-Control-Request-Method` でメソッドを、 `Access-Control-Request-Headers` で追加のヘッダーを示します。実際のリクエストを送るのは、応答が200や204などの成功ステータスを持ち、両方を許可している場合だけです。

概念を判断に生かす

「CORSとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 アカウント、パスワード、権限を整理し、業務に必要なアクセスだけを付与し、退職者の権限は当日に解除します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Linuxサーバーを守る1,000件のRBACルールを作成 — 1,000 セキュリティ上の穴を塞ぐアクセスルール

Linuxサーバーを守る1,000件のRBACルールを作成 お問い合わせ →

なぜ重要なのか

APIを分離したウェブアプリケーションのほぼすべてが、初日にCORSに出会います。フロントエンドは一つのオリジン、APIは別のオリジンから配信されます。最初の `fetch` は、APIが適切なヘッダーを送るまで失敗し、エラーがブラウザーのコンソールに出ます。この形で出会うチームは、CORSを開発の妨げと捉え、エラーを消すためなら何でも許可するように設定しがちです。そこにリスクがあります。

ヘッダーが決めるのは、利用者のブラウザーを通して、誰がその利用者のデータを読めるかです。アプリケーションにログインした利用者は、セッションCookieを持っています。APIが、どのオリジンからでも認証情報付きのリクエストを送り、その答えを読むことを許せば、利用者が訪れたどのページでも、その人になり代わってAPIを呼び、結果を読めます。Fetch Standardは、認証情報付きの応答の共有について、「混乱した代理人の問題を避けるには、極めて慎重に扱う必要がある」と明言しています。利用者の認証情報を持つブラウザーが代理人であり、不注意なCORSポリシーは、見知らぬ相手のページがブラウザーに指図できるようにしてしまいます。

CORSが何をしないかを知ることも、同じように重要です。サーバーを保護するものではありません。単純なリクエストは、後から応答を共有するかどうかに関係なく、サーバーに到達して実行されます。そのため、データを書き換える `POST` は、ブラウザーがCORSヘッダーを一つでも見る前に、データを変更しています。ブラウザーの外では、 `Origin` ヘッダーは呼び出し元が書いたとおりの

値であり、OWASPのテストガイドも偽装可能だと注意しています。誰がエンドポイントを呼べるか、そこで何ができるかは、毎回、サーバー上の認証と認可で決めます。CORSが決めるのは、ブラウザー内で、どのウェブページが応答を読めるかだけです。

実際の活用例

`https://app.example.com` のシングルページアプリケーションが、ログイン時にAPIの設定したセッションCookieを使い、 `https://api.example.com` のAPIを通してノートを保存するとします。

```
await fetch("https://api.example.com/v1/notes/42", {
  method: "PUT",
  credentials: "include",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ text: "金曜日にリリースする" }),
});
```

JSONの本文を持つ `PUT` は、フォームが送れる範囲を超えるため、ブラウザーは先に尋ねます。プリフライトにはCookieを付けず、実際のリクエストが使うものだけを示します。

```
OPTIONS /v1/notes/42 HTTP/1.1
Host: api.example.com
Origin: https://app.example.com
Accept: */*
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: content-type
```

APIは、自分の一覧に照らしてオリジンを確認し、答えます。

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: https://app.example.com
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: PUT
Access-Control-Allow-Headers: content-type
Access-Control-Max-Age: 600
Vary: Origin
```

ブラウザーは `PUT` を、Cookieと `Origin` ヘッダーを付けて送ります。APIの応答も、 `Access-Control-Allow-Origin` と `Access-Control-Allow-Credentials: true` を繰り返し含めます。どちらか一方でも欠けると、ノートはサーバーに保存されても、スクリプトはそのことを知れません。 `Access-Control-Max-Age` によって、ブラウザーは毎回尋ねる代わりに、最大600秒、プリフライトの答えを再利用できます。このヘッダーがなければ、標準では五秒保持します。 `Vary: Origin` が重要なのは、要求元のオリジンによって応答が変わるからです。それを無視するキャッシュは、あるオリジン向けの答えを別のオリジンへ渡す可能性があります。

職業や組織に関するデータを持つプラットフォームでは、データベースに何かが届く前に、API層でレート制限、CORS、リクエストサイズの制限、入力検証を行います。同じページは、CORSがこの一覧の一項目であり、防御全体ではない理由も説明しています。プラットフォームはクライアントが悪意を持つと仮定し、ブラウザーが何を許可したかに関係なく、サーバーで各リクエストの権限を改めて確認します。

問題が起きるところ

サーバーが、どの `Origin` もそのまま `Access-Control-Allow-Origin` へコピーします。 コンソールのエラーを消す最短の方法ですが、 `Access-Control-Allow-Credentials: true` も加えると、インターネット上のすべてのウェブサイトに、利用者から見えるAPIの内容を渡してしまいます。アクセスを認めるオリジンの一覧を明示的に持ち、届いた `Origin` と完全一致で比較します。一致したときだけ、ヘッダーを送ります。

一覧を前方一致や部分文字列で確認します。 `https://app.example.com` で始まるすべてのオリジンを認める規則は、 `https://app.example.com.attacker.example` も認めます。部分文字列の一致はさらに危険です。シリアライズされたオリジン全体を比較します。シリアライズしたオリジンには、パスも末尾のスラッシュもありません。そのため、一覧に `https://app.example.com/` を記しても、何にも一致しません。

認証情報があっても `*` が使えると思ってしまいます。 リクエストが認証情報を含むと、ブラウザーは `*` を受け付けません。これは、 `Access-Control-Allow-Origin` 、 `Access-Control-Allow-Methods` 、 `Access-Control-Allow-Headers` 、 `Access-Control-Expose-Headers` のすべてに当てはまります。 `Access-Control-Allow-Credentials` は、小文字の `true` と正確に一致する必要もあります。Cookieのない、公開された読み取り専用APIは、安全に `*` を返せます。セッションに依存するAPIは、オリジンを指定しなければなりません。

許可する一覧に `null` があります。 `data:` URLから読み込んだページは、不透明なオリジンを持ちます。オリジンをまたいでリダイレクトされたリクエストも `Origin: null` を報告するため、互いに無関係な多数の文書が同じ値を示します。`null` を認めれば、そのすべてを認めることになります。一覧に含めないようにします。

`Vary: Origin` がありません。 APIが `Access-Control-Allow-Origin` を送るのが、一致する `Origin` を持つリクエストに限られる場合、キャッシュは、そのヘッダーのない応答を保存して後のCORSリクエストに返すか、あるオリジンの応答を保存して別のオリジンに返す可能性があります。Fetch Standardは、応答がオリジンによって変わるときは常に `Vary: Origin` を求めています。許可するオリジンの値が `*` か、一つの固定されたオリジンなら、すべての応答に送り、`Vary` は省きます。

`Access-Control-Allow-Headers: *` がすべてを含むと思ってしまいます。 ワイルドカードは `Authorization` を一切含まず、名前を明示する必要があります。認証情報付きのリクエストでは、何も含みません。クライアントが実際に送るヘッダーを指定します。

CORSをアクセス制御として扱います。 `Origin` が不適切に見えるとリクエストを拒み、そうでなければ受け入れるサーバーの検査は、ブラウザーの外のスクリプトならどれでも通り抜けられます。呼び出し元を認証し、サーバーで操作を認可します。CORSには、ブラウザーがページに何を見せるかを決めるという、一つの役割を任せます。

参考資料

- Fetch Standard：CORSプロトコル — リクエストと応答のヘッダー、プリフライト、認証情報の表、`Vary` に関する助言を含む定義です。
- RFC 6454：ウェブのオリジンという概念 — オリジンの意味、スキーム、ホスト、ポートによる決定と、`Origin` ヘッダーへのシリアライズ方法を説明します。
- OWASPウェブセキュリティテストガイド：オリジン間リソース共有のテスト — ワイルドカードのオリジンや、確認せずにそのまま返すオリジンを、テスト担当者がどう調べるかを説明します。



用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「CORSとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/cors/