SAMPLE CHAPTER

CORS IN ACTION

Creating and consuming cross-origin APIs

Monsur Hossain

FOREWORD BY Eric Bidelman

MANNING

CORS in Action

by Monsur Hossain

Chapter 1

brief contents

1 The Core of CORS

This chapter covers

- Which issues CORS solves
- How a CORS request works
- The benefits of CORS

Suppose you're building a web mashup to load photos from the New York Public Library's (NYPL) Flickr page and display them on your own page. What would the code look like? You could start with an HTML page to display the photos, add JavaScript code to load the photos from the Flickr page, and display them on the page. Pretty straightforward, right?

But if you were to run this code in the browser, it wouldn't work because the browser's same-origin policy limits client code from making HTTP requests to different origins. This means that a web page running from your desktop or web server can't make an HTTP request to Flickr.com.

Cross-Origin Resource Sharing, or CORS, was built to help solve this issue. Before CORS, developers would need to go to great lengths to access APIs from JavaScript clients in the browser. CORS enables cross-origin requests in a safe, standard manner. From a client's perspective, CORS is awesome because it opens up a new world of APIs that previously wasn't available to browser JavaScript. From a server's

perspective, CORS is awesome because it allows the server to open up its APIs to a new world of users.

This chapter gives an overview of what CORS is and how it's used. It begins by reviewing CORS's features and benefits. It then walks through the code to make a CORS request.

1.1 What is CORS?

CORS is simply a way of making HTTP requests from one place to another. This is a trivial thing in other programming languages. But it's difficult to do in client-side JavaScript, because for years the browser's same-origin policy has explicitly prevented these types of requests.

This may make CORS sound like a contradiction. How can CORS allow cross-origin requests if the same-origin policy explicitly forbids them? The key is that CORS puts servers firmly in charge of who can make requests, and what type of requests are allowed. A server has the choice to open up its API to all clients, open it up to a small number of clients, or prevent access to all clients.

So if browsers enforce a same-origin policy, how does CORS work? The secret lies in the request and response headers. The browser and the server use HTTP headers to communicate how cross-origin requests behave. Using the response headers, the server can indicate which clients can access the API, which HTTP methods or HTTP headers are allowed, and whether cookies are allowed in the request.

Figure 1.1 shows what an end-to-end CORS request to the Flickr API looks like.

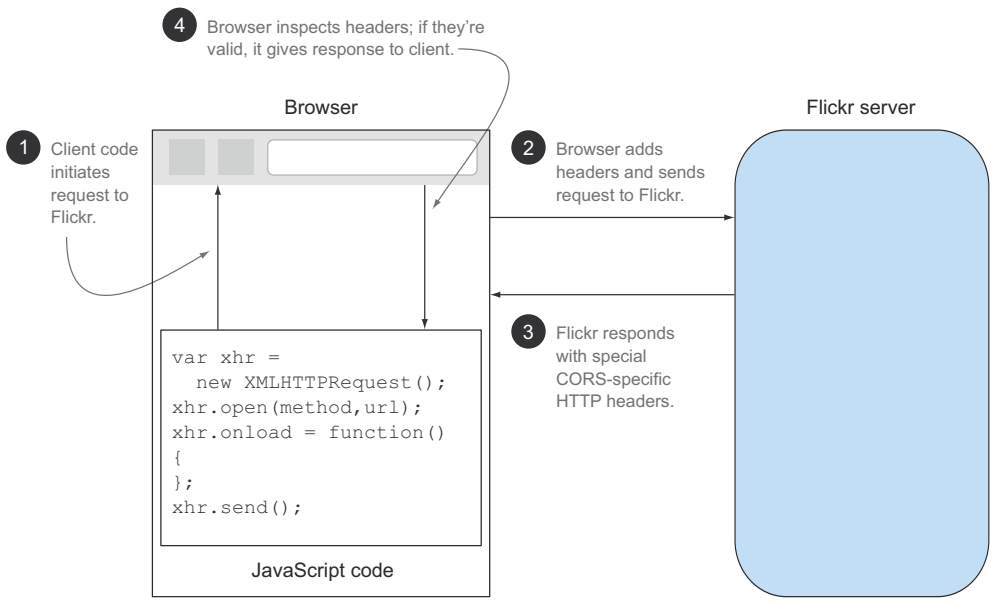

Figure 1.1 End-to-end CORS request flow

Here is a simplified look at the steps to making a CORS request (there are a few more nuances to some CORS requests, which we'll cover in later chapters):

1. The CORS request is initiated by the JavaScript client code.
2. The browser includes additional HTTP headers on the request before sending the request to the server.
3. The server includes HTTP headers in the response that indicate whether the request is allowed.
4. If the request is allowed, the browser sends the response to the client code.

If the headers returned by the server don't exist, or aren't what the browser expects, the response is rejected and the client can't view the response. In this way, browsers can still enforce the same-origin policy on servers that don't allow cross-origin requests. Now that you have a sense of what CORS is, let's turn our attention to making a CORS request.

1.2 CORS by example

Let's demonstrate how CORS works by building a Flickr sample app. Figure 1.2 shows the app, which loads photos from the NYPL's Flickr site and displays them on the page.

The following listing shows the code behind this sample.

Listing 1.1 Making a CORS request

```
<!DOCTYPE html>
<html>
<body onload="loadPhotos();">
<div id="photos">Loading photos...</div>
<script>
function loadPhotos() {
  var api_key = '<YOUR API KEY HERE>';
  var method = 'GET';
  var url = 'https://api.flickr.com/services/rest/?' +
      'method=flickr.people.getPublicPhotos&' +
      'user_id=32951986%40N05&' +
      'extras=url_q&format=json&nojsoncallback=1&' +
      'api_key=' + api_key;

  var xhr = new XMLHttpRequest();
  if (!('withCredentials' in xhr)) {
    alert('Browser does not support CORS.');
    return;
  }
  xhr.open(method, url);

  xhr.onerror = function() {
    alert('There was an error.');
  };

  xhr.onload = function() {
    var data = JSON.parse(xhr.responseText);
    if (data.stat == 'ok') {
      var photosDiv = document.getElementById('photos');
      photosDiv.innerHTML = '';
```

Request to Flickr API

Makes sure browser supports CORS

```
        var photos = data.photos.photo;
        for (var i = 0; i < photos.length; i++) {
          var img = document.createElement('img');
          img.src = photos[i].url_q;
          photosDiv.appendChild(img);
        }
      } else {
        alert(data.message);
      }
    };

    xhr.send();
}
</script>
</body>
</html>
```

Displays photos on page

NOTE If you'd like to run this sample in your browser, you'll need to obtain an API key from Flickr and substitute it for the `<YOUR API KEY HERE>` string in the code. You can obtain an API key at www.flickr.com/services/apps/create/.

Figure 1.2 Loading photos from Flickr using CORS

If you save this code to an HTML file (and set the API key as mentioned in the preceding note) and then open that file in your browser, you should see a bunch of photos. The key thing to note about this example is that although the web page is running from your local filesystem, it's making a request to the server at api.flickr.com. Let's walk through the code to get a better understanding of what each section is doing.

1.2.1 Setting up the request

The code starts by creating a new `XMLHttpRequest` object:

```
var xhr = new XMLHttpRequest();
if (!('withCredentials' in xhr)) {
  alert('Browser does not support CORS.');
  return;
}
xhr.open(method, url);
```

The first and last lines are the same for same-origin and cross-origin requests. The first line creates the `XMLHttpRequest` object, the last sets the HTTP method and URL.

The three middle lines highlight the difference between a same-origin-capable browser and a cross-origin-capable browser. If the browser fully supports CORS, the `XMLHttpRequest` object will contain a `withCredentials` property. You can use this property to check if the browser supports CORS. The preceding code alerts you if the browser doesn't support CORS.

1.2.2 Sending the request

Once the request is set up, you send the request to the server using the `send` method:

```
xhr.send();
```

This method initiates the HTTP request to the server. Chrome's Network tab gives you a better understanding of what the request looks like, as shown in figure 1.3. The figure shows the fact that even though the request originates from the filesystem ❶, the destination server is in fact flickr.com ❷.

Notice how the response has a header set to `Access-Control-Allow-Origin: *` ❸. The Access-Control-Allow-Origin header is the magic behind CORS. The server uses this header to indicate that cross-origin requests are allowed. The Access-Control-Allow-Origin header must always be present on a CORS response, but it's just one of many headers that can be used to configure CORS behavior. Part 2 of this book will cover these headers in greater detail.

1.2.3 Processing the response

Once the browser receives the response, it checks the response headers to verify that the cross-origin request is valid. If the request isn't valid, the browser will log an error

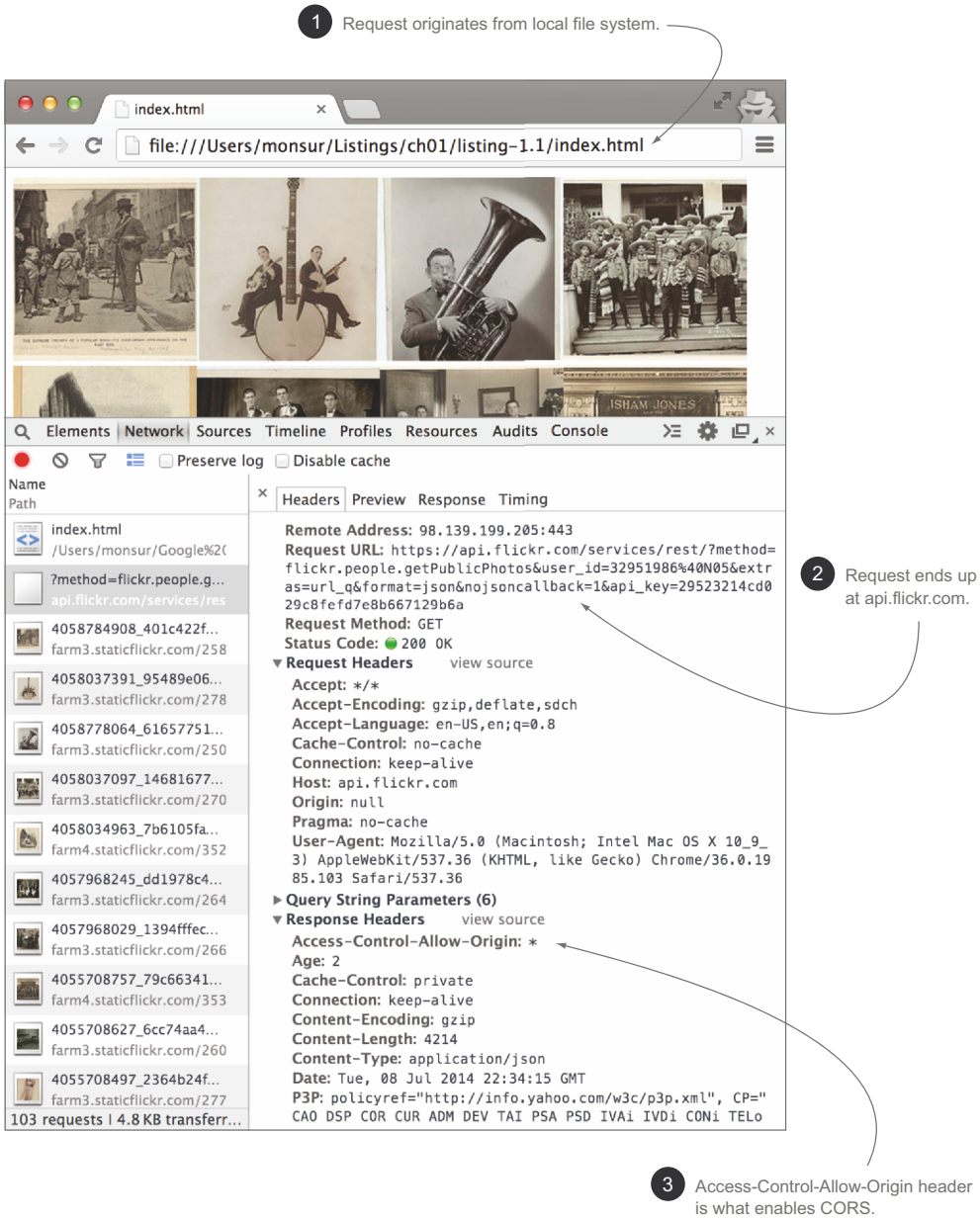

Figure 1.3 Details on the HTTP request and response from the Flickr API

to the console, then fire the `XMLHttpRequest`'s `onerror` event. But because the response here is valid, the browser fires the `XMLHttpRequest`'s `onload` event:

```
xhr.onload = function() {
  var data = JSON.parse(xhr.responseText);
```

```
    if (data.stat == 'ok') {
      var photosDiv = document.getElementById('photos');
      photosDiv.innerHTML = '';
      var photos = data.photos.photo;
      for (var i = 0; i < photos.length; i++) {
        var img = document.createElement('img');
        img.src = photos[i].url_q;
        photosDiv.appendChild(img);
      }
    } else {
      alert(data.message);
    }
  };
```

This code parses the response text into a JavaScript Serialized Object Notation (JSON) object, grabs the images from the object, and displays them on the page. In addition to the response text, the `XMLHttpRequest` object also has properties for the HTTP status, HTTP status text, and methods that retrieve response headers.

1.3 Benefits of CORS

The example from the previous section gave you a sense of power of CORS. Now let's turn our attention to some of the benefits CORS provides.

1.3.1 Wider audience

If you're building a public API, you want to open up access to as wide an audience as possible. Developers in other languages can easily use native libraries to make API requests. For example, a Python developer can use the httplib2 library to make HTTP requests to any server, regardless of where the request originates. The following snippet shows what a Python request to Flickr looks like. The httplib2 library doesn't care whether the server is CORS-enabled; it indiscriminately makes the request and processes the response, as this sample code shows:

```
import httplib2
h = httplib2.Http(".cache")
resp, content = h.request("http://www.flickr.com/photos/nypl/", "GET")
```

JavaScript developers don't have that advantage because the browser's same-origin policy limits HTTP requests to a single domain. CORS enables JavaScript developers to use an API the same way a developer in another language could.

1.3.2 Servers stay in charge

Safety is an important factor when making cross-domain requests. Any cross-domain mechanism needs to be careful not to break the browser's same-origin policy and inadvertently send requests to an unsuspecting server.

CORS achieves this safety by allowing servers to opt-in to CORS. For the request to succeed, the server must use response headers to explicitly acknowledge that the request

is allowed. That way, if a CORS request is made to a server that doesn't support CORS or doesn't have the right CORS headers, the request fails.

1.3.3 Flexibility

CORS gives servers the flexibility to configure cross-origin access in a variety of ways. The server can specify various features:

- Which domains are allowed to make requests
- Which HTTP methods are allowed (for example, `GET/PUT/POST/DELETE`)
- Which headers are allowed on the HTTP request
- Whether or not requests may include cookie data
- Which response headers the client can read

These rich settings put the server firmly in control of how the CORS request works.

1.3.4 Easy for developers

Because CORS is a standardized specification, it works in a consistent manner across sites. Developers need only learn about CORS once; then they can use the same techniques across all sites that support CORS.

While CORS requires new configuration on the server side, the client-side developer experience remains largely unchanged. JavaScript's `XMLHttpRequest` object has been available in browsers for over 10 years. Developers make CORS requests using the same `XMLHttpRequest` object they are familiar with. There isn't any new code for the developer to learn. From the developer's perspective, same-origin and cross-origin requests look mostly the same. (There are some slight differences, which we'll cover in chapter 2.)

1.3.5 Reduced maintenance overhead

There are ways to make cross-origin requests without using CORS (appendix D covers some of these techniques). But these techniques require custom code, custom servers, or additional documentation. This leads to an additional maintenance burden for the server developer.

Conversely, CORS only requires a few additional response headers. This reduced maintenance means that API owners can focus their attention on other things, rather than worrying about reinventing and maintaining new cross-domain mechanisms. Because CORS is a published specification with broad browser support, site owners can rest assured that their implementation is stable and that details won't change.

1.4 Summary

CORS allows client code to make cross-origin requests to remote servers. CORS is necessary because the browser's same-origin policy traditionally disallows cross-origin requests, which makes it difficult to load data from other sites. Here are some benefits of CORS:

- Opens an API to a wider audience
- Puts servers in charge of how CORS behaves
- Allows flexible configuration options
- Makes it easy for client developers to use
- Reduces maintenance overhead for server developers

The next chapter will dive into the details of how to make CORS requests from the browser.

WEB DEVELOPMENT

CORS IN ACTION

Monsur Hossain

Suppose you need to share some JSON data with another application or service. If everything is hosted on one domain, it's a snap. But if the data is on another domain, the browser's "same-origin" policy stops you cold. Cross-Origin Resource Sharing (CORS) is a new web standard that enables safe cross-domain access without complex server-side code. Mastering CORS makes it possible for web and mobile applications to share data simply and securely.

CORS in Action introduces Cross-Origin Resource Sharing from both the server and the client perspective. It starts with making and enabling CORS requests and then explores performance, debugging, and security. You'll learn to build apps that can take advantage of APIs hosted anywhere and how to write APIs that expand your products to a wider range of users.

What's Inside

- CORS from the ground up
- Serving and consuming cross-domain data
- Best practices for building CORS APIs
- When to use CORS alternatives like JSON-P and proxies

For web developers comfortable with JavaScript. No experience with CORS is assumed.

Monsur Hossain is an engineer at Google who has worked on API-related projects such as the Google JavaScript Client, the APIs Discovery Service, and CORS support for Google APIs.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/CORSinAction

Free eBook
SEE INSERT

"A well-rounded resource for developers wanting to learn the entire spectrum of CORS."
—From the Foreword by Eric Bidelman, Google

"All you need to know about CORS in one well-explained book."
—Roger Keizer, HUSS B.V.

"The right balance of application and theory."
—Roger Le, Coder Vox

"Stop getting cross-eyed from cross-domain problems."
—Christopher Haupt
Mobirobo, Inc.

ISBN 13: 978-1-617291-82-1
ISBN 10: 1-617291-82-X
54999
9 781617 291821

MANNING $49.99 / Can $52.99 [INCLUDING eBOOK]