Consuming Cross-Domain RESTful Web API in AJAX

IT 4403 Advanced Web and Mobile Applications

Jack G. Zheng

Fall 2019

KENNESAW STATE UNIVERSITY

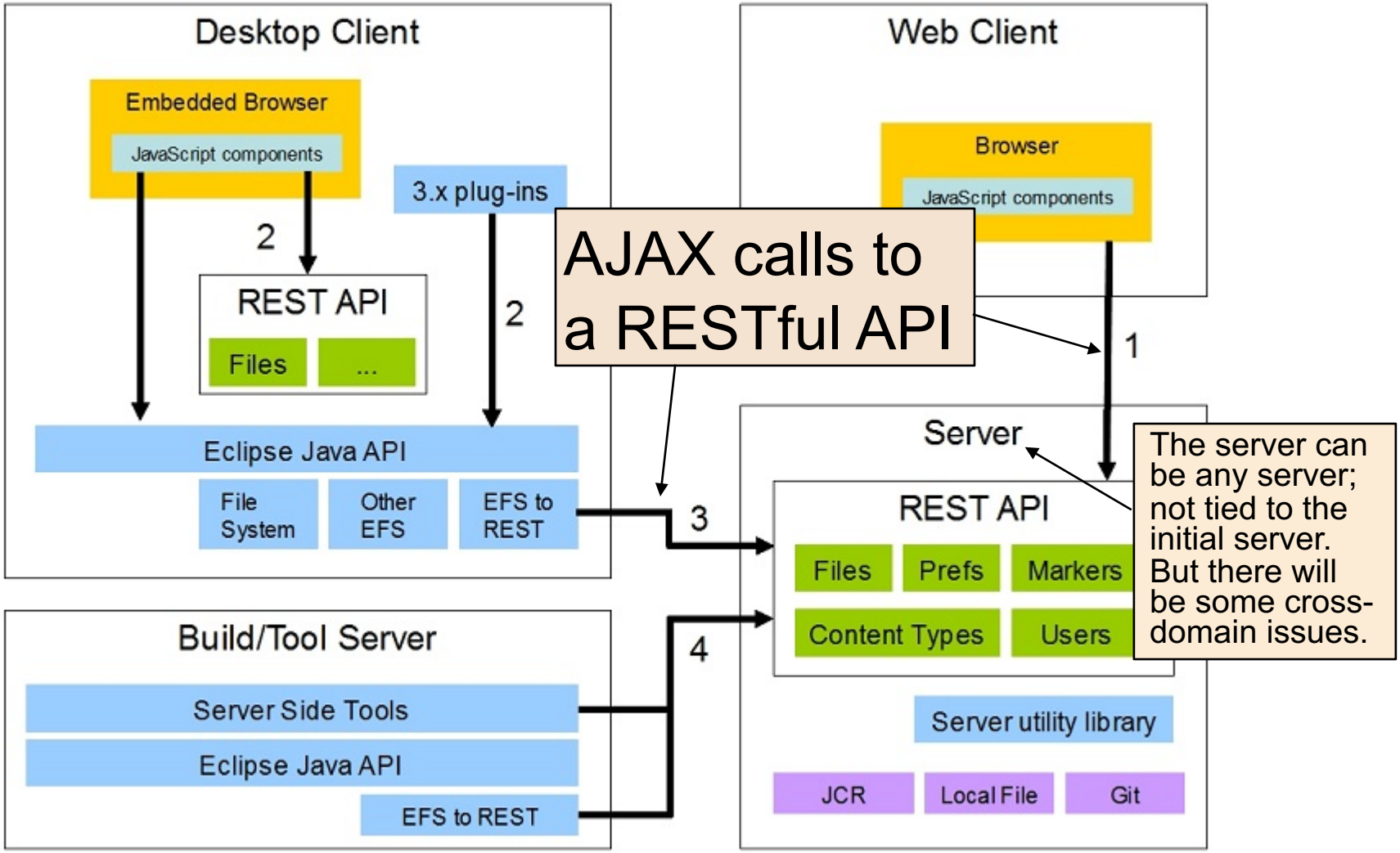
Overview

- Cross-domain (same origin) issues of consuming RESTful resources using AJAX
- Major techniques to bypass the restriction
  - JSON-P
  - CORS HTTP setting
  - Server proxy

AJAX and Services

- AJAX change the “client” from browser to the web app holder
  - The web app is separated to the app holder (the client) and app content
  - The initial app holder is downloaded from the server
  - The client consumes services (which provide app content) from the server
- AJAX enables the “client” (at the browser side) to directly consumes RESTful Web services, without server side support.
  - Which makes the client more independent and self contained.
- With AJAX, the presentation logic and business logic can completely move from server to client; data access logic remain on the server
  - Business logic is expose to the client; if not desired, business logic should remain on the server
- In many ways, AJAX applications follow the REST design principles. Each XMLHttpRequest can be viewed as a REST service request, sent using GET. And the response is often in JSON, a popular response format for REST.
- http://rest.elkstein.org/2008/02/ajax-and-rest.html

AJAX + (RESTful) Services



http://stage.vambenepe.com/archives/1801

Key Problem

- Due to browser security restrictions, most "AJAX" requests are subject to the same origin policy; the request can not successfully retrieve data from a different domain, subdomain, port, or protocol.
- Same origin policy
  - https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- Note this restriction is enforced by the browser.

Definition of Origin/Domain

- Two pages have the same origin if the protocol, port (if one is specified), and host are the same for both pages.
- The following table gives examples of origin comparisons to the URL http://store.company.com/dir/page.html

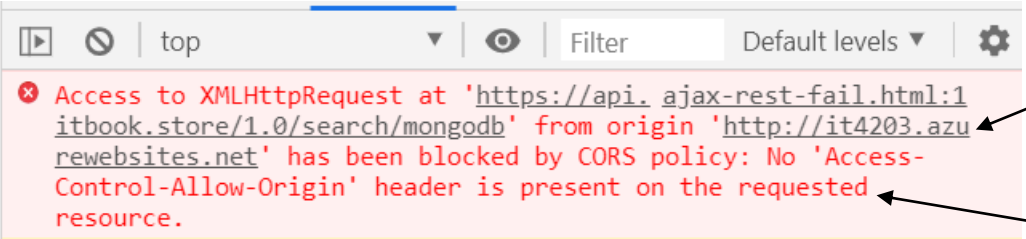
URL	Outcome	Reason
http://store.company.com/dir2/other.html	Success	
http://store.company.com/dir/inner/another.html	Success	
https://store.company.com/secure.html	Failure	Different protocol
http://store.company.com:81/dir/etc.html	Failure	Different port
http://news.company.com/dir/other.html	Failure	Different host

https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Example

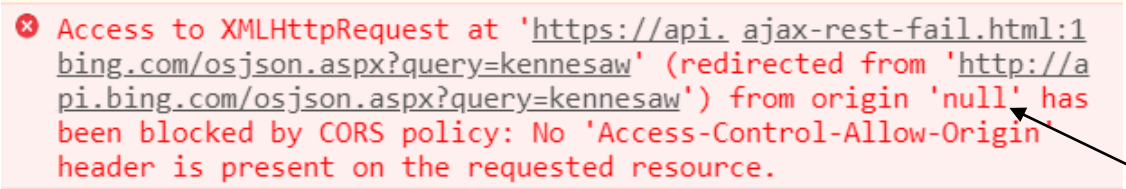
- Use the following testing client to request some samples services directly. Check the error messages in developer tools console
  - http://it4203.azurewebsites.net/demo/ajax-rest/ajax-rest-fail.html

- We will get an error: if “access-control-allow-origin” HTTP response header from the service provider is missing, or its value does not match your current domain, or your current domain is “null”. Example APIs that require this:
  - https://api.itbook.store/1.0/search/mongodb
  - https://api.bing.com/osjson.aspx?query=kennesaw
  - https://joke-api-strict-cors.appspot.com/random_joke



Requested from a page on a hosted site/domain

The required header is missing.



Null means it is requested from a local HTML file (no hosting server/site)

Services That Support CORS

- The following service allows cross origin requests with “access-control-allow-origin” HTTP response header set to * (all)
  - http://ziptasticapi.com/30303
  - http://it-ebooks-api.info/v1/search/web
  - https://www.googleapis.com/books/v1/volumes?q=ajax
  - https://openlibrary.org/api/books?bibkeys=ISBN:1593274874&jscmd=data&format=json
- Note the server may also require authentication and directly and dynamically set the attribute to its request domain.

```
× Headers Preview Response Cookies »
Status Code: 200 OK
▼Response Headers view source
Access-Control-Allow-Headers: origin,
x-requested-with, content-type
Access-Control-Allow-Methods: PUT, GE
T, POST, DELETE, OPTIONS
Access-Control-Allow-Origin: *
CF-RAY: 237eee44993b37f2-ATL
Connection: keep-alive
Content-Encoding: gzip
```

```
▼Response Headers view parsed
HTTP/1.1 200 OK
Date: Thu, 20 Oct 2016 18:31:25 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=d31c3022b2bf0c151707
18:31:24 GMT; path=/; domain=.ziptastica
Access-Control-Allow-Origin: *
```

```
Response Headers view source
Access-Control-Allow-Headers: origin, x-requested-with, content-type
Access-Control-Allow-Methods: PUT, GET, POST, DELETE, OPTIONS
Access-Control-Allow-Origin: *
```

Three Work-around Techniques

- CORS (Cross Origin Resource Sharing)
  - The CORS standard provides the solution through the use of HTTP headers at the protocol level.
- JSONP (JSON Padding)
  - A programming hack which make use of HTML elements (essentially the script tag) that has no cross domain restrictions.
  - Will cover more in the following slides
- Server proxy
  - Use the server which hosts the current application to request the service from the server side and then transfer the response to the client.
  - Need server support

CORS

- Cross-Origin Resource Sharing (CORS) is a W3C spec that allows cross-domain communication from the browser.
- CORS support requires coordination between both the server and client
- It makes use of HTTP headers
  - The service provider will determine the rules to respond with proper HTTP headers, especially "**Access-Control-Allow-Origin**".
  - Usually used with authentication
  - https://en.wikipedia.org/wiki/Cross-origin_resource_sharing
- The client side process is managed by the browser, not the JavaScript program
  - Browser support http://caniuse.com/#feat=cors

Server Proxy

- The cross domain restriction does not apply to servers. So the server can act as a middle man.

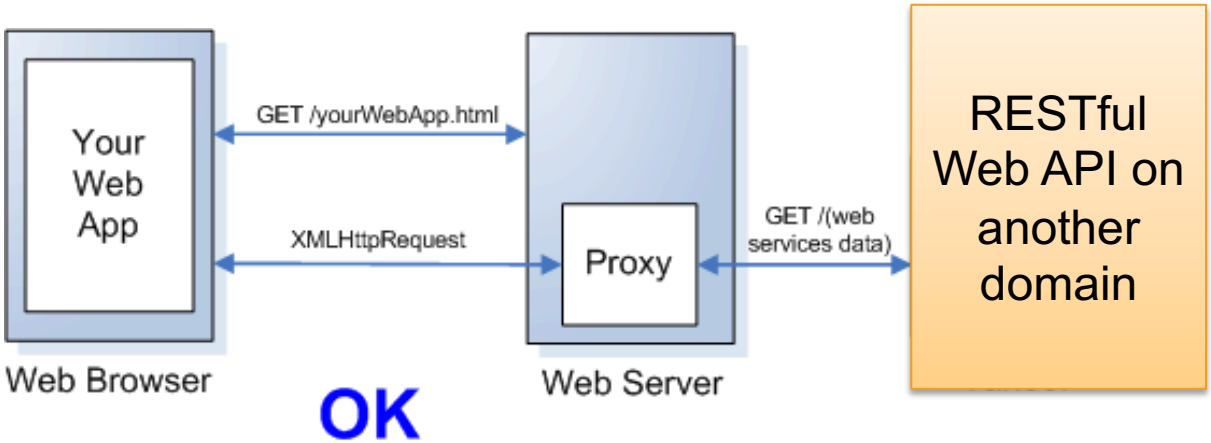


Image from https://www.sitepoint.com/working-around-origin-policy/

- Two methods
  - CORS proxy: https://cors-anywhere.herokuapp.com/ - see example “itbook.store-cors.html”
  - Server side app proxy on your own server. For example, develop a PHP app to call the service and redirect the response to the client. See examples “itbook.store-proxy.html” and “itbook.store-proxy.php”

JSONP

- JSON Padding is a technique to avoid the same origin constraint by using the script tag that can download JavaScript from any domain.
  - Strictly speaking, JSONP is not AJAX. It does not use the XHR to manage request and response.
  - But it provides a kind of asynchronous communication and supports partial page loading through dynamic script insertion.
  - JSONP needs specific web API support
- Three key elements in JSONP
  - The service provider needs to provide a response that wraps the JSON content in a callback function – not pure JSON (the JSON content is actually a value passed to the function)
  - The client JavaScript program needs to define the callback function that processes the JSON passed through the function
  - Static or dynamic script insertion in the client JavaScript program

Code Examples

- Live demo of all examples at http://it4203.azurewebsites.net/demo/ajax2
- **Examples can be downloaded on the content page. Please put all files to your server and see its effects. These AJAX examples need sever side support.**

Files	Example
ajax-rest-fail.html	A simple example to demo some failed cross-domain AJAX calls.
ajax-rest-cors-proxy.html itbook.store-cors.html	These examples demos the use of server CORS proxy https://cors-anywhere.herokuapp.com/
itbook.store-proxy.html itbook.store-proxy.php	These two files demos the use of a PHP server proxy to bypass CORS. itbook.store-proxy.php is the proxy app that actually calls the API.
bing-suggest-jsonp.html	Bing suggest app using JSONP method to bypass the same-origin restriction.
googlebooks-jsonp-getscript.html googlebooks-jsonp-getjson.html	Consuming Google Books service in a JSONP way. The first file shows the complete steps. The second file shows the use of getJSON function which handles the callback function and dynamic insertion behind the scene.
openlibrary-jsonp2.html	Open library API also needs to be called through JSONP in plain JavaScript. The example demos the use of JS variable instead of callback function. The service provider supports the JS variable function.

Two Techniques of JSONP

- Using a callback function – most popular way
  - Wrap JSON data in a JavaScript function as a parameter
  - Example: https://openlibrary.org/api/books?bibkeys=ISBN:1593274874&jscmd=data&callback=handleResponse

The response JSON is a parameter of a JavaScript function "handleResponse"

openlibrary.org/api/books?bibkeys=ISBN:1593274874&jscmd=data&callback=handleResponse


```
handleResponse({"ISBN:1593274874": {"publishers": [{"name": "No Starch Press"}], "pagination": "x
"level": 0}, {"title": "Structure and semantics", "label": "", "pagenum": "", "level": 0}, {"titl
{"title": "Modern JavaScript", "label": "", "pagenum": "", "level": 0}, {"title": "Device APIs",
"", "level": 0}, {"title": "Multimedia", "label": "", "pagenum": "", "level": 0}, {"title": "Web
"", "level": 0}, {"title": "Browser support as of March 2013", "label": "", "pagenum": "", "level
"https://openlibrary.org/books/OL25427547M/The_Modern_Web", "identifiers": {"isbn_13": ["97815932
```


- Using a JavaScript variable
  - Directly assign JSON data to a JavaScript variable
  - Example: https://openlibrary.org/api/books?bibkeys=ISBN:1593274874&jscmd=data

The response JSON is assigned to a JavaScript variable "_OLBookInfo"

openlibrary.org/api/books?bibkeys=ISBN:1593274874&jscmd=data


```
var _OLBookInfo = {"ISBN:1593274874": {"publishers": [{"name": "No Starch Pr
"", "level": 0}, {"title": "Structure and semantics", "label": "", "pagenum"
{"title": "Modern JavaScript", "label": "", "pagenum": "", "level": 0}, {"ti
"", "level": 0}, {"title": "Multimedia", "label": "", "pagenum": "", "level"
"", "level": 0}, {"title": "Browser support as of March 2013", "label": "",
"https://openlibrary.org/books/OL25427547M/The_Modern_Web", "identifiers": {
"https://covers.openlibrary.org/b/id/7257680-L.jpg", "medium": "https://cove
"https://openlibrary.org/subjects/web site development", "name": "Web site c
```

Example

See the “googlebooks-jsonp-getscript.html” example

```
<script>
    $(function ()
    {
        $("#btnSearch").click(function ()
        {
            $("#results").html("Searching ...");
            var service_point="https://www.googleapis.com/books/v1/volumes";
            var parameter="?q="+$("#txtTerm").val();
           $.getScript(service_point+parameter+"&callback=handleResponse");
         });
    });

    function handleResponse(json)
    {
        var resultHTML = "";
        for (i in json.items)
        {
            var cover = json.items[i].volumeInfo.imageLinks.smallThumbnail;
            resultHTML += "<img height='100' src='" + cover + "'/> ";
        }
        $("#results").html(resultHTML);
    }
</script>
```

```
// API callback
handleResponse({
 "kind": "books#volumes",
 "totalItems": 1561,
 "items": [
  {
   "kind": "books#volume",
```

1. The changed service point that returns JSON wrapped in a callback function

2. Define a callback function to process the JSON in the service response

3. Dynamic insertion of the script, which basically execute the callback function.

jQuery JSONP

- Use jQuery to handle callback functions is much easier
- Use the same AJAX functions include
  - .getJSON
  - .get
  - .ajax (see https://learn.jquery.com/ajax/working-with-jsonp/)
- These functions will handle the naming of callback functions and dynamic insertion of the script (as a response of the service)

getJSON

- If the service point URL includes the string "callback=?" (or similar, as the parameter name is defined by the server-side API), the request is treated as JSONP instead.
  - Basically we use the ? to indicate that getJSON should handle the naming of the callback function.
- Reference: http://api.jquery.com/jquery.getjson/

Example

See the “googlebooks-jsonp-getjson.html” example

```
$(function ()
  {
    $("#btnSearch").click(function ()
    {
                                $("#results").html("Searching ...");
      var service_point = "https://www.googleapis.com/books/v1/volumes";
      var parameter = "?q=" + $("#txtTerm").val();
      $.getJSON(service_point + parameter + "&callback=?").done(function (json)
      {
        … // JSON processing statements.
      });
    })
  });
```

The ? replaces the normal specific naming so jQuery will handle it.

```
// API callback
jQuery31103094721677038847 5_1477019763751({
 "kind": "books#volumes",
 "totalItems": 1274,
 "items": [
  {
   "kind": "books#volume",
```

Limitations of JSONP

- Only GET requests are allowed
- JSONP has to be supported by the source server (much like CORS). Fortunately, most websites/server configurations support JSONP (partly because of the limitations of CORS, and partly because it is an older technology compared to CORS)
- Because JSONP is not an AJAX request, you cannot make synchronous requests. The Asynchronous behavior is the result of your browser processing the script tag in the HTML page (as explained above)
- Security risk: The website/server returns a script. So essentially, you are trusting an external server to send over JavaScript code that could potentially read anything your browser client has access to (for e.g. cookies, other data on your current webpage like usernames, emails, etc. etc.). Hence it is safer to go with reputed sites if you are using JSONP.

Cross-Domain from Browser (Chrome) Extensions

- Chrome extensions support cross-domain requests in a two different ways:
- Include domain in manifest.json - Chrome extensions can make cross-domain requests to any domain *if* the domain is included in the "permissions" section of the manifest.json file:
- "permissions": ["http://*.html5rocks.com"]
- The server doesn't need to include any additional CORS headers or do any more work in order for the request to succeed.
- CORS request - If the domain is not in the manifest.json file, then the Chrome extension makes a standard CORS request. The value of the Origin header is "chrome-extension://[CHROME EXTENSION ID]". This means requests from Chrome extensions are subject to the same CORS rules described in this article.
- http://stackoverflow.com/questions/11127025/how-does-google-chromes-advanced-rest-client-make-cross-domain-post-requests

Good Resources

- Working With and Around the Same-Origin Policy: https://www.sitepoint.com/working-around-origin-policy/
- CORS: https://en.wikipedia.org/wiki/Cross-origin_resource_sharing
- JSONP: https://en.wikipedia.org/wiki/JSONP
- jQuery AJAX with JSONP: http://www.dotnetcurry.com/jquery/1095/json-jsonp-jquery-basic-tutorial
- https://blog.algolia.com/jsonp-still-mandatory/
- AJAX+REST as the latest architectural mirage: http://stage.vambenepe.com/archives/1801
- Mozilla resources:
  - https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS
- http://enable-cors.org