IO Lab: Working with APIs

September 23, 2013

Info 290TA (Information Organization Lab)
Kate Rushton & Raymon Sutedjo-The
HTTP Overview
HTTP Request

http://web-sniffer.net/

HTTP Request Header

Connect to 170.149.168.130 on port 80 ... ok

GET / HTTP/1.1[CRLF]
Host: www.nytimes.com[CRLF]
Connection: close[CRLF]
User-Agent: Web-sniffer/1.0.46 (+http://web-sniffer.net/)[CRLF]
Accept-Encoding: gzip[CRLF]
Accept-Charset: ISO-8859-1, UTF-8; q=0.7, *; q=0.7[CRLF]
Cache-Control: no-cache[CRLF]
Accept-Language: de, en; q=0.7, en-us; q=0.3[CRLF]
Referer: http://web-sniffer.net/[CRLF]
[CRLF]
HTTP Response

### HTTP Response Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>Date</td>
<td>Thu, 19 Sep 2013 16:35:11 GMT</td>
</tr>
<tr>
<td>Server</td>
<td>Apache</td>
</tr>
<tr>
<td>expires</td>
<td>Thu, 01 Dec 1994 16:00:00 GMT</td>
</tr>
<tr>
<td>cache-control</td>
<td>no-cache</td>
</tr>
<tr>
<td>pragma</td>
<td>no-cache</td>
</tr>
<tr>
<td>Set-Cookie</td>
<td>RMID=007f01002e39523b27bf0076; Expires=Fri, 19 Sep 2014 16:35:11 GMT; Path=/; Domain=.nyt</td>
</tr>
<tr>
<td>Set-cookie</td>
<td>adxcas=-; path=/; domain=.nytimes.com</td>
</tr>
<tr>
<td>Content-Type</td>
<td>text/html; charset=UTF-8</td>
</tr>
<tr>
<td>Transfer-Encoding</td>
<td>gzip</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>chunked</td>
</tr>
</tbody>
</table>

Content (encoded: 41.86 KiB / decoded: 180.80 KiB)

```html
<!DOCTYPE html>
<!--[if IE]><!--> <![endif]-->
<html lang="en">
```

HTTP Verbs

• GET
• POST
• PUT*
• DELETE* *less common
Which Type of Request?

- Add a book to the collection
- View a list of books
- Check the price of the book with the ID 1
- Remove the book with the ID 1
- Change the price of all books published before 1960
GET Request (Example)
GET Request (Example)

Query string: ?safe=active&hl=en&q=penguins
POST Request (Example)
POST Request (Example)

<form name="myForm" action="result.php" method="post">
Posting Title: <input type="text" name="title">
<input type="submit" value="Submit">
</form>
cURL

- **GET**
  
  ```bash
  $ curl http://www.myurl.com
  ```
cURL

- **POST**
  
  ```
  $ curl --data "key1=val1&key2=val2" http://myurl.com
  ```
Synchronous

Source: http://courses.cs.washington.edu/courses/cse190m/
Synchronous

**OskiCat UCB Library Catalog**

Quick Search | Advanced Keyword Search | Find Articles | Course Reserves | Renew Items | Questions?

---

Keyword(s)    penguins    Entire Collection

Limit to items not checked out

226 results found. Sorted by relevance | date | ti

Result Page 1 2 3 4 5 Next

**Keywords (1-50 of 226)**

**Most relevant titles entries 1-64**

1. **Penguins: natural history and conservation / edited by Pablo Garcia Borboroglu and P. Dee Boersma.**
   - Location: Seattle: University of Washington Press, [2013]
   - Cell No.: Bioscience & Natural Resources QL696.5473 P4527 2013
   - Status: AVAILABLE

2. **Penguins of the world / text and photographs by Wayne Lynch.**
   - Lynch, Wayne.
   - Request
Asynchronous

Source: http://courses.cs.washington.edu/courses/cse190m/
Asynchronous
AJAX

• Asynchronous
• JavaScript
• And
• XML (or HTML, JSON, etc.)
AJAX

- Not a programming language
- A particular way of using JavaScript
- The client (web browser) downloads data in the background and dynamically updates the page
- Avoids the click-wait-refresh cycle
Personality Embodiment in Multicharacter Online Games

Divya Anand, Evie Phan, Kate Rushton, Taeil Kwak, Yang Zhao

Introduction

According to industry reports, seventy-two percent of American households play computer or video games. Many of the top-selling games are in the action, role-playing, shooter, and strategy genres, which often require the player to create or select a character at the start of play (ESA 2011). From that
Web Applications
XMLHttpRequest

- JavaScript object for performing AJAX requests
- Problem: Cross-browsing compatibility annoyances
- Problem: Ugly syntax
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    alert(xhr.responseText);
  }
}
xhr.open('GET', 'http://example.com', true);
xhr.send(null);
jQuery + AJAX

- jQuery to the rescue!

```javascript
$.get('http://example.com',
    function(responseText) {
        alert(responseText);
    });
```
jQuery AJAX Request

1. user clicks, invoking an event handler function in your code
2. the handler creates an AJAX request (XMLHttpRequest)
3. the browser sends the request to the server
4. when the server responds, jQuery fires another event (callback)
5. The callback function updates the page
jQuery AJAX Request

```html
<div>
    <div id='output'></div>
    <button id='loadButton'>Load</button>
</div>
```
jQuery AJAX Request

```javascript
$('#loadButton').click(function() {
    $.get('http://myurl.com',
        function(data) {
            $('#output').text(data);
        });
});
```
Example 1
jQuery AJAX Variations

- Default: $.ajax()
- Shorthand: $.get()
  $.post()
  $.getJSON()
  $.load()

http://api.jquery.com/category/ajax/shorthand-methods/
jQuery AJAX Variations

$.ajax(
    {
        url: 'http://mywebsite.com/search',
        type: 'GET',
        data: {'q': 'penguins'},
        dataType: 'json',
        success: function(data) {
            //do something with data
        }
    }
);
Example 2
XSS (Cross-Site Scripting)

• In general, browsers must adhere to the same origin policy.

• AJAX requests to a server must come from the same place – the domain, protocol, and port must match.

• If they do not, your AJAX request will fail with an error.
Why?
Workarounds

- JSONP
- Cross-Origin Resource Sharing (CORS)
- Proxy Server
JSONP

- Requires explicit server support
- Kind of hacky – dynamically generates a `<script>` tag that loads JavaScript from a remote server
- The remote server’s code in turn calls a function in your code (the “callback” function)
CORS

- Requires explicit server support
- Server-side code can set the Access-Control-Allow-Origin header to explicitly allow a cross-domain request
- Either return the name of the server if it is approved, or "*" for all domains (only use with public resources)
- Does not work in IE 😞
Proxy Server

- Does not require any special behavior from the remote server
- AJAX requests are made to the same origin web server
- The server-side code communicates to the remote server on behalf of the client, then returns the requested data
AJAX Request (Client Only)

www.mysite.com → DATA PLZ → www.twitter.com

www.mysite.com ← LOL NOPE ← www.twitter.com
AJAX Request (Via Proxy)

DATA PLZ

HERE YOU GO!

www.mysite.com
So. Much. Data.

Source: http://www.ischool.berkeley.edu/
APIs

• A way for your code to interact with other software/systems

• Web API: An intentionally arranged collection of web resources and the interactions they support

• Accessed through HTTP requests
APIs

- Basically, an API (and its documentation) defines for developers:
  - What resources are available?
  - What properties do those resources have?
  - How do I access them?
  - How do I create my own?
# Last.fm API

<table>
<thead>
<tr>
<th>API Methods</th>
<th>Album</th>
<th>Artist</th>
<th>Geo</th>
<th>Track</th>
<th>Group</th>
<th>Library</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Album</strong></td>
<td>Album.addTags</td>
<td>Artist.addTags</td>
<td>Geo.getEvents</td>
<td>Track.addTags</td>
<td>Group.getHype</td>
<td>Library.addAlbum</td>
<td>User</td>
</tr>
</tbody>
</table>
album.getInfo

Params

- **artist** (Required (unless mbid)) : The artist name
- **album** (Required (unless mbid)) : The album name
- **mbid** (Optional) : The musicbrainz id for the album
- **autocorrect[0|1]** (Optional) : Transform misspelled artist names into correct artist names, returning the correct version instead. The corrected artist name will be returned in the response.
- **username** (Optional) : The username for the context of the request. If supplied, the user's playcount for this album is included in the response.
- **lang** (Optional) : The language to return the biography in, expressed as an ISO 639 alpha-2 code.
- **api_key** (Required) : A Last.fm API key.

Auth

This service does **not** require authentication.

Sample Response

```xml
<album>
  <name>Believe</name>
  <artist>Cher</artist>
  <id>2026126</id>
  <mbid>61bf0388-b8a9-48f4-81d1-7eb02706dfb0</mbid>
  <url>http://www.last.fm/music/Cher/Believe</url>
  <releasedate>6 Apr 1999, 00:00</releasedate>
  <image size="small">...</image>
  <image size="medium">...</image>
  <image size="large">...</image>
  <listeners>47602</listeners>
</album>
```
album.addTags

**Params**

- **artist** (Required): The artist name
- **album** (Required): The album name
- **tags** (Required): A comma delimited list of user supplied tags to apply to this album. Accepts a maximum of 10 tags.
- **api_key** (Required): A Last.fm API key.
- **api_sig** (Required): A Last.fm method signature. See [authentication](#) for more information.
- **sk** (Required): A session key generated by authenticating a user via the authentication protocol.

**Auth**

This service requires authentication. Please see our [authentication](#) how-to.

This is a write service and must be accessed with an HTTP POST request. All parameters should be sent in the POST body, including the 'method' parameter. See [rest requests](#) for more information.

**Sample Response**

```
<lfm status="ok">
</lfm>
```

**Errors**

- 2: Invalid service - This service does not exist
- 3: Invalid Method - No method with that name in this package
- 4: Authentication Failed - You do not have permissions to access the service
- 5: Invalid format - This service doesn't exist in that format
- 6: Invalid parameters - Your request is missing a required parameter
Delicious API

https://delicious.com/developers
Flickr Gallery

https://github.com/krushton/iolab-lecture6