IS 651: Distributed Systems
Chapter 7: REST Web Services

Jianwu Wang
Spring 2018
Learning Outcomes

• After learning chapter 7, you should be able to
  ▪ Understand the features of REST Web service and its differences from SOAP based Web service
  ▪ Know how to use proper method to call a REST Web service
  ▪ Know JSON and its differences from XML
  ▪ Know the cross-domain restriction and how to work around it
REST Basics

• REST is a term coined by Roy Fielding in his Ph.D. dissertation to describe an architecture style of networked systems. REST is an acronym standing for Representational State Transfer. It is easier to understand using representational resource state transfer.
  ▪ Resource: Resources are any addressable object (as something with a URI on the web), such as a book or student record.
  ▪ Representation: Resource representations for client and resources at server side are separated
  ▪ REST: access and manipulate resource states using a representational approach. We don’t care how the resource is actually stored/managed on the server, we only care its representation from the client view
REST Basics (2)

• REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages.

• An HTTP REST Web service follows three basic design principles
  ▪ Use HTTP methods explicitly (HTTP is stateless)
  ▪ Expose directory structure-like URIs
  ▪ Transfer XML, JavaScript Object Notation (JSON), or both

 ▪ There are no standards/specifications for REST Web service
Use HTTP methods explicitly

• Methods
  ▪ *POST*: create, sending data
  ▪ *GET*: read, list, retrieve
  ▪ *PUT*: replace, update
  ▪ *DELETE*: delete

• GET Examples
  ▪ From your web browser:
    http://www.webservicex.net/uszip.asmx/GetInfoByZIP?USZip=21250
  ▪ Curl command: curl -v
    http://www.webservicex.net/uszip.asmx/GetInfoByZIP?USZip=21250
POST vs. GET

• POST should be used for creating resources

• Common error:
  ▪ Wrong: GET /adduser?name=Robert HTTP/1.1
  ▪ Correct

```
POST /users HTTP/1.1
Host: myserver Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Robert</name>
</user>
```
GET vs. PUT

• GET is for data retrieval only. GET is an operation that should be free of side effects, a property also known as **idempotence**.

GET /users/Robert HTTP/1.1
Host: myserver
Accept: application/xml

• Common error:
  - Wrong: `GET /updateuser?name=Robert&newname=Bob HTTP/1.1`
  - Correct:

```plaintext
PUT /users/Robert HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Bob</name>
</user>
```

• Similarly, DELETE should be used rather than a deleteuser function with GET.
Be Stateless

• A complete, independent request doesn't require the server, while processing the request, to retrieve any kind of application/client context or state.
  ▪ Treats each request as an independent transaction that is unrelated to any previous request

• A REST Web service application/client includes, within the HTTP headers and body of a request, all of the parameters, context, and data needed by the server-side component to generate a response.

• Stateless server-side components are less complicated to design, write, and distribute across load-balanced servers.
Be Stateless (2)

- A stateless service not only performs better, it shifts most of the responsibility of maintaining state to the client application.

- In a RESTful Web service, the server is responsible for generating responses and for providing an interface that enables the client to maintain application state on its own.

- For example, in the request for a multipage result set, the client should include the actual page number to retrieve instead of simply asking for next.

- The principle of *loose-coupling* implies statelessness.
Compare a Stateless and Stateful Service

No state information is kept!
Expose Directory Structure-like URIs

- REST Web service URIs should be intuitive to the point where they are easy to guess.
  - Think of a URI as a kind of self-documenting interface that requires little, if any, explanation or reference for a developer to understand what it points to and to derive related resources.
Directory Structure-like URI Examples

• Example URIs from the book (not real):
  ▪ http://www.w3schools.com/catalog/cds (for cd list)
  ▪ http://www.w3schools.com/catalog/cds/2 (for the detailed info of the cd)
  ▪ http://www.w3schools.com/getCD.php?cd=2 (not the best url, but same as above)

• Examples from the #5 reference (not real):
  ▪ http://www.parts-depot.com/parts (for a parts list)
  ▪ http://www.parts-depot.com/parts/00345 (for a part)
  ▪ http://www.parts-depot.com/parts/getPart?id=00345 (not the best url, but same as above)
Transfer XML, JSON, or both

• The web service makes available a URL to submit a purchase order (PO).
  ▪ The client creates a PO instance document which conforms to the PO schema that Parts Depot has designed (and publicized in a WSDL document).
  ▪ The client submits PO.xml as the payload (i.e. entity body) of an HTTP POST message.

• The payload (HTTP entity body) should be in XML or JSON.
JSON

• Alternative serialization.

```json
{ "menu": {  "id": "file",  "value": "File:",  "popup": {    "menuitem": [      {"value": "New", "onclick": "CreateNewDoc()"},      {"value": "Open", "onclick": "OpenDoc()"},      {"value": "Close", "onclick": "CloseDoc()"}    ]  } }
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()"/>
    <menuitem value="Open" onclick="OpenDoc()"/>
    <menuitem value="Close" onclick="CloseDoc()"/>
  </popup>
</menu>
```
JSON Example

• The figure shows how JSON works in JavaScript.
  ▪ **Program link**

• The JSON is included directly in the program here, but could easily be the result of a REST query.

• It results in the simple text output to the browser: *Sally Green 27*
Guardian API

• Most Web applications offer a REST API such as Twitter, Flickr, YAHOO, Facebook, The New York Times, NPR, and the Guardian Newspaper.

• The Guardian News

• Guardian Open Platform API docs

• http://content.guardianapis.com/search?q=syria&section=news&from-date=2013-09-01&api-key=xyz (JSON)

• http://content.guardianapis.com/search?q=syria&section=news&from-date=2013-09-01&format=xml&api-key=xyz (XML)

• You can use command to call the Rest web service
  ▪ curl -v "http://content.guardianapis.com/search?q=syria&section=news&from-date=2013-09-01&api-key=xyz"

• You need to replace the xyz in the last two links with your api-key to make them work.
Cross-Domain Restriction

• One problem that comes up is that web browsers have a security limitation that requires any program running in the browser (using JavaScript with AJAX) can only return results from the same domain that the original web page came from.
  ▪ This is called the **cross-domain restriction**.
  ▪ Our Guardian example would therefore not work for a web page we created on gl since our web page is from umbc.edu and Guardian is on the guardian.com.

• Demo:  
  [https://userpages.umbc.edu/~jianwu/is651/programs/ch7/cross_domain_restriction.html](https://userpages.umbc.edu/~jianwu/is651/programs/ch7/cross_domain_restriction.html)
CURL and PHP

• One way around this is to use a server-side program to retrieve the XML or JSON from Guardian and then send it back to the user that requested the web page from gl.
  ▪ In order to do this, we will use PHP to issue the request using the Curl library.
  ▪ The Curl library offers a way to send a URL programmatically and handle the response.
CURL and PHP Example

```php
<?php
    $querystring='q=debates&section=news&from-date=2013-09-01&api-key=xyz';
    $host = 'https://content.guardianapis.com/search';
    $request = $host.'?'.$querystring;

    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $request); // set url
    curl_setopt($ch, CURLOPT_FAILONERROR, 1);
    curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1); // allow redirects
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1); // return variable
    curl_setopt($ch, CURLOPT_TIMEOUT, 4); // times out after 4s
    curl_setopt($ch, CURLOPT_HTTPGET, true); // set GET method

    $json = curl_exec($ch);
    curl_close($ch);
    header("Content-type: application/json"); //send http header
    echo "jsonProcessFn(".$json.");"; //wraps result in a function call
?>
```
JSONP (JSON with Padding)

• The client-side program can use JSONP to call the PHP and process the result.
  • JSONP sends JSON data without worrying about cross-domain issues.
  • JSONP uses the <script> tag, instead of the XMLHttpRequest object.
  • JSONP requires the data is wrapped by function name.

• Example: guardian.html.
<!DOCTYPE html>
<html>
<style>
  h3 { background: red; }
  span { color: green; }
</style>
<body>
<p id="output"></p>
<script>
  function jsonProcessFn(data) {
    var h3 = document.createElement("h3"); //create an h3 element
    h3.innerHTML = "User Tier = " + data.response.userTier;
    //output element is defined in the html.
    document.getElementById("output").appendChild(h3);
    //response and results are at php results.
    var arr = data.response.results;
    for (var i = 0, len = arr.length; i < len; i++) {
      newsProcessFn(arr[i]);
    }
  }
</script>
</body>
</html>
function newsProcessFn(news) {
    var li = document.createElement("li");
    var span1 = document.createElement("span");
    span1.innerHTML = "Date= ";
    var span2 = document.createElement("span");
    span2.innerHTML = "Title= ";
    li.appendChild(span1);
    li.appendChild(document.createTextNode(news.webPublicationDate + " "));
    li.appendChild(span2);
    li.appendChild(document.createTextNode(news.webTitle));
    document.getElementById("output").appendChild(li);
} </script>
<script src="guardian.php"></script>
</body>
</html>
Ch7 Exercise and Homework