RESTful web services

61FIT3NPR -Network Programming

Faculty of Information Technology
Hanoi University
Fall 2020

What's a Web Service?

- A web service is just a web page meant for a computer to request and process
- More precisely, a Web service is a Web page that's meant to be consumed by an autonomous program as opposed to a Web browser or similar UI tool
- Web Services require an architectural style to make sense of them, because there's no smart human being on the client end to keep track

What's a Web Service? (cont.)

- The pre-Web techniques of computer interaction don't scale on the Internet
- They were designed for small scales and single trust domains

REST

- Introduced and defined in 2000 by Roy Fielding in his doctoral dissertation at University of California
- Abstracts the architectural elements within a distributed hypermedia system
- A distillation of the way the Web already works

Introduction to REST

- REST stands for Representational State Transfer
- It is an architectural ***pattern*** for developing web services as opposed to a ***specification***.
- REST web services communicate over the HTTP specification, using HTTP vocabulary:
  - Methods (GET, POST, etc.)
  - HTTP URI syntax (paths, parameters, etc.)
  - Media types (xml, json, html, plain text, etc)
  - HTTP Response codes.

Introduction to REST (cont.)

- Representational
  - Clients possess the information necessary to identify, modify, and/or delete a web resource.
- State
  - All resource state information is stored on the client.
- Transfer
  - Client state is passed from the client to the service through HTTP.

Introduction to REST (cont.)

- The six characteristics of REST:
  - Uniform interface
  - Decoupled client-server interaction
  - Stateless
  - Cacheable
  - Layered
  - Extensible through code on demand (optional)
- Services that do not conform to the above required contstraints are not strictly RESTful web services.

REST defined

- Resources are identified by uniform resource identifiers (URIs)
- Resources are manipulated through their representations
- Messages are self-descriptive and stateless
- Multiple representations are accepted or sent
- Hypertext is the engine of application state

REST Web Services

- Representational State Transfer - basically means that each unique URL is a representation of some object
  - You can get the contents of that object using an HTTP GET, to delete it.
  - You can use a POST, PUT, or DELETE to modify the object
    - Most of services use POST

Who's using REST?

- Yahoo's web services: Flickr, del.icio.us
- eBay and Amazon have web services for both REST and SOAP
- Advantages:
  - Lightweight - not a lot of extra xml markup
  - Human Readable Results
  - Easy to build - no toolkits required

Data and Media Types

- A RESTful service can offer a selection of data types, and it's very common to offer multiple types.
  - JSON
  - XML
  - Others

RESTful URLs

- Example: the GitHub API
  - https://api.github.com/users/example
  - https://api.github.com/users/example/repos
  - https://api.github.com/users/example/gists

→ allow users to guess what will be found when visiting them

→ easily navigate around a predictable and clearly designed system

→ describe what data will be found there, and what to expect

HTTP Features in REST

- REST makes the most of HTTP's best features
  - Placing all the metadata about the request and response into the headers
  - Reserving the main body of the communications for the actual content
- Create Resources:
  - Resources are created by making a POST request
  - Successful status code will be included with the response when the resource has been successfully created
    - Code 201: Created
    - Code 200: Accepted, but not completed
    - Code 400: Bad Request
    - Code 406: Not Accepted
    - Code 409: Conflict

HTTP Features in REST (cont.)

- Read Records:
  - Resources are fetched by making a GET request without sending any body content with the GET request
  - Status code:
    - Code 200: Successfully Retrieved
    - Code 302: Found
    - Code 304: Not Modified
    - Code 404: Not Found
    - Code 402: Not Authorized
    - Code 403: Forbidden
    - Code 429: Too many requests

HTTP Features in REST (cont.)

- Update Records:
  - Resource should be retrieved by GET
  - The representation of the resource can be altered as needed then should be PUT back to its original URI
  - the whole resource will be fetched and sent back for the update
- Delete Records:
  - A DELETE request to the URI of the item to be deleted
  - No body content necessary

Additional Headers in RESTful Services

- Authorization Headers
  - This can be used with a variety of different techniques for authenticating users
  - Familiar to web developers
- Caching Headers
  - can help when an API server needs to handle a lot of traffic

HTTP-REST Request Basics

- The **HTTP request** is sent *from the client*.
  - Identifies the location of a **resource**.
  - Specifies the **verb**, or HTTP **method** to use when accessing the resource.
  - Supplies optional **request headers** (name-value pairs) that provide additional information the server may need when processing the request.
  - Supplies an optional **request body** that identifies additional data to be uploaded to the server (e.g. form parameters, attachments, etc.)

HTTP-REST Request Basics (cont.)

Sample Client Requests:

- A typical client GET request:

```
GET /view?id=1 HTTP/1.1
User-Agent: Chrome
Accept: application/json
[CRLF]
```

Requested Resource (path and query string)

Request Headers

(no request body)

- A typical client POST request:

```
POST /save HTTP/1.1
User-Agent: IE
Content-Type: application/x-www-form-urlencoded
[CRLF]
name=x&id=2
```

Requested Resource (typically no query string)

Request Headers

Request Body (e.g. form parameters)

HTTP-REST Response Basics

- The **HTTP response** is sent *from the server*.
  - Gives the **status** of the processed request.
  - Supplies **response headers** (name-value pairs) that provide additional information about the response.
  - Supplies an optional **response body** that identifies additional data to be downloaded to the client (html, xml, binary data, etc.)

HTTP-REST Response Basics (cont.)

- Sample Server Responses:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1337
[CRLF]
<html>
  <!-- Some HTML Content. -->
</html>
```

Response Status

Response Headers

Response Body (content)

```
HTTP/1.1 500 Internal Server Error
```

Response Status

```
HTTP/1.1 201 Created
Location: /view/7
[CRLF]
Some message goes here.
```

Response Status

Response Header

Response Body

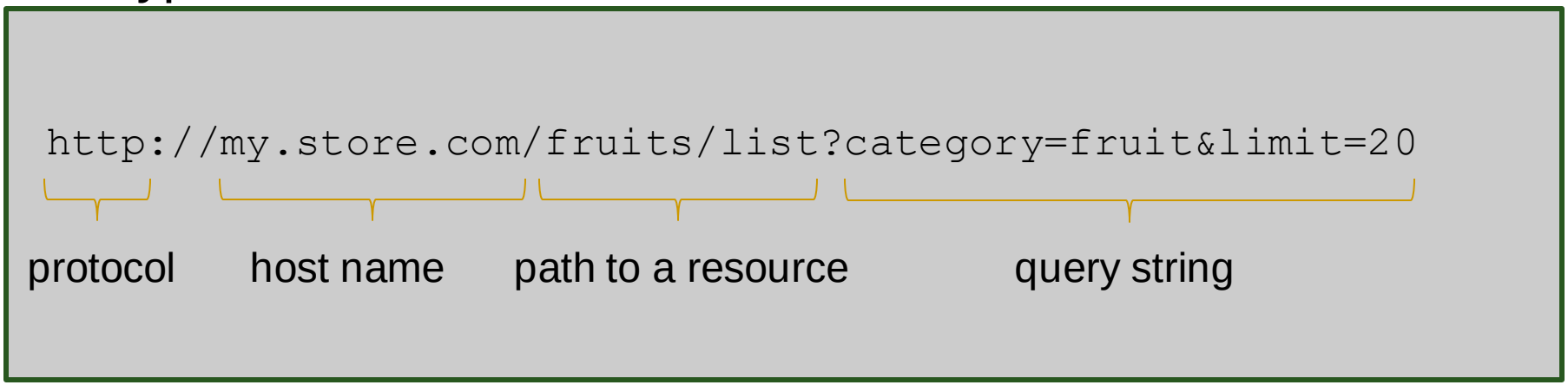
HTTP-REST Vocabulary

HTTP Methods supported by REST:

- GET – Requests a resource at the request URL
  - Should not contain a request body, as it will be discarded.
  - May be cached locally or on the server.
  - May produce a resource, but should not modify on it.
- POST – Submits information to the service for processing
  - Should typically return the new or modified resource.
- PUT – Add a new resource at the request URL
- DELETE – Removes the resource at the request URL
- OPTIONS – Indicates which methods are supported
- HEAD – Returns meta information about the request URL

HTTP-REST Vocabulary (cont.)

A typical HTTP REST URL:



- The **protocol** identifies the transport scheme that will be used to process and respond to the request.
- The **host name** identifies the server address of the resource.
- The **path** and **query string** can be used to identify and customize the accessed resource.

HTTP and REST

A REST service framework provides a **controller** for routing HTTP requests to a request handler according to:

- The HTTP method used (e.g. GET, POST)
- Supplied path information (e.g /service/listItems)
- Query, form, and path parameters
- Headers, cookies, etc.

Principles of REST web service design

1. Identify all the conceptual entities that we wish to expose as services. (Examples we saw include resources such as : parts list, detailed part data, purchase order)
2. Create a URL to each resource.
3. Categorize our resources according to whether clients can just receive a representation of the resource (using an HTTP GET), or whether clients can modify (add to) the resource using HTTP POST, PUT, and/or DELETE).

Principles of REST web service design

4. All resources accessible via HTTP GET should be side-effect free. That is, the resource should just return a representation of the resource. Invoking the resource should not result in modifying the resource.
5. Put hyperlinks within resource representations to enable clients to drill down for more information, and/or to obtain related information.
6. Design to reveal data gradually. Don't reveal everything in a single response document. Provide hyperlinks to obtain more details.

Principles of REST web service design

7. Specify the format of response data using a schema (DTD, W3C Schema, RelaxNG, or Schematron). For those services that require a POST or PUT to it, also provide a schema to specify the format of the response.
8. Describe how our services are to be invoked using either a WSDL document, or simply an HTML document.

REST vs. SOAP

Simple web service as an example: querying a phonebook application for the details of a given user.

- Using Web Services and SOAP, the request would look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
          <soap:body pb="http://www.acme.com/phonebook">
                    <pb:GetUserDetails>
                              <pb:UserID>12345</pb:UserID>
                    </pb:GetUserDetails>
          </soap:Body>
</soap:Envelope>
```

REST vs. SOAP (cont.)

- And with REST the query will probably look like this:

http://www.acme.com/phonebook/UserDetails/12345

```
GET /phonebook/UserDetails/12345 HTTP/1.1
Host: www.acme.com
Accept: application/xml
```

REST vs. SOAP (cont.)

- SOAP is definitely the heavyweight choice for Web service access. It provides the following advantages when compared to REST:
  - Language, platform, and transport independent (REST requires use of HTTP)
  - Works well in distributed enterprise environments (REST assumes direct point-to-point communication)
  - Standardized
  - Provides significant pre-build extensibility in the form of the WS* standards
  - Built-in error handling
  - Automation when used with certain language products

REST vs. SOAP (cont.)

- REST is easier to use for the most part and is more flexible. It has the following advantages when compared to SOAP:
  - No expensive tools require to interact with the Web service
  - Smaller learning curve
  - Efficient (SOAP uses XML for all messages, REST can use smaller message formats)
  - Fast (no extensive processing required)
  - Closer to other Web technologies in design philosophy

Deciding Between SOAP and REST

- The focus of your decision often centers on which Web service best meets your needs, rather than which protocol to use.
- The best way to discover whether SOAP or REST works best for you is to try a number of free Web services:
  - Free-Web-Services.com
  - WebserviceX.NET
  - SwaggerHub
  - XMethods

JSON

Overview and parsing

Overview

- What is JSON?
- Comparisons with XML
- Syntax
- Data Types
- Live Examples

What is JSON?

JSON example

- "JSON" stands for "JavaScript Object Notation"
  - Despite the name, JSON is a (mostly) language-independent way of specifying objects as name-value pairs
- Example (http://secretgeek.net/json_3mins.asp):

```
{"skillz": {
    "web":[
        { "name": "html",
          "years": 5
        },
        { "name": "css",
          "years": 3
        }]
    "database":[
        { "name": "sql",
          "years": 7
        }]
}}
```

What is JSON

- JavaScript Object Notation

- Used to format data
- Commonly used in Web as a vehicle to describe data being sent between systems

JSON is…

- A lightweight text based data-interchange format
- Completely language independent
- Based on a subset of the JavaScript Programming Language
- Easy to understand, manipulate and generate

JSON is NOT…

- Overly Complex
- A “document” format
- A markup language
- A programming language

Why use JSON?

- Straightforward syntax
- Easy to create and manipulate
- Can be natively parsed in JavaScript using **eval()**
- Supported by all major JavaScript frameworks
- Supported by most backend technologies

JSON vs. XML

Much Like XML

- Plain text formats
- “Self-describing“ (human readable)
- Hierarchical (Values can contain lists of objects or values)

Not Like XML

- Lighter and faster than XML
- JSON uses typed objects. All XML values are type-less strings and must be parsed at runtime.
- Less syntax, no semantics
- Properties are immediately accessible to JavaScript code

Knocks against JSON

- Lack of namespaces
- No inherit validation (XML has DTD and templates, but there is JSONlint)
- Not extensible
- It's basically just ***not*** XML

Syntax

JSON example

- "JSON" stands for "JavaScript Object Notation"
  - Despite the name, JSON is a (mostly) language-independent way of specifying objects as name-value pairs
- Example (http://secretgeek.net/json_3mins.asp):

```
{"skillz": {
    "web":[
        { "name": "html",
          "years": 5
        },
        { "name": "css",
          "years": 3
        }]
    "database":[
        { "name": "sql",
          "years": 7
        }]
}}
```

JSON Example

```
var employeeData = {
  "employee_id": 1234567,
  "name": "Jeff Fox",
  "hire_date": "1/1/2013",
  "location": "Norwalk, CT",
  "consultant": false
};
```

JSON syntax

- An *object* is an unordered set of name/value pairs
  - The pairs are enclosed within braces, { }
  - There is a colon between the name and the value
  - Pairs are separated by commas
  - Example: { "name": "html", "years": 5 }
- An *array* is an ordered collection of values
  - The values are enclosed within brackets, [ ]
  - Values are separated by commas
  - Example: ["html", ”xml", "css"]

JSON syntax

- A *value* can be: A string, a number, true, false, null, an object, or an array
  - Values can be nested
- *Strings* are enclosed in double quotes, and can contain the usual assortment of escaped characters
- *Numbers* have the usual C/C++/Java syntax, including exponential (E) notation
  - All numbers are decimal--no octal or hexadecimal
- *Whitespace* can be used between any pair of tokens

JSON Array Example

```
var employeeData = {
  "employee_id": 1236937,
  "name": "Jeff Fox",
  "hire_date": "1/1/2013",
  "location": "Norwalk, CT",
  "consultant": false,
   "random_nums": [ 24,65,12,94 ]
};
```

Data Types

Data Types: Strings

- Sequence of 0 or more Unicode characters
- Wrapped in "double quotes“
- Backslash escapement

Data Types: Numbers

- Integer
- Real
- Scientific
- No octal or hex
- No NaN or Infinity – Use **null** instead.

Data Types: Booleans & Null

- Booleans: true or false
- Null: A value that specifies nothing or no value.

Data Types: Objects & Arrays

- Objects: Unordered key/value pairs wrapped in { }
- Arrays: Ordered key/value pairs wrapped in [ ]

How to turn JSON into JavaScript object –eval(*)

- The JavaScript eval(***string***) method compiles and executes the given string
  - The string can be an expression, a statement, or a sequence of statements
  - Expressions can include variables and object properties
  - eval returns the value of the last expression evaluated
- When applied to JSON, eval returns the described object

JSON and—methods?

- In addition to instance variables, objects typically have methods
- There is nothing in the JSON specification about methods
- However, a method can be represented as a string, and (when received by the client) evaluated with eval
  - Obviously, this breaks language-independence
  - Also, JavaScript is rarely used on the server side

Comparison of JSON and XML

- Similarities:
  - Both are human readable
  - Both have very simple syntax
  - Both are hierarchical
  - Both are language independent
  - Both can be used by Ajax
  - Both supported in APIs of many programming languages
- Differences:
  - Syntax is different
  - JSON is less verbose
  - JSON can be parsed by JavaScript's eval method
  - JSON includes arrays
  - Names in JSON must not be JavaScript reserved words
  - XML can be validated

JSON in AJAX

- JSON can be used in AJAX as follows:
- Include it in HTML directly
- <html>... <script> var data = *JSONdata*; </script>... </html>
- JSON is used with XMLHttpRequest and can be converted into a JavaScript structure
- responseData = eval('(' + responseText + ')');

We have not yet spoken about AJAX --- revisit this again after you have learned about AJAX

Why is JSON better suited for AJAX?

- JSON is widely used in AJAX. The X in AJAX stands for XML.
- E.g.
- {
- "fullname": "Swati Kumar",
- "org": "Columbia",
- }

- <?xml version='1.0‘ encoding='UTF-8'?>
- <element>
- <fullname>Swati Kumar</fullname>
- <org>Columbia</org>
- </element>

- JSON response at client side is:
- *var name = eval('(' + req.responseText + ')').fullname.value;*
- To access a composite element
- *eval('(' + req.responseText + ')').xyz.abc.value;*
- Thus, any level deep elements can be easily accessed.

XML and JavaScript

- XML response at client side is:
- *var root = req.responseXML;*
- *var name = root.getElementsByTagName('fullname');*
- To access a composite element
- *root.getElementsByTagName('xyz')[0].firstChild*
- To access deeper levels we need more overhead.
- Reduced extensibility in XML

JSON and Java

How to parse JSON in Java

Mapping between JSON and Java entities

JSON	Java
string	java.lang.String
number	java.lang.Number
true|false	java.lang.Boolean
null	null
array	java.util.List
object	java.util.Map

JSON.simple maps entities from the left side to the right side while decoding or parsing, and maps entities from the right to the left while encoding.
On decoding, the default concrete class of *java.util.List* is*org.json.simple.JSONArray* and the default concrete class of *java.util.Map* is*org.json.simple.JSONObject*.

JSON reading in Java example

- http://www.tutorialspoint.com/json/json_java_example.htm

There is an interface called JSONParser in javax.json.stream you can implement

- javax.json.stream
- **Interface JsonParser**

Simple Example

Simple Demo

- Build a JSON data object in code
- Display raw output
- Display formatted output
- Manipulate via form input

Resources

- Simple Demo on Github: https://github.com/jfox015/BIFC-Simple-JSON-Demo
- Another JSON Tutorial: http://iviewsource.com/codingtutorials/getting-started-with-javascript-object-notation-json-for-absolute-beginners/
- JSON.org: http://www.json.org/


THANK
YOU