REST Introduction

Sang Shin
"Code with Passion!"

Topics

- What is and Why REST?
- Principles of RESTful architecture
- REST is everywhere
- REST vs SOAP
- JAX-RS
- Tools
- SpringMVC REST APIs vs Spring REST/JAX-RS APIs

What is and Why REST?

The World Before REST

- Many different communication standards
  - RMI, SOAP, CORBA, DCOM, etc.
- From many different parties
  - Sun, Microsoft, IBM, etc
- Caused many problems
  - Bad interoperability
  - Hard to implement
  - Vendor 'lock-in'

What is and Why REST?

- REpresentational State Transfer
  - Introduced by Roy Fielding's in his doctoral thesis “Architectural Styles and the Design of Network-based Software Architecture”
- He tried to address the following questions
  - Why is the Web so prevalent and ubiquitous?
  - What makes Web scale?
  - How can I apply the architecture of the Web to the “applications”?
- He found the same set of architectural styles that make Web so successful can be applied to the development/deployment/usage of “applications” and he calls it REST

Benefits of REST: Deployment standpoint

- Scalable
- High performance
- Loosely coupled
- Fault-tolerant
- Secure
- Interoperable

Benefits of REST: Development standpoint

- Simple, intuitive, consistent
- Friendly to developer
- Programming language independent
- Most languages have REST support
- Explorable via HTTP tool

REST Architectural Principles

REST Architectural Principles

- Addressability
- Uniform interface
- Representation-oriented
- Stateless
- HATEOAS (Hypermedia As The Engine Of Application State)
- Cacheable

Addressability

- Every resource has a unique address in the form of a URI
- URI structure
  - http://host:port/path?query1=value1&query2=value2#fragment
  - https://host:port/path?query1=value1&qiery2=value2#fragment
- Characters allowed
  - a-z, A-Z, 0-9, ., -, *, _
  - Other characters get encoded (space to +, others to %xx)
- Each resource having a unique URI enables “resources linking” (HATEOAS)
  - These links can be embedded into the document
  - The links embedded in the document carry states

Uniform Interface

- HTTP has a fixed set of methods, each of which has specific purpose and pre-defined behavior
  - GET, PUT, DELETE, POST, HEAD, PATCH, OPTIONS
- Safety and Idempotency
  - Safe methods are HTTP methods that do not modify resources
  - An idempotent HTTP method is a HTTP method that can be called many times without different outcomes

Benefits of Uniform Interface

- Simplicity
  - There is no need for IDL-like contract (IDL in CORBA, WSDL in SOAP) that specifies what methods are available
- Easy accessibility
  - The client does not need any special library or stub (like in the case of SOAP) in order to access the service: all they need is HTTP client library
- Interoperability
  - Due to simple requirements, REST clients and REST services are highly interoperable (since there are a lot less moving parts)
- Scalability
  - You can take advantage of built-in caching capability of HTTP

CRUD Operations are Performed through HTTP method + URI

CRUD Operations	4 main HTTP methods	
	Verb	**Noun**
Create (Single)	POST	Collection URI
Read (Multiple)	GET	Collection URI
Read (Single)	GET	Entry URI
Update (Single)	PUT	Entry URI
Delete (Single)	DELETE	Entry URI

Representation-oriented

- Each service (or resource) is addressable through a specific URI and representations are exchanged between client and server
- With GET, you get current representation of the resource
- With PUT or POST, you pass a representation of a resource to the server so that the underlying resource state can change
- The representation has self-descriptive messages

Stateless Communication

- What does “stateless” mean?
  - It does NOT mean your RESTful application can't have a state – it simply means the server does not maintain client session data
  - If client session data needs to be maintained, it should be maintained by the client and transferred to the server with each request as needed
- Benefits of stateless communication
  - Scalability (because server does not maintain the client session)
  - Reliability (because there is no lost client session data in case of server crash)

HATEOAS

- Hypermedia As The Engine of Application State (HATEOAS)
- It enables document centric approach with embedding links (for other resources) within that document
  - In the same way the web pages we, humans, visited contains links
- Each document that is returned guides the client to the other resources
  - In the same way the web pages we, humans, visited gave us the links to click through

Cacheable

- Response messages from the service to its consumers are explicitly labeled as cacheable or non-cacheable
- The service, the consumer, or one of the intermediary middleware components can cache the response for reuse in later requests

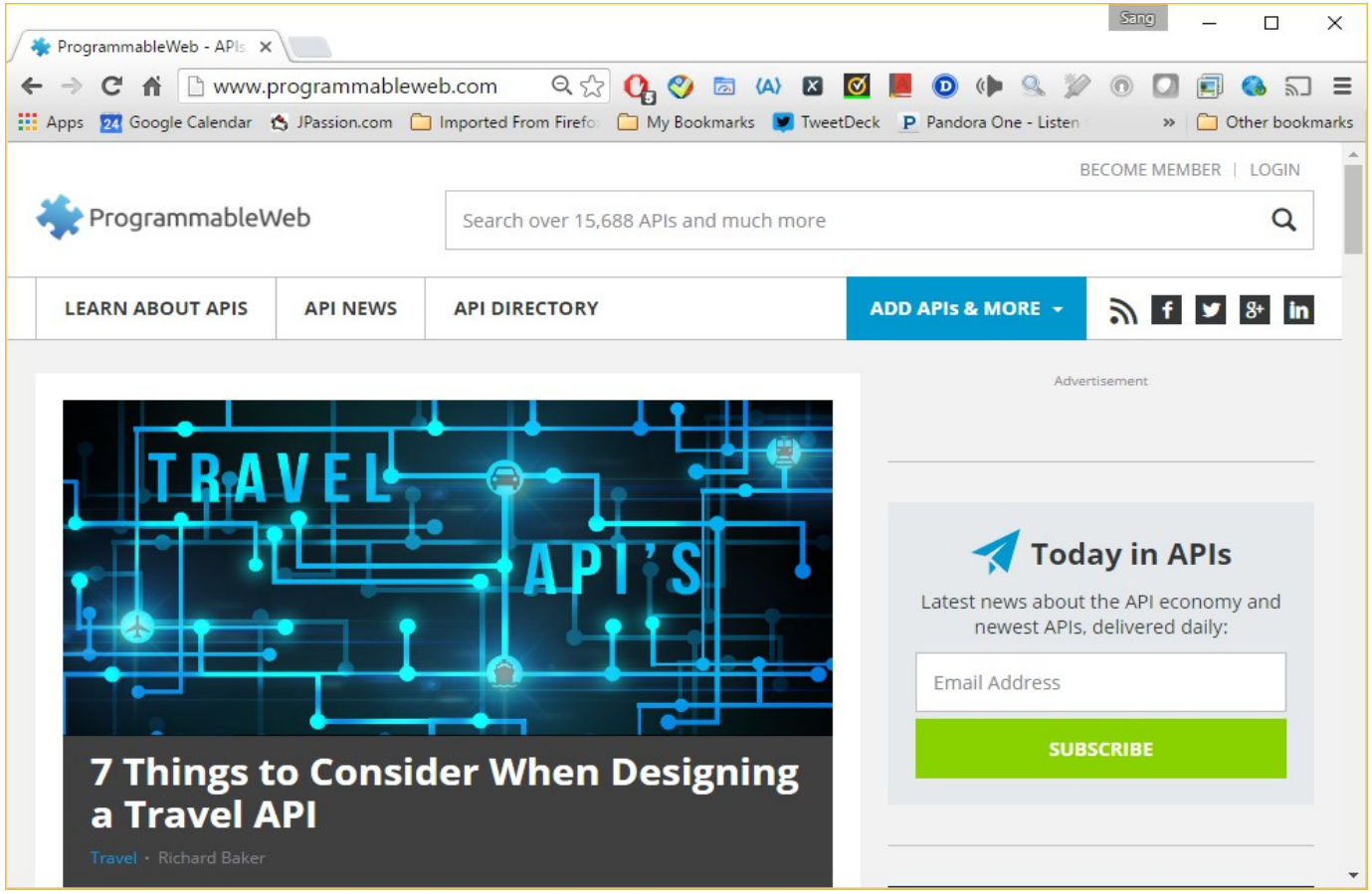
REST is Everywhere

REST is Everywhere

- Pretty much all services on the internet are exposed as RESTful services
  - Amazon
  - Google
  - Facebook
  - Tweeter
  - LinkedIn
  - ...

programmableweb.com

- Maintains all REST APIs on the net



Lab:

Exercise 2: Explore REST APIs
4361_javarest_introduction.zip

REST vs. SOAP

REST vs SOAP

- SOAP-based web service
  - Few URIs (nouns), many custom methods (verbs)
    - `musicPort.getRecordings("beatles")`
  - Uses HTTP as transport for SOAP messages
- RESTful web service
  - Many resources (nouns), few fixed methods(verbs)
    - `GET /music/artists/beatles/recordings`
  - HTTP is the protocol

SOAP Service and REST Resource

- SOAP based web services is about services
  - Stock quote service

  quoteService.purchase(“goog”, 2000);

- REST is Resource-Oriented Architecture
  - Stock quote resource
  - Resources are manipulated by exchanging representations
  - Eg. **purchasing** stock
    - Manipulate my portfolio resource
    - Handle a POST in a stock resource that I own
    - **`POST /mystocks/goog`**

Advantages of SOAP over REST

- Transport independence
  - You can use SOAP over any kind of transport (HTTP/S, JMS, SMTP) while REST works only over HTTP/S
- Well-defined standards
  - SOAP has well-defined standards in the area of security, transaction, reliability while REST lacks the standards in these areas
  - SOAP better suits with stateful operations
- Well-defined service contact via WSDL
  - REST now has WADL

Advantages of REST over SOAP

- Simplicity
- REST works with different representations (XML, JSON, XHTML, etc) while SOAP works only with XML (unless you use different encoding style)

JAX-RS
(Java Specification for REST)

Problem in Using Servlet API For Exposing a Resource (Too much coding)

```
public class Artist extends HttpServlet {

    public enum SupportedOutputFormat {XML, JSON};

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String accept = request.getHeader("accept").toLowerCase();
        String acceptableTypes[] = accept.split(",");
        SupportedOutputFormat outputType = null;
        for (String acceptableType: acceptableTypes) {
            if (acceptableType.contains("*/*") || acceptableType.contains("application/*") ||
                acceptableType.contains("application/xml")) {
                outputType=SupportedOutputFormat.XML;
                break;
            } else if (acceptableType.contains("application/json")) {
                outputType=SupportedOutputFormat.JSON;
                break;
            }
        }
        if (outputType==null)
            response.sendError(415);
        String path = request.getPathInfo();
        String pathSegments[] = path.split("/");
        String artist = pathSegments[1];
        if (pathSegments.length < 2 && pathSegments.length > 3)
            response.sendError(404);
        else if (pathSegments.length == 3 && pathSegments[2].equals("recordings")) {
            if (outputType == SupportedOutputFormat.XML)
                writeRecordingsForArtistAsXml(response, artist);
            else
                writeRecordingsForArtistAsJson(response, artist);
        } else {
            if (outputType == SupportedOutputFormat.XML)
                writeArtistAsXml(response, artist);
            else
                writeArtistAsJson(response, artist);
        }
    }
    private void writeRecordingsForArtistAsXml(HttpServletResponse response, String artist) { ... }
    private void writeRecordingsForArtistAsJson(HttpServletResponse response, String artist) { ... }
    private void writeArtistAsXml(HttpServletResponse response, String artist) { ... }
    private void writeArtistAsJson(HttpServletResponse response, String artist) { ... }
}
```

Design Goals of JAX-RS: Java API for RESTful Web Services

- Support REST concepts
  - Everything is a resource
  - Every resource is address'able via URI
  - HTTP methods provides uniform interface
  - Representations (formats)
  - HATEOAS
- Support High level and Declarative programming model
  - Use @ annotation in POJOs
- Generate or hide the boilerplate code
  - No need to write boilerplate code for every app

Implementations of JAX-RS (JSR 311)

- Jersey – reference implementation of JAX-RS
  - Download it from http://jersey.dev.java.net
  - Comes with Glassfish, other Java EE 6+ servers

- Other open source implementations of JAX-RS
  - Apache CXF
  - JBoss RESTEasy
  - Restlet

Tools

Development Tools

- IDE – for general purpose RESTful Web service development
  - Eclipse, Intellij IDEA, NetBeans
- Client tools – for sending HTTP requests
  - “Postman” Chrome Application
  - RESTClient
  - Several command line tools
    - curl http://curl.haxx.se/
  - soapUI
- Browser

Lab:

Exercise 1: Tools
4361_javarest_introduction.zip

SpringMVC REST APIs vs Spring REST/JAX-RS APIs

Comparison

- JAX-RS is designed with REST in mind
  - SpringMVC REST is an extension to MVC with REST features
- JAX-RS is more feature rich than SpringMVC REST
  - More APIs are available
- JAX-RS more portable than SpringMVC REST
  - It is Java standard

You can build Spring REST application using either SpringMVC REST APIs or JAX-RS APIs

How to use JAX-RS APIs in Spring

- Add the following dependency

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jersey</artifactId>
</dependency>
```

- Add JerseyConfig.java

```
@Component
@ApplicationPath("/resources")
public class JerseyConfig extends ResourceConfig {

   public JerseyConfig() {
      register(CustomersResource.class);
      register(OrdersResource.class);
   }

}
```

Code with Passion!
JPassion.com