6 Restful Web Services: Overview

Objectives

After completing this lesson, you should be able to do the following:

- Describe RESTful architecture and how it can be applied to web services
- Design a RESTful web service and identify resources
- Navigate a RESTful web service by using hypermedia
- Select the correct HTTP method to use when duplicate requests must be avoided
- Identify web service result status by HTTP response code

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 1: Introduction to Web Services
- Lesson 2: Creating XML Documents
- Lesson 3: Processing XML with JAXB
- Lesson 4: SOAP Web Services Overview
- Lesson 5: Creating JAX-WS Clients

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- **Lesson 6: Exploring REST Services** — **You are here!**
- Lesson 7: Creating REST Clients
- Lesson 8: Bottom Up JAX Web Services
- Lesson 9: Top Down JAX Web Services
- Lesson 10: Implementing JAX RS Web Services

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 11: Web Service Error Handling
- Lesson 12: Java EE Security and Securing JAX WS

REST is a *"Representational State Transfer architectural style for distributed hypermedia systems".*

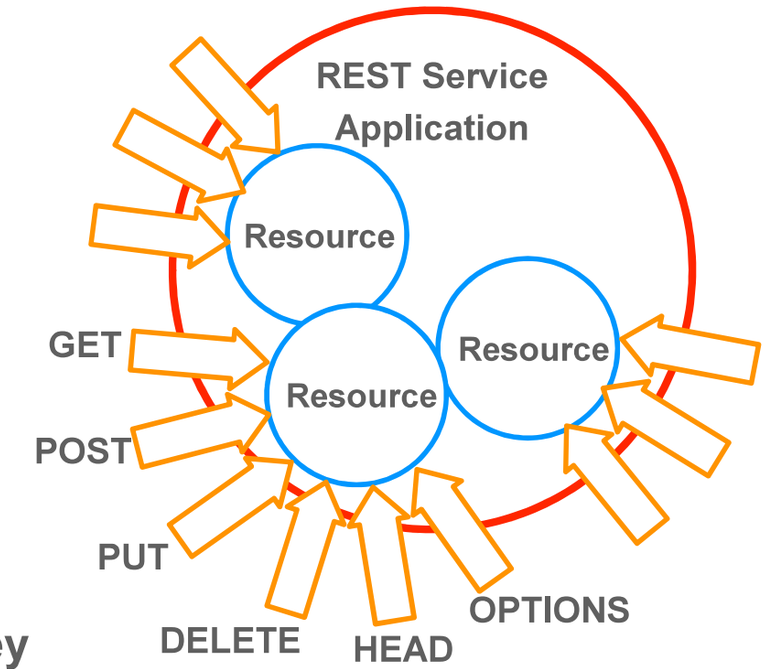
- RESTful web services represent a different approach to web services than SOAP-based services.
- REST is a style or architecture.
  - Developers have different styles.
  - Roy Fielding's doctoral dissertation describes REST.

http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

- The tools (JAX-RS) and best practices are still evolving.
- It is designed to limit system behavior to only what is required.

REST Service Conventions and Resources

- REST Service Application represents one or more Resource
- Each Resource represents a business entity and is mapped to its own URL
- Each Resource defines a number of operations mapped to HTTP Methods
- Typical conventional use of HTTP Methods:
  - GET receives (queries) a collection of elements or a specific element identified by client
  - POST creates new element
  - PUT updates existing element
  - DELETE removes an element
  - Other operations are sometimes used to retrieve metadata



❖ **Unlike SOAP, there is no standard for REST Services - they are considered to be a coding style rather than an actual protocol.**

REST Communication Model

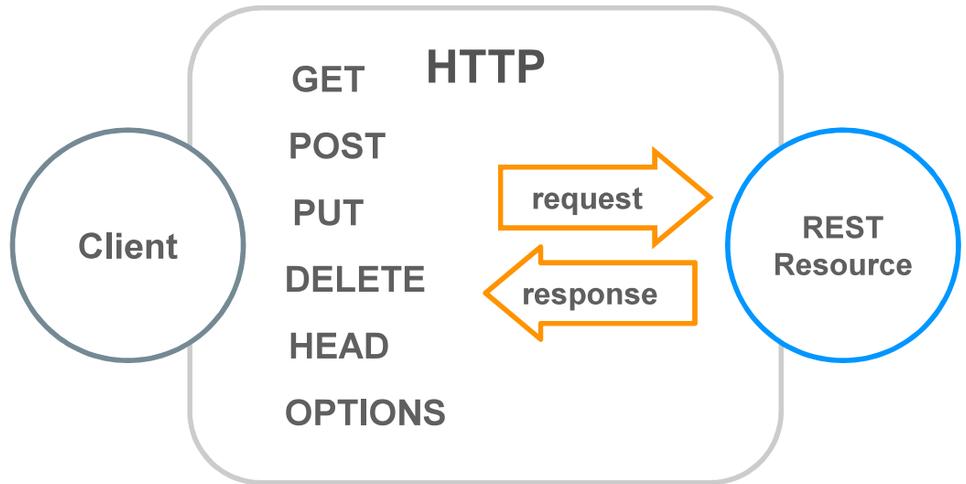
- REST Resources and REST Clients characteristics:
  - REST Clients are often implemented by using JavaScript in Browser or Mobile Applications.
  - Use HTTP as a transport layer and dispatch requests by using the GET, POST, PUT, DELETE, HEAD, and OPTIONS methods containing data that the client wants to pass to the REST Service.
  - Clients handle responses that contain HTTP status codes and may also contain a body with server-generated content.
  - Use of HTTP methods is conventional – REST services are not required to handle these in the same way.
  - REST Resources may use different media types such as:
    - JSON
    - Plain Text
    - XML
    - And so on

XML

```
<Product>
<id>101</id>
<name>Tea</name>
<price>1.99</price>
</Product>
```

JSON

```
{"Product": {
 "id": "101",
"name": "Tea",
"price": "1.99"
}}
```



REST Constraints

The properties of REST architecture are:

- Client-server based
- **Stateless communication:** Each request should contain all information needed to understand the request.
- **Uniform interface:** An analogy would be designing a single general-purpose Java interface and using it for every class you will ever create.
- **Layered System:** In a layered system (tiers), a client communicates only with an adjacent layer. A service should not expose any implementation details.
- Hypermedia based

REST Resources

REST is centered around an abstraction known as a “resource.” Any named piece of information can be a resource.

- A resource is identified by a URI.
- Clients request and submit representations of resources.
  - There may be many different representations available to represent the same resource.
- A representation consists of data and metadata.
  - Metadata often takes the form of HTTP headers.
- Resources are interconnected by hyperlinks.
- A RESTful web service is designed by identifying the resources.
  - This is similar to OOA&D where you identify nouns in use-cases.

Resource Design

A RESTful web service contains multiple resources.

- Resources are linked together (hypermedia).
- Design a tree of resources with a single base resource.
- It is comparable to an object graph in Java or hierarchy of elements in an XML document.
- Resources are (usually) nouns or things.
- Each resource has a limited number of general-purpose operations that it may support (`GET`, `PUT`, `POST`, `DELETE`).
- A resource is uniquely identified by a URL.

http://llocalhost:7001/myapp/resources/users

A collection is a resource.

http://llocalhost:7001/myapp/resources/users/matt

A specific resource in a collection.

Action Resources

Sometimes when designing noun-only resources, you may have a use-case that spans resources or does not map to a standard operation. An action resource can be used in this situation.

http://localhost/bank/accounts/matt

http://localhost/bank/accounts/tom

- Matt wants to give money to Tom. With two separate resources, you cannot atomically transfer money.

http://localhost/bank/transfers

- The verb in this use-case was transfer; by “nounifying” the verb, you can correctly identify the needed resource—a record of account transfers, which can be added to in order to create a new transfer.

Data Input

A web service needs input data to perform operations. Because HTTP is typically used by RESTful web services, there are several options for attaching input data to a request.

- **URL Path Parameter:** http://host/collection/{id} Use the URL to identify a resource or a collection of resources. Don't use http://host/collection/filterA/filterC/filterB.
- **Query Parameters:** http://host/collection?filter=A&filter=B Use query parameters to sort, filter, or limit collections.
- **Resource Representation (the HTTP body):** To update a resource client, download a representation, modify it, and send the modified version to the server that validates the changes.
- **Headers:** Use HTTP headers to add metadata or links to resources that do not support hypermedia (images).

Resource Representation Formats

A RESTful web service may support one or more representational formats. The `Accept` and `Content-Type` HTTP headers are used to negotiate and identify formats.

- **JSON:** A web browser (JavaScript) is commonly used as a web service client.
  - By returning a data format that is easily understood by JavaScript, you enable cleaner and more efficient applications.
  - JSON is not as verbose as XML, but can usually convey similar structure.
- **XML:** XML does have features that make it desirable including:
  - XML Schemas
  - Robust library support in programming libraries

HATEOAS Principle

Hypermedia as the Engine of Application State

- REST applications should provide at most one fixed URL. All application URLs, with related resources, should be “dynamically” discovered and navigated, starting from that fixed URL, with the help of hypermedia links throughout all resources.
- If you are asked to download Java from the oracle.com website, you may or may not know the download URL already. By reading web pages you can discover the download location. If you always begin at the base URL, it will not matter if the site is reorganized; you will still be able to discover the download location. The URL does not matter.

Hyperlinking

- Returning absolute URLs to clients instead of relative URLs makes it easier for clients to follow links.
- Use link headers to add non-structural links or enable hyperlinking in non-hypermedia.
- URLs matter. Some recommend that URLs should be fixed:
  - Cool URIs do not change. http://www.w3.org/Provider/Style/URI.html
- Client developers will probably save a URL that is seven links deep that they need instead of trying to discover it.
  - Try to make cool URLs that are discoverable.
  - Use link relations to enable the discovery of a link's purpose.

```
<link rel="stylesheet" href="example.css"/>
```

HTTP Methods

HTTP methods are the methods in your web service's API.

Method	Purpose
`GET`	Read, possibly cached
`POST`	Update or create without a known ID
`PUT`	Update or create with a known ID
`DELETE`	Remove
`HEAD`	Read headers, has version changed?
`OPTIONS`	List the "Allow"ed methods

HTTP Status Codes

- 1xx – Informational
  - Request received, continuing process
- 2xx – Success
  - Action successfully received, understood, and accepted
- 3xx – Redirection
  - Client must take additional action to complete the request.
- 4xx – Client Error
  - Request contains bad syntax or cannot be fulfilled.
- 5xx – Server Error
  - Server failed to fulfill an apparently valid request.

HTTP Status Codes :

2xx Status Codes

Code	Meaning	Description
200	OK	Successful HTTP request
201	Created	New resource created
202	Accepted	Accepted for processing, but not yet completed
204	No Content	Successfully processed, but no return content
206	Partial Content	Only part of the resource returned

HTTP Status Codes :

3xx Status Codes

Code	Meaning	Description
300	Multiple Choices	Multiple options for the resource that the client may follow
301	Moved Permanently	This and all future requests should be directed to the given URI.
303	See Other	Response can be found at URI using `GET` method.
304	Not Modified	Resource has not been modified since last requested.

HTTP Status Codes:

4xx Status Codes

Code	Meaning	Description
400	Bad Request	Request contains bad syntax or cannot be fulfilled.
401	Unauthorized	Request refused, when authentication is possible but has failed, or has not yet been provided.
403	Forbidden	Request was legal, but the server refuses to respond to it.
404	Not Found	Resource could not be found but may be available again in the future.

4xx Status Codes

Code	Meaning	Description
405	Method Not Allowed	Request made using method not supported by that resource.
406	Not Acceptable	Resource can only generate content not acceptable given `Accept` headers sent in.
409	Conflict	Request could not be processed due to conflict in the request.

HTTP Status Codes:

4xx Status Codes

Code	Meaning	Description
410	Gone	Resource no longer available and will not be available again.
412	Precondition Failed	Server does not meet precondition put on the request.
415	Unsupported Media Type	Request did not specify any media types that the resource supports.
417	Expectation Failed	Server cannot meet requirement of `Expect` header field.
418	I'm a teapot	Response entity "MAY be short and stout".

Idempotence

Some HTTP requests, when repeated, will have no effect whereas other methods will.

- `GET`: Read only and idempotent. Never changes the state of the resource.
- `PUT`: An idempotent insert or update of a resource. Idempotent because it is repeatable without side effects.
- `DELETE`: Resource removal and idempotent
- `POST`: Non-idempotent, “anything goes” operation

Resource Collections

It is very common to represent collections of resources; the collection itself is a resource.

- `GET /collection`: Returns a listing of hyperlinks to the elements in the collection
- `DELETE /collection`: Deletes everything
- `PUT /collection`: Replaces everything
- `POST /collection`: Creates one new thing in the collection, returns the generated ID of that new resource
- `PUT /collection/{id}`: Updates an item in the collection
- `PUT /collection/{id}`: If clients do not need the IDs generated, then you `PUT` to the item ID instead of `POST` to the collection to create a new item.

Given:

http://localhost/bank/accounts/matt

http://localhost/bank/accounts/tom

http://localhost/bank/transfers

How do you transfer money and not duplicate the transfer when there is an error?

`ID = POST` http://localhost/bank/transfers until you get the ID of a blank transfer resource.

`PUT` http://localhost/bank/transfers/{ID} to update the transfer with amounts until 200 status.

RESTful Web Service Documentation

Developers must understand a web service's API to use it.

- **Single URL:** The service fully embraces hypermedia. Starting at the base URL, all other URLs can be discovered. By using the `OPTIONS` HTTP method, all methods for each resource can be determined.
- **Human-generated developer docs:** Typically as a web page, the docs list the available URLs and provide example requests.
- **Machine-generated developer docs:** A tool either discovers all URLs by using a root URL and HTTP `OPTIONS` requests and then generates a human-readable list, or platform-specific code reads the resources (source code files) and generates documentation, possibly a WADL file.

Richardson Maturity Model

Leonard Richardson created a model of RESTful maturity that can be used to quickly describe architectural elements of a web service.

- **Level 0:** HTTP is used as a tunneling technology. A single method (`POST`) to a single URL is used to transfer XML documents that are tied to a collection of methods.
- **Level 1 (nouns):** Multiple endpoints (resources) with identifying path params
- **Level 2 (verbs):** Multiple endpoints and correct use of multiple HTTP methods
- **Level 3 (hypermedia):** HATEOAS. A single root resource can be used to discover all other resources.

Versioning

When you are ready to create version 2 of your web service, do you retire version 1?

- Private internal services are easier to retire, public services less so.
- No breaking changes: If possible, enhance your existing service by adding additional functionality (new URL or HTTP methods) but do not change anything that is in use by clients.
- Parallel versions: Each version has a different root resource URL. /v1 and /v2 are common.
- Keep the version 1 URLs, but use custom content types to switch to different behaviors.
- Use custom types: application/vnd.mycompany-v1+xml

Which HTTP status code range represents client errors?

a. 1xx
b. 2xx
c. 3xx
d. 4xx
e. 5xx

Which HTTP methods are idempotent?

a. GET
b. PUT
c. DELETE
d. POST

Resources

Topic	Website
Architectural Styles and the Design of Network-based Software Architectures	http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
REST APIs must be hypertext-driven	http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven
Hypertext Transfer Protocol -- HTTP/1.1	http://www.rfc-editor.org/rfc/rfc2616.txt
Richardson Maturity Model	http://martinfowler.com/articles/richardsonMaturityModel.html
Oracle Fusion Middleware Developing RESTful Web Services for Oracle WebLogic Server 12c	http://docs.oracle.com/cd/E24329_01/web.1211/e24983/toc.htm
Web Linking	http://www.rfc-editor.org/rfc/rfc5988.txt

Summary

In this lesson, you should have learned how to:

- Describe RESTful architecture and how it can be applied to web services
- Design a RESTful web service and identify resources
- Navigate a RESTful web service by using hypermedia
- Select the correct HTTP method to use when duplicate requests must be avoided
- Identify web service result status by HTTP response code
- Version RESTful web services

Practice 6 : Overview

This practice covers the following topics:

- Exploring REST Services
- Enabling RESTful Management Services for WebLogic
- Exploring WebLogic RESTful Management Services
- Updating Jersey (JAX-RS)
- Creating a Basic RESTful Web Service with JAX-RS
- Exploring a REST Service with cURL