Cloud Computing:
Rest based Web Services

Vijay Dialani, PhD
Boise State University
vijaydialani@boisestate.edu

What is REST?

1. REST (REpresentational State Transfer) is an architectural style for designing systems
2. REpresentational State Transfer was promoted in PhD by Roy Fielding (2001)
3. REST is not a protocol and is an alternative to WS-* and SOAP

Properties of RESTful Systems

1. Stateless
   - RESTful Systems communicate using stateless information
2. Addressable
   - Everything is a resource and is uniquely identifiable (URI)
   - Everything can be referenced
3. Limited action set
   - Every operation on URI can be accomplished by

     HTTP: GET, POST, PUT, DELETE, etc
   - Actions on the URI are idempotent

URI

Collections and operations can be represented as URI

http://estore.com/orders/all
http://estore.com:8080/orders?custId=8977

From a URI we know

The protocol (How do we communicate - http, https, ftp, smtp?)
The host and the port number (Where it is on network)
The resource path(What resources and operations are being supported?)

Type System

RESTful services do not explicitly define a type system, but use MIME types and XML to specify the structure of the object.

```
<?xml version="1.0" encoding="utf-8"?>
<Customer CustomerID="GREAL">
        <CompanyName>Great Lakes Food Market</CompanyName>
        <ContactName>Howard Snyder</ContactName>
        <ContactTitle>Marketing Manager</ContactTitle>
        <Phone>503-555-7555</Phone>
        <FullAddress>
            <Address>2732 Baker Blvd.</Address>
            <City>Eugene</City>
            <Region>OR</Region>
            <PostalCode>97403</PostalCode>
            <Country>USA</Country>
        </FullAddress>
</Customer>
```

Limited Actions

- How can we build applications with only 4+ methods?
  - SQL only has 4 operations: INSERT, UPDATE, SELECT, DELETE
  - Look at it, if you have a BLOB of data all you could do is:
    - Create a new BLOB
    - Add to an existing BLOB
    - Update a BLOB
    - Delete a BLOB
    - Search a BLOB
- A well-defined fixed and finite set of operations
  - Resources can only use these operations
  - Each operation has well-defined, explicit behavior
  - In HTTP land, these methods are GET, POST, PUT, DELETE

- TAKE THIS SOAP and WS-* !!! Scripting people love US not you "Type System People"

Implications of limited Action set

- Uniform and Predictable behavior across all resources
  - GET - readonly and idempotent.
  - PUT - an idempotent insert or update of a resource.
  - DELETE - resource removal and idempotent.
  - POST - non-idempotent, “anything goes” operation

  Key/Value store community loves this!!!

- Roll based Access Control is easily applicable

Designing in REST is Easy … Step 1: Define an Interface

```
public interface CustomerService {

    void createCustomer(Customer cust);
    void deleteCustomer(int custId);
    Customer[] getCustomers();
    Customer findCustomer(String phone);
}

public interface OrderService {

    void placeOrder(Order order);
    Order[] getAllOrders();
    void updateAnOrder(Order order);
    void cancelAnOrder(int orderId);
    Order[] getOrdersForCustomer(Customer customer);
}
```

Designing in REST is Easy … Step 2: Define URIs

Define the URIs for the resource

/customers
- GET - list all customers
- POST - create a new customer

/customers/{cust-id}
- GET - get a customer representation
- DELETE- remove a customer

/customers/find/{cust-id}
- GET – find a customer with id

/orders
- GET - list all orders
- POST - submit a new order

/orders/{order-id}
- GET - get an order representation
- PUT - update an order
- DELETE - cancel an order

/orders/customer/{cust-id}
- GET – get all orders for a customer

What makes REST attractive?

- It is the way that web sclaes. Its Stateless!!!
- Easily embeddable in online resources
- Isolates client from changes on the server, built in fault tolerance.
- A RESTFul application does not maintain sessions/conversations on the server
- Doesn't mean an application can't have state, it is held at the client and is transferred as a part of each request.

JAX-RS

@Path

Defines URI mappings and templates

@ProduceMime, @ConsumeMime

What MIME types does the resource produce and consume

@GET, @POST, @DELETE, @PUT, @HEADER

Identifies which HTTP method the Java method is interested in

@PathParam

Allows you to extract URI parameters/named URI template segments

@QueryParam

Access to specific parameter URI query string

@HeaderParam

Access to a specific HTTP Header

@CookieParam

Access to a specific cookie value

@MatrixParam

Access to a specific matrix parameter

Above annotations can automatically map HTTP request values to

String and primitive types

Class types that have a constructor that takes a String parameter

Class types that have a static valueOf(String val) method

List or Arrays of above types when there are multiple values

@Context

Access to contextual information like the incoming UR

JAX-RS

```
@Path(“/orders”)
public class OrderService {

    @Path(“/{order-id}”)
    @GET
    @ProduceMime(“application/xml”)
    String getOrder(@PathParam(“order-id”) int id) {
     …
    }
}
```

JAX-RS

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @ProduceMime("application/xml")
    String getOrder(@PathParam("order-id") int id) {
     …
    }
}
```

Base URI path to resource

JAX-RS

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @ProduceMime("application/xml")
    String getOrder(@PathParam("order-id") int id) {
     …
    }
}
```

Extension to base URI
that getOrder() method maps to

JAX-RS

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @ProduceMime("application/xml")
    String getOrder(@PathParam("order-id") int id) {
     …
    }
}
```

Defines a URI path segment parameter

JAX-RS

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @ProduceMime("application/xml")
    String getOrder(@PathParam("order-id") int id) {
     …
    }
}
```

HTTP method Java getOrder() maps to

JAX-RS

What's the CONTENT-TYPE returned?

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @ProduceMime("application/xml")
    String getOrder(@PathParam("order-id") int id) {
      …
    }
}
```

JAX-RS

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @ProduceMime("application/xml")
    String getOrder(@PathParam("order-id") int id) {
     …
    }
}
```

Inject value of URI segment into the *id* Java parameter

JAX-RS

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @ProduceMime("application/xml")
    String getOrder(@PathParam("order-id") int id) {
     …
    }
}
```

Automatically convert URI string segment into an integer

JAX-RS

```
@Path("/orders")
public class OrderService {

    @POST
    @ConsumeMime("application/xml")
    void submitOrder(String orderXml) {
      …
    }
}
```

What CONTENT-TYPE is this method expecting from client?

JAX-RS

```
@Path("/orders")
public class OrderService {

    @POST
    @ConsumeMime("application/xml")
    void submitOrder(String orderXml) {
      …
    }
}
```

Un-annotated parameters assumed to be incoming message body.
There can be only one!

References:

Book:

RESTFUL Java with JAX-RS 2.0, Bill Bruke

Special Thanks to Bill Bruke's for the JAX-RS examples