REST

Pushpendra Singh

Disclaimer: these slides have been prepared by using contents from resources mentioned in the reference slide

RESTful Web Services

- The three questions [Roy Fielding]
  - *Why is the Web so prevalent and ubiquitous?*
  - *What makes the Web scale?*
  - *How can I apply the architecture of the Web to my own applications?*

Source: Burke, Bill (2013-11-12). RESTful Java with JAX-RS 2.0. O'Reilly Media.

RESTful Web Services

- Addressable resources
  - *The key abstraction of information and data in REST is a resource, and each resource must be addressable via a URI (Uniform Resource Identifier).*
- A uniform, constrained interface
  - *Use a small set of well-defined methods to manipulate your resources.*

Source: Burke, Bill (2013-11-12). RESTful Java with JAX-RS 2.0. O'Reilly Media. Kindle Edition.

RESTful Web Services

- Representation-oriented
  - *You interact with services using representations of that service.*
  - *A resource referenced by one URI can have different formats.*
    - Different platforms need different formats. For example, browsers need HTML, JavaScript needs JSON (JavaScript Object Notation), and a Java application may need XML.
- Communicate statelessly
  - *Stateless applications are easier to scale.*
- Hypermedia As The Engine Of Application State (HATEOAS)
  - *Let your data formats drive state transitions in your applications.*

Source: Burke, Bill (2013-11-12). RESTful Java with JAX-RS 2.0. O'Reilly Media. Kindle Edition.

Addressability

- Every object and resource in your system is reachable through a unique identifier
- Addressability is the prime requirement
  - *SOA requires that disparate applications must integrate and interact.*

- Challenge: Standardized object identity isn't available in many environments.

Source: Burke, Bill (2013-11-12). RESTful Java with JAX-RS 2.0. O'Reilly Media. Kindle Edition.

Addressability

- REST uses URIs (Universal Resource Identifier)
- Each HTTP request must contain the URI of the object
- scheme:// host:port/ path? queryString# fragment
  - *Scheme: the protocol*
  - *host: dns name or ip address*
  - *Path: delimited by /, gives path of the resource on the machine*
  - *Optional query string as name-value pairs separated by # and joined by &*
- E.g.
  - *http:// example.com/ customers? lastName = Burke& zipcode = 02115*
- URIs have their encoding scheme
  - *Space is represented by +*

Source: Burke, Bill (2013-11-12). RESTful Java with JAX-RS 2.0. O'Reilly Media. Kindle Edition.

The Uniform, Constrained Interface

- How does having a constrained interface help?
- Use only the methods of HTTP for your web services

GET

- Read-only operation
- Idempotent and safe
  - *Idempotent means that no matter how many times you apply the operation, the result is always the same.*
  - *Safe meants that invoking a GET does not change the state of the server at all.*

PUT

- PUT requests that the server store the message body sent with the request under the location provided in the HTTP message.
  - *Usually modeled as an insert or update.*
  - *Creates a resource at the URI*
- URI in a PUT request identifies the entity enclosed with the request -- the user agent knows what URI is intended and the server MUST NOT attempt to apply the request to some other resource.
  - Server may send a 301 (Moved Permanently) response
- Idempotent?
- Safe?

DELETE

- DELETE is used to remove resources.
- Idempotent?
- Safe?

POST

- POST sends data to a specific URI and expects the resource at that URI to handle the request.
- The web server at this point can determine what to do with the data in the context of the specified resource.
- The URI in a POST request identifies the resource that will handle the enclosed entity. That resource might be a data-accepting process, a gateway to some other protocol, or a separate entity that accepts annotations.
- Idempotent?
- Safe?

Others

- HEAD: is exactly like GET except that instead of returning a response body, it returns only a response code and any headers associated with the request.
- OPTIONS: is used to request information about the communication options of the resource you are interested in. It allows the client to determine the capabilities of a server and a resource without triggering any resource action or retrieval.

The Uniform, Constrained Interface

- How does having a constrained interface help?
- Familiarity
- Interoperability
- Scalability

Representation-Oriented

- Services should be representation-oriented.
  - *Each service is addressable through a specific URI and representations are exchanged between the client and service.*
  - *With a GET operation, you are receiving a representation of the current state of that resource.*
  - *A PUT or POST passes a representation of the resource to the server so that the underlying resource's state can change.*
- Representations define the complexity of the client-server.
  - *These representations could be XML, JSON, or any format you can come up with.*

Representation-Oriented

- HTTP uses the Content-Type header to tell the client or server what data format it is receiving.
  - *type/ subtype; name = value; name = value…*
    - text/ plain
    - text/ html; charset = iso-8859-1
- Accept header allows a client to list its preferred response formats.
  - *Ajax can ask for JSON; Java client for XML*

- Burke, Bill (2013-11-12). RESTful Java with JAX-RS 2.0 (Kindle Locations 403-409). O'Reilly Media. Kindle Edition.

Communicate Statelessly

- Stateless means that there is no client session data stored on the server.
- The server only records and manages the state of the resources it exposes.
- If there needs to be session-specific data, it should be held and maintained by the client and transferred to the server with each request as needed.

HATEOAS

- Hypermedia As The Engine Of Application State
- A document-centric approach with added support for embedding links to other services and information within that document format.
- Transitioning links between resources results in change of the state of an application
- On each interaction, the service and consumer exchange representations of resource state, not application state.
  - *A transferred representation includes links that reflect the state of the application*
  - *The links advertise the application state transitions, but the application state isn’t recorded explicitly in the representation received by the consumer*

HATEOS Example

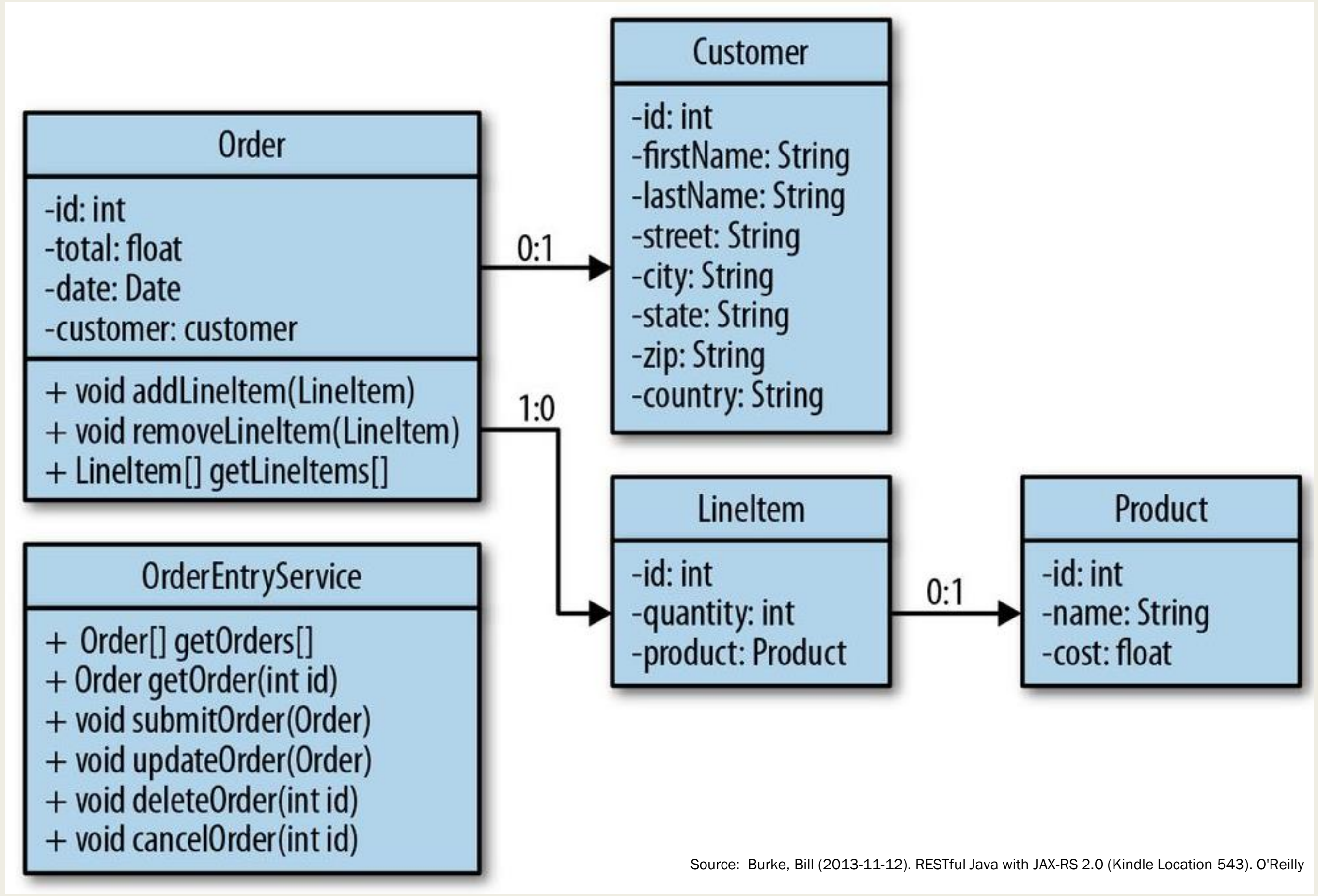
```
<order xmlns=”http://schemas.restbucks.com”>
        <location>takeAway</location>
        <item>
                <name>latte</name>
                <quantity>1</quantity>
                <milk>whole</milk>
                <size>small</size>
        </item>
        <status>pending</status>
</order>
```

```
<order xmlns=”http://schemas.restbucks.com”>
        <location>takeAway</location>
        <item>
                <name>latte</name>
                <quantity>1</quantity>
                <milk>whole</milk>
                <size>small</size>
        </item>
        <cost>2.0</cost>
        <status>payment-expected</status>
        <payment>https://restbucks.com/payment/1234</payment>
</order>
```

A RESTFUL SERVICE

Disclaimer: these slides have been prepared by using contents from resources mentioned in the reference slide

The Object Model



Source: Burke, Bill (2013-11-12). RESTful Java with JAX-RS 2.0 (Kindle Location 543). O'Reilly

Each order in the system represents a single transaction or purchase and is associated with a particular customer.

Orders are made up of one or more line items.

Line items represent the type and number of each product purchased.

Disclaimer: these slides have been prepared by using contents from resources mentioned in the reference slide

Model the URIs

- /orders
- /orders/{ id}
- /products
- /products/{ id}
- /customers
- /customers/{ id}

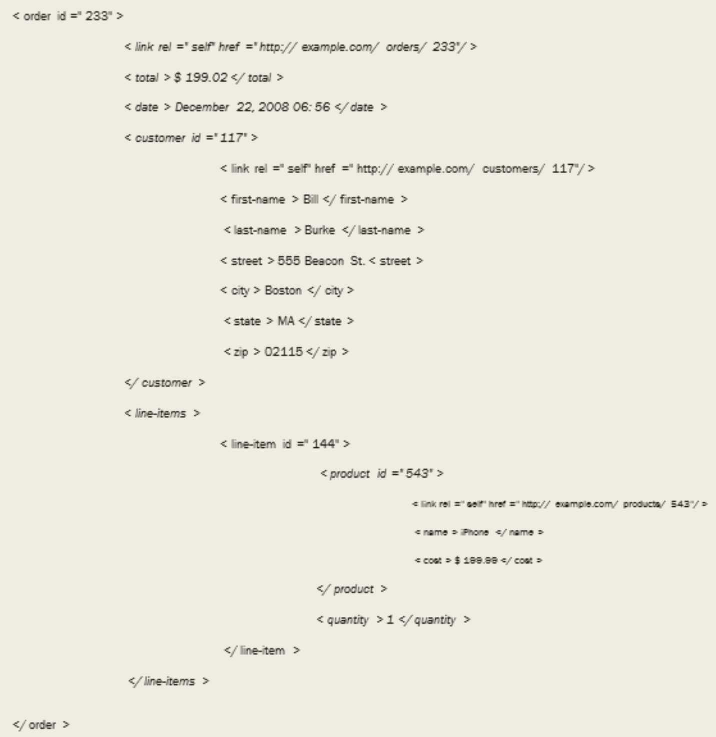
- URIs should not identify operations

Data Format

```
< customer id =" 117" >
        < link rel =" self" href =" http:// example.com/ customers/ 117"/ >
        < first-name > Bill </ first-name >
        < last-name > Burke </ last-name >
        < street > 555 Beacon St. < street >
        < city > Boston </ city >
        < state > MA </ state >
         < zip > 02115 </ zip >
</ customer >
```

Data Format

```
< order id =" 233" >
        < link rel =" self" href =" http:// example.com/
        orders/ 233"/ >
        < total > $ 199.02 </ total >
        < date > December 22, 2008 06: 56 </ date >
        < customer id =" 117" >
              < link rel =" self" href =" http:// example.com/
              customers/ 117"/ >
              < first-name > Bill </ first-name >
               < last-name > Burke </ last-name >
              < street > 555 Beacon St. < street >
              < city > Boston </ city >
               < state > MA </ state >
               < zip > 02115 </ zip >
</ order >
```



```
< order id =" 233" >
        < link rel =" self" href =" http:// example.com/ orders/ 233"/ >
        < total > $ 199.02 </ total >
        < date > December 22, 2008 06: 56 </ date >
        < customer id =" 117" >
              < link rel =" self" href =" http:// example.com/ customers/ 117"/ >
              < first-name > Bill </ first-name >
               < last-name > Burke </ last-name >
              < street > 555 Beacon St. < street >
              < city > Boston </ city >
               < state > MA </ state >
               < zip > 02115 </ zip >
        </ customer >
        < line-items >
              < line-item id =" 144" >
                     < product id =" 543" >
                           < link rel =" self" href =" http:// example.com/ products/ 543"/ >
                           < name > iPhone </ name >
                           < cost > $ 199.99 </ cost >
                     </ product >
                     < quantity > 1 </ quantity >
              </ line-item >
        </ line-items >
</ order >
```

Disclaimer: these slides have been prepared by using contents from resources mentioned in the reference slide

Browsing all orders, customers, or products

- GET /products HTTP/ 1.1

```
< products >
        < product id =" 111" >
        < link rel =" self" href =" http:// example.com/ products/ 111"/ >
         < name > iPhone </ name >
        < cost > $ 199.99 </ cost >
         </ product >
        < product id =" 222" >
        ...
        </ product >
        ...
< products >
```

Browsing all orders, customers, or products

- GET /orders? startIndex = 0& size = 5 HTTP/ 1.1
- GET /products? startIndex = 0& size = 5 HTTP/ 1.1
- GET /customers? startIndex = 0& size = 5 HTTP/ 1.1

Obtaining Individual Orders, Customers, or Products

- GET /orders/ 233 HTTP/ 1.1

```
HTTP/ 1.1 200 OK
Content-Type: application/ xml
< order id =" 233" >... </ order >
```

Creating Order

- PUT /orders/ 233 HTTP/ 1.1
- PUT /customers/ 112 HTTP/ 1.1
- PUT /products/ 664 HTTP/ 1.1

Creating Order

```
POST /orders HTTP/ 1.1
Content-Type: application/ xml
< order >
        < total > $ 199.02 </ total >
        < date > December 22, 2008 06: 56 </ date >
        ...
</ order >
```

Order Response

```
HTTP/ 1.1 201 Created
Content-Type: application/ xml
Location: http:// example.com/ orders/ 233
< order id =" 233" >
        < link rel =" self" href =" http:// example.com/ orders/ 233"/ >
        < total > $ 199.02 </ total >
        < date > December 22, 2008 06: 56 </ date >
         ...
</ order > HTTP requires
```

Updating Order, Customer, Product

```
PUT /orders/ 233 HTTP/ 1.1
Content-Type: application/ xml
< product id =" 111" >
        < name > iPhone </ name >
        < cost > $ 149.99 </ cost >
</ product >
```

When a resource is updated with PUT, the HTTP specification requires that you send a response code of 200, “OK,” and a response message body or a response code of 204, “No Content,” without any response body.

Removing an Order, Customer, Product

- The client simply invokes the DELETE method on the exact URI that represents the object we want to remove.
- When a resource is removed with DELETE, the HTTP specification requires that you send a response code of 200, “OK,” and a response message body or a response code of 204, “No Content,” without any response body.

State versus Operations

Disclaimer: these slides have been prepared by using contents from resources mentioned in the reference slide

References

- Burke, Bill (2013-11-12). RESTful Java with JAX-RS 2.0. O'Reilly Media
- Rest in Practice: Hypermedia and Systems Architecture
  - *Publisher: Shroff/O'Reilly*
- RESTful Java with JAX-RS 2.0
  - *Publisher: O'Reilly*
- Developing RESTful Services with JAX-RS 2.0, WebSockets, and JSON
  - *Publisher: PACKT*
- Building a RESTful Web Service with Spring
  - *Publisher: PACKT*
- WebSocket
  - *Publisher: O'Reilly*