REST support implemented with Jersey

**1.1. What is REST?**

REST is an architectural style which is based on web-standards and the HTTP (stateless) protocol. REST was first described by Roy Fielding in 2000.

In a REST based architecture everything is a resource. A resource is accessed via a common interface based on the HTTP standard methods.

In a REST based architecture you typically have a REST server which provides access to the resources and a REST client which accesses and modifies the REST resources.

Every resource should support the HTTP common operations. Resources are identified by global IDs (which are typically URIs).

REST allows that resources have different representations, e.g., text, XML, JSON etc. The REST client can ask for a specific representation via the HTTP protocol (content negotiation).

<http://sle-ccalg-001.corp.ebay.com:8080/OnCallLogApp>

http://sle-ccalg-001.corp.ebay.com:8080/OnCallLogApp/onCallLog/v1/onCallLogs/priorityCache

<display-name>OnCallLog</display-name>

<servlet>

<servlet-name>OnCall REST Service</servlet-name>

<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>com.ebay.oncall.rest</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>OnCall REST Service</servlet-name>

<url-pattern>/onCallLog/\*</url-pattern>

</servlet-mapping>

The following principles encourage ***RESTful***applications to be simple, lightweight, and fast:

* **Resource identification through URI**: A ***RESTful***web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs, which provide a global addressing space for resource and service discovery. See The ***@Path***Annotation and URI Path Templates for more information.
* **Uniform interface**: Resources are manipulated using a fixed set of four **create, read, update, delete** operations: **PUT,GET, POST, and DELETE**. **PUT creates a new resource**, which can be then **deleted by using DELETE**. **GET retrieves the current state of a resource in some representation**. **POST transfers a new state onto a resource**. See Responding to HTTP Methods and Requests for more information.
* **Self-descriptive messages**: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as **HTML, XML, plain text, PDF, JPEG, JSON, and others**. Metadata about the resource is available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control. See Responding to HTTP Methods and Requests and Using Entity Providers to Map HTTP Response and Request Entity Bodies for more information.
* **Stateful interactions through hyperlinks**: Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction. See Using Entity Providers to Map HTTP Response and Request Entity Bodies and “Building URIs” in the JAX-RS Overview document for more information.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Resource** | **GET** | **PUT** | **POST** | **DELETE** |
| **Collection URI, such ashttp://example.com/resources/** | **List** the URIs and perhaps other details of the collection's members. | **Replace** the entire collection with another collection. | **Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation. | **Delete**the entire collection. |
| **Element URI, such ashttp://example.com/resources/item17** | **Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type. | **Replace** the addressed member of the collection, or if it doesn't exist,**create** it. | Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it. | **Delete**the addressed member of the collection |

**Creating a *RESTful*Root Resource Class**  
**Root resource classes** are ***POJOs***that are either annotated with***@Path*** or have at least one method annotated with ***@Path*** or a**request method designator**, such as ***@GET, @PUT, @POST, or @DELETE***. **Resource methods** are methods of a resource class annotated with a request method designator. This section explains how to use ***JAX-RS*** to annotate Java classes to create***RESTful***web services.

**Developing *RESTful*Web Services with *JAX-RS***  
**JAX-RS** is a Java programming language API designed to make it easy to develop applications that use the REST architecture.  
The**JAX-RS API** uses Java programming language annotations to simplify the development of ***RESTful*** web services. Developers decorate Java programming language class files with **JAX-RS** annotations to define resources and the actions that can be performed on those resources. JAX-RS annotations are runtime annotations; therefore, runtime reflection will generate the helper classes and artifacts for the resource. A Java EE application archive containing **JAX-RS**resource classes will have the resources configured, the helper classes and artifacts generated, and the resource exposed to clients by deploying the archive to a Java EE server.

|  |  |
| --- | --- |
| **Annotation** | **Description** |
| @Path | The @Path annotation’s value is a relative URI path indicating where the Java class will be hosted: for example, /helloworld. You can also embed variables in the URIs to make a URI path template. For example, you could ask for the name of a user and pass it to the application as a variable in the URI: /helloworld/{username}. |
| @GET | The @GET annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP GET requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @POST | The @POST annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP POST requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @PUT | The @PUT annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP PUT requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @DELETE | The @DELETE annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP DELETE requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @HEAD | The @HEAD annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP HEAD requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @PathParam | The @PathParam annotation is a type of parameter that you can extract for use in your resource class. URI path parameters are extracted from the request URI, and the parameter names correspond to the URI path template variable names specified in the @Path class-level annotation. |
| @QueryParam | The @QueryParam annotation is a type of parameter that you can extract for use in your resource class. Query parameters are extracted from the request URI query parameters. |
| @Consumes | The @Consumes annotation is used to specify the MIME media types of representations a resource can consume that were sent by the client. |
| @Produces | The @Produces annotation is used to specify the MIME media types of representations a resource can produce and send back to the client: for example, "text/plain". |
| @Provider | The @Provider annotation is used for anything that is of interest to the JAX-RS runtime, such as MessageBodyReader and MessageBodyWriter. For HTTP requests, theMessageBodyReader is used to map an HTTP request entity body to method parameters. On the response side, a return value is mapped to an HTTP response entity body by using a MessageBodyWriter. If the application needs to supply additional metadata, such as HTTP headers or a different status code, a method can return a Response that wraps the entity and that can be built using Response.ResponseBuilder. |

