Ojp-howto-custom-jpa-processor

How to write Custom OData JPA Processor

Overview

OData JPA processor provides a way to transform an existing JPA model as EDM with no or minimalistic coding. It also processes the OData request and generates OData response. However at times it is required for an application to perform some pre-processing of request and post-processing of response. To enable pre and post processing in the application following steps needs to be performed.

```
The feature is supported from Apache Olingo release 1.1.0 onwards.
```

Step 1: Write a Custom OData JPA Processor

You could write a custom OData JPA processor by extending the class org.apache.olingo.odata2.jpa.processor.api.ODataJPAProcessor.

```
public class CustomODataJPAProcessor extends ODataJPAProcessor{

  @Override
  public ODataResponse readEntitySet(final GetEntitySetUriInfo uriParserResultView, final String contentType)
      throws ODataException {

    /* Pre Process Step */
    preprocess ( );

    List<Object> jpaEntities = jpaProcessor.process(uriParserResultView);

    /* Post Process Step */
    postProcess( );

    ODataResponse oDataResponse =
        responseBuilder.build(uriParserResultView, jpaEntities, contentType);

    return oDataResponse;
  }

}
```

In the above code snippet preprocess and postProcess are the two private methods that can be written to process the request and response. The instance variable (part of ODataJPAProcessor) **jpaProcessor** can be used to process the OData request. The jpaProcessor returns the JPA entities after processing the OData request. The instance variable **responseBuilder** can be used for building the OData response from the processed JPA entities.

Step 2: Write a Custom OData JPA Service Factory

As a second step implement an OData JPA service factory to create an OData service with custom OData JPA processor. The default service factory **org.apache.olingo.odata2.jpa.processor.api.ODataJPAServiceFactory** part of the library cannot be used. Hence create a class by extending **org.apache.olingo.odata2.api.ODataServiceFactory**. Copy the entire code from ODataJPAServiceFactory and just replace the code as shown below.

```
ODataSingleProcessor odataJPAProcessor = accessFactory.createODataProcessor(oDataJPAContext);

with

ODataSingleProcessor odataJPAProcessor = new CustomODataJPAProcessor(oDataJPAContext);
```

With the above two steps a custom OData JPA processor can be hooked to the existing flow.