Ojp-handling-blob-proposal

Handling Blob and Clob data types

Overview

JPA Entities can have properties that are typed as **java.sql.Blob** or **java.sql.Clob**. Internally JPA entities can instantiate a JPA provider (Eclipse Link or Hibernate or ...) specific implementation of the above two interfaces and bind them to the properties. Now to enable write on such properties using OData JPA processor an additional access modifier is required to be added to the JPA entities. Following is the proposal on how OData JPA processor handles java.sql.Blob and java.sql.Clob during meta data generation and run time processing.

EDM generation

Following is the pseudo code for generating the EDM when the JPA entity property is typed as **java.sql.Blob**

Step 1 - Check if JPA entity property is of type byte[] or (of type java.sql.Blob and annotated with @Lob annotation).

Step 2 - If Step 1 is true then generate an EDM property with type as Edm.Binary

In case of **java.sql.Clob**
Step 1 - Check if JPA entity property is of type java.sql.Clob and annotated with @Lob annotation.

Step 2 - If Step 1 is true then generate an EDM property with type as Edm.String with no max length unless a max length is specified.

Run time Processing

Following is the pseudo code for handling the **java.sql.Blob** and **java.sql.Clob** during run time

Before discussing the pseudo code, it is mandatory for developers to implement the callback interface org.apache.olingo.odata2.jpa.processor.api.OnJPAWriteContent. The implemented interface needs to be registered with the service via the method **setOnJPAWriteContent** method part of **ODataJPAServiceFactory**.

Example

```
/* Callback Implementation */

public class OnDBWriteContent implements OnJPAWriteContent {

  @Override
  public Blob getJPABlob(byte[] binaryData) throws ODataJPARuntimeException {
    try {
      return new JDBCBlob(binaryData);
    } catch (SerialException e) {
      ODataJPARuntimeException.throwException(ODataJPARuntimeException.INNER_EXCEPTION, e);
    } catch (SQLException e) {
      ODataJPARuntimeException.throwException(ODataJPARuntimeException.INNER_EXCEPTION, e);
    }
    return null;
  }

  @Override
  public Clob getJPAClob(char[] characterData) throws ODataJPARuntimeException {
    try {
      return new JDBCClob(new String(characterData));
    } catch (SQLException e) {
      ODataJPARuntimeException.throwException(ODataJPARuntimeException.INNER_EXCEPTION, e);
    }
    return null;
  }
}


/* Call Back registration in Service Factory */
public class JPAReferenceServiceFactory extends ODataJPAServiceFactory {

  public static final OnJPAWriteContent onDBWriteContent = new OnDBWriteContent();

  @Override
  public ODataJPAContext initializeODataJPAContext()
      throws ODataJPARuntimeException {

    ....
    ........
    ..............

    setOnWriteJPAContent(onDBWriteContent); //Register Call Back

    return oDataJPAContext;
  }
}
```

OData JPA processor internally does the following

Step 1 - Check if the JPA entity property is of type java.sql.Blob or java.sql.Clob.

Step 2 - If Step 1 is true then invoke the registered callback method getJPABlob or getJPAClob respectively

Step 3 - If callback method is not defined throw exception.

Step 3 is needed because the OData library is not bound to any specific implementation of Blob or Clob interface.