Hibernate Xdoclet Mapping transformation to JPA

Article *in* SSRN Electronic Journal · June 2019

CITATIONS
0

READS
289

2 authors, including:

Mustafa Shuaieb Sabri
Harrisburg University
498 PUBLICATIONS **461** CITATIONS

SEE PROFILE

International Journal of Management, IT & Engineering
Vol. 9 Issue 6, June 2019,
ISSN: 2249-0558 Impact Factor: 7.119
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gage as well as in Cabell's Directories of Publishing Opportunities, U.S.A

Hibernate Xdoclet Mapping transformation to JPA

Vijay Kumar Pandey*

Abstract: -Hibernateis an open source object relational mapping (ORM) framework for the java-based technologies. Hibernate was started in 2001 before Java language annotations (feature of Java 5) came into existence, this led to a lot of enterprise systems build using Xdoclet, whichprovided a way to specify metadata as comments for Hibernate ORM within the java entity classes. Ant (a build tool) was then used to convert these Xdoclet metadata to generate hibernate mapping files (hbm files) which were then processed by the Hibernate runtime engine. With the advent of Java annotations, Hibernate also created anannotations module to provide a way to use standard java annotations within the entity classes, making the Hibernate Xdoclet annotations obsolete. Based on the community feedback and popularity of ORM based engines, Java Persistence Architecture (JPA) was first released in 2006 to provide a standard way of utilizing ORM products and its annotations. Enterprise systems that might have thousands of entities which still utilize Xdoclet way of managing metadata, have a hard time of moving to JPA as manual transformation can be buggy and expensive and that's where an automated mechanism of transforming java source codebase to not only convert Xdoclet metadata mapping to standard JPA annotations but also move the proprietary use of Hibernate API's to JPA API's across the codebase.If Hibernate engine is used as the JPA provider, one can use Hibernate API where JPA doesn't provide that specific feature.

Keywords: -ORM, JPA, Xdoclet, Hibernate, Java, API, JDBC (Java Database Connectivity)

*Director of Technology Solutions, Intueor Consulting, Inc. Irvine CA – USA, pandey@intueor.com

I. INTRODUCTION

This paperprovides a detailed mechanism of how to automatically convert an Xdoclet based hibernate mapping to JPA 2.1 based annotations. One of the most important aspect of this transformation is to understand the major differences between Hibernate mapping based on Xdoclet and JPA annotations.The pictorial view below provides a side by side view of the differences between Xdoclet mapping (left side) and the JPA annotations (right side) covering areas such as mapping at the entity class, primary key property, simple property and a complex *OneToMany* type property.

Figure1

The entity mapping code below shows the mapping configuration generated through Ant task for the Xdoclet mapping as shown above. The Xdoclet mapping tags cannot be read during runtime by Hibernate engine, but that's where generated mapping files provide the mapping configuration for the entity. On the other hand, any JPA provider including Hibernate can read the JPA annotations, hence no extra entity mapping file is needed.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class table="AUDIT" name="research.AuditData">
      <id column="AUDIT_ID" name="auditId">
            <generator class="sequence">
              <param name="sequence">SEQ_AUDIT</param>
            </generator>
      </id>
    <property name="week" not-null="true" type="integer" column="WEEK"/>
      <set lazy="false" cascade="all" name="auditSet" inverse="true">
        <key column="AUDIT_ID"/>
        <one-to-many class="research.SampleData"/>
      </set>
  </class>
</hibernate-mapping>
```

Figure2

II. HIBERNATE INITIALIZATION - METADATA EXTRACTION

To automate the transformation of Hibernate based metadata generated through Xdoclet mapping, hibernate entity metadata needs to be extracted from the hibernate engine. This entity metadata can then be used to generate JPA based annotations. The code outlined below shows how to perform this task:

A. Hibernate Configuration File

The hibernate configuration file as shown below will be used for Hibernate to initialize its runtime.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
      <session-factory>
            <property name="hibernate.connection.driver_class">org.hsqldb.jdbcDriver</property>
            <property name="connection.driver.class">org.hsqldb.jdbcDriver</property>
          <property name="connection.url">jdbc:hsqldb:mem:hsql_mem_db_hibernate</property>
          <property name="connection.username">sa</property>
          <property name="connection.password"></property>

            <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

            <mapping resource="research/AuditData.hbm.xml"/>
            <mapping resource="research/SampleData.hbm.xml"/>
      </session-factory>
</hibernate-configuration>
```

Figure 3

B. Hibernate Runtime Initialization – Standalone Java Environment

The code below represents the Hibernate 4 way of initializing its runtime using the above file named as 'hibernate.cfg.xml'. This initialization of hibernate runtime will give access to some of the important hibernate runtime classes to read entity metadata such as *org.hibernate.cfg.Configuration*, *org.hibernate.SessionFactory* and *org.hibernate.internal.SessionFactoryImpl*

```
//Read the hibernate configuration file
InputStream hibCfgStream = Thread.currentThread().getContextClassLoader().getResourceAsStream("hibernate.cfg.xml");
String cfgAsString = org.apache.commons.io.IOUtils.toString(hibCfgStream, Charset.defaultCharset());

//Use the dom/sax parsers
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document hibernateConfigDoc = builder.parse(new InputSource(new StringReader(cfgAsString)));

//configure the Hibernate config object
org.hibernate.cfg.Configuration hibernateConfiguration = new Configuration();
hibernateConfiguration.configure(hibernateConfigDoc);

//initialize the hibernate runtime
ServiceRegistry serviceRegistry = new ServiceRegistryBuilder()
                    .applySettings(hibernateConfiguration.getProperties())
                    .buildServiceRegistry();

//build the hibernate runtime
org.hibernate.SessionFactory sessionFactory = hibernateConfiguration.buildSessionFactory(serviceRegistry);
org.hibernate.internal.SessionFactoryImpl sessionFactoryImpl = (SessionFactoryImpl)sessionFactory;
```

Figure 4

C. Hibernate Entity Metadata

Hibernate keeps the entity metadata as part of its class '*org.hibernate.metadata.ClassMetadata*'. The SessionFactory built in the above step provides a way to access the entity metadata.

```
Map<String, ClassMetadata> metadatas =sessionFactory.getAllClassMetadata();
ClassMetadata auditDataMD = metadatas.get("research.AuditData");
```

D. Parts of Entity Metadata

Once the entity metadata is accessed for a specific entity, it can be used to gain access to metadata for its various properties, identifier (primary key) property. The code below shows the mechanism of how to access these property level metadata.

```
//Basic functionality for persisting an entity via JDBC AbstractEntityPersister
persister = (AbstractEntityPersister)auditDataMD;

//entity mapping
PersistentClass persistentClass = hibernateConfiguration.getClassMapping(auditDataMD.getEntityName());

//Centralizes metamodel information about an entity
EntityMetamodel metaModel = persister.getEntityMetamodel();

//Represents a defined entity identifier (primary key) property within the Hibernate runtime-metamodel
IdentifierProperty identifier = persister.getEntityMetamodel().getIdentifierProperty();

//some of the important attributes related to entity mapping
String[] propertyNames = persister.getPropertyNames();

StandardProperty[] properties = persister.getEntityMetamodel().getProperties();

String[] columnNames = persister.getPropertyColumnNames(propertyIndex);
```

Figure 5

E. Generate JPA Identifier Annotations from Metadata

Once identifier property is accessed as shown above, its related metadata can be extracted and with that information JPA annotations can be formed. The algorithmwith pseudo code below shows this process.

```
//Get the identifier property from the entity metamodel
IdentifierProperty identifierProp = persister.getEntityMetamodel().getIdentifierProperty();

//Get the column names
String[] columnNames = persister.getKeyColumnNames();
String columnName = columnNames !=null && columnNames.length >0 ? columnNames[0] : null;

//form the column annotation
String columnAnntValue = "@Column(name = \"%s\" %s)";
StringBuilder sb = new StringBuilder();

//form the column length
if(identifierProp.getType() !=null && identifierProp.getType().getClass().getName().equals(StringType.class.getName())) {
        Property property = persistentClass.getProperty(identifierProp.getName());
        if(property !=null) {
                int length = ((Column) property.getColumnIterator().next()).getLength();
                if(length > 0 && length < Column.DEFAULT_LENGTH) {
                        sb.append(String.format(", length = %d",length));
                }
        }
}

//Default identifier annotation
String id = "@Id";

IdentifierGenerator idenGen = persister.getIdentifierGenerator();

String generatorAnnt = null;
String genValueAnnt = null;

//if sequence generator
if(idenGen !=null && idenGen instanceof SequenceGenerator) {
        SequenceGenerator seqGen = (SequenceGenerator)idenGen;
        String generatorFmt = "@SequenceGenerator(name = \"%s\", sequenceName = \"%s\", allocationSize = 1)";
        String genValueFmt = "@GeneratedValue(generator = \"%s\", strategy = GenerationType.SEQUENCE)";

        generatorAnnt = String.format(generatorFmt, seqGen.getSequenceName() + "_Gen",seqGen.getSequenceName());
        genValueAnnt = String.format(genValueFmt, seqGen.getSequenceName() + "_Gen");

//if identity generator
}else if (idenGen !=null && idenGen instanceof IdentityGenerator) {
        IdentityGenerator identityGen = (IdentityGenerator)idenGen;
        String genValueFmt = "@GeneratedValue(strategy=GenerationType.IDENTITY)";

//if foreign generator
}else if (idenGen !=null && idenGen instanceof ForeignGenerator) {
        ForeignGenerator foreignGen = (ForeignGenerator)idenGen;
        id = "@MapsId";
        //don't generate the column annotation - it should be generated as join column at the relationship level
        columnAnntValue = null;

//if id is assigned
  }else if (idenGen !=null && idenGen instanceof Assigned) {
         Assigned assigned = (Assigned)idenGen;

//if identifier is composite nested generated
}else if (idenGen !=null && idenGen instanceof CompositeNestedGeneratedValueGenerator) {
        CompositeNestedGeneratedValueGenerator compositeId = (CompositeNestedGeneratedValueGenerator)idenGen;
        id="@EmbeddedId";
        //only id is needed

        //don't generate the column annotation - it should be generated on the embedded class or use attribute override
        columnAnntValue = null;
}
```

Figure 6

F. Generate JPA Non-IdentifierProperty Annotations from Metadata

This section details how to generate JPA annotations of non-association standard property of an entity with the help of its metadata. Once the JPA annotations are formed, it could easily be added to the JPA based entities. The algorithm with the pseudo code describes how to achieve the formation of the annotations.

```
//Loop over the standard entity properties
StandardProperty[] properties = persister.getEntityMetamodel().getProperties();

//Note: Below algorithm is how to transform simple property and not association proerty
StandardProperty standardProp = properties[propertyIndex];

Type propertyType = standardProp.getType();

//Get the column names
String[] columnNames = persister.getKeyColumnNames();
String columnName = columnNames !=null && columnNames.length >0 ? columnNames[0] : null;

//form the column annotation
String columnAnntValue = "@Column(name = \"%s\" %s)";
StringBuilder sb = new StringBuilder();

//check if the column should be insertable or updateable
boolean updtInser = true;
if(persister instanceof SingleTableEntityPersister && persister.getRootEntityName() !=null &&
                persister.getRootEntityName().equals(persister.getEntityName()) &&
persister.getEntityMetamodel().getSubclassEntityNames().size() > 1) {
        String disColName = persister.getDiscriminatorColumnName();
        if(disColName !=null && disColName.equals(columnName)) {
                updtInser = false;
        }
}

//check if column is null
if(!standardProp.isNullable()) {
        sb.append(", nullable = false");
}

//check if column cannot be inserted
if(!standardProp.isInsertable() || !updtInser) {
        sb.append(", insertable = false");
}

//check if column cannot be updated
if(!standardProp.isUpdateable() || !updtInser) {
        sb.append(", updatable = false");
}

//form the column length for String type, others can be added
if(standardProp.getType() !=null && standardProp.getType().getClass().getName().equals(StringType.class.getName())) {
        Property property = persistentClass.getProperty(standardProp.getName());
        int length = ((Column) property.getColumnIterator().next()).getLength();
        if(length > 0 && length < Column.DEFAULT_LENGTH) {
                sb.append(String.format(", length = %d",length));
        }
}

//form the full JPA column annotation
String column = String.format(columnAnntValue, columnName, sb.toString());

//check if the property is lazy - then generate the Basic with Lazy annotation
if(standardProp.isLazy()) {
        String basicAnntValue = "@Basic(fetch = FetchType.LAZY)";
}

//Once the above annotations are formed - they can be added to the entity property
```

Figure 7

G. Generate JPA association OneToMany Property Annotations from Metadata

This section details how to generate JPA annotations of association property such as *OneToMany*of an entity with the help of its metadata. This algorithm is divided in two parts (Part 1 and Part 2) below.

```
Part 1:
//Loop over the standard properties
StandardProperty[] properties = persister.getEntityMetamodel().getProperties();

//Note: Below algorithm is how to transform OneToMany mapping to JPA annotations
StandardProperty standardProp = properties[propertyIndex];

//If the type is org.hibernate.type.SetType - Most probably its OneToMany association Type
propertyType = standardProp.getType();

SetType setType = (SetType)propertyType;

//find the associated collection metadata
org.hibernate.metadata.CollectionMetadata collMetaData = sessionFactory.getCollectionMetadata(setType.getRole());

//check if its ManyToMany association
Type elementType = collMetaData.getElementType();
boolean manyToMany = false;
BasicCollectionPersister colPersister = collMetaData instanceof BasicCollectionPersister ?
                        (BasicCollectionPersister) collMetaData : null;
if(colPersister !=null && colPersister.isManyToMany()) {
        manyToMany = true;
}

String manyTypeAnnt = manyToMany ? "@ManyToMany" : "@OneToMany";
StringBuilder oneToManySB = new StringBuilder();

//  if collection is lazy
if(!collMetaData.isLazy()) {
        if(oneToManySB.length() >0) {
                oneToManySB.append(',');
        }
        oneToManySB.append("fetch = FetchType.EAGER");
}

//if this side is the inverse
if(collPersister.isInverse()) {
        Type[] types = collPersister.getElementPersister().getPropertyTypes();
        int index = 0;
        for(Type type : types) {
                if(type.getClass().getName().equals(ManyToOneType.getFullyQType())) {
                        org.hibernate.type.ManyToOneType manyType = (org.hibernate.type.ManyToOneType) type;
                        if(manyType.getAssociatedEntityName().equals(persister.getEntityName())) {
                                if(oneToManySB.length() >0) {
                                        oneToManySB.append(',');
                                }
                                oneToManySB.append(String.format("mappedBy = \"%s\"",collPersister.
                                        getElementPersister().getPropertyNames()[index]));
                                foundMappedBy = true;
                                break;
                        }
                }
                index++;
        }
        if(!foundMappedBy) {
                index = 0;
                for(Type type : types) {
                        if(type.getClass().getName().equals(OneToOneType.getFullyQType())) {
                                org.hibernate.type.OneToOneType manyType = (org.hibernate.type.OneToOneType) type;
                                if(manyType.getAssociatedEntityName().equals(persister.getEntityName())) {
                                        if(oneToManySB.length() >0) {
                                                oneToManySB.append(',');
                                        }
                                        oneToManySB.append(String.format("mappedBy = \"%s\"",

        collPersister.getElementPersister().getPropertyNames()[index]));
                                        foundMappedBy = true;
                                        break;
                                }
                        }
                        index++;
                }
        }
        // continued     .
```

Figure 8

Part 2:

```
		if(!foundMappedBy) {
			index = 0;
			for(Type type : types) {
				if(type.getClass().getName().equals(OneToOneType.getFullyQType())) {
					org.hibernate.type.OneToOneType manyType = (org.hibernate.type.OneToOneType) type;
					if(manyType.getAssociatedEntityName().equals(persister.getEntityName())) {
						if(oneToManySB.length() >0) {
							oneToManySB.append(',');
						}
						oneToManySB.append(String.format("mappedBy = \"%s\"",
								collPersister.getElementPersister().getPropertyNames()[index]));
						foundMappedBy = true;
						break;
					}
				}
				index++;
			}
		}

		if(!foundMappedBy) {
			index = 0;
			for(Type type : types) {
				if(type.getClass().getName().equals(ManyToOneType.getFullyQType())) {
					org.hibernate.type.ManyToOneType manyType = (org.hibernate.type.ManyToOneType) type;
					//subclasses mapped
					if(persister.getEntityName().equals(persister.getRootEntityName()) &&
							persister.getEntityMetamodel().getSubclassEntityNames().
								contains(manyType.getAssociatedEntityName())) {
						if(oneToManySB.length() >0) {
							oneToManySB.append(',');
						}
						oneToManySB.append(String.format("mappedBy = \"%s\"",
								collPersister.getElementPersister().getPropertyNames()[index]));
						foundMappedBy = true;
						break;
					}
				}
				index++;
			}
		}
	}else {
		//form the join column JPA annotation if its non inverse side
		if(!manyToMany && collPersister.getKeyColumnNames() !=null && collPersister.getKeyColumnNames().length == 1) {
			String joinColFmt = "@JoinColumn(name = \"%s\" %s)";
			StringBuilder sb = new StringBuilder();
			Collection collMapping = hibernateConfiguration .getCollectionMapping(collType.getRole());

			if(!collMapping.getKey().isNullable()) {
				sb.append(", nullable = false");
			}
			String joinColumnAnnt = String.format(joinColFmt, collPersister.getKeyColumnNames()[0], sb.toString());
		}
	}

	//form the final OneToMany JPA annotation
	String oneToManyAnnt = null;
	if(oneToManySB.length() >0) {
		oneToManyAnnt = manyTypeAnnt + "(" + oneToManySB.toString() + ")";
	}else {
		oneToManyAnnt = manyTypeAnnt;
	}
```

Figure 9

III. CONCLUSION

This paper presents a thorough analysis of how to extract the hibernate entity metadata that were provided through Xdoclet and then converted to hibernate mapping configuration. This paper also deep dives into the internals of the Hibernate 4 engine to extract this metadata information for generating the JPA annotations. Property types such as identifier, a standard and a collection based OneToMany association-based properties were thoroughly investigated and detailed algorithm along with pseudo code was provided to unravel the complexities of how to convert these ORM provider specific mapping to standard JPA annotations which could be easily analysed by any ORM provider. The thorough understanding of this transformation process can help enterprise systems to transform to standard JPA annotations that are still using now legacy Xdoclet along with non-standard hibernate mapping files.

REFERENCES

[1] Hibernate 4.2, website - https://hibernate.org/orm/documentation/4.2/
[2] Ishaq Azhar Mohammed, "A literature review on the application of AI to Identity Access Management", International Journal of Emerging Technologies and Innovative Research (www.jetir.org | UGC and issn Approved), ISSN:2349-5162, Vol.5, Issue 10, page no. pp96-99, October-2018, Available at : http://www.jetir.org/papers/JETIR1810A65.pdf
[3] Hibernate 4.2 API, website - http://docs.jboss.org/hibernate/orm/4.2/javadocs/
[4] JPA 2.1 API, website - http://docs.jboss.org/hibernate/jpa/2.1/api/
[5] Hibernate Xdoclet, website - http://xdoclet.sourceforge.net/xdoclet/tags/hibernate-tags.html
[6] JPA 2.1 Specification, website - https://download.oracle.com/otndocs/jcp/persistence-2_1-fr-eval-spec/index.html
[7] Ishaq Azhar Mohammed, "Identity and Access Management for the Internet of Things", International Journal of Emerging Technologies and Innovative Research (www.jetir.org), ISSN:2349-5162, Vol.5, Issue 5, page no.1299-1303, May-2018, Available :http://www.jetir.org/papers/JETIR1805954.pdf
[8] JPA Annotations, website - https://www.objectdb.com/api/java/jpa/annotations