Metadata handling

by Armin Waibel

Table of contents

1. Introduction

To make OJB proper work information about the used databases (more info see connection handling) and sequence managers is needed. Henceforth these metadata information is called **connection metadata**.

Further on OJB needs information about the persistent objects and object relations, henceforth this information is called **(persistent) object metadata**.

All metadata information need to be stored in the OJB repository file.

The *connection metadata* are completely decoupled from the *persistent object metadata*. Thus it is possible to use the same *object metadata* on different databases.
But it is also possible to use different *object metadata* profiles .

In OJB there are several ways to make metadata information available:

- using xml configuration files parsed at start up by OJB
- set metadata instances at runtime by building metadata class instances at runtime
- parse additional xml configuration files (additional repository files) and merge at runtime

All classes used for managing metadata stuff can be find under
`org.apache.ojb.broker.metadata.*`-package.
The main class for metadata handling and entry point for metadata manipulation at runtime is
`org.apache.ojb.broker.metadata.MetadataManager` .

2. When does OJB read metadata

By default all metadata is read at startup of OJB, when the first call to
`PersistenceBrokerFactory` (directly or by a top-level api) or `MetadataManager` class was done.

OJB expects a repository file at startup, but it is also possible to start OJB without an repository file or only load connection metadata and object metadata at runtime or what ever combination fit your requirements.

3. Connection metadata

The *connection metadata* encapsulate all information referring to used database and must be declared in OJB repository file.
For each database a *jdbc-connection-descriptor* must be declared. This element encapusaltes the connection specific metadata information.

The *JdbcConnectionDescriptor* instances are managed by
`org.apache.ojb.broker.metadata.ConnectionRepository`

3.1. Load and merge connection metadata

It is possible to load additional connection metadata at runtime and merge it with the existing one. The used repository files have to be valid against the repository.dtd:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE descriptor-repository SYSTEM "repository.dtd">

<descriptor-repository version="1.0" isolation-level="read-uncommitted">
    <jdbc-connection-descriptor
        jcd-alias="runtime"
```

```
        platform="Hsqldb"
        jdbc-level="2.0"
        driver="org.hsqldb.jdbcDriver"
        protocol="jdbc"
        subprotocol="hsqldb"
        dbalias="../OJB_FarAway"
        username="sa"
        password=""
        batch-mode="false"
    >

        <object-cache
class="org.apache.ojb.broker.cache.ObjectCacheDefaultImpl">
            <attribute attribute-name="timeout" attribute-value="900"/>
            <attribute attribute-name="autoSync" attribute-value="true"/>
        </object-cache>

        <connection-pool
            maxActive="5"
            whenExhaustedAction="0"
            validationQuery="select count(*) from OJB_HL_SEQ"
        />

        <sequence-manager
className="org.apache.ojb.broker.util.sequence.SequenceManagerHighLowImpl">
            <attribute attribute-name="grabSize" attribute-value="5"/>
        </sequence-manager>
    </jdbc-connection-descriptor>

    <!-- user/passwd at runtime required -->
    <jdbc-connection-descriptor
        jcd-alias="minimal"
        platform="Hsqldb"
        jdbc-level="2.0"
        driver="org.hsqldb.jdbcDriver"
        protocol="jdbc"
        subprotocol="hsqldb"
        dbalias="../OJB_FarAway"
    >
    </jdbc-connection-descriptor>
</descriptor-repository>
```

In the above additional repository file two new *jdbc-connection-descriptor* (new databases) *runtime* and *minimal* are declared, to load and merge the additional connection metadata the *MetadataManager was used:*

```
// get MetadataManager instance
MetadataManager mm = MetadataManager.getInstance();

// read connection metadata from repository file
ConnectionRepository cr = mm.readConnectionRepository("valid path/url to
repository file");

// merge new connection metadata with existing one
mm.mergeConnectionRepository(cr);
```

After the merge the access to the new databases is ready for use.

4. Persistent object metadata

The *object metadata* encapsulate all information referring to the persistent capable java objects and the associated tables in database. *Object metadata* must be declared in OJB repository file.
Each persistence capable java object must be declared in a corresponding class-descriptor.

The *ClassDescriptor* instances are managed by `org.apache.ojb.broker.metadata.DescriptorRepository` . Per default OJB use only **one global instance** of this class - it's the repository file read at startup of OJB. But it is

possible to change the global use repository:

```
// get MetadataManager instance
MetadataManager mm = MetadataManager.getInstance();

mm.setDescriptor(myGlobalRepository, true);
```

4.1. Load and merge object metadata

It is possible to load additional object metadata at runtime and merge it with the existing one. The used repository files have to be valid against the repository.dtd:

Note:

When using the dynamic mapping technique described below, all objects in the structure must implementet `java.io.Serializable` for OJB to be able to created cloned copies. OJB currently uses SerializationUtils from Commons Lang Core Language Utilities for all deep-cloning operations.

An additional repository file may look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE descriptor-repository SYSTEM "repository.dtd">

<descriptor-repository version="1.0" isolation-level="read-uncommitted">

    <class-descriptor
      class="org.my.MyObject"
      table="MY_OBJ"
    >
        <field-descriptor
         name="id"
         column="OBJ_ID"
         jdbc-type="INTEGER"
         primarykey="true"
         autoincrement="true"
        />

        <field-descriptor
         name="name"
         column="NAME"
         jdbc-type="VARCHAR"
        />
    </class-descriptor>
</descriptor-repository>
```

To load and merge the object metadata of the additional repository files first read the metadata using the *MetadataManager* .

```
// get MetadataManager instance
MetadataManager mm = MetadataManager.getInstance();

// read the additional repository file
DescriptorRepository dr = mm.readDescriptorRepository("valid path/url to
repository file");

// merge the new class-descriptor with existing object metadata
mm.mergeDescriptorRepository(dr);
```

It is also possible to keep the different *object metadata* for the same classes parallel by using *metadata profiles* .

4.2. Global object metadata changes

The *MetadataManager* provide several methods to read/set and manipulate object metadata.

Per default OJB use a global instance of class *DescriptorRepository* to manage all *object metadata*.

This means that all *PersistenceBroker* instances (kernel component used by all top-level api) use the same object metadata.

So changes of the object metadata (e.g. remove of a CollectionDescriptor instance from a ClassDescriptor) will be seen immediately by all *PersistenceBroker* instances. This is in most cases not the favoured behaviour and OJB supports per thread changes of object metadata.

4.3. Per thread metadata changes

Per default the manager handle one global *DescriptorRepository* for all calling threads (keep in mind PB-api is not threadsafe, thus each thread use it's own PersistenceBroker instance), but it is ditto possible to use different *metadata profiles* in a per thread manner - profiles means different instances of DescriptorRepository objects. Each thread/PersistenceBroker instance can be associated with a specific *DescriptorRepository* instance. All made object metadata changes only will be seen by the PersistenceBroker instances using the same DescriptorRepository instance. In theory each PersistenceBroker instance could be associated with a separate instance of object metadata, but the recommended way is to use metadata profiles.

To enable the use of different *DescriptorRepository* instances for each thread do:

```
MetadataManager mm = MetadataManager.getInstance();
// tell the manager to use per thread mode
mm.setEnablePerThreadChanges(true);
...
```

This can be done e.g. at start up or at runtime when it's needed. If method `setEnablePerThreadChanges` is set *false* only the *global DescriptorRepository* was used. Now it's possible to use dedicated DescriptorRepository instances per thread:

```
// e.g get a coppy of the global repository
DescriptorRepository dr = mm.copyOfGlobalRepository();
// now we can manipulate the persistent object metadata of the copy
......

// set the changed repository for current thread
mm.setDescriptor(dr);

// now let this thread lookup a PersistenceBroker instance
// with the modified metadata
// all other threads use still the global object metadata
PersistenceBroker broker =
PersistenceBrokerFactory.createPersistenceBroker(myKey)
```

Note:

Set object metadata (setting of the DescriptorRepository) before lookup the PersistenceBroker instance for current thread, because the metadata was bound to the PersistenceBroker instance at lookup.

4.4. Object metadata profiles

MetadataManager was shipped with a simple mechanism to add, remove and load different *persistent objects metadata profiles* (different DescriptorRepository instances) in a per thread manner. Use method *addProfile* to add different persistent object metadata profiles, method *removeProfile* to remove profiles and *loadProfile* load a profile for the calling thread.

```
// get MetadataManager instance
MetadataManager mm = MetadataManager.getInstance();

// enable per thread mode if not done before
mm.setEnablePerThreadChanges(true);
```

```
// Load additional object metadata by parsing an repository file
DescriptorRepository dr_1 = mm.readDescriptorRepository("pathOrURLtoFile_1");
DescriptorRepository dr_2 = mm.readDescriptorRepository("pathOrURLtoFile_2");

// add  profiles
mm.addProfile("global", mm.copyOfGlobalRepository());
mm.addProfile("guest", dr_1);
mm.addProfile("admin", dr_2);

// now load a specific profile
mm.loadProfile("admin");
broker = PersistenceBrokerFactory.defaultPersistenceBroker();
```

After the *loadProfile* call all PersistenceBroker instances will be associated with the *admin* profile.

Note:

Method *loadProfile* only proper work if the per thread mode is enabled.

4.5. Reference runtime changes on per query basis

FIXME (arminw):

Changes of reference settings on a per query basis will be supported with next upcoming release 1.1

4.6. Pitfalls

OJB's flexibility of *metadata handling* demanded specific attention on object caching. If a global cache (shared permanent cache) was used, be aware of side-effects caused by runtime metadata changes.

For example, using two metadata profiles *A* and *B*. In profile A all fields of a class are showed, in profile B only the 'name filed' is showed. Thread 1 use profile A, thread 2 use profile B. It is obvious that a global shared cache will cause trouble.

5. Questions

5.1. Start OJB without a repository file?

It is possible to start OJB without any repository file. In this case you have to declare the `jdbc-connection-descriptor` and `class-descriptor` at runtime. See Connect to database at runtime? and Add new persistent objects (class-descriptors) at runtime? for more information.

5.2. Connect to database at runtime?

There are two possibilities to connect your database at runtime:

- load connection metadata by parsing additional repository files
- create the *JdbcConnectionDescriptor* at runtime

The first one is described in section load and merge connection metadata. For the second one a new instance of class `org.apache.ojb.broker.metadata.JdbcConnectionDescriptor` is needed. The prepared instance will be passed to class *ConnectionRepository*:

```
ConnectionRepository cr = MetadataManager.getInstance().connectionRepository();

JdbcConnectionDescriptor jcd = new JdbcConnectionDescriptor();
jcd.setJcdAlias("testConnection");
jcd.setUserName("sa");
```

```
jcd.setPassWord("sa");
jcd.setDbAlias("aAlias");
jcd.setDbms("aDatabase");
// .... the other required setter

// add new descriptor
cr.addDescriptor(jcd);

// Now it's possible to obtain a PB-instance
PBKey key = new PBKey("testConnection", "sa", "sa");
PersistenceBroker broker = PersistenceBrokerFactory.
createPersistenceBroker(key);
```

Please read this section from beginning for further information.

5.3. Add new persistent objects metadata (class-descriptor) at runtime?

There are two possibilities to add new *object metadata* at runtime:

- load object metadata by parsing additional repository files
- create new metadata objects at runtime

The first one is described in section load object metadata.

To create and add new metadata objects at runtime we create new `org.apache.ojb.broker.metadata.ClassDescriptor` instances at runtime and using the `MetadataManager` to add them to OJB:

```
DescriptorRepository dr = MetadataManager.getInstance().getRepository();

ClassDescriptor cld = new ClassDescriptor(dr);
cld.setClassOfObject(A.class);
//.... other setter

// add the fields of the class
FieldDescriptor fd = new FieldDescriptor(cld, 1);
fd.setPersistentField(A.class, "someAField");
cld.addFieldDescriptor(fd);

// now we add the the class descriptor
dr.setClassDescriptor(cld);
```

Please read this section from beginning for further information.