App Engine ORM User Guide

Introduction

App Engine ORM provides transparent persistence of your application's model objects using your choice of JDO or JPA, both of which are standard java persistence apis. The bulk of the implementation of these standards comes from an open source product called DataNucleus. DataNucleus has a plugin model that enables extensions for other types of datastores. The App Engine ORM is a DataNucleus plugin that adds support for the App Engine Datastore. The goal of this document is to describe how to configure your Java App Engine application to use App Engine ORM. This document does **not** teach you how to use JDO or JPA. App Engine ORM is an implementation of the JDO and JPA specifications, so you should assume that the behavior and api of App Engine ORM is consistent with these specifications unless explicitly called out in this guide.

What's Inside appengine-orm.jar?

File or Directory	Description
asm-3.1.jar	Byte code manipulation library used by DataNuclues
datanucleus-core-1.1-SNAPSHOT-sources.jar	Source for DataNucleus core
datanucleus-enhancer-1.0-SNAPSHOT-sources.jar	Source DataNuclues enhancer
datanucleus-enhancer-1.0-SNAPSHOT.jar	DataNucleus enhancer
datanucleus-jpa-1.1-SNAPSHOT.jar	DataNucleus JPA support
datanucleus-rdbms-1.1-SNAPSHOT-sources.jar	Source for DataNucleus rdbms support
datanucleus-rdbms-1.1-SNAPSHOT.jar	DataNucleus rdbms support. Even though the App Engine d relational database, we do make use of some of DataNucleu related features.
jdo2-api-2.2-SNAPSHOT.jar	JDO api
jta.jar	Java Transaction API
persistence-api-1.0.2.jar	Java Persistence API
datanucleus-core-1.1-SNAPSHOT-000X.jar	DataNuclues core, currently split up across multiple jars to g limitations.
datanucleus-appengine-0.2.jar	DataNucleus AppEngine Datastore plugin
datanucleus-appengine-0.2-sources.jar	DataNucleus AppEngine Datastore plugin sources
README	Release Notes
demos (directory)	Contains ORM demo applications. Run 'ant' in any subdirect that application using the dev appserver.
docs	App Engine ORM documentation

Enhancing Your Classes

App Engine ORM makes your POJOs transparently persistable using the DataNucleus bytecode enhancer. If you're just running your app in the dev appserver you can enable runtime bytecode enhancement by adding the following jvm flag:

```
-javaagent:lib/datanucleus-enhancer-1.0-SNAPSHOT.jar=package.with.classes.needing.enhancement
```

You'll pay a performance penalty, but hey, it's just the dev appserver. No big deal, right?

This is certainly the fastest way to get your app up and running, but you'll need to enhance your actual classfiles before you deploy your code to Google's servers. Fortunately, DataNucleus includes an ant task that makes this pretty straightforward. Here's a snippet from demos/helloorm/build.xml that performs enhancement:

```
  <path id="enhancer.classpath">
    <pathelement location="${orm.lib.dir}/jdo2-api-2.2-SNAPSHOT.jar"/>
    <pathelement location="persistence-api-1.0.2.jar"/>
    <pathelement location="${orm.lib.dir}/datanucleus-enhancer-1.0-SNAPSHOT.jar"/>
    <pathelement location="${orm.lib.dir}/datanucleus-core-1.1-SNAPSHOT-0000.jar"/>
    <pathelement location="${orm.lib.dir}/datanucleus-core-1.1-SNAPSHOT-0001.jar"/>
    <pathelement location="${orm.lib.dir}/datanucleus-core-1.1-SNAPSHOT-0002.jar"/>
    <pathelement location="${orm.lib.dir}/datanucleus-core-1.1-SNAPSHOT-0003.jar"/>
    <pathelement location="${orm.lib.dir}/datanucleus-jpa-1.1-SNAPSHOT.jar"/>
    <pathelement location="${orm.lib.dir}/datanucleus-rdbms-1.1-SNAPSHOT.jar"/>
    <pathelement location="${orm.lib.dir}/asm-3.1.jar"/>
    <pathelement location="classes"/>
  </path>

  <target name="enhance" description="enhance" depends="compile">
    <taskdef name="datanucleusenhancer" classpathref="enhancer.classpath"
            classname="org.datanucleus.enhancer.tools.EnhancerTask"/>

    <datanucleusenhancer classpathref="enhancer.classpath" failonerror="true">
      <fileset dir="classes">
        <include name="**/*.class"/>
      </fileset>
    </datanucleusenhancer>
  </target>
```

JPA Guide
Configuration

Create a persistence.xml file. Here's a sample persistence.xml for App Engine ORM, taken from demos/helloorm/src/META-INF/persistence.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">

    <persistence-unit name="helloorm">
        <class>com.google.appengine.helloorm.Flight</class>
        <properties>
            <property name="datanucleus.NontransactionalRead" value="true"/>
            <property name="datanucleus.NontransactionalWrite" value="true"/>
            <property name="datanucleus.ConnectionURL" value="appengine"/>
        </properties>
    </persistence-unit>
<persistence>
```

And then in your code...

```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("helloorm");
// off to the races
```

JDO Guide
Configuration

Create a datanucleus.properties file and place it at the root of your classpath. Here's a sample datanucleus.properties for App Engine ORM, taken from demos/helloorm/src/datanucleus.properties:

```
javax.jdo.PersistenceManagerFactoryClass=org.datanucleus.jdo.JDOPersistenceManagerFactory
javax.jdo.option.ConnectionURL=appengine
datanucleus.NontransactionalRead=true
datanucleus.NontransactionalWrite=true
```

And then in your code...

```
PersistenceManagerFactory pmf =
JDOHelper.getPersistenceManagerFactory("datanucleus.properties");
// off to the races
```

Primary Keys
The unique identifier for an Entity in the App Engine Datastore consists of your application id, the kind of the entity, and an additional value provided either by the datastore or by the user. **As of this writing, primary keys for POJOs that can be written to the datastore using App Engine ORM must have a String primary key field.** When you create a new object that you want persisted to the datastore you get to choose whether you want the datastore to provide this value for you or if you want to choose it yourself. If you want the datastore to provide it, just leave your primary key field null. App Engine ORM will treat this as a request for the datastore to assign an id. If you want to provide it yourself, set the primary key field to the value you want before making your object persistent. App Engine ORM will modify the value of the Primary Key field during the process of making your object persistent.

Regardless of which approach you take, once you make your object persistent the primary key field's value will now be a String encoding of a com.google.apphosting.api.datastore.Key object. If you wanted the datastore to generate the id for you the generated id can be retrieved by calling com.google.api.datastore.KeyFactory.decodeKey(pkString).getId(). If you wanted to set the value yourself you can verify that your value was properly set by calling com.google.api.datastore.KeyFactory.decodeKey(pkString).getName().

Inserting POJOs With Ancestors
When you persist a POJO to the datastore, one of the things you get to choose is which persistent object key, if any, is the parent of the POJO you are persisting. App Engine ORM supports this datastore feature through a JPA/JDO extension. Here's an example for JPA:

```
@Entity
public class HasAncestorJPA {
  @Extension(vendorName="datanucleus", key="ancestor-pk", value="true")
  private String ancestorId;

  @Id
  @GeneratedValue(strategy= GenerationType.IDENTITY)
  private String id;

  public HasAncestorJPA(String ancestorId) {
    this(ancestorId, null);
  }
 }
```

and here's the corresponding example for JDO:

```
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class HasAncestorJDO {
  @Persistent
  @Extension(vendorName="datanucleus", key="ancestor-pk", value="true")
  private String ancestorId;

  @PrimaryKey
  @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
  private String id;

  public HasAncestorJDO(String ancestorId) {
    this(ancestorId, null);
  }
}
```

Adding the Extension annotaiton to a field tells App Engine ORM to use the value of this field as the parent key when it creates the persistent Entity. Currently this field must be a String. Note that creating a field for the ancestor and annotating it as above is not a commitment to providing an ancestor for every POJO of this type - null is a perfectly valid value for this field.

Querying POJOs By Ancestor

The datastore can fulfill ancestor queries. Even though ancestors are not part of the JPA or JDO specifications, you can issue JPQL and JDOQL queries that restrict by ancestor without any special syntax.

In JPQL:

```
    Query q = entityMgr.createQuery(
        "select from " + HasAncestor.class.getName() + " where ancestorId =
:ancId");
    q.setParameter("ancId", ancestorKeyStr);
    return q.getResultList();
```

In JDOQL:

```
    Query q = persistenceMgr.newQuery(
        "select from " + HasAncestorJDO.class.getName() + " where
```

```
ancestorId == ancId parameters String ancId");
    return q.execute(ancestorKeyStr);
```

Query Limitations

Although JDOQL and JPQL are feature-rich query languages, App Engine ORM only supports the features of these query languages that can be natively fulfilled by the datastore. Please read http://code.google.com/appengine/docs/datastore/queryclass.html for information on the types of queries supported by the datastore. You should be able to conceptually map the datastore query capabilities to the corresponding features in JDOQL and JPQL, but here's a summary:

- You can have as many equals filters as you want as long as they're separated by AND.
- You can have exactly one inequality filter (>, >=, <=, <).
- No aggregation functions (min, max, avg, etc.).
- No joins.

Transactions

You'll need to read and understand http://code.google.com/appengine/docs/datastore/transactions.html before you can make proper user of transactions with App Engine ORM. While we support standard transaction management apis, the semantics are probably a bit different from what you're used to because the App Engine Datastore does not support global transactions. More information to be added by Erick.....