Google code

Google App Engine

Using the Datastore with JDO

Storing data in a scalable web application can be tricky. A user could be interacting with any of dozens of web servers at a given time, and the user's next request could go to a different web server than the one that handled the previous request. All web servers need to be interacting with data that is also spread out across dozens of machines, possibly in different locations around the world.

With Google App Engine, you don't have to worry about any of that. App Engine's infrastructure takes care of all of the distribution, replication and load balancing of data behind a simple API—and you get a powerful query engine and transactions as well.

The App Engine datastore is one of several services provided by App Engine with two APIs: a standard API, and a low-level API. By using the standard APIs, you make it easier to port your application to other hosting environments and other database technologies, if you ever need to. Standard APIs "decouple" your application from the App Engine services. App Engine services also provide low-level APIs that exposes the service capabilities directly. You can use the low-level APIs to implement new adapter interfaces, or just use the APIs directly in your app.

App Engine includes support for two different API standards for the datastore: Java Data Objects (JDO) and Java Persistence API (JPA). These interfaces are provided by DataNucleus Access Platform, an open source implementation of several Java persistence standards, with an adapter for the App Engine datastore.

For the guest book, we'll use the JDO interface to retrieve and post messages left by users.

Setting Up DataNucleus Access Platform

Access Platform needs a configuration file that tells it to use the App Engine datastore as the backend for the JDO implementation. In the final WAR, this file is named `jdoconfig.xml` and resides in the directory `war/WEB-INF/classes/META-INF/`.

If you are using Eclipse, this file has been created for you as `src/META-INF/jdoconfig.xml`. This file is automatically copied into `war/WEB-INF/classes/META-INF/` when you build your project.

If you are not using Eclipse, you can create the directory `war/WEB-INF/classes/META-INF/` directly, or have your build process create it and copy the configuration file from another location. The Ant build script described in Using Apache Ant copies this file from `src/META-INF/`.

The `jdoconfig.xml` file should have the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<jdoconfig xmlns="http://java.sun.com/xml/ns/jdo/jdoconfig"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://java.sun.com/xml/ns/jdo/jdoconfig">

    <persistence-manager-factory name="transactions-optional">
        <property name="javax.jdo.PersistenceManagerFactoryClass"

value="org.datanucleus.store.appengine.jdo.DatastoreJDOPersistenceManagerFactory"/>
        <property name="javax.jdo.option.ConnectionURL" value="appengine"/>
        <property name="javax.jdo.option.NontransactionalRead" value="true"/>
        <property name="javax.jdo.option.NontransactionalWrite" value="true"/>
        <property name="javax.jdo.option.RetainValues" value="true"/>
        <property name="datanucleus.appengine.autoCreateDatastoreTxns"
value="true"/>
    </persistence-manager-factory>
</jdoconfig>
```

JDO Class Enhancement

When you create your JDO classes, you use Java annotations to describe how instances should be stored in the datastore, and how they should be recreated when retrieved from the datastore. Access Platform connects your data classes to the implementation using a post-compilation processing step, which DataNucleus calls "enhancing" the classes.

If you are using Eclipse and the Google Plugin, the plugin performs the JDO class enhancement step automatically as part of the build process.