db4o-spring: db4o Spring integration

Reference Documentation

Version 0.1

13.11.2006

Costin Leau

Table of Contents

Chapter 1. Introduction

The aim of db4o-spring project is to facilitate integration between Spring framework and db4o allowing easier resource management, DAO implementation support and transaction strategies. In many respects, db4o-spring is similar in structure, naming and functionality to Spring core modules for Hibernate, JPA or JDO - user familiar with Spring data access packages should feel right at home when using db4o-spring.

The project contains a sample web application named *Recipe Manager* but also some examples from db4o distribution (mainly chapter 1) which have been simplified using db4o-spring code.

Before moving on, please note that this document is a work in progress - if you have any suggestions about db4o-spring project please use the mailing list or Spring/Db4o forums.

Chapter 2. Configuration

Before being used, db4o has to be configured. db4o-spring makes it easy to externalize db4o configuration (be it client or server) from the application into Spring application context xml files, reducing the code base and allowing decoupling of application from the environment it runs in. The core class for creating db4o's `ObjectContainer` is the `ObjectContainerFactoryBean`. Based on the various parameter passed to it, the objectcontainer can be created from a db4o database file, from an `ObjectServer` or based on a `Configuration` object.

2.1. Configuring an ObjectContainer

The FactoryBean will create `ObjectContainers` based on its properties, using the algorithm below:

1. if the databaseFile is set, a local file based client will be created
2. if memoryFile is set, a local memory based client will be instantiated
3. if a server property is set, a client `ObjectContainer` will be created within the VM using the given server object
4. if all the above fail, a connection to a (possibly) remote machine will be opened using the hostName, port, user and password properties.

For example in order to create a memory based file `ObjectContainer`, the following configuration can be used:

```
<bean id="memoryContainer" class="org.db4ospring.ObjectContainerFactoryBean">
  <property name="memoryFile">
   <bean class="com.db4o.ext.MemoryFile"/>
  </property>
</bean>
```

An an `ObjectContainer` connected to a (remote) server, as follows:

```
<bean id="remoteServerContainer" class="org.db4ospring.ObjectContainerFactoryBean">
  <property name="hostName" value="localhost"/>
  <property name="port" value="123"/>
  <property name="user" value="foo"/>
  <property name="password" value="bar"/>
</bean>
```

While a database file based, local `ObjectContainer` using a bean definition such as:

```
<bean id="fileContainer" class="org.db4ospring.ObjectContainerFactoryBean">
  <property name="databaseFile" value="classpath:db4o-file.db"/>
</bean>
```

2.2. Configuring an ObjectServer

ObjectServerFactoryBean can be used for creating and configuring an `ObjectServer`:

```
<bean id="server" class="org.db4ospring.ObjectServerFactoryBean">
  <property name="userAccessLocation" value="user-access.properties"/>
  <property name="databaseFile" value="file://./db4o.db"/>
  <property name="port" value="123"/>
```

```
</bean>
```

Note the *userAccessLocation* property which specifies the location of a `Properties` file that will be used for user acess - the properties file keys will be considered the user names while the values as their passwords.

2.3. Using db4o's Configuration object

When a complex configuration is required, `ConfigurationFactoryBean` offers an extensive list of db4o parameters which can be used set and which will affect all db4o `ObjectContainers` created afterwards. For example to specify a custom activation depth and a custom message level, the following xml can be used:

```
<bean id="configurationObject" class="org.db4ospring.ConfigurationFactoryBean">
  <property name="messageLevel" value="2"/>
  <property name="activationDepth" value="3"/>
</bean>
```

Chapter 3. Inversion of Control: Template and Callback

The core classes of db4o-spring that are used in practice, are `Db4oTemplate` and `Db4oCallback`. The template translated db4o exceptions into Spring Data Access exception hierarchy (making it easy to integrate db4o with other persistence frameworks supported by Spring) and maps most of db4o `ObjectContainer` and `ExtObjectContainer` interface methods, allowing one-liners:

```
db4oTemplate.activate(personObject, 4);
// or
db4oTemplate.releaseSemaphore("myLock");
```

Chapter 4. Transaction Management

db4o-spring provides integration with Spring's excellent transaction support through `Db4oTransactionManager` class. Since db4o statements are always executed inside a transaction, Spring transaction demarcation can be used for commiting or rolling back the running transaction at certain points during the execution flow.

Consider the following example (using Spring 2.0 transactional namespace):

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

  <!-- this is the service object that we want to make transactional -->
  <bean id="fooService" class="x.y.service.DefaultFooService"/>

  <!-- the transactional advice (i.e. what 'happens'; see the <aop:advisor/> bean below) -->
  <tx:advice id="txAdvice" transaction-manager="txManager">
    <!-- the transactional semantics... -->
    <tx:attributes>
      <!-- all methods starting with 'get' are read-only -->
      <tx:method name="get*" read-only="true"/>
      <!-- other methods use the default transaction settings (see below) -->
      <tx:method name="*"/>
    </tx:attributes>
  </tx:advice>

  <!-- ensure that the above transactional advice runs for any execution
      of an operation defined by the FooService interface -->
  <aop:config>
    <aop:pointcut id="fooServiceOperation" expression="execution(* x.y.service.FooService.*(..))"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceOperation"/>
  </aop:config>

  <bean id="txManager" class="org.db4ospring.Db4oTransactionManager">
    <property name="objectContainer" ref="objectContainer"/>
  </bean>

  // more bean definition follow
</beans>
```

Chapter 5. Outside the Spring container

It is important to note that db4o-spring classes rely as much as possible on db4o alone and they work with objects either configured by the user or Spring framework. The template as well as the `FactoryBeans` can be instantiated either by Spring or created programatically through Java code.