tapestry-jdo guide

Version Status : 0.0.2 second release
Tested with Tapestry 5.2.x and 5.3.x

Introduction

The Tynamo Tapestry-JDO module allows you to work with a JDO3 (http://www.oracle.com/technetwork/java/index-jsp-135919.html) backed persistence layer, similarly to the way you would work w/ a JPA or Hibernate based persistence layer. JDO's main appeal is in its extensive support for ORM into relational databases (e.g. MySQL) as well as non-relational data stores (e.g. MongoDB through Datanucleus MongoDB Support, Google AppEngine JDO API, etc)

Configuration

In order to have a working configuration, the following are required:

- A dependency on org.tynamo:tapestry-jdo
- A JDO implementation (we use datanucleus in our examples, but others such as Apache JDO should work as well). The implementations typically have some way of integrating the JDO Enhancement process into the build; although not absolutely necessary (e.g. you could probably do that manually after you run your build and before packaging your app), it would probably be a good idea
- A properly configured jdoconfig.xml or persistence.xml

Add dependency on Tapestry-JDO

Just adding this dependency will bring transitive dependency on jdo3.

Maven dependency for tapestry-jdo

```
<!-- tynamo jdo related dependencies -->
<dependency>
   <groupId>org.tynamo</groupId>
   <artifactId>tapestry-jdo</artifactId>
   <version>0.0.2</version>
</dependency>
```

Add your JDO implementation

Add your own dependencies for your JDO implementation if you're using something different:

JDO Implementation Dependencies

```
<dependency>
     <groupId>org.datanucleus</groupId>
     <artifactId>datanucleus-core</artifactId>
     <version>3.1.2</version>
 </dependency>
<dependency>
     <groupId>org.datanucleus</groupId>
     <artifactId>datanucleus-api-jdo</artifactId>
     <version>3.1.2</version>
</dependency>
<dependency>
     <groupId>org.datanucleus</groupId>
     <artifactId>datanucleus-rdbms</artifactId>
     <version>3.1.2</version>
</dependency>
```

Configure build to instrument your JDO classes:

JDO Enhancer Configuration in Maven

```
<build>
   ..
   <plugins>
          <!--
    This plug-in "enhances" your domain model objects (i.e. makes them
    persistent for datanucleus)
   -->
            <plugin>
                <groupId>org.datanucleus</groupId>
                <artifactId>maven-datanucleus-plugin</artifactId>

                <version>3.1.2</version>
                <configuration>
                    <metadataDirectory>target/classes</metadataDirectory>
                    <metadataIncludes>**/entities/*.class</metadataIncludes>
                    <verbose>true</verbose>
                    <enhancerName>ASM</enhancerName>
                    <api>JDO</api>
                </configuration>
                <executions>
                    <execution>
                        <phase>compile</phase>
                        <goals>
                            <goal>enhance</goal>
                        </goals>
                    </execution>
                </executions>
                <dependencies>
                    <dependency>
                        <groupId>org.datanucleus</groupId>
                        <artifactId>datanucleus-core</artifactId>
                        <version>3.1.2</version>
```

```
                    <scope>runtime</scope>
                    <exclusions>
                        <exclusion>
                            <groupId>javax.transaction</groupId>
                            <artifactId>transaction-api</artifactId>
                        </exclusion>
                    </exclusions>
                </dependency>
                <dependency>
                    <groupId>org.datanucleus</groupId>
                    <artifactId>datanucleus-api-jdo</artifactId>
                    <version>3.1.2</version>
                </dependency>
                <dependency>
                    <groupId>org.datanucleus</groupId>
                    <artifactId>datanucleus-rdbms</artifactId>
                    <version>3.1.2</version>
                </dependency>
                <dependency>
                    <groupId>org.datanucleus</groupId>
                    <artifactId>datanucleus-enhancer</artifactId>
                    <version>3.1.1</version>
                </dependency>
            </dependencies>
        </plugin>
....
```

```
      </plugins>
</build>
....
```

Add your persistence store driver :

JDBC Driver Dependency

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

Setup your JDO configuration (jdoconfig.xml):

jdoconfig.xml

```
<?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">
    <!--property name="javax.jdo.option.NontransactionalWrite" value="true"/-->
    <persistence-manager-factory name="testapp-pm">
        <property name="javax.jdo.PersistenceManagerFactoryClass"
value="org.datanucleus.api.jdo.JDOPersistenceManagerFactory"/>
        <property name="javax.jdo.option.RetainValues" value="true"/>
  <property name="javax.jdo.option.ConnectionDriverName" value="org.h2.Driver" />
  <property name="javax.jdo.option.ConnectionURL"
value="jdbc:h2:~/.h2/test-jdo-database" />
  <property name="javax.jdo.option.ConnectionUserName" value="" />
  <property name="javax.jdo.option.ConnectionPassword" value="" />
  <property name="datanucleus.autoCreateSchema" value="true" />
  <property name="datanucleus.validateTables" value="false" />
  <property name="datanucleus.validateConstraints" value="false" />
    </persistence-manager-factory>
</jdoconfig>
```

Contribute the PersistenceManagerFactory name to your AppModule:

Contribute Persistence Manager Factory name

```
public static void contributeApplicationDefaults(
           MappedConfiguration<String, String> configuration) {
        configuration.add(JDOSymbols.PMF_NAME, "testapp-pm");
    }
```

Usage

Inject PersistenceManager into your pages:

Injecting JDO Persistence Manager

```
import javax.jdo.PersistenceManager
public class Page1 {
 @Inject private PersistenceManager pm;

 public List<TestEntity> getEntities() {
  return (List<TestEntity>) pm.newQuery(TestEntity.class).execute();
 }
}
```

Auto-Commit in your pages

@CommitAfter in pages

```
import javax.jdo.PersistenceManager;
import org.tynamo.jdo.annotations.CommitAfter;
public class Page2 {
 @Inject private PersistenceManager pm

 @CommitAfter
 public void onActionFromAddEntity() {
  TestEntity te = new TestEntity();
  te.setValue(new Date().getTime()+"");
  pm.makePersistent(te);
 }
}
```

Configure service transactional behavior

Service transactions

```
8.3. To add transactional behavior to your own services, instrument them in the module
and use @CommitAfter in the interface definition, e.g.
//.. create and annotate a service ...
public interface TestService {
 @CommitAfter void addTestEntity();
 @CommitAfter void removeTestEntity(long id);
}

//... in AppModule....
@Match("*Service")
public static void adviseTransactions(JDOTransactionAdvisor advisor,
MethodAdviceReceiver receiver) {
 advisor.addTransactionCommitAdvice(receiver);
}
```

Use JDO object as page contex

Classes that use a simple primary key (e.g. Long, String, etc) can be used as an acitvation/passivation context directly

Activate/Passivate

```
public class Page3 {

    @Property
    private User user;


    @SuppressWarnings("unchecked")
    User onPassivate() {
        List<User> users = (List<User>) pm.newQuery(User.class).execute();
        if (users.isEmpty()) {
            return null;
        }
        return users.get(0);
    }

    void onActivate(User user) {
        this.user = user;
    }
}
```