Enterprise Systems

Lecture 15

A view of Architecture via Spring and Hibernate

Note the separation of duties!

Presentation tier
JSP, Wicket, GWT...

Business Services
Layer: independent
of persistence layer

 DAO interface
for persistent
operations

Hibernate
ORM
DAO interface
Implementation
using ORM

Persistent domain
model
POJO

Recap

• Why Spring?
  • Description of the problem
• Overall view
• Inspecting a sample of an Spring-Hibernate Application → List of Questions!
• Detailed study of Spring:
  • Dependency Injection
  • Beans and their wiring
  • Interaction with db
  • Transactions
  • Security

Part 1 of Solution: Dependency injection
1. Create interfaces to hide the details of implementation behind it
2. provide different implementations
   (GoodImplementation(), Better...)
3. Instead of making the class (Investor) responsible for getting the Investment, assign the Investment to the Investor (this is done via wiring: an xml file)
4. Use Spring container to implement the wiring by wiring the components together.

Part 2 of Solution: AoP

Aspect oriented Programming allows you
1) define crosscutting concerns in one place as an object called aspect.
2) declaratively specify when and where the functionality should be applied
Hence
• You modify them in one place (easier and cheaper maintenance)
• Reduce the code at the business logic

Four methods of doing AoP in Spring
1. Classic proxy based method (Java-based implements one of aop interfaces to produce an advice)
   Example: logfile example earlier slides
2. Pure-POJO aspects (using XML config file)
   shop-hbr.spring.jar example
   We will use this in transactions mostly.
   There are two other methods using AspectJ, which we will not study in this course
Creating advice - the classic way
Implement one of the following interfaces.
Around a method:
org.aopalliance.intercept.MethodInterceptor
Before a method call:
org.springframework.aop.MethodBeforeAdvice
After-returning from a method call:
org.springframework.aop.AfterReturningAdvice

Around a method
Around a method is a combination of the other three.

Interface MethodInterceptor requires implementation of only one method
invoke() which has a try/catch and three parts
1) The code before the target method
2) Then proceed() is called, remember
Object obj =
    methodInvocation.proceed();
3) The code after the target

Declaring pure-POJO aspects
Declare AOP information in XML config. files via these tags:
<aop:config> top-level element containing all other elements
<aop:aspect> defines an aspect.
<aop:pointcut> defines pointcuts

Also
<aop:after>, <aop:after-returning>,
<aop:after-throwing>, <aop:around>

Sample code
In shop-spring-bib.xml we have:
<aop:config>
    <aop:pointcut
        id="shopServiceOperation"
        expression="execution
(\*shop.usecases..*(\..))="/>
    <aop:advisor advice-ref="txAdvice"
        pointcutref="shopServiceOperation"
    />
</aop:config>
We will study these in the transactions section

Data Access Object (DAO)
Service objects accesses data through a DAO interface
Advantages:
1. Better testing (use dummy implementation for DAO)
2. Different DB technologies can be used with minimal changes
   (Services and DB are decoupled)
Data Access Object (DAO)

View code now
- Interfaces in shop.dao directory - inspect
- DAO implementations in Hibernate directory
- For a JDBC implementation we require a JDBC directory
Now we will focus on DAO implementation

DAO Impl.: Template & callback

In spring data access process has two parts:
1. Templates (JDBC, Hibernate, ...)
2. Callbacks //What is a callback?

Callback: technology dependent - creating statements, binding parameters ...
Templates: implements transaction control, managing of resources, and handling exceptions via JdbcTemplate or HibernateTemplate ...
Callback part is a snippet of Hibernate code, but how does Template works?

how does Template works?

Templates require following configurations:
1) Configure data source (database, password, username,...)
2) configuring Hibernate (sessions, mapping files,...)
3) extending DAO support classes
(remember bulletpoint extends HibernateDaoSupport)
Next we will look at the above three.

DAO Impl.: Template & callback

DAO Template DAO Callback
1. Prepare resources
2. Start Transactions 3. Execute Transaction
4. Return Data
5. Commit/Rollback Transaction
6. Close resources + Handle Exception

Look at HibernateCallback Object
1) Configure data source

This will configure database, username, password, arguments about pool size, etc.

There are three methods, we use Database Connection Pools (DBCP)

See jakarta.apache.org/commons/dbcp

Look at Lines 16-32 of the boilerplate code at shop-spring-hib.xml

Pay attention to wiring of password by “ref=” to shop.Main.getPassword(). For password setting with value use

```
<property name="password" value=""/>
```

2) configuring Hibernate part

Remember in Hibernate save, update, delete, … was done through Session which was obtained through a SessionFactory:

Lines 54-56:

```xml
<bean id="hibernateTemplate"
 class="org.springframework.orm.hibernate3.HibernateTemplate">
 <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

2) configuring Hibernate part

We looked at code

Declare

1) dataSource (see earlier slider)

2) List of Hibernate mappingResources

3) Dialect of hibernate

3) extending DAO support classes

It is possible to manually wire the beans, but it is easier that “Dao Impl.” classes extend HibernateDaoSupport

(see BookDAOImpl.java)

3) extending DAO support classes

Call getHibernateTemplate() method and then invoke hibernate methods such as

- execute()
- save()
- saveOrUpdate()
- ...

A few words on Exceptions

Remember we argued that: although some exceptions if caught are useful, such as rollback … But most are not!

If a connection is failed- no matter what the reason, we can’t do much 😞

Spring has a large set of exceptions for data access which are independent from the technology (Hibernate, JDBC, …)

In addition you can leave the exceptions unchecked! Spring will handle them for you. But you must use Templates.
Samples of exceptions

CannotAcquireLockException
CannotSerializeTransactionException
CleanupFailureDataAccessException
ConcurrencyFailureException
DataAccessException
DataAccessResourceFailureException
DataIntegrityViolationException
DataRetrievalFailureException
DeadlockLoserDataAccessException
EmptyResultDataAccessException

...some more

...Very self explanatory!