Enterprise Systems

Lecture 16

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

ORM
- DAO interface
- Implementation

Persistent domain model
- POJO

Hibernate
- ORM

Recap

Spring uses
1) dependency injection
2) aspect

We looked at sample code.

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

Q: Can I do classic aop with spring? Yes.
We don't use it, discussion is for the sake of completeness
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

After-throwing an exception:
- org.springframework.aop.ThrowsAdvice

Around a method

Around a method is a combination of the other three.

Interface MethodInterceptor require
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
Around a method

Then PerformanceException is caught. You must remember to call proceed(). Failure to call proceed() will result in the advice being applied but the target method never being executed. But then again, that may be what you want. Perhaps you'd like to prevent execution of a method under certain conditions.

--

We can get away with using “around” always!
For details of the other three, see the Spring manual.

Declaring pure-POJO aspects

In our course we don't use the “classic way”. Instead declare AoP information in XML configuration files.

1) give an advice
2) say where the advice is applied

Creating advice (recap)

<title>tx:advice id="transactionAdvice" transaction-manager="transactionManager">
<title>tx:attributes>
<title>tx:method name="*" propagation="REQUIRED" />
<title>tx:attributes>
</title>advice>
Can be applied to any method!

Applying advice to pointcut (recap)

<title>aop:config proxy-targetclass="true">
<title>aop:pointcut expression="execution(* uk.ac.bham.cs.book.hibernate.ShopServiceImpl.*(..))" id="shopServiceOperation" />
<title>aop:advisor advice-ref="transactionAdvice" pointcut-ref="shopServiceOperation" />
</title>aop:config>

Available AOP configuration elements

You can control the behaviour of aspect via the following elements:
<title>aop:config> top-level element containing all other elements
<title>aop:aspect> defines an aspect.
<title>aop:pointcut> defines pointcuts

Similar to classic ways you can use the following elements:
<title>aop:after>, <title>aop:after-returning>, <title>aop:after-throwing>, <title>aop:around>
Data Access Object (DAO)

- Service objects accesses data through a DAO interface
- DAO Interface
- DAO implementation
- Service objects accesses data through a DAO interface
- 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

Advantages:
1. Better testing (use dummy implementation for DAO)
2. Different DB technologies can be used with minimal changes (Services and DB are decoupled)

Comparing

Now we compare the two styles of Hibernate with the new one
Why?
1) understand old code
2) understand how to use spring with jdbc, jpa, ... (if you decided not to use hibernate

Old style is discouraged.

getBookByISBN in earlier versions

public Book getBookByISBN(final String isbn)
{
    return (Book) getHibernateTemplate().execute(new HibernateCallback()
    {
        public Object doInHibernate(Session session)
        {
            return session.createQuery("from Book b where b.isbn = :b_isbn")
                        .setString("b_isbn", isbn)
                        .uniqueResult();
        }
    });
}

Data Access Object (DAO) (older versions)

Apart from the following bullet points, the rest is hibernate code - what is going on?
- getHibernateTemplate()
- extends HibernateDaoSupport
- HibernateCallback()
- doInHibernate(Session session)
Template & callback (older versions)

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

Template & callback (older versions)

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

Template & callback (older versions)

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 things fit together (older versions)

It is possible to manually wire the beans, but it is easier that “Dao Impl.” classes extend HibernateDaoSupport
( org.springframework.orm.hibernate3.support.HibernateDaoSupport)

3) extending DAO support classes

Call getHibernateTemplate() method and then invoke hibernate methos such as
• execute()
• save()
• saveorUpdate()
• ...

We will not look any further at older version
Comparison and discussion

The following are supported in Hibernate5

- getHibernateTemplate()
- HibernateCallback()
- doInHibernate(Session session)

How do the two ways compare?
SessionFactory.getCurrentSession() is more elegant?
You can write pure hibernate code.
Spring takes care of opening closing flushing through transaction manager...

Spring discourages the use of templates

Inorg.springframework.orm.hibernate5.HibernateTemplate API says:

NOTE: Hibernate access code can also be coded in plain Hibernate style. Hence, for newly started projects, consider adopting the standard Hibernate style of coding data access objects instead, based on SessionFactory.getCurrentSession(). This HibernateTemplate primarily exists as a migration helper for Hibernate 3 ...

Templates can be used for exception handling.
If you are not using hibernate for jdbc and jpa you can use jdbcTemplate and jpaTemplate and benefit from Spring (dependency injection, aop, transactions,...)