Chapter 1 - Overview of Spring Boot Database Integration

Objectives
Key objectives of this chapter: ■ DAO Support in Spring ■ Various data access technologies supported by Spring

1.1 DAO Support in Spring

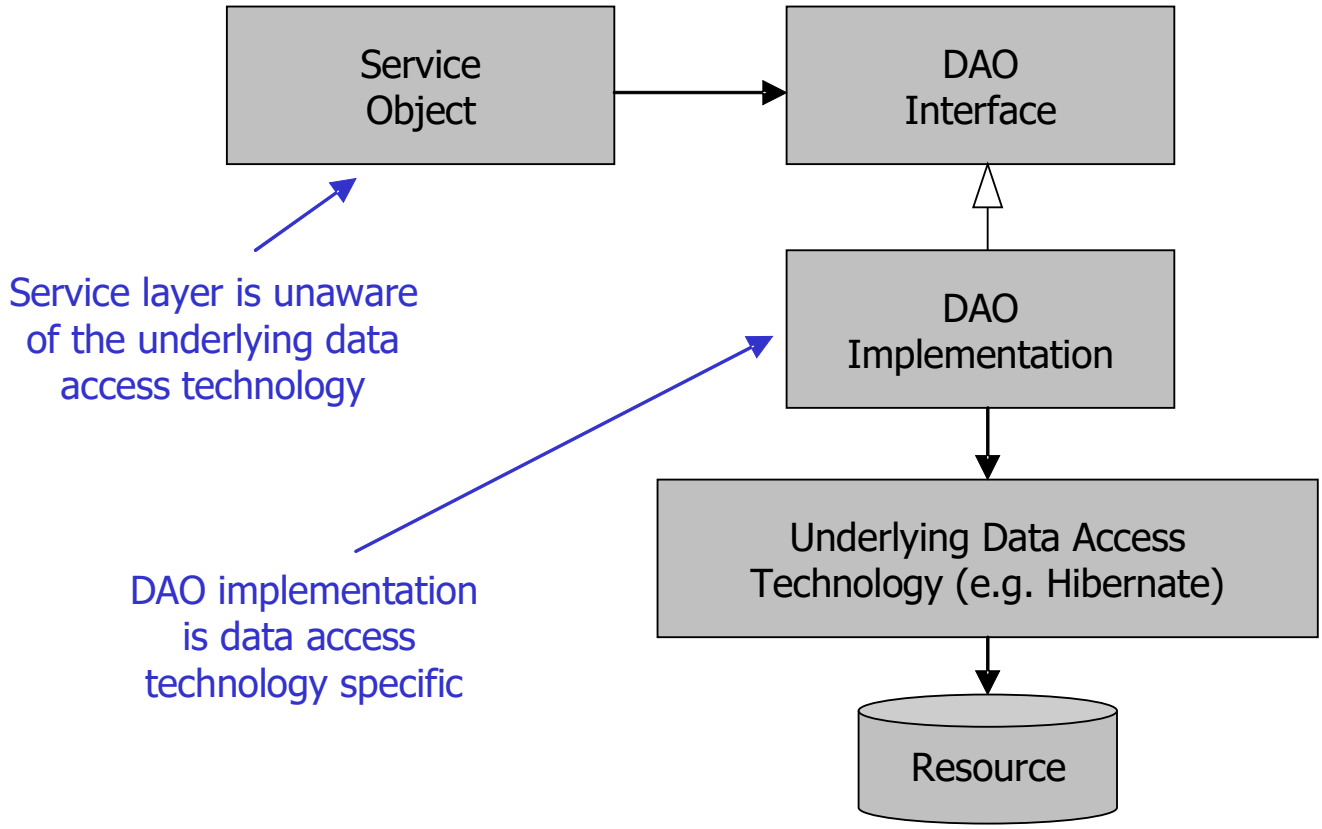
- A data access object (DAO) is a mechanism to read and write data from a database
- DAOs use underlying data access technologies such as JDBC or an ORM framework like Hibernate
- Spring DAO support is designed so it is easy to switch from one data access technology to another
  - E.g. JDBC, Hibernate, JDO, etc

DAO Support in Spring

The data access object design pattern is a proven pattern in J2EE application architecture. Its primary purpose is to provide a way of keeping upper software layers decoupled from lower level data access technologies and infrastructural details so upper layers can concern themselves with business logic and not have to worry about data persistence.

Spring recognizes the value of this pattern and provides a sophisticated framework based around it.

1.2 DAO Support in Spring

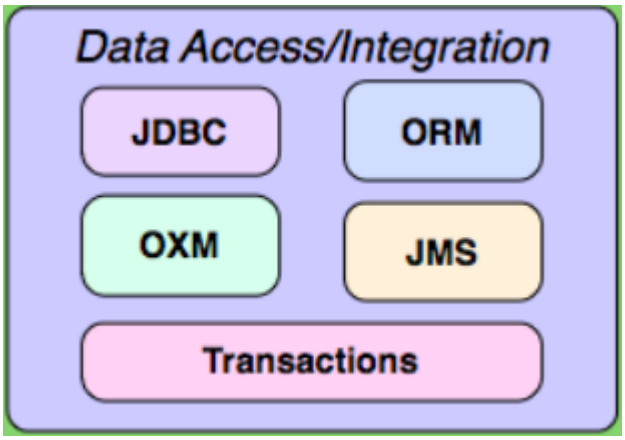


DAO Support in Spring

Spring promotes coding to interfaces and the area of DAOs is no exception. By coding service layers to DAO interfaces, you gain at least two advantages. First, the code is easier to test. Mock DAO implementations can more easily be swapped in during testing of service layers allowing you to fully test a service object without the need for the real DAO implementation and all of its dependencies.

Second, using an interface allows you the flexibility of swapping in a different DAO implementation at deployment time. For example, if you choose to change from JDBC to Hibernate as your data access technology, your service layers are oblivious to this fact and should not need to change.

1.3 Spring Data Access Modules



- Spring has 2 modules providing support for various data access options
  - JDBC module
  - ORM module
    - The ORM module depends on the JDBC module
- Both modules depend on the Spring Transaction module
  - Data access is transactional in nature but Spring supports this with a separate module

1.4 Spring JDBC Module

- The Spring JDBC module provides a framework and supporting classes for simplifying JDBC code
  - It is a good fit for applications that already use direct JDBC access and have not migrated to some Object Relational Mapping framework (ORM)
- The Spring JDBC module also has classes for DataSource access
  - This includes DataSource implementations that could be used in testing or when running an application outside a Java EE server
  - This also includes support for an embedded database which could be used in testing

- The DataSource support from this module is used even when an application is using the Spring ORM module which is why the ORM module depends on the JDBC module

- Much of the classes from this module used directly in applications are some form of JdbcTemplate class

1.5 Spring ORM Module

- The Spring ORM module has supporting classes for the following ORM frameworks:
  - Hibernate
  - Java Persistence API (JPA)
    - This is the relatively new Java standard way to do persistence
  - Java Data Objects (JDO)
  - iBATIS SQL maps
- An application would typically use only one of these frameworks although the Spring ORM module contains support for all four
  - For applications not already using one of these frameworks the JPA standard is generally the default choice since it is a Java standard

1.6 Spring ORM Module

- Code written using one of the ORM frameworks supported by Spring typically uses the persistence framework directly
  - Spring provides supporting classes to integrate other Spring features like transactions with the ORM framework but these are often not seen in the code of the application itself

- Spring configuration files is the area that largely contains the integration with the ORM framework
- Spring has various XXXTemplate classes, like HibernateTemplate, but these generally should NOT be used as Spring 3 can properly integrate when the application uses the ORM API directly

1.7 DataAccessException

- The Spring DAO framework insulates higher layers from technology-specific exceptions
  - E.g. SQLException and HibernateException
- Instead, it throws DataAccessException
  - Root cause is still available using getCause()
- This "Exception translation" is one of the key benefits of Spring data access support
  - This keeps higher layers decoupled from lower level technologies
- DataAccessException is a RuntimeException
  - RuntimeExceptions are "unchecked" which means they do not need to be caught in order for the code to compile
  - This is important because it means that application code is not required to catch these Exceptions since nothing can often be done about them anyway
    - Why add a bunch of code to catch Exceptions when you aren't doing anything except ignoring the Exception?

DataAccessException

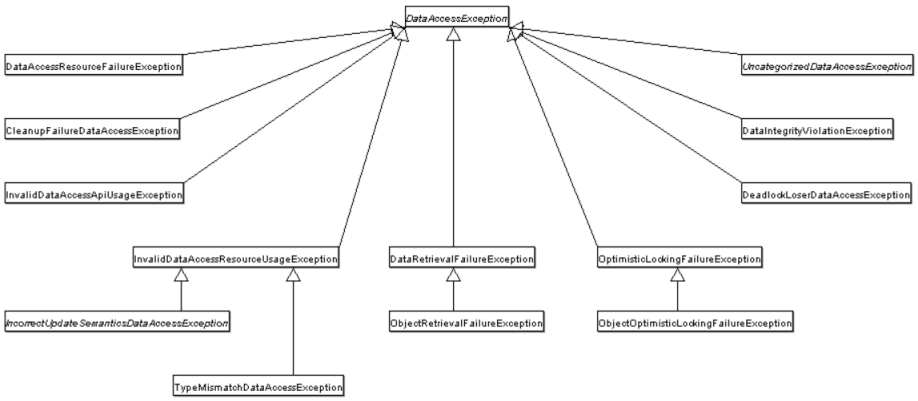
By throwing DataAccessException from your DAO layer instead of underlying data access specific

exceptions like SQLException, you keep upper software layers decoupled from the underlying data access technology. For example, you can change from JDBC to Hibernate without upper layers even knowing about it because they don't deal with SQLExceptions generated by JDBC.

Since DataAccessException is a RuntimeException, you are not forced to catch exceptions that you probably cannot recover from. For example, if the database goes down, there probably isn't much that the application can do about it. Even if there is, you can still catch the exception, you just aren't forced to.

1.8 DataAccessException

- Spring provides a hierarchy of DataAccessException subclasses representing different types of exceptions
  - E.g. DataRetrievalFailureException – Data could not be retrieved
- Spring converts data access technology errors to subclasses in this hierarchy
  - Spring understands some database specific error codes and ORM specific exceptions
- Spring exceptions are more descriptive



DataAccessException

The Spring DataAccessException hierarchy allows higher layers to code to technology independent exception classes, further supporting the ability to change data access technologies without impacting higher layers. Springs exceptions can be more descriptive than the underlying database error codes/exceptions because the creators of Spring have gone to great lengths to understand and handle database errors from different vendors.

Some examples exceptions include:

TypeMismatchDataAccessException – Type mismatch between Java type and data type

DataIntegrityViolationException – A write operation resulted in a database integrity violation of some kind (e.g. foreign key violation)

DeadlockLoserDataAccessException – The current process entered a database deadlock and was chosen to be the loser

1.9 @Repository Annotation

- The DataAccessException translation described previously can easily be added to a DAO implementation by adding the @Repository annotation
- This annotation is one of the Spring "stereotype" annotations for declaring Spring components with an annotation
- This annotation is the only code that needs to be added to a class to enable the Exception translation functionality

```
@Repository
public class PurchaseDAOJDBCImpl implements PurchaseDAO {
```

1.10 Using DataSources

- Spring Boot will auto-configure a datasource based on the 'application.properties' file

```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

- All you need to do then is annotate a DataSource property with '@Autowired'
- For extra convenience, Spring Boot also sets up a JdbcTemplate and NamedParameterJdbcTemplate instance
  - Again, just declare a reference to the JdbcTemplate type and annotate it '@AutoWired'

Using DataSources

A DataSource provides Connection objects to the underlying database and often provide a connection pool for resource management purposes.

The jndi:lookup element is new to Spring 2.x. It requires the use of the new XML schema-based configuration syntax. In the example on the slide, a DataSource object is looked up using JNDI and assigned to the id "dataSource".

DriverManagerDataSource is a simple data source that uses a DriverManager. It is useful for testing. It has properties for driverClassName, url, username, and password.

1.11 DAO Templates

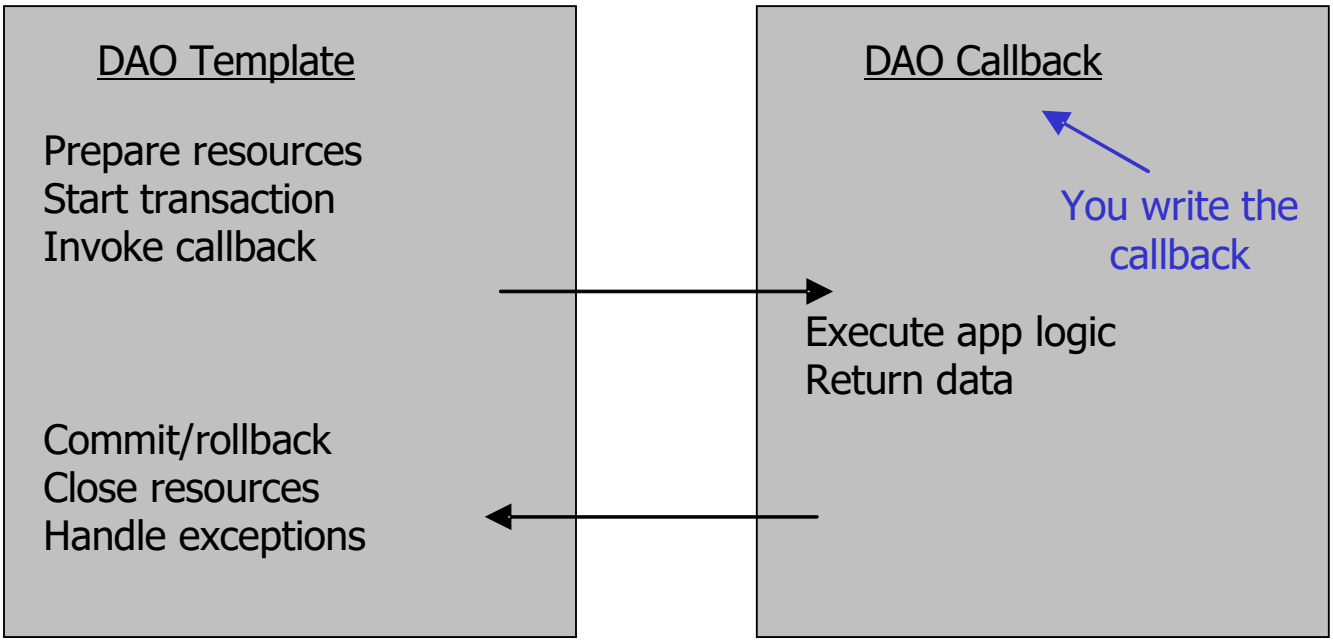
- Data access logic often consists mostly of infrastructural code
  - Resource management, exception handling, connection management, etc
- Spring uses the template method pattern to provide the infrastructural logic for you
- The developer focuses on creating the application-specific logic using a

"DAO callback"

DAO Templates

The template method design pattern is illustrated in the classic book "Design Patterns – Elements of Reusable Object-Oriented Software" by Erich Gamma, et al (the "Gang of Four").

1.12 DAO Templates and Callbacks



DAO Templates and Callbacks

The diagram on the slide illustrates templates and callbacks. The DAO template is responsible for all of the data access "plumbing" logic such as resource management and exception handling. Spring provides this for you.

The DAO callback is where the application developer writes his/her application specific data access logic.

1.13 ORM Tool Support in Spring

- ORM (object/relational mapping) frameworks provide functionality over JDBC such as:
  - Lazy loading, caching, cascading updates, etc
- Spring provides integration support for several ORM frameworks including:
  - Hibernate, JDO, iBATIS SQL Maps, Apache OJB
- Spring also provides services on top of these frameworks including:
  - Transaction management
  - Exception handling
  - Template classes
  - Resource management

ORM Tool Support in Spring

Spring does not provide an ORM framework since several excellent frameworks already exist. However, it does provide sophisticated integration with many of these frameworks as well as adding additional services on top of them.

1.14 Summary

- Spring provides modules to simplify data access code
  - These modules are some of the most used features of Spring
- No matter what technology is used Spring provides a common architecture so it is easy to switch data access technologies with minimum impact on the rest of an application
- The Spring @Repository annotation should be used on DAO components to enable Spring's data Exception translation