

Database Access Using Hibernate, JPA2 and iBATIS

Duration:	4 days
Type:	intermediate

Description

Java developers traditionally use handwritten JDBC code for database access, in the same way that C# developers use ADO or Perl developers DBI. However this low level approach is increasingly unworkable as Enterprise Java applications grow in size and complexity.

Developers on modern JSE and JEE projects almost always use frameworks to simplify the task of moving data from objects to tables and vice versa. These frameworks can be divided into those that simplify the JDBC API for ease of use and those that act as full object-relational mapping tools.

This course introduces Java developers to the three most commonly used frameworks for database access, these being iBATIS, Hibernate and the JPA. During the course delegates implement a full data access layer for a sample application using each technology.

Prerequisites

Delegates must be experienced Java developers who are familiar with the core JDBC API and SQL

List of Modules

Core Java Database Concepts

- Running SQL statements using JDBC
- Transaction support built into JDBC
- Distributed transactions, XA and 2 phase commit
- Using the Java Transaction API (JTA)

Simplifying Database Access with iBATIS

- Installing and configuring iBATIS
- Using mapped statements to create objects
- Running inserts, updates and deletes
- Working with stored procedures
- Dynamic SQL and caching

The Evolution of ORM Frameworks

- Problems caused by the Object Relational Mismatch
- Key benefits of automating data access using tools like Hibernate
- The evolution of EJB3 and the Java Persistence API (JPA)
- Comparing features offered by Hibernate, JPA and JPA 2

Creating Hibernate Based Projects

- Adding Hibernate and its dependencies
- Creating the *hibernate.cfg.xml* configuration file
- Choosing the driver, dialect and logging settings

Specifying Mapping Information

- Creating and organizing XML mapping files for classes
- Specifying mapping information using JPA annotations
- Associating classes with tables and fields with columns
- Choosing an appropriate strategy for generating object ids
- Mapping basic (many-to-one) relations between objects

Development Using the Hibernate API

- Creating *SessionFactory* and *Session* objects
- Starting and committing transactions
- Initializing and saving persistent objects
- Retrieving a persistent object via its key
- Navigating associations between persistent objects
- Finding groups of objects via HQL queries
- Using native SQL queries instead of HQL

Development Using the JPA EntityManager

- Acquiring an entity manager in JSE and JEE applications
- Entity managers, transactions and persistence contexts
- Persisting, loading, removing, detaching and merging objects
- Finding objects using the Java Persistence Query Language (JPQL)

Designing a Persistence Layer Part 1

- Differentiating between application and database transactions
- How objects are categorized as transient, persistent or detached
- Defining how Hibernate treats associations via the *cascade* attribute
- Deciding if, when and how to store sessions within a Web Application

Designing a Persistence Layer Part 2

- Producing a fine grained object model using value types and components
- Specifying and working with Sets, Lists and Maps of value types
- Modeling one-to-many and many-to-many associations
- Breaking down many-to-many relationships
- Different ways of modeling inheritance

Tuning Hibernate for Performance

- Deciding whether to load objects eagerly or lazily
- Choosing the correct level of transaction isolation
- Controlling the level 1 cache by evicting objects
- Picking and configuring a level 2 cache implementation
- Special considerations that apply to the query cache