Library DAO Tutorial

Step 7: Hibernate Version – Model and DAO Interface Changes

The Learning House

427 S 4th Street #300

Louisville KY 40202

Step 7: Hibernate Version – Model and DAO Interface Changes

Overview

For illustrative purposes, we will implement a parallel Model and DAO for Hibernate. We will create another DAO interface and implementation and we will create a new Model object to represent Books (the changes to the other model objects can coexist with the JdbcTemplate implementation).

HBook

We'll start with the Model object for our Book. You will notice two main differences in this new Model object:

- We now have references to Publisher and Author objects rather than publisher and author ids.
- Our model object contains Hibernate/JPA annotations that tell Hibernate how and where to map our data into the database.

Notes:

- @Entity marks the class as an entity to the JPA/Hibernate framework.
- @Table indicates the table to which this entity maps.
- @Id indicates that the annotated property represents the primary key.
- @GeneratedValue indicates that this value is determined by the database.
- @Column indicates the column to which the annotated property maps.
- @ManyToOne indicates that the annotated property is part of a many-to-one relationship.
- @JoinColumn indicates the column in this table that contains the foreign key for the many-to-one relationship.
- @ManyToMany indicates that the annotated property (in this case a List of Authors) is part of a many-to-many relationship.
    - **fetch** indicates when these related objects should be retrieved from the database. Can be EAGER (get them right away) or LAZY (only get them if someone actually tries to access the objects).
    - **cascade** indicates what should be done when inserting/updating/deleting objects in the relationship.
- @JoinTable indicates which bridge table should be used for the many-to-many relationship.
    - joinColumns indicates which column(s) in the bridge table represents this object (Book, in this case).
    - inverseJoinColumns indicates which column(s) in the bridge table represents the other object in the relationship (Author, in this case).
- We are using a Set to hold the Author objects for this book. An HBook object contains its associated Authors and an Author object contains its associated HBooks, which means that there are times when we are asking Hibernate to get both of these groups at once. A Set is used in this case because Hibernate cannot fetch two Lists at once, but it can fetch two Sets at once.

```java
package com.swcguild.library.model;

import java.math.BigDecimal;
import java.util.Date;
import java.util.List;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

/**
 *
 * @author apprentice
 */
@Entity
@Table(name = "books")
public class HBook {

    @Id
    @GeneratedValue
    @Column(name = "book_id")
    private int bookId;
    @Column(name = "isbn")
    private String isbn;
    @Column(name = "title")
    private String title;
    @ManyToOne
    @JoinColumn(name = "publisher_id")
    private Publisher publisher;
    @ManyToMany(fetch = FetchType.EAGER, cascade = {CascadeType.ALL})
    @JoinTable(name = "books_authors",
            joinColumns = {
                @JoinColumn(name = "book_id")},
            inverseJoinColumns = {
                @JoinColumn(name = "author_id")})
    private Set<Author> authors;
    @Column(name = "price")
    private BigDecimal price;
    @Column(name = "publish_date")
    private Date publishDate;

    public int getBookId() {
        return bookId;
    }
    public void setBookId(int bookId) {
        this.bookId = bookId;
    }
    public String getIsbn() {
        return isbn;
    }
    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
```

```
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public Publisher getPublisher() {
        return publisher;
    }
    public void setPublisher(Publisher publisher) {
        this.publisher = publisher;
    }
    public Set<Author> getAuthors() {
        return authors;
    }
    public void setAuthors(Set<Author> authors) {
        this.authors = authors;
    }
    public BigDecimal getPrice() {
        return price;
    }
    public void setPrice(BigDecimal price) {
        this.price = price;
    }
    public Date getPublishDate() {
        return publishDate;
    }
    public void setPublishDate(Date publishDate) {
        this.publishDate = publishDate;
    }
}
```

Author

Next, we'll make changes to the Author class. We don't need to make a separate HAuthor class because the changes required for Hibernate can coexist with the JdbcTemplate implementation. We'll make two changes:

1. Add Hibernate annotations for all fields.
2. Add a field to hold all the books that the author has written; this will be a Set of HBooks.

```
package com.swcguild.library.model;

import java.util.Set;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

/**
 *
 * @author apprentice
 */
@Entity
@Table(name = "authors")
public class Author {
```

```
@Id
@GeneratedValue
@Column(name = "author_id")
private int authorId;
@Column(name = "first_name")
private String firstName;
@Column(name = "last_name")
private String lastName;
@Column(name="street")
private String street;
@Column(name="city")
private String city;
@Column(name="state")
private String state;
@Column(name="zip")
private String zip;
@Column(name="phone")
private String phone;
@ManyToMany(fetch = FetchType.EAGER, mappedBy="authors")
private Set<HBook> books;

public int getAuthorId() {
    return authorId;
}
public void setAuthorId(int authorId) {
    this.authorId = authorId;
}
public String getFirstName() {
    return firstName;
}
public void setFirstName(String firstName) {
    this.firstName = firstName;
}
public String getLastName() {
    return lastName;
}
public void setLastName(String lastName) {
    this.lastName = lastName;
}

public String getStreet() {
    return street;
}
public void setStreet(String street) {
    this.street = street;
}
public String getCity() {
    return city;
}
public void setCity(String city) {
    this.city = city;
}
public String getState() {
    return state;
}
public void setState(String state) {
    this.state = state;
}
public String getZip() {
    return zip;
}
```

```
    public void setZip(String zip) {
        this.zip = zip;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
     public Set<HBook> getBooks() {
         return books;
     }

     public void setBooks(Set<HBook> books) {
         this.books = books;
     }
}
```

Publisher

Next, we'll make changes to the Publisher class. We don't need to make a separate HPublisher class because the changes required for Hibernate can coexist with the JdbcTemplate implementation. We'll make two changes:

1. Add Hibernate annotations for all fields.
2. Add a field to hold all the books that the publisher has published; this will be a Set of HBooks.

```
package com.swcguild.library.model;

import java.util.List;

import java.util.Set;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.Table;
```

```
/**
*
* @author apprentice
*/

@Entity
@Table(name="publishers")
public class Publisher {

    @Id
    @GeneratedValue
    @Column(name="publisher_id")
    private int publisherId;
    @Column(name="name")
    private String name;
    @Column(name="street")
    private String street;
    @Column(name="city")
    private String city;
    @Column(name="state")
    private String state;
    @Column(name="zip")
    private String zip;
    @Column(name="phone")
    private String phone;
    @OneToMany(fetch = FetchType.EAGER, mappedBy = "publisher")
    private Set<HBook> books;
```

```
public Set<HBook> getBooks() {
    return books;
}

public void setBooks(Set<HBook> books) {
    this.books = books;
}

public int getPublisherId() {
    return publisherId;
}

public void setPublisherId(int publisherId) {
    this.publisherId = publisherId;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getStreet() {
    return street;
}
```

```
public void setStreet(String street) {
    this.street = street;
}

public String getCity() {
    return city;
}

public void setCity(String city) {
    this.city = city;
}

public String getState() {
    return state;
}

public void setState(String state) {
    this.state = state;
}

public String getZip() {
    return zip;
}

public void setZip(String zip) {
    this.zip = zip;
}
```

```
    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }
}
```

HLibraryDAO

The DAO interface has to change in three ways:

1. We need to reference HBook instead of Book
2. The Delete operations now take an object reference instead of an id
3. The methods getAuthorsByBookId and getPublisherByBookId are no longer needed because an HBook object already has a List of all of its Author objects and a reference to its Publisher object

```
package com.swcguild.library.dao;

import com.swcguild.library.model.Author;
import com.swcguild.library.model.HBook;
import com.swcguild.library.model.Publisher;
import java.util.List;

public interface HLibraryDao {
        public void addAuthor(Author author);
    public void deleteAuthor(Author author);
    public void updateAuthor(Author author);
    public Author getAuthorById(int id);
    // Just get the book - the author objects will be there...
    //public List<Author> getAuthorsByBookId(int bookId);
    public List<Author> getAllAuthors();

    public void addBook(HBook book);
    public void deleteBook(HBook book);
    public void updateBook(HBook book);
    public HBook getBookById(int id);
    //public List<Book> getBooksByAuthorId(int authorId);
    //public List<Book> getBooksByPublisherId(int publisherId);
    public List<HBook> getAllBooks();

    public void addPublisher(Publisher publisher);
    public void deletePublisher(Publisher publisher);
    public void updatePublisher(Publisher publisher);
    public Publisher getPublisherById(int id);
    //public Publisher getPublisherByBookId(int bookId);
    public List<Publisher> getAllPublishers();
}
```