Hibernate Tutorial 05 Collection Mapping

By Gary Mak
hibernatetutorials@metaarchit.com
September 2006

1. Mapping collections of values

In our online bookshop application, we want to show the chapter information of a book to the users before they make a decision of buying it. Each book may contain many chapters. We first treat each chapter as a string (title only) and use a java.util.List to store all the chapters of a book. List is one kind of "collections" in the Java collections framework. The other kinds of collections will be introduced later. For our collections are used to store values (e.g. string, integer, double) but not persistent objects, they are also called "collections of values".

```
public class Book {
    private Long id;
    private String isbn;
    private String name;
    private Publisher publisher;
    private Date publishDate;
    private Integer price;
    private List chapters;

    // Getters and Setters
}
```

To define a collection property, we first choose which type of collection to be used. The simplest collection type in Hibernate may be <bag>. It is a list of objects without ordering, and which can contain duplicated elements. The corresponding type in Java is java.util.List. Moreover, for concentrating on the collection itself, let's ignore the many-to-one association temporarily.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        <id name="id" type="long" column="ID">
            <generator class="native"/>
        </id>
        <property name="isbn" type="string">
            <column name="ISBN" length="50" not-null="true" unique="true" />
        </property>
        <property name="name" type="string">
            <column name="BOOK_NAME" length="100" not-null="true" />
        </property>
        <property name="publishDate" type="date" column="PUBLISH_DATE" />
        <property name="price" type="int" column="PRICE" />
```

```
        <many-to-one name="publisher" class="Publisher" column="PUBLISHER_ID"
            lazy="false" fetch="join" cascade="save-update,delete" />
        <bag name="chapters" table="BOOK_CHAPTER">
            <key column="BOOK_ID" />
            <element column="CHAPTER" type="string" length="100" />
        </bag>
    </class>
</hibernate-mapping>
```

This mapping will use a separate table BOOK_CHAPTER for storing the chapter information. It contains two columns, BOOK_ID and CHAPTER. Each chapter is stored as a row in this table. Don't forget to run the schema update task for reflecting the changes to database.

1.1. Lazy initialization

Suppose we use the following code to retrieve a book object from database. The collection of chapters should also be retrieved at the same time.

```
Session session = factory.openSession();
try {
    Book book = (Book) session.get(Book.class, id);
    return book;
} finally {
    session.close();
}
```

But when we access the chapter collection through book.getChapters() outside the session, an exception will occur.

```
for (Iterator iter = book.getChapters().iterator(); iter.hasNext();) {
    String chapter = (String) iter.next();
    System.out.println(chapter);
}
```

The reason for this exception is due to the "lazy initialization" of the collection. We can initialize it explicitly for accessing outside the session.

```
Session session = factory.openSession();
try {
    Book book = (Book) session.get(Book.class, id);
    Hibernate.initialize(book.getChapters());
    return book;
} finally {
    session.close();
}
```

Or we can even turn off the lazy initialization for this collection. You must consider carefully before you do that, especially for a collection.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        ...
        <bag name="chapters" table="BOOK_CHAPTER" lazy="false">
            <key column="BOOK_ID" />
            <element column="CHAPTER" type="string" length="100" />
        </bag>
    </class>
</hibernate-mapping>
```

1.2. Fetching strategies

By default, the fetch mode of a collection is “select”. Hibernate will issue two SELECT statements for querying the book object and the chapter collection separately. We can change the fetch mode to “join” telling Hibernate to issue a single SELECT statement with table join.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        ...
        <bag name="chapters" table="BOOK_CHAPTER" lazy="false" fetch="join">
            <key column="BOOK_ID" />
            <element column="CHAPTER" type="string" length="100" />
        </bag>
    </class>
</hibernate-mapping>
```

In HQL, we need to use a special syntax to specify the joining fetch strategy. This can also force the collection to be initialized, if it is lazy.

```
Session session = factory.openSession();
try {
    Query query = session.createQuery(
        "from Book book left join fetch book.chapters where book.isbn = ?");
    query.setString(0, isbn);
    Book book = (Book) query.uniqueResult();
    return book;
} finally {
    session.close();
}
```

You must be careful when choosing the fetch strategy for a collection, because it will multiply your result set by the collection size. Keep in mind that you should use “select” fetch for large collection.

2. Collection types in Hibernate

So far, we have introduced the <bag> collection type in Hibernate. There are actually many other collection types supported by Hibernate. Each of them has different characteristics.

2.1. Bag

A <bag> is an unordered collection, which can contain duplicated elements. That means if you persist a bag with some order of elements, you cannot expect the same order retains when the collection is retrieved. There is not a “bag” concept in Java collections framework, so we just use a java.util.List to correspond to a <bag>.

A table BOOK_CHAPTER will be used to persist the following <bag> collection. It contains two columns, BOOK_ID and CHAPTER. Each element will be stored as a row in this table.

```
public class Book {
    ...
    private List chapters;
}

<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        ...
        <bag name="chapters" table="BOOK_CHAPTER">
            <key column="BOOK_ID" />
            <element column="CHAPTER" type="string" length="100" />
        </bag>
    </class>
</hibernate-mapping>
```

2.2. Set

A <set> is very similar to a <bag>. The only difference is that set can only store unique objects. That means no duplicated elements can be contained in a set. When you add the same element to a set for second time, it will replace the old one. A set is unordered by default but we can ask it to be sorted, which we will discuss later. The corresponding type of a <set> in Java is java.util.Set.

The table structure of a <set> is the same as a <bag>, i.e. contains two columns, BOOK_ID and CHAPTER.

```
public class Book {
    ...
    private Set chapters;
}
```

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        ...
        <set name="chapters" table="BOOK_CHAPTER">
            <key column="BOOK_ID" />
            <element column="CHAPTER" type="string" length="100" />
        </set>
    </class>
</hibernate-mapping>
```

2.3. List

A <list> is an indexed collection where the index will also be persisted. That means we can retain the order of the list when it is retrieved. It differs from a <bag> for it persists the element index while a <bag> does not. The corresponding type of a <list> in Java is java.util.List.

The table structure for a <list> has one more column, CHAPTER_INDEX, than a <bag>. It is used for storing the element index. The type of this column is int.

```
public class Book {
    ...
    private List chapters;
}
```

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        ...
        <list name="chapters" table="BOOK_CHAPTER">
            <key column="BOOK_ID" />
            <list-index column="CHAPTER_INDEX"/>
            <element column="CHAPTER" type="string" length="100" />
        </list>
    </class>
</hibernate-mapping>
```

2.4. Array

An <array> has the same usage as a <list>. The only difference is that it corresponds to an array type in Java, not a java.util.List. It is seldom used unless we are mapping for legacy applications. In most cases, we should use <list> instead. That is because the size of an array cannot be increased or decreased dynamically, where a list can.

The table structure of an <array> is the same as a <list>, i.e. contains three columns, BOOK_ID, CHAPTER and CHAPTER_INDEX.

```
public class Book {
    ...
    private String[] chapters;
}

<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        ...
        <array name="chapters" table="BOOK_CHAPTER">
            <key column="BOOK_ID" />
            <list-index column="CHAPTER_INDEX"/>
            <element column="CHAPTER" type="string" length="100" />
        </array>
    </class>
</hibernate-mapping>
```

2.5. Map

A <map> is very similar to a <list>. The difference is that a map uses arbitrary keys to index the collection, not an integer index using in a list. A map stores its entries in key/value pairs. You can lookup the value by its key. The key of a map can be of any data types. The corresponding type of a <map> in Java is java.util.Map.

The table structure of the following <map> contains three columns, BOOK_ID, CHAPTER and CHAPTER_KEY. Here we choose string as the data type of our key column.

```
public class Book {
    ...
    private Map chapters;
}

<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        ...
        <map name="chapters" table="BOOK_CHAPTER">
            <key column="BOOK_ID" />
            <map-key column="CHAPTER_KEY" type="string" />
            <element column="CHAPTER" type="string" length="100" />
        </map>
    </class>
</hibernate-mapping>
```

3. Sorting the collections

Sometimes we want our collections can be sorted automatically when retrieval. Hibernate supports two ways of sorting a collection.

3.1. Sorting in memory

The first way of sorting a collection is utilizing the sorting features provided by the Java collections framework. The sorting occurs in the memory of JVM which running Hibernate, after the data being read from database. Notice that for large collections this kind of sorting may not be efficient. Only <set> and <map> supports this kind of sorting.

To ask a collection to be sorted in natural order, we define the sort attribute to be "natural" in the collection. Hibernate will use the compareTo() method defined in the java.lang.Comparable interface to compare the elements. Many basic data types, such as String, Integer and Double have implemented this interface.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        ...
        <set name="chapters" table="BOOK_CHAPTER" sort="natural">
            <key column="BOOK_ID" />
            <element column="CHAPTER" type="string" length="100" />
        </set>
    </class>
</hibernate-mapping>
```

If you don’t satisfy with the natural ordering, you can write your own comparator instead. You write a custom comparator by implementing the java.util.Comparator interface. The comparing logic should be put inside the overridden compare() method. To make use of this comparator, you pass it to the sort attribute of the collection.

```
public class ChapterComparator implements Comparator {
    public int compare(Object o1, Object o2) {
        // if o1 and o2 don’t instantiate the same class, throw an exception
        // if o1 is less than o2, return a negative number
        // if o1 is equal to o2, return a zero
        // if o1 is greater than o2, return a positive number
        ...
    }
}

<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        ...
```

```
        <set name="chapters" table="BOOK_CHAPTER"
            sort="com.metaarchit.bookshop.ChapterComparator">
            <key column="BOOK_ID" />
            <element column="CHAPTER" type="string" length="100" />
        </set>
    </class>
</hibernate-mapping>
```

3.2. Sorting in database

If your collection is very large, it will be more efficient to sort it in the database. We can specify the order-by condition for sorting this collection when retrieval. Notice that this order-by attribute should be a SQL column, not a property name in Hibernate.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        ...
        <set name="chapters" table="BOOK_CHAPTER" order-by="CHAPTER">
            <key column="BOOK_ID" />
            <element column="CHAPTER" type="string" length="100" />
        </set>
    </class>
</hibernate-mapping>
```

In fact, we can use any SQL expression, which is valid in the “order by” clause, for this order-by attribute, e.g. using commas to separate multiple columns, using “asc” or “desc” to specify in ascending or descending order. Hibernate will copy this order-by attribute to the “order by” clause during SQL generation.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Book" table="BOOK">
        ...
        <set name="chapters" table="BOOK_CHAPTER" order-by="CHAPTER desc">
            <key column="BOOK_ID" />
            <element column="CHAPTER" type="string" length="100" />
        </set>
    </class>
</hibernate-mapping>
```