Hibernate Tutorial 07 Component Mapping

By Gary Mak
hibernatetutorials@metaarchit.com
September 2006

1. Using a component

In our online bookshop application, a customer can place an order for purchasing some books. Our staff will process his order and deliver the books to him. The customer can specify different recipients and contact details for different day periods (weekdays and holidays). First, we add a new persistent class Order to our application.

```
public class Order {
    private Long id;
    private Book book;
    private Customer customer;
    private String weekdayRecipient;
    private String weekdayPhone;
    private String weekdayAddress;
    private String holidayRecipient;
    private String holidayPhone;
    private String holidayAddress;

    // Getters and Setters
}
```

Then we create a mapping definition for this persistent class. We just map the properties of this class as usual.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Order" table="BOOK_ORDER">
        <id name="id" type="long" column="ID">
            <generator class="native" />
        </id>
        ...
        <property name="weekdayRecipient" type="string" column="WEEKDAY_RECIPIENT" />
        <property name="weekdayPhone" type="string" column="WEEKDAY_PHONE" />
        <property name="weekdayAddress" type="string" column="WEEKDAY_ADDRESS" />
        <property name="holidayRecipient" type="string" column="HOLIDAY_RECIPIENT" />
        <property name="holidayPhone" type="string" column="HOLIDAY_PHONE" />
        <property name="holidayAddress" type="string" column="HOLIDAY_ADDRESS" />
    </class>
</hibernate-mapping>
```

One may feel that our Order class is not well designed, since the “recipient”, “phone”, “address” properties have been duplicated two times for weekdays and holidays. From the object-oriented perspective, we should create a class say Contact to encapsulate them.

```
public class Contact {
    private String recipient;
    private String phone;
    private String address;

    // Getters and Setters
}

public class Order {
    ...
    private Contact weekdayContact;
    private Contact holidayContact;

    // Getters and Setters
}
```

Now the changes are done for Java. But how can we modify the Hibernate mapping definition to reflect the changes? According to the techniques we have learned before, we can specify Contact as a new persistent class and use a one-to-one association (the simplest way is to use a <many-to-one> association with unique="true") to associate Order and Contact.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Contact" table="CONTACT">
        <id name="id" type="long" column="ID">
            <generator class="native" />
        </id>
        <property name="recipient" type="string" column="RECIPIENT" />
        <property name="phone" type="string" column="PHONE" />
        <property name="address" type="string" column="ADDRESS" />
    </class>
</hibernate-mapping>

<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Order" table="BOOK_ORDER">
        ...
        <many-to-one name="weekdayContact" class="Contact" column="CONTACT_ID"
            unique="true" />
        <many-to-one name="holidayContact" class="Contact" column="CONTACT_ID"
            unique="true" />
    </class>
</hibernate-mapping>
```

In this case, modeling the Contact class as a standalone persistent class seems not suitable. This is because it is meaningless once departed from an order. Its function is much on grouping some values logically. It should not be a complete persistent object with object identifier. Hibernate is providing a concept called “components” for mapping this kind of objects.

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Order" table="BOOK_ORDER">
        ...
        <component name="weekdayContact" class="Contact">
            <property name="recipient" type="string" column="WEEKDAY_RECIPIENT" />
            <property name="phone" type="string" column="WEEKDAY_PHONE" />
            <property name="address" type="string" column="WEEKDAY_ADDRESS" />
        </component>
        <component name="holidayContact" class="Contact">
            <property name="recipient" type="string" column="HOLIDAY_RECIPIENT" />
            <property name="phone" type="string" column="HOLIDAY_PHONE" />
            <property name="address" type="string" column="HOLIDAY_ADDRESS" />
        </component>
    </class>
</hibernate-mapping>
```

There is no new persistent object introduced. All the columns mapped for these components are on the same table as their parent object. Components do not have an identity, and exist only if their parent does. They are most suitable for grouping several properties as a single object.

2. Nested components

Components can be even defined to be nested, i.e. components embedded within other components. For example, we can define the phone property as another component and embed it into the contact component.

```
public class Phone {
    private String areaCode;
    private String telNo;

    // Getters and Setters
}

public class Contact {
    ~~private String phone;~~
    private Phone phone;

    // Getters and Setters
}
```

```
<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Order" table="BOOK_ORDER">
        ...
        <component name="weekdayContact" class="Contact">
            <property name="recipient" type="string" column="WEEKDAY_RECIPIENT" />
            <property name="phone" type="string" column="WEEKDAY_PHONE" />
            <component name="phone" class="Phone">
                <property name="areaCode" type="string" column="WEEKDAY_PHONE_AREA_CODE" />
                <property name="telNo" type="string" column="WEEKDAY_PHONE_TEL_NO" />
            </component>
            <property name="address" type="string" column="WEEKDAY_ADDRESS" />
        </component>
        <component name="holidayContact" class="Contact">
            ...
        </component>
    </class>
</hibernate-mapping>
```

3. References in component

A component can have a reference to its parent object through a <parent> mapping.

```
public class Contact {
    private Order order;
    private String recipient;
    private Phone phone;
    private String address;

    // Getters and Setters
}

<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Order" table="BOOK_ORDER">
        ...
        <component name="weekdayContact" class="Contact">
            <parent name="order" />
            <property name="recipient" type="string" column="WEEKDAY_RECIPIENT" />
            <component name="phone" class="Phone">
                <property name="areaCode" type="string" column="WEEKDAY_PHONE_AREA_CODE" />
                <property name="telNo" type="string" column="WEEKDAY_PHONE_TEL_NO" />
            </component>
            <property name="address" type="string" column="WEEKDAY_ADDRESS" />
        </component>
    </class>
</hibernate-mapping>
```

A component can be used to group not only normal properties, but also many-to-one and one-to-one associations. Suppose we want to associate the address of an order to the address in our customer database.

```
public class Contact {
    private Order order;
    private String recipient;
    private Phone phone;
    ~~private String address;~~
    private Address address;

    // Getters and Setters
}

<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Order" table="BOOK_ORDER">
        ...
        <component name="weekdayContact" class="Contact">
            ...
            ~~<property name="address" type="string" column="WEEKDAY_ADDRESS" />~~
            <many-to-one name="address" class="Address" column="WEEKDAY_ADDRESS_ID" />
        </component>
    </class>
</hibernate-mapping>
```

4. Collection of components

Suppose we need to support a more flexible contact mechanism for our book ordering. A customer can specify several contact points for a book delivery as he may not sure which one is most suitable for a specified time period. Our staff will try to contact these points one by one when they deliver the books. We use a java.util.Set to hold all the contact points for an order.

```
public class Order {
    ...
    ~~private Contact weekdayContact;~~
    ~~private Contact holidayContact;~~
    private Set contacts;

    // Getters and Setters
}
```

To map many contact points for an order in Hibernate, we can use a collection of components. We use <composite-element> to define the components in a collection. For simplicity, we first rollback our Contact class to the original form.

```
public class Contact {
    private String recipient;
    private String phone;
    private String address;

    // Getters and Setters
}

<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Order" table="BOOK_ORDER">
        ...
        <set name="contacts" table="ORDER_CONTACT">
            <key column="ORDER_ID" />
            <composite-element class="Contact">
                <property name="recipient" type="string" column="RECIPIENT" />
                <property name="phone" type="string" column="PHONE" />
                <property name="address" type="string" column="ADDRESS" />
            </composite-element>
        </set>
    </class>
</hibernate-mapping>
```

In additional to normal properties, we can define nested components and associations in a collection as well. We use <nested-composite-element> to define nested components in a collection. Let's perform the necessary changes to our Contact class first.

```
public class Contact {
    private String recipient;
    private Phone phone;
    private Address address;

    // Getters and Setters
}

<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Order" table="BOOK_ORDER">
        ...
        <set name="contacts" table="ORDER_CONTACT">
            <key column="ORDER_ID" />
            <composite-element class="Contact">
                <property name="recipient" type="string" column="RECIPIENT" />
                <property name="phone" type="string" column="PHONE" />
                <nested-composite-element name="phone" class="Phone">
                    <property name="areaCode" type="string" column="PHONE_AREA_CODE" />
                    <property name="telNo" type="string" column="PHONE_TEL_NO" />
                </nested-composite-element>
```

```
                <property name="address" type="string" column="ADDRESS" />
                <many-to-one name="address" class="Address" column="ADDRESS_ID" />
            </composite-element>
        </set>
    </class>
</hibernate-mapping>
```

5. Using components as Map keys

Suppose we want to extend our collection of contact points to be of java.util.Map type. We use date periods as the keys of the map. The customer can now specify the most suitable contact point for a particular date period. We create a class Period to encapsulate the start date and end date of a period and use it as the key type of our map. This Period class is also a component since it should not be a standalone persistent object with identifier.

```
public class Period {
    private Date startDate;
    private Date endDate;

    // Getters and Setters
}

<hibernate-mapping package="com.metaarchit.bookshop">
    <class name="Order" table="BOOK_ORDER">
        ...
        <map name="contacts" table="ORDER_CONTACT">
            <key column="ORDER_ID" />
            <composite-map-key class="Period">
                <key-property name="startDate" type="date" column="START_DATE" />
                <key-property name="endDate" type="date" column="END_DATE" />
            </composite-map-key>
            <composite-element class="Contact">
                <property name="recipient" type="string" column="RECIPIENT" />
                <nested-composite-element name="phone" class="Phone">
                    <property name="areaCode" type="string" column="PHONE_AREA_CODE" />
                    <property name="telNo" type="string" column="PHONE_TEL_NO" />
                </nested-composite-element>
                <many-to-one name="address" class="Address" column="ADDRESS_ID" />
            </composite-element>
        </map>
    </class>
</hibernate-mapping>
```

For the map key component to work properly, we should override the equals() and hashCode() methods of the component class.

```
public class Period {
    ...
    public boolean equals(Object obj) {
        if (!(obj instanceof Period)) return false;
        Period other = (Period) obj;
        return new EqualsBuilder().append(startDate, other.startDate)
                                  .append(endDate, other.endDate)
                                  .isEquals();
    }

    public int hashCode() {
        return new HashCodeBuilder().append(startDate)
                                    .append(endDate)
                                    .toHashCode();
    }
}
```