2 Kopplingsregler

Koppling mellan objekt- och datamodell

Hibernate stödjer idag tre sätt att specificera kopplingen mellan modellerna:

- Med en kopplingsfil utformad i XML
  - ➜ Det vanliga är att skapa en separat kopplingsfil för varje klass av lagringsbara objekt
  - ➜ Kopplingsfilen är helt skild från källkoden
  - ➜ Kopplingsfilen ska ha suffixet `.hbm.xml`
  - ➜ Konventionen är att ge filen samma stamnamn som klassen, t ex `Track.hbm.xml`
  - ➜ Placeras intill klassens objektkodsfil i katalogträdet under rotkatalogen för objektkod
- Med Hibernate XDoclet-anvisningar direkt i källkoden (före J2SE 5.0)
  - ➜ Ett verktyg genererar kopplingsfiler från anvisningarna
  - ➜ Kopplingsfilerna hanteras därefter som i första alternativet
- Med Java Persistence API-anvisningar direkt i källkoden (fr o m J2SE 5.0)
  - ➜ Anvisningarna finns med i den genererade objektkoden, där Hibernate kan läsa dem

Vi kommer i denna kurs endast att visa den första och tredje tekniken.

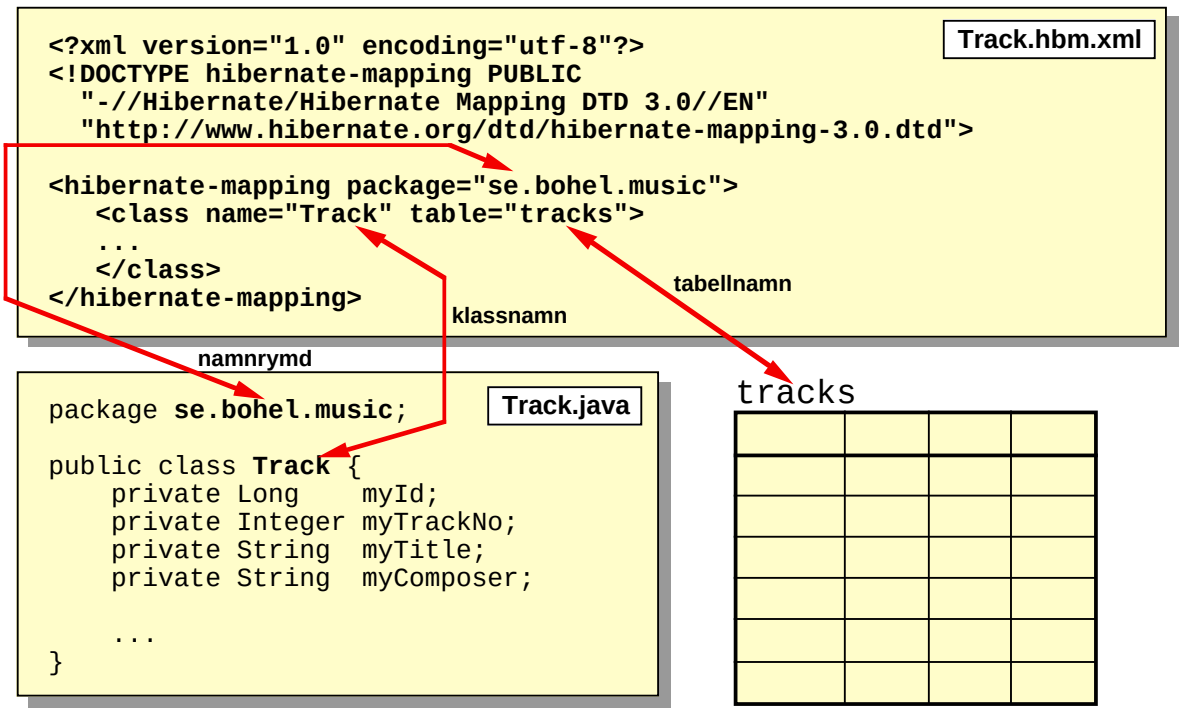
Nästa steg är att specificera kopplingen mellan objekt- och datamodell. I den nuvarande versionen av Hibernate stöds tre olika sätt att ange kopplingen:

- Det traditionella sättet, som funnits från början i Hibernate, är att specificera kopplingen i en särskild *kopplingsfil*. Det normala är att vi skapar en separat kopplingsfil för varje klass, även om det inte är ett krav. Kopplingsfiler är helt skilda från den vanliga källkoden och utformas i XML. De ska ha suffixet `.hbm.xml`, och konventionsmässigt ges de samma stamnamn som klassen de specificerar kopplingen för. Kopplingsfilen för klassen `Track` blir sålunda `Track.hbm.xml`. De placeras i samma katalog som klassens objektkodsfil befinner sig i när den placerats korrekt i katalogträdet under rotkatalogen för objektkod.
- Det finns också tekniker med vars hjälp kopplingen anges direkt i källkoden med hjälp av *anvisningar* (*annotations* i engelsk Java-terminologi). Före version 5.0 av Java 2 Standard Edition understöddes inte anvisningar i själva programspråket Java. En vanlig teknik var att tillföra dem med verktyget **XDoclet**. Det finns en uppsättning sådana XDoclet-anvisningar som kan användas för att ange kopplingen mellan objekt- och datamodell. Ett verktyg kan därefter generera en kopplingsfil per klass från anvisningarna. Före J2SE 5.0 är XDoclet-anvisningar den enda vägen att göra detta direkt i källkoden.
- Fr o m version 5.0 av Java 2 Standard Edition går det även att ange kopplingen direkt i källkoden med anvisningar från **Java Persistence API** (JPA), som bl a används för liknande syften fr o m version 5 av **Java Enterprise Edition**. Dessa anvisningar har stöd i språket och finns med i den genererade objektkoden, där Hibernate kan läsa dem direkt, utan att kopplingsfiler behöver genereras som mellansteg.

Vi kommer i denna kurs endast att visa den första och tredje tekniken, dvs hur kopplingsfiler i XML resp Java Persistence API-anvisningar används för att specificera kopplingen.

Kopplingsfiler i XML

Relationen mellan klass och databastabell



En kopplingsfil uttryckt i XML ska som alla XML-filer inledas med en rad som anger vilken version av XML vi utnyttjar:

```
<?xml version="1.0" encoding="utf-8"?>
```

XML-filen ska ha ett rotelement med namnet `hibernate-mapping`. På samma sätt som när vi konfigurerade Hibernate via en XML-fil finns det även här en DTD-fil som specificerar vilka element som får resp måste förekomma inom rotelementet `hibernate-mapping`. Därför fortsätter filen med ett `DOCTYPE`-direktiv som anger att det existerar en DTD-fil, och var den finns att hämta. Detta gör det möjligt för XML-verktyg att verifiera att filens innehåll inte enbart är giltig XML utan även stämmer överens med vad som får förekomma inom elementet `hibernate-mapping`.

Det finns flera attribut som kan användas för rotelementet `hibernate-mapping`. Det vanligaste är `package`, som anger en namnrymd som de klasser som nämns i kopplingsfilen tillhör. Om `package` angivits, blir det alltså möjligt att referera till dessa klasser utan att använda deras fullständiga namn. I vår fil refererar vi än så länge endast till klassen `Track`:

```
<hibernate-mapping package="se.bohel.music">
   ...
</hibernate-mapping>
```

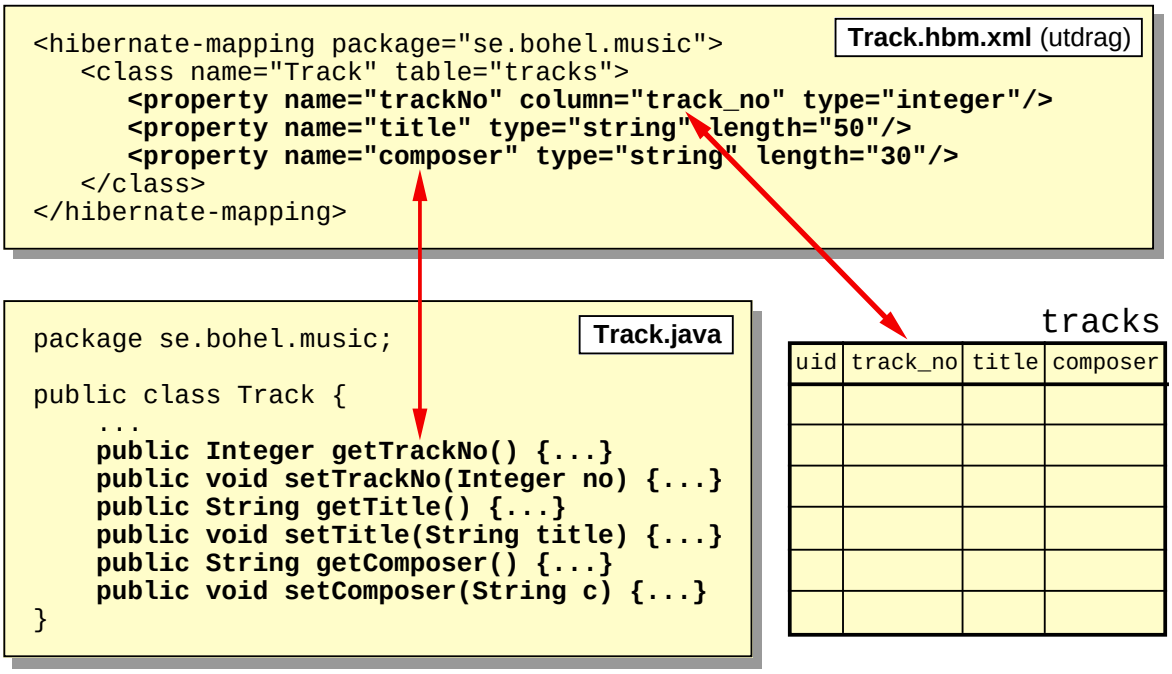
Det viktigaste underelementet till `hibernate-mapping` är `class`. Det är här vi anger vilken databastabell (attributet `table`) som motsvarar vår klass (attributet `name`):

```
<class name="Track" table="tracks">
```

Det finns en lång rad andra attribut som kan användas tillsammans med elementet `class`.

Kopplingsfiler i XML

Lagring av exponerade egenskaper



En central uppgift för en kopplingsfil är att specificera hur egenskaperna i de objekt som ska lagras ska transformeras till olika kolumner i en tabell i databasen. Detta anges framför allt med underelementet `property` till elementet `class` i kopplingsfilen:

```
<property name="trackNo" column="track_no" type="integer"/>
```

I XML är detta egentligen en förkortning av ett start- och ett slutelement utan underelement:

```
<property name="trackNo" column="track_no" type="integer">
</property>
```

De olika attributen anger detaljer om kopplingen:

- Attributet `name` anger egenskapens namn i klassen, vilket alltså betyder att en egenskap som t ex `trackNo` matchas mot åtkomsttjänsterna `getTrackNo()` resp `setTrackNo()` i klassen.
- Attributet `column` anger namnet på den kolumn (i den tabell som attributet `table` i elementet `class` anger) som egenskapens värde ska lagras i resp hämtas från. Om attributet inte anges, förutsätts kolumnen ha samma namn som egenskapen.
- Attributet `type` anger egenskapens typ, uttryckt som en av Hibernates typer (se nästa sida). Om attributet saknas, kommer Hibernate att försöka härleda attributet genom att inspektera klassens objektkod, men det är inte alltid det är tillräckligt.
- Därutöver kan en lång rad ytterligare attribut förekomma i särskilda fall. Exempelvis anger attributet `length` kolumnens bredd för de typer där detta måste anges, om inte förvalda värden godtas. I vårt fall gäller detta de kolumner som innehåller text.

Kopplingsfiler i XML

Egenskapstyper

Egenskaper i Hibernate kan vara av ett stort antal olika typer:

- Fördefinierade Hibernate-typer
  - ➜ Ett stort antal fördefinierade typer – se nästa sida!
- Motsvarande primitiva typer och klasser i Java
- Klasser i Java som implementerar gränssnittet `java.io.Serializable`
- Egendefinierade Hibernate-typer
  - ➜ Klasser som implementerar `org.hibernate.usertype.UserType`
  - ➜ Klasser som implementerar `org.hibernate.usertype.CompositeUserType`
  - ➜ Beskriver inte de värden som lagras, utan hur värden av någon annan klass lagras
  - ➜ En vanlig tillämpning är att efterlikna en uppräknad datatyp

Som vi såg på föregående sida kan det vara nödvändigt att ange typen för en egenskap i kopplingsfilen. Hibernate tillåter att vi använder ett ganska rikt system av typer:

- Det finns ett antal *fördefinierade Hibernate-typer*. De viktigaste av dessa presenteras på nästa sida, tillsammans med vilka typer och klasser de motsvarar i Java.
- De primitiva datatyper och klasser i Javas standardbibliotek som motsvarar de fördefinierade Hibernate-typerna kan också användas.
- Alla klasser i Java som implementerar gränssnittet `Serializable` från namnrymden `java.io` får användas som egenskapstyper.
- Den som har återkommande behov av en egen skräddarsydd typ kan utveckla en *egendefinierad Hibernate-typ*. En sådan är en klass som implementerar något av gränssnitten `UserType` eller `CompositeUserType` från namnrymden `org.hibernate.usertype`. Dessa klasser beskriver inte de objekt som lagras, utan hur objekt av en annan klass ska lagras. En vanlig tillämpning av detta är att skapa motsvarigheten till en uppräknad datatyp.

Kopplingsfiler i XML

Fördefinierade typer i Hibernate

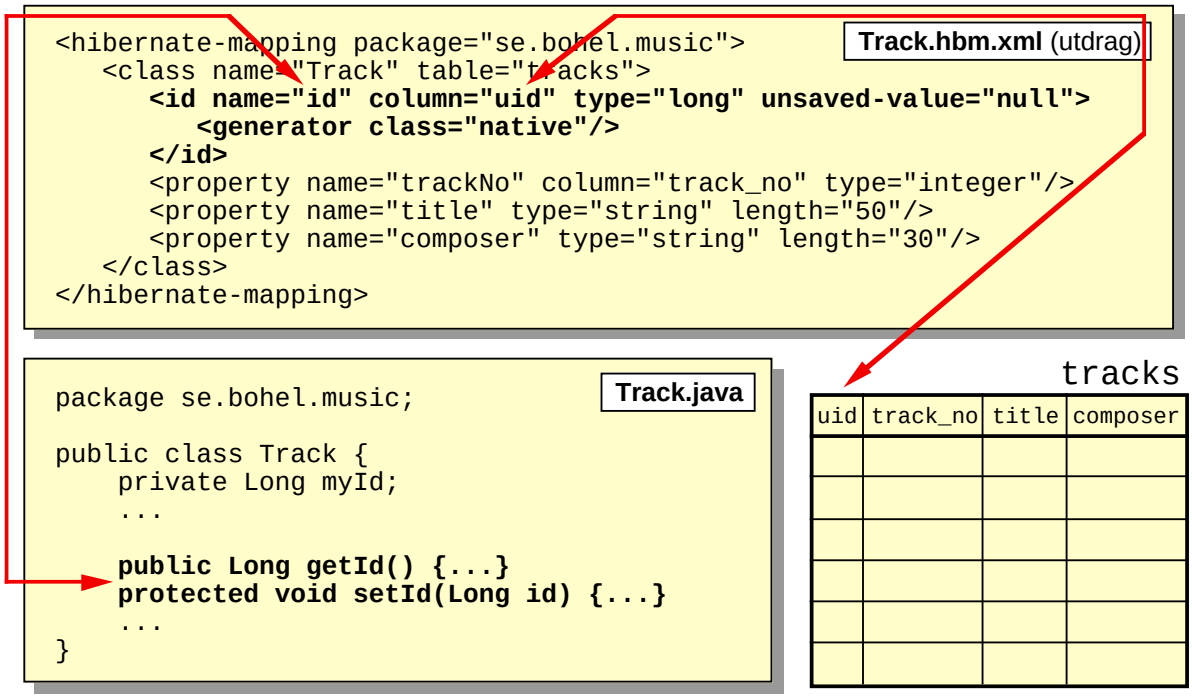
Typer i Hibernate	*Motsvarande typer i Java*
`integer`	`int`, `java.lang.Integer`
`long`	`long`, `java.lang.Long`
`double`	`double`, `java.lang.Double`
`boolean`, `yes_no`, `true_false`	`boolean`, `java.lang.Boolean`
`string`, `text`	`java.lang.String`
`date`, `time`, `timestamp`	`java.util.Date`
`calendar`, `calendar_date`	`java.util.Calendar`
`big_integer`	`java.math.BigInteger`
`big_decimal`	`java.math.BigDecimal`
`binary`	`byte[]`
`blob`	`java.sql.Blob`
`clob`	`java.sql.Clob`

Det finns ett stort antal fördefinierade typer i Hibernate. Tabellen ovan, som inte är uttömmande, visar några av de vanligaste samt vilka primitiva datatyper och klasser i Javas standardbibliotek som de motsvarar.

Som synes finns det ibland flera Hibernate-typer som motsvarar samma typ eller grupp av typer i Java. Detta är en förklaring till varför det kan vara nödvändigt att specificera den rätta typen i `property`-elementet i en kopplingsfil, trots att Hibernate har vissa möjligheter att härleda typen genom att inspektera objektkoden för klassen med hjälp av Javas **Reflection API**.

Kopplingsfiler i XML

Primära nycklar



Varje tabell som Hibernate ska kunna hantera måste ha en *primär nyckel*. Denna specificeras med underelementet `id` till elementet `class` i kopplingsfilen:

```
<id name="id" column="uid" type="long" unsaved-value="null">
```

- Attributet `name` anger nyckelns egenskapsnamn i klassen, vilket alltså betyder att en nyckel som t ex `id` matchas mot åtkomsttjänsterna `getId()` resp `setId()` i klassen. Om attributet saknas, tolkas detta som att nyckeln inte lagras i objekten (det var ju ett tillåtet om än inte rekommenderat alternativ).
- Attributet `column` anger namnet på den kolumn (i den tabell som attributet `table` i elementet `class` anger) som nyckelns värde ska lagras i resp hämtas från. Om attributet inte anges, förutsätts kolumnen ha namnet som attributet `name` anger (och som då måste vara explicit angivet).
- Attributet `type` anger nyckelns typ, uttryckt som en av Hibernates typer. Vilka typer en nyckel kan ha är delvis beroende av hur nyckeln skapas – se nedan!
- Attributet `unsaved-value` anger vilket värde Hibernate använder som tomt värde för en nyckel som ännu inte satts till sitt värde. Genom att utnyttja detta attribut kan det bli mer hanterbart att även använda primitiva datatyper som nycklar.

Underelementet `generator` specificerar hur nycklar genereras. Attributet `class` kan ges ett tiotal olika värden som beskriver det valda alternativet. Här har vi valt värdet `native`, vilket betyder att vi låter Hibernate välja en stabil form av genererade nycklar som understöds av den aktuella databasen. Vad som händer blir därmed beroende av den dialekt vi valt under konfigureringen. Detta tenderar att vara ett portabelt alternativ som drar så mycket nytta som möjligt av databasen. Ett annat värde är `assigned`, som innebär att programmet ansvarar för att tilldela nyckelns värde. Detta kan t ex vara nödvändigt att använda för att kunna integrera vårt program med en befintlig databas.