第2章

映射简介

我们已经开始使用Hibernate了。不过现在，我们要先停下向前迈进的脚步，站在一定高度观察一下Hibernate的全貌，以免迷失在Hibernate安装和配置的繁文缛节之中，这是非常必要的。像Java这样的面向对象语言提供了强大而方便的抽象层，可以在运行时以对象（类的实例）的形式来处理信息。这些对象之间可以通过各种方式链接起来，除了它们本身所拥有的原始数据外，还可以包含规则和行为。但是，当程序运行结束时，所有对象都会消失得无影无踪。

对于那些在程序多次运行之间需要保存下来的信息，或者是需要在不同程序或系统之间共享的信息，实践证明关系数据库是难以击败的最佳解决方案。关系数据库具有高度的可伸缩性、可靠性、高效性以及灵活性。所以，我们需要有一种方式可以把信息从SQL数据库中提取出来，再将其转换成Java对象，反之亦然。

实现这一功能有许多不同的方法，从完全手工的数据库设计和编码，到高度自动化的工具，都有相应的解决方案。这个普遍性问题就是人们所谓的对象/关系数据库映射（Object/Relational Mapping）问题，而Hibernate正是Java中一种轻量级的O/R映射服务。

所谓“轻量级”（lightweight）是指，和其他一些可用的工具相比，Hibernate的设计相当易于学习和使用，同时对系统资源的需求也在合理的范围内。虽然如此，Hibernate还是设法达到了用途广泛而又有技术的深度。Hibernate的设计者对实际项目中需要完成的工作进行了细致的研究，并对这些工作提供良好的支持。

Hibernate的使用方法有很多种，这要取决于你开始时着手的对象。如果需要交互的数据库已经存在，则可以用Hibernate的一些工具对现有的数据库模式（schema）进行分析，以此作为映射（mapping）的起点，之后，它再帮助你编写一些用于表示那些数据的Java类。如果你已经有了Java类，并希望把这些类的实例中的数据保存在数据库中，则可以从这些Java类开始，用Hibernate工具生成相应的映射文件，再生成用于创建数据库表的模式脚本。

在本书中，我们将要带领你从一个全新的项目开始，没有现成的Java类或数据库表，让

Hibernate帮助你生成这些类和数据库表。当像这样从头做起时，最佳的切入点就是二者之间的一个中间点，也就是我们打算在程序对象和存储这些对象的数据表之间使用的映射的抽象定义。附录E就如何深入学习Hibernate列举了一些建议；如果你愿意使用Eclipse，第11章还得介绍如何在Eclipse中使用Hibernate。

对于那些已经习惯处理Java对象，而对抽象模式比较陌生的开发人员来说，他们在熟悉这种方法之前，可能会遇到些小麻烦，或许只是为一个外部的XML文件费神而已。第7章会演示如何使用Java 5的标注，在你的数据模型类中嵌入映射信息。搞清楚基于XML的映射是很重要的，所以我们就从它开始学习Hibernate。

在我们的示例中，我们将处理一个数据库，用它来驱动一个应用程序界面，用户可以保存许多个人音乐收藏数据，还可以方便地进行搜索、浏览和欣赏音乐（第1章最后创建了一些数据库文件，从其中的文件名你或许可以猜到这些吧）。

编写映射文档

传统的Hibernate使用XML文档来维护Java类和关系数据库表之间的映射关系。这种映射文档设计为可由开发人员阅读和手工编辑的。你也可以用图形化的CASE（Computer Aided Software Engineering，计算机辅助软件工程）工具（例如Togethor（注1）、Rose（注2）以及Poseidon（注3））来建立表示数据模型的UML图表，再将这些图表输入到AndroMDA（注4）（http://www.andromda.org/）中，就可以生成相应的Hibernate映射文档。前面提到的Hibernate Tools包也可以为你提供这些功能（第11章将会演示在Eclipse中使用它们是多么容易）。

要牢记，Hibernate及其扩展工具可以让你用其他方式进行开发。如果你已经有了Java类或数据，也可以从它们开始着手。我们还会学习一种更新的方法——Hibernate标注，它可以让你完全脱离开XML映射文档，这种方法将在第7章加以介绍。

这里，我们先手工编写XML映射文档，这是一种相当实用的方法。我们要为曲目(Track)对象创建映射文件。曲目就是各个可以单独欣赏的乐曲，或是可以收录在乐曲集或演奏列表中的乐曲。首先，我们要记录曲目的标题、存储实际音乐的文件路径、它的播放时间、曲目添加进数据库的日期以及播放曲目应该用的音量（不同乐曲在录制时采用的音量不见得相同，总用预设的默认音量播放不一定合适）。

注1： http://www.borland.com/us/products/together/index.html.
注2： http://www-306.ibm.com/software/awdtools/developer/thechnical/.
注3： http://genteware.com/index.php.
注4： http://www.andromda.org/.

为什么选择这个示例

你可能根本不需要创建一个新的系统来保存音乐曲目，但在建立这个示例的映射文档时所涉及的概念和处理过程，可以在你的实际项目中加以应用。

应该怎么做

注意：你可能会觉得这一映射文件中包含了太多的信息。正如你所看到的，确实如此，可以用它来创建很多有用的项目资源。

打开你喜欢用的文本编辑器，在前面1.7节中创建的src/com/oreilly/hh/data目录下创建一个名为Track.hbm.xml的文件(如果你跳过了这一节，就得回去再阅读一下1.7节，因为本示例要依赖当时所建立的项目目录结构和工具)。在该映射文档中输入例2-1所示的内容。

例2-1：曲目对象的映射文档：Track.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"> ❶

<hibernate-mapping>

  <class name="com.oreilly.hh.data.Track" table="TRACK"> ❷
    <meta attribute="class-description"> ❸
      Represents a single playable track in the music database.
      @author Jim Elliott (with help from Hibernate)
    </meta>

    <id name="id" type="int" column="TRACK_ID"> ❹
      <meta attribute="scope-set">protected</meta>
      <generator class="native"/> ❺
    </id>

    <property name="title" type="string" not-null="true"/> ❻

    <property name="filePath" type="string" not-null="true"/>

    <property name="playTime" type="time"> ❼
      <meta attribute="field-description">Playing time</meta>
    </property>

    <property name="added" type="date">
      <meta attribute="field-description">When the track was created</meta>
    </property>

    <property name="volume" type="short" not-null="true">
      <meta attribute="field-description">How loud to play the track</meta>
    </property>
```

```
    </class>
</hibernate-mapping>
```

❶ 最前面的3行是有效的XML文档所必需的预定义部分，用于声明该XML文档符合Hibernate映射使用的文档类型定义。实际的映射内容将位于hibernate-mapping标签内部。

❷ 我们正在为一个类（com.oreilly.hh.data.Track）定义它的映射，这个类的名称和包名与已经创建的文件的名称和路径是一致的。但这种关系不是必需的，可以在一个映射文档中为任意数量的类定义它们的映射，为映射文件起任意名字，并把它放在任何地方，只要告诉Hibernate怎么找到这个文件就可以了。不过，根据映射到的类来命名映射文件，并将它和映射到的类放在一起，这一惯例带来的好处是：当需要使用这个类时，Hibernate能够自动定位该映射文档。当类的数量不多时，这样可以简化 Hibernate的配置和使用。

如果映射类的数量比较多，那么你可能会希望使用XML格式的Hibernate配置文件，再从这个配置文件中引用所有的映射文档，而不必像本书第1版那样，在所有示例源代码中再涉及它们（从下一章开始，就会看到我们换用了这种新方法）。此外，在使用基于XML的配置文件时，将各个映射文档和Hibernate配置文件放在一起，而不是和映射类分散地保存，这样做更有意义。

在class元素的开始标签中，我们指定这个类要保存在一个名为TRACK的数据库表中。

❸ meta标签并不会直接影响映射，相反，这个标签为使用其他不同工具而提供额外的信息。在这个例子中，通过将attribute属性值指定为“ class-description ”，我们告诉Java代码生成工具：需要为Track类关联什么样的JavaDoc文本信息。这个标签完全是可选的，在本章稍后的“ 生成Java类 ”一节中，就会看到包含JavaDoc信息的结果。

❹ 映射内容的其他部分用于设定我们想保存到数据库中的信息，也就是类的属性以及数据库表中与之相关的字段。虽然我们在介绍这个例子时没有提及，但每个曲目对象都需要一个id属性。按照数据库的最佳实践标准，我们应该用一个没有意义的代理键（surrogate key，一个没有语义含义的值，只用来标识特定的数据行）。在Hibernate中，key/id（键/属性）之间的映射关系是用id标签来设置的。我们准备使用一个int类型的值把id保存在数据库字段Track_id中。这里又包含了一个meta标签，用于和Java代码生成器进行通信，告诉它这个id属性的set方法应该具有protected类型的访问权限，因为应用程序代码本身没有必要去修改曲目对象的ID。

❺ generator标签用于设置Hibernate如何为新的实例创建其id值（注意，该标签与普通的对象/关系映射操作有关，而与Java代码生成器无关，而且也不经常使用代码生成器；generator要比可选的meta标签发挥更多的功能）。在这个标签中可以从中选择很多不同的ID生成策略，甚至可以编写自己的策略。在这个例子中，我们告诉Hibernate使用底层数据库最自然的主键生成方法（稍后我们会看到Hibernate如何知道我们正在使用什么数据库）。在

HSQLDB中，会采用一个标识（identity）字段。

❻ 在ID之后，我们只列举出关心的各种曲目属性。title是一个字符串属性，而且不能为null。filePath也有同样的属性设置，而除了volume以外，其他属性都可以为null。

❼ playTime是time类型，added是date类型，而volume是short类型。最后三个属性用了另一个新的meta属性“field-description”，用它为单个属性指定JavaDoc文本信息，但目前的代码生成器对此还有些限制。

该映射文件严格而简练地指定了我们需要表达的音乐曲目的各种数据，而且Hibernate也可以处理这种映射格式。接下来我们就看看Hibernate实际上是如何处理的(Hibernate能够表示更为复杂的信息，包括表示对象之间的关系，我们将在接下来的几章中加以介绍。附录E也讨论了一些与深入学习本书内容相关的方法)。

生成Java类

我们的映射文件中包含的是有关数据库、Java类以及它们之间的映射关系的信息。可以用它来帮助我们创建数据库表和Java类。首先，我们来看看Java类。

应该怎么做

你在第1章安装的Hibernate Tools中包含一个工具，可以生成匹配映射文档规定的Java源代码。只要用一个Ant任务就能方便地在Ant的构建文件中调用这一工具。编辑build.xml，把例2-2中的黑体部分添加进去。

例2-2：Ant构建文件（已经为生成Java代码而做了相应的更新）

```
<?xml version="1.0"?>
<project name="Harnessing Hibernate 3 (Developer's Notebook Second Edition)"
         default="db" basedir="."
         xmlns:artifact="antlib:org.apache.maven.artifact.ant">

  <!-- Set up properties containing important project directories -->
  <property name="source.root" value="src"/>
  <property name="class.root" value="classes"/>
  <property name="data.dir" value="data"/>

  <artifact:dependencies pathId="dependency.class.path">
    <dependency groupId="hsqldb" artifactId="hsqldb" version="1.8.0.7"/>
    <dependency groupId="org.hibernate" artifactId="hibernate"
                version="3.2.5.ga">
      <exclusion groupId="javax.transaction" artifactId="jta"/>
    </dependency>
    <dependency groupId="org.hibernate" artifactId="hibernate-tools"
                version="3.2.0.beta9a"/>
    <dependency groupId="org.apache.geronimo.specs"
```

```
                artifactId="geronimo-jta_1.1_spec" version="1.1"/>
      <dependency groupId="log4j" artifactId="log4j" version="1.2.14"/>
    </artifact:dependencies>

    <!-- Set up the class path for compilation and execution -->
    <path id="project.class.path">
        <!-- Include our own classes, of course -->
        <pathelement location="${class.root}" />
        <!-- Add the dependencies classpath -->
        <path refid="dependency.class.path"/>
    </path>

    <!-- Teach Ant how to use the Hibernate Tools -->
    <taskdef name="hibernatetool" ❶
             classname="org.hibernate.tool.ant.HibernateToolTask"
             classpathref="project.class.path"/>

    <target name="db" description="Runs HSQLDB database management UI
  against the database file--use when application is not running">
        <java classname="org.hsqldb.util.DatabaseManager"
              fork="yes">
           <classpath refid="project.class.path"/>
           <arg value="-driver"/>
           <arg value="org.hsqldb.jdbcDriver"/>
           <arg value="-url"/>
           <arg value="jdbc:hsqldb:${data.dir}/music"/>
           <arg value="-user"/>
           <arg value="sa"/>
        </java>
    </target>

    <!-- Generate the java code for all mapping files in our source tree -->
    <target name="codegen" ❷
            description="Generate Java source from the O/R mapping files">
      <hibernatetool destdir="${source.root}">
        <configuration>
          <fileset dir="${source.root}">
            <include name="**/*.hbm.xml"/>
          </fileset>
        </configuration>
        <hbm2java/>
      </hibernatetool>
    </target>

  </project>
```

我们在构建文件中新添加了一个taskedf（任务定义）元素，以及一个用于创建文件的target元素：

❶ 这一行的任务定义教给Ant一个新技巧：它告诉Ant如何使用Hibernate Tools中包含的hibernate tool任务（有个专门的类为该目的提供帮助）。注意，它也指定了调用这个工具时所需的类路径，并引用project.calss.path定义（这个定义包含了Maven Ant Tasks为我们管理

的所有依赖文件）。当Ant需要使用hibernate工具时，通过这种办法，就可以找到它们。

❷ codegen构建目标使用Hibernate Tools（hbm2java）模式来运行Hibernate的代码生成器，处理src源代码目录中找到的任何映射文件，生成相应的Java源代码。“ **/*.hbm.xml ”这样的匹配模式（pattern）是说“ 在指定目录或其任何子目录下（无论有多深），任何以.hbm.xml结尾的文件 ”。

让我们先试一下吧！从你的项目目录的命令行中，输入以下命令：

```
ant codegen
```

你应该看到类似以下内容的输出（假设你运行过前面章节中ant db示例，运行那个例子会下载所有必需的依赖文件；如果你还没有运行过那个例子，则此时在下载这些依赖文件时，会多显示许多行信息）：

```
Buildfile: build.xml

codegen:
[hibernatetool] Executing Hibernate Tool with a Standard Configuration
[hibernatetool] 1. task: hbm2java (Generates a set of .java files)
[hibernatetool] log4j:WARN No appenders could be found for logger
(org.hibernate.cfg.Environment).
[hibernatetool] log4j:WARN Please initialize the log4j system properly.

BUILD SUCCESSFUL
Total time: 2 seconds
```

以上提示信息大致说的是，我们在第1章配置构建文件要安装log4j，但还没有建立它需要的配置文件，我们将在2.3节介绍解决这一问题的办法。现在，如果你去看看src/com/oriellyhh/data目录，就会发现出现了一个名为Track.java的新文件，其内容如例2-3所示。

例2-3：根据Track映射文档生成的Java源代码

```
package com.oreilly.hh.data;
// Generated Sep 2, 2007 10:27:53 PM by Hibernate Tools 3.2.0.b9

import java.util.Date;

/**
 *       Represents a single playable track in the music database. ❶
 *       @author Jim Elliott (with help from Hibernate)
 *
 */
public class Track  implements java.io.Serializable {

    ❷
    private int id;
    private String title;
    private String filePath;
    /**
     * Playing time
    */
```

```
    private Date playTime;
    /**
     * When the track was created
    */
    private Date added;
    /**
     * How loud to play the track
    */
    private short volume;

❸
    public Track() {
    }

    public Track(String title, String filePath, short volume) {
        this.title = title;
        this.filePath = filePath;
        this.volume = volume;
    }
    public Track(String title, String filePath, Date playTime, Date added,
short volume) {
       this.title = title;
       this.filePath = filePath;
       this.playTime = playTime;
       this.added = added;
       this.volume = volume;
    }

    public int getId() {
        return this.id;
    }

    protected void setId(int id) { ❹
        this.id = id;
    }
    public String getTitle() {
        return this.title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
    public String getFilePath() {
        return this.filePath;
    }

    public void setFilePath(String filePath) {
        this.filePath = filePath;
    }
    /**
     *      * Playing time
     */
    public Date getPlayTime() {
        return this.playTime;
    }
```

```
        public void setPlayTime(Date playTime) {
            this.playTime = playTime;
        }
        /**
         *      * When the track was created
         */
        public Date getAdded() {
            return this.added;
        }

        public void setAdded(Date added) {
            this.added = added;
        }
        /**
         *      * How loud to play the track
         */
        public short getVolume() {
            return this.volume;
        }

        public void setVolume(short volume) {
            this.volume = volume;
        }

    }
```

这个文件是怎么生成的？Ant会找出源代码树中所有以.hbm.xml结尾的文件（目前只有一个），把它们传给Hibernate的代码生成器，该代码生成器会分析映射文件，并生成一个满足Track映射文件中指定映射要求的Java类文件。很明显，这个工具可以为我们节省很多时间，并帮我们完成一些重复性的工作！

注意：Hibernate可以为我们节省许多时间并完成很多繁琐的操作。要不然，我们一定会被它们搞得晕头转向。

比较生成的Java源代码和映射文件中的映射规定（例2-1），你会有所收获。源代码以相应的包（package）声明作为开始，对于hbm2java而言，根据映射文件中指定的完全限定的类名，就可以容易地确定正确的包名：

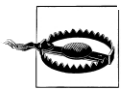
❶ 类级的JavaDoc看起来应该很眼熟，因为它来自映射文档中meta标签的“ class-description ”。

❷ 字段声明是来自于映射文档中定义的id和property标签。源代码中所用到的Java类型都是源自于映射文档中的property类型声明。学习有关Hibernate支持的全部值类型，可以参阅附录E提到的资源。目前而言，映射文件内的类型和已生成的代码内的Java类型两者间的关系应该相当明确。

❸ 字段声明之后是三个构造函数的定义。第一个构造函数是创建实例时不用任何参数（如果想让你的类能够作为标准的bean来使用，例如在JSP（Java Server Page）内使用，这是这种

数据类非常常见的用法）；第二个构造函数只需要提供映射文档中标明不得为null的值；最后一个构造函数是为所有属性赋值。注意，这些构造函数都没设置id属性的值，当我们从数据库把对象取出或者第一次将对象数据插入数据库时，Hibernate负责帮我们做这件事。

❹ 同样，setId()方法的访问类型是protected，与id的映射配置的要求是一样的。后面的getter和setter方法就没什么特别之处了，都是些照本宣科式的程序代码（我们都写过无数次了），这就是让Hibernate工具为我们生成这些代码的妙处所在。



如果你想使用Hibernate生成的代码作为起点，并在生成的Java类中添加某些业务逻辑或其他功能，一定要记住，你所做出的修改在下一次运行代码生成器时，都会被“悄悄”地丢弃。在这样的项目中，你需要确保手工修改过的类不会被任何Ant的构建目标任务重新生成。一种常用的技巧是，把需要手工修改的类扩展成Hibernate生成的类。这也是为什么我们要将映射生成的Java类隔离到它们自己的代码包和子目录中的原因之一。

虽然此例中Hibernate为我们生成了数据类，但需要指出的一个要点是，Hibernate创建的getter和setter方法不仅仅是为了样子好看。对于你需要持久化的任何属性，在持久类中都必须具有getter和setter方法。这是因为Hibernate底层的持久化架构是通过反射机制来访问JavaBeans风格的属性的。如果你不希望类的属性是公共（public）的，它们可以不必是public的（即使属性被声明为protected或private，Hibernate还是有办法取得这些属性的值），但它们必须具有访问器和修改器方法，即getter（访问器）和setter（修改器）方法。这一点也是优秀的面向对象设计应该遵循的准则；Hibernate团队希望把实际的实例变量（instance variable）的实现细节与底层持久化保存机制完全分离开来。

编制数据库Schema

刚才真简单，对吧？根据映射来创建数据库表也差不多是这样，你会很愉快地学习下去。和代码生成一样，只要利用映射文件，几乎所有工作都做完了。剩下的就是设置和运行schema生成工具。

应该怎么做

第一步是我们在第1章间接提到的一些事情。我们必须告诉Hibernate，我们要使用什么数据库，这样，Hibernate才知道应该使用什么SQL“方言”（dialect）。没错，SQL是标准，但每种数据库系统在标准SQL基础上，还都有各自在某些方面的扩展，有一套会影响到实际应用的特定功能和限制。为了解决这一现实问题，Hibernate提供了一组类来封装常见数据库环境的独特功能，这些类位于org.hibernate.dialect包中。你只需要告诉Hibernate想要使用哪一

种数据库（如果你要使用的数据库是Hibernate目前尚未支持的，也可以自行实现必需的SQL方言）。

在我们的这个例子中，使用的是HSQLDB数据库，所以要用HSQLDialect。配置Hibernate最简单的方法就是创建一个名为hibernate.properties的属性配置文件，再将它放在项目类路径的根目录中。在src目录的顶层创建这个文件，将例2-4的内容放进去。

例2-4：设置hibernate.properties的内容

```
hibernate.dialect=org.hibernate.dialect.HSQLDialect
hibernate.connection.driver_class=org.hsqldb.jdbcDriver
hibernate.connection.url=jdbc:hsqldb:data/music
hibernate.connection.username=sa
hibernate.connection.password=
hibernate.connection.shutdown=true
```

除了设置要使用的SQL方言以外，这个属性配置文件也会告诉Hibernate如何使用封装在HSQLDB数据库JAR文件内的JDBC驱动程序来建立数据库的连接，而且数据应该放在data目录下名为music的数据库中。在经过第1章的实践以后，对用户名和空密码（事实上，所有都是这个值）应该都很熟悉了。最后，我们告诉Hibernate，当处理完成后，应该显式关闭数据库连接，这是在嵌入模式下处理HSQLDB的一个artifact。如果不关闭连接的话，当工具退出时，对数据库的修改就不一定会写入到数据库中，所以你的数据库模式总是莫名其妙地为空。

如前所述，你也可以使用XML格式来保存配置信息，但就此处的简单需求而言，这样做没有什么价值。我们将在第3章介绍XML格式的配置方法。

你也可以把.properties文件存放在其他地方，并提供不同的文件名称，或者以完全不同的方式把这些设置属性传递给Hibernate。但这里是Hibernate寻找配置文件的默认位置，因此这是个最方便的路径（或者，我猜想也是需要最少运行时配置的方法）。

我们也需要在Ant构建文件中加入一些新项，如例2-5所示，在build.xml文件末尾的</project>结束标签前增加了一些新的目标。

例2-5：生成schema所需要的Ant构建文件

```
<!-- Create our runtime subdirectories and copy resources into them -->
<target name="prepare" description="Sets up build structures"> ❶
  <mkdir dir="${class.root}"/>

  <!-- Copy our property files and O/R mappings for use at runtime -->
  <copy todir="${class.root}" >
    <fileset dir="${source.root}" >
      <include name="**/*.properties"/>
      <include name="**/*.xml"/> ❷
    </fileset>
```

```
    </copy>
  </target>

  <!-- Generate the schemas for all mapping files in our class tree -->
  <target name="schema" depends="prepare" ❸
          description="Generate DB schema from the O/R mapping files">

    <hibernatetool destdir="${source.root}">
      <configuration>
        <fileset dir="${class.root}">
          <include name="**/*.hbm.xml"/>
        </fileset>
      </configuration>
    <hbm2ddl drop="yes" /> ❹
    </hibernatetool>
  </target>
```

❶ 首先，我们加了一个prepare构建目标，以供其他构建目标使用，而不是从命令行直接调用。其用途是在必要时创建classes目录，以便将编译好的Java代码放在这个目录中，再把src目录中找到的任何.properties文件和映射文件都复制到对应目录的classes层。这种层次式的复制是Ant相当棒的功能（使用特殊的“ **/* ”匹配模式），让我们可以定义和编辑源代码文件会使用到的相关资源，同时在运行时通过类加载器（class loader）来使用这些资源。

❷ 这一设置会让Ant复制所有找到的XML文件，而不仅仅是映射文档。虽然我们现在还不需要这样的配置文件，但以后当我们切换到基于XML的Hibernate配置文件时，这一设置就显得很重要了。

❸ schema构建目标需要依赖prepare构建目标，让它把映射文档复制到正确的位置以供运行时使用。它会用hbm2ddl模式来调用Hibernate工具，让它们为在classes目录中找到的任何映射文档生成相应的数据库模式（如前所述，在下一章我们使用功能更强大的Hibernate XML配置文件时，这样就会更简单些）。

❹ 可以为schema生成工具指定很多参数，以配置其工作方式。在这个例子中，我们告诉schema生成工具在根据映射文档生成新的数据表定义之前，先删除原来可能已经存在的（drop=yes）。有关这一配置和其他配置选项的更多细节，可以查阅Hibernate Tools参考手册。Hibernate甚至可以检查现有的数据表，并尝试计算出如何修改schema，才能反映出新映射文件的变化。

在添加了这些内容以后，就可以准备为我们的TRACK表生成数据库schema了。Hibernate可以为实现我们的目标而完成许多奇特的操作，并井然有序地完成处理。我们只需要为一定的消息配置好日志处理，就可以很容易地观察到Hibernate进行的操作过程。为此，我们需要配置Log4j，这是Hibernate所使用的日志处理环境。最简单的配置方法是直接在类路径下放一个log4j.properties文件。我们可以利用现有的prepare target，在Ant复制Hibernate的.properties文件时，顺便将log4j. properties文件从src目录复制到classes目录。

在src目录下创建一个名为log4j.properties的文件，内容如例2-6所示。（一种简单的做法就是从你下载的Hibernate发行包中的doc/tutorial/src目录将这个文件复制出来，因为该文件就是给Hibernate发行包所带的示例使用的。如果你自己输入，则可以跳过注释块的内容，它们只是介绍一些有用的日志功能的其他选项。）

例2-6：log4j.properties日志配置文件

```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

### direct messages to file hibernate.log ###
#log4j.appender.file=org.apache.log4j.FileAppender
#log4j.appender.file.File=hibernate.log
#log4j.appender.file.layout=org.apache.log4j.PatternLayout
#log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

### set log levels - for more verbose logging change 'info' to 'debug' ###

log4j.rootLogger=warn, stdout

log4j.logger.org.hibernate=info
#log4j.logger.org.hibernate=debug

### log HQL query parser activity
#log4j.logger.org.hibernate.hql.ast.AST=debug

### log just the SQL
#log4j.logger.org.hibernate.SQL=debug

### log JDBC bind parameters ###
log4j.logger.org.hibernate.type=info
#log4j.logger.org.hibernate.type=debug

### log schema export/update ###
log4j.logger.org.hibernate.tool.hbm2ddl=debug

### log HQL parse trees
#log4j.logger.org.hibernate.hql=debug

### log cache activity ###
#log4j.logger.org.hibernate.cache=debug

### log transaction activity
#log4j.logger.org.hibernate.transaction=debug

### log JDBC resource acquisition
#log4j.logger.org.hibernate.jdbc=debug

### enable the following line if you want to track down connection ###
### leakages when using DriverManagerConnectionProvider ###
#log4j.logger.org.hibernate.connection.DriverManagerConnectionProvider=trace
```

有了日志配置文件以后，你可能会想编辑build.xml中的codegen目标，使其也依赖于新的prepare目标。这样可以确保无论何时调用codegen，都配置了日志处理，而且Hibernate配置也可以使用，消除了我们第一次使用它时的麻烦。

现在可以制作数据库模式了！在项目目录的命令行下，执行命令ant prepare，接着再执行ant schema。你会看到类似例2-7内容的输出，同时还会创建classes目录，并在其中生成了各种资源文件；再接着就会运行模式生成器（schema generator）。

例2-7：使用HSQLDB的嵌入式数据库服务器来建立数据库模式

```
% ant prepare
Buildfile: build.xml

prepare:
    [mkdir] Created dir: /Users/jim/svn/oreilly/hib_dev_2e/current/examples/
ch02/classes
     [copy] Copying 3 files to /Users/jim/svn/oreilly/hib_dev_2e/current/
examples/ch02/classes

BUILD SUCCESSFUL
Total time: 0 seconds

% ant schema
Buildfile: build.xml

prepare:

schema:
[hibernatetool] Executing Hibernate Tool with a Standard Configuration
[hibernatetool] 1. task: hbm2ddl (Generates database schema)
[hibernatetool] 22:38:21,858  INFO Environment:514 - Hibernate 3.2.5
[hibernatetool] 22:38:21,879  INFO Environment:532 - loaded properties from reso
urce hibernate.properties: {hibernate.connection.username=sa, hibernate.connecti
on.password=****, hibernate.dialect=org.hibernate.dialect.HSQLDialect, hibernate.
connection.shutdown=true, hibernate.connection.url=jdbc:hsqldb:data/music, hibe
rnate.bytecode.use_reflection_optimizer=false, hibernate.connection.driver_class
=org.hsqldb.jdbcDriver}
[hibernatetool] 22:38:21,897  INFO Environment:681 - Bytecode provider name : cg
lib
[hibernatetool] 22:38:21,930  INFO Environment:598 - using JDK 1.4 java.sql.Time
stamp handling
[hibernatetool] 22:38:22,108  INFO Configuration:299 - Reading mappings from fil
e: /Users/jim/Documents/Work/OReilly/svn_hibernate/current/examples/ch02/classes/
com/oreilly/hh/data/Track.hbm.xml
[hibernatetool] 22:38:22,669  INFO HbmBinder:300 - Mapping class: com.oreilly.hh.
data.Track -> TRACK
[hibernatetool] 22:38:22,827  INFO Dialect:152 - Using dialect: org.hibernate.di
alect.HSQLDialect
[hibernatetool] 22:38:23,186  INFO SchemaExport:154 - Running hbm2ddl schema exp
ort
[hibernatetool] 22:38:23,194 DEBUG SchemaExport:170 - import file not found: /im
port.sql
```

```
[hibernatetool] 22:38:23,197  INFO SchemaExport:179 - exporting generated schema
 to database
[hibernatetool] 22:38:23,232  INFO DriverManagerConnectionProvider:41 - Using Hi
bernate built-in connection pool (not for production use!)
[hibernatetool] 22:38:23,234  INFO DriverManagerConnectionProvider:42 - Hibernat
e connection pool size: 20
[hibernatetool] 22:38:23,241  INFO DriverManagerConnectionProvider:45 - autocomm
it mode: false
[hibernatetool] 22:38:23,255  INFO DriverManagerConnectionProvider:80 - using dr
iver: org.hsqldb.jdbcDriver at URL: jdbc:hsqldb:data/music
[hibernatetool] 22:38:23,258  INFO DriverManagerConnectionProvider:86 - connecti
on properties: {user=sa, password=****, shutdown=true}
[hibernatetool] drop table TRACK if exists;
[hibernatetool] 22:38:23,945 DEBUG SchemaExport:303 - drop table TRACK if exists;
[hibernatetool] create table TRACK (TRACK_ID integer generated by default as ide
ntity (start with 1), title varchar(255) not null, filePath varchar(255) not nul
l, playTime time, added date, volume smallint not null, primary key (TRACK_ID));
[hibernatetool] 22:38:23,951 DEBUG SchemaExport:303 - create table TRACK (TRACK_
ID integer generated by default as identity (start with 1), title varchar(255) n
ot null, filePath varchar(255) not null, playTime time, added date, volume small
int not null, primary key (TRACK_ID));
[hibernatetool] 22:38:23,981  INFO SchemaExport:196 - schema export complete
[hibernatetool] 22:38:23,988  INFO DriverManagerConnectionProvider:147 - cleanin
g up connection pool: jdbc:hsqldb:data/music

BUILD SUCCESSFUL
Total time: 2 seconds
```

在模式导出（schema export）部分的末尾，你能够看到Hibernate用于创建TRACK表的真实SQL语句。如果你查看data目录下music.script文件的开头部分，就会发现它已经整合到数据库了。为了采用更友好的（也许是更方便的）方式来查看数据，可以执行ant db来启动HSQLDB图形界面，如图2-1所示。

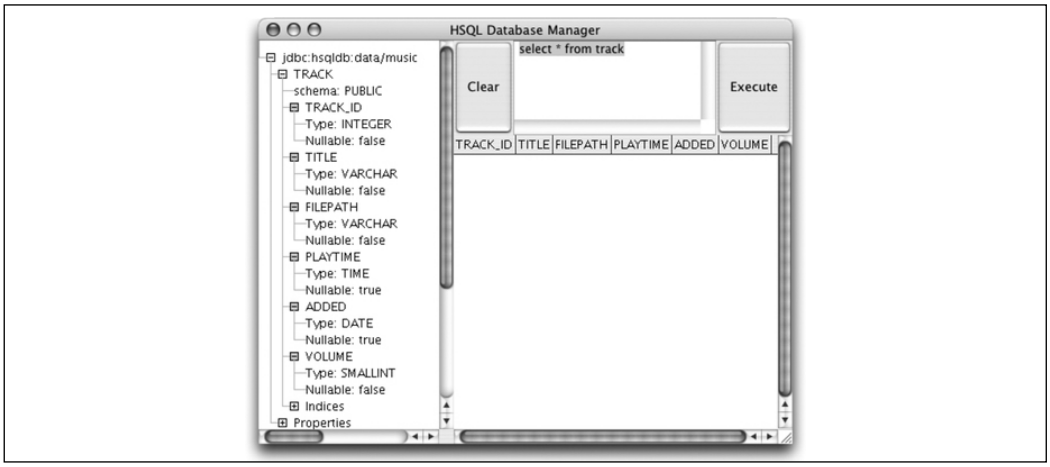


图2-1：显示新的TRACK表的内容以及一个查询的数据库管理器界面

细心的读者可能会问，既然schema target已经依赖于prepare了，那为什么要调用两次Ant，第一次是运行prepare，第二次是运行schema。这是一种所谓的步步为营法（bootstrapping）的问题，只在第一次创建环境时才会遇到。具体问题是，Ant在启动时会处理属性定义，以决定项目类路径的内容。直到prepare至少运行一次以前，classes目录还不存在，也不会包含hibernate.properties文件，所以在类路径中也不包含这个目录。在prepare运行以后，才会在classes目录生成这个文件，而classes目录也才会包含在类路径中。但是Ant在下一次运行以前，不会重新设置属性定义。所以，如果你试图在第一次Ant运行时就运行schema目标，Hibernate就会出现异常，向你报告没有指定一个数据库方言，因为它找不到配置属性文件。这种问题经常会导致令人困惑的尴尬。不过，从现在起，可以安全地直接运行schema目标了，并通过它来调用prepare，按照需要复制新版本的属性文件，因为它们在Ant开始运行时，就已经在类路径中存在了。

发生了什么事

刚才我们已经能够使用Hibernate创建一个数据表了，以后我们可以将Hibernate为我们创建的Java类实例持久地保存在这个数据表中。我们没有输入任何一行SQL或Java语句！当然，我们的数据库表目前还是空的。来改变它吧！下一章会介绍你可能最想学习的主题：在Java程序中使用Hibernate将Java对象转换为数据库中的数据项，并进行相反的转换。

在深入探讨那种非常酷的任务之前，我们应该再回想一下通过一些XML和.properties文件所能做到的诸多事项，这是非常有价值的。希望你已经开始看到Hibernate的强大功能和使用上的便利了。

其他

其他ID生成方法呢？主键（Key）在整个数据库中或全世界都是惟一的吗？Hibernate支持很多为存储在数据库中的对象选择相应的主键的方法。这是由generator标签控制的，如例2-1的第5行所示。在这个示例中，我们告诉Hibernate，对其遇到的数据库类型使用最自然的主键生成方法（native）。其他方法还包括流行的“hi/lo”算法、全局UUID、完全由Java程序代码决定的方法以及其他多种方法。具体细节，可以参阅Hibernate参考手册文档中“Basic O/R Mapping”那一章的“generator”一节。此外，如果内建的各种方法都满足不了你的需要（时常如此），还可以提供你自己的类来做你想做的事情（实现org.hibernate.id.IdentifierGenerator接口，再把实现类的名称放在generator标签内）。

如果你想学习一下用Hibernate连接你所更熟悉的数据库的示例，可以先看看第10章，这一章将演示如何用Hibernate处理MySQL数据库。