Hibernate - 通过连接表进行多对多映射

韩博孙

2016 年 1 月 4 日 女士-PL

评价我: ★★★★★ 5.00/5 (2 票)

多对多关联如何与Hibernate配合使用的详细说明，使用MySQL和Spring

在这篇文章中，我将列出我在处理一个小项目时必须处理的一些问题，在该项目中我需要在两个实体之间使用多对多映射。一旦我解决了所有问题，就有了一种以多对多关系表示两种实体类型的好方法。本教程将向您展示为什么以及如何如此出色。

下载源代码 - 8.1 KB

介绍

最近，我正在做一个小项目，我需要在两个实体之间使用多对多映射。网上有很多例子。他们中的大多数人没有解释我必须解决的一些问题。这迫使我编写本教程——让读者了解我必须处理的一些问题。事实是，一旦我解决了所有问题，这是在多对多关系中表示两种实体类型的好方法。本教程将向您展示为什么以及如何如此出色。

那么什么是多对多关系呢？想象一下，您有一个博客，并且想要管理上传到博客的照片。您可以通过图库对照片进行分类。可以有五个图像（图像 1 到图像 5）。并且可以有两个画廊（gal-1，gal-2）。图像 image-1、image-3 和 image-4 在 gal-1 中；image-2、image-3、image-4 和 image-5 在 gal-2 中。如您所见，图像 image-3 和 image-4 都在两个画廊中。这些关联可以被识别为多对多。以前我尽量避免这种复杂的关系，只处理两个有直接映射关系的实体，使用一对一或者一对多。对于这种多对多关系，它可以用三个表并使用某种类型的一对多映射（画廊和`join`表之间的一对多，图像和表之间的一对多）。`join`桌子）。但是现在，我意识到这种复杂的映射可能不是一个好主意。我特别关心我必须对后端数据库进行的显式 SQL 调用的数量。通过适当的多对多映射，我认为 Hibernate 可以帮助我简化我感兴趣的操作。

那么我对什么类型的操作感兴趣？好吧，这里有一些：

- 我喜欢创建画廊，上传图像，然后将图像与画廊相关联。
- 我喜欢删除画廊或图像。为此，我不必在删除之前明确删除关联。
- 我喜欢在画廊中找到所有图像并进行分页。
- 我喜欢添加或删除图库和图像之间的关联，但不删除图库或图像。

背景

听起来很简单。我们如何用普通的 SQL 脚本来做这些？好吧，我可以在图库表中插入一行，然后`row`在`image`表中插入另一行。最后，我为这两个添加了一个`row`into `imagetogallery`（这是`join`-table）。现在，如果我删除图库或图像行，SQL DB 有一种方法可以自动删除`join`-table 中的行。我也可以删除`join`表中的行，严重的图像和图库之间的关系。如果我想在图库中查找所有图像，我会使用两个内部`join`s 的一个查询来完成。

为了说明我的操作，这是我将要创建的测试表（顺便说一下，我使用的是 MySQL）：

SQL 复制代码

```sql
DROP TABLE IF EXISTS imagetogallery;
DROP TABLE IF EXISTS gallery;
DROP TABLE IF EXISTS image;

CREATE TABLE image (
   id int NOT NULL PRIMARY KEY,
   filepath VARCHAR(256) NULL
);

CREATE TABLE gallery (
   id int NOT NULL PRIMARY KEY,
   name VARCHAR(128) NULL
);

CREATE TABLE imagetogallery (
   id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
   imageid int NOT NULL,
   galleryid int NOT NULL,

   FOREIGN KEY (galleryid) REFERENCES gallery(id)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
   FOREIGN KEY (imageid) REFERENCES image(id)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);
```

前三行基本上删除了已经存在的表。表库和图像各有两列，第一列是主键“ id”。" id" 具有整数类型，不能是NULL。为了便于测试，我将id在 SQL 测试和启用 Hibernate 的 Java 程序中明确设置“ ”值。最后一个表 imagetogallery更复杂。它有一个主键“ id”。并且其值设置为自动递增。join当我开始使用 Hibernate 时，每当插入一行时自动提供此表的新 id 值非常重要。当我遇到它时，我会解释这一点。该join表还具有两个外键，一到gallery表一到image表。update和delete。这对于运行 SQL 语句或使用 Hibernate 再次很重要。再次，当我到达那里时，我会解释原因。

创建这些表后，我认为使用普通 SQL 语句对设置运行一些模拟是个好主意。我做的第一件事是：

SQL 复制代码

```sql
INSERT INTO gallery (id, name) VALUES (1, 'My Gallery');

INSERT INTO image (id, filepath) VALUES (2, 'c://images//testimg1.jpg');

INSERT INTO image (id, filepath) VALUES (3, 'c://images//testimg2.jpg');

INSERT INTO imagetogallery (imageid, galleryid) VALUES (2, 1);

INSERT INTO imagetogallery (imageid, galleryid) VALUES (3, 1);
```

上面的代码片段将在图库表中创建一行，在图像表中创建两行，然后将这两个图像行与图库行相关联。接下来，我想确保我可以进行查询以找到属于画廊且 id 等于 1 的所有图像。这是我的查询：

SQL 复制代码

```sql
SELECT image.* FROM image
INNER JOIN imagetogallery ON image.id = imagetogallery.imageid
WHERE imagetogallery.galleryid = 1
```

查询应该成功并产生以下输出：

ID	文件路径
2	*c://images//testimg1.jpg*
3	*c://images//testimg2.jpg*

接下来，我将尝试删除image表中的一行，比如idequals 2。这是通过以下 SQL 语句完成的：

SQL 复制代码

```
DELETE FROM image WHERE image.id = 2;
```

我使用相同的查询来查找画廊 ID 等于 1 的图像。查询返回：

ID	文件路径
3	*c://images//testimg2.jpg*

发生了什么？好吧，我确实提到，当我到达 CASCADEdelete和 时update，我将解释它们的用途以及它们为何重要。这里是。创建join表时，我不必声明 CASCADEdelete或update. 然后，如果我delete在图像表上执行该操作，我将收到一条错误消息，表明该操作将因违反外键约束而失败。原因是该join表有一行引用了我将要删除的图像。为了更正这个错误，我必须先删除join表中引用图像的行，然后才能删除图像。这比较尴尬。现在有了 CASCADEdelete在外键上，我可以只删除图像或图库表中的一行，join表中引用图库或图像的行将被自动删除。哇！那是革命性的！那么CASCADE有update什么作用呢？想象一下，假设我必须更新id图库或图像的值。这是主键的更新（危险！），它可能会因错误而失败，因为join表中的一行或多行可能引用了此图像。但它可能发生并且可以完成。声明了 CASCADE 更新后，我可以更新该图像的 id（只要我选择的 id 尚未在图像表中使用）。和DB引擎将自动更新imageid中join桌子。事实上，如果我想对图库执行此操作，我也可以在不手动更新join表的情况下执行此操作。神奇！我喜欢它！

通过所有这些测试，我对桌子设计感到满意。现在我想将所有这些移动到 Hibernate 应用程序中。它没什么特别的，只是一个普通的旧 Java 控制台应用程序。

休眠应用程序

我抢救了一个旧的基于 Spring 的程序，并转换成这个应用程序。该程序具有以下文件结构：

复制代码

```
project base directory
|____DB
     |____table1.sql
|____logs
     |____{nothing in this folder}
|____src
     |____main
          |____java
               |____org
                    |____hanbo
                         |____hibernate
                              |____experiment
                                   |____entities
                                        |____Gallery.java
                                        |____Image.java
                                        |____ImageGalleryRepository.java
                                   |____Main.java
          |____resources
               |____application-context.xml
               |____log4j.properties
|____pom.xml
```

第 1 步——POM 文件

我将首先显示*pom.xml*。在这个文件中，它包含我的实验应用程序所需的所有依赖项。它看起来像这样：

XML 缩小▲ 复制代码

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```

```
                        http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.hanbo.hibernate-experiment</groupId>
  <artifactId>hibernate-manytomany</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>hibernate-manytomany</name>
  <url>http://maven.apache.org</url>

  <properties>
     <spring.version>3.2.11.RELEASE</spring.version>
     <slf4j.version>1.7.5</slf4j.version>
  </properties>
  <dependencies>
     <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
        <version>${spring.version}</version>
     </dependency>
     <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
     </dependency>
     <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
     </dependency>
     <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>${spring.version}</version>
     </dependency>
     <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>${spring.version}</version>
     </dependency>
     <dependency>
       <groupId>log4j</groupId>
       <artifactId>log4j</artifactId>
       <version>1.2.17</version>
     </dependency>
     <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>1.1.3</version>
     </dependency>
     <dependency>
        <groupId>commons-pool</groupId>
        <artifactId>commons-pool</artifactId>
        <version>1.6</version>
     </dependency>
     <dependency>
        <groupId>commons-lang</groupId>
        <artifactId>commons-lang</artifactId>
        <version>2.6</version>
     </dependency>
     <dependency>
        <groupId>commons-dbcp</groupId>
        <artifactId>commons-dbcp</artifactId>
        <version>1.4</version>
     </dependency>
     <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.2.15.Final</version>
     </dependency>
```

```
      <dependency>
         <groupId>org.hibernate.javax.persistence</groupId>
         <artifactId>hibernate-jpa-2.0-api</artifactId>
         <version>1.0.1.Final</version>
      </dependency>
      <dependency>
         <groupId>org.jboss.logging</groupId>
         <artifactId>jboss-logging</artifactId>
         <version>3.1.4.GA</version>
      </dependency>
      <dependency>
         <groupId>javax.transaction</groupId>
         <artifactId>jta</artifactId>
         <version>1.1</version>
      </dependency>
      <dependency>
         <groupId>mysql</groupId>
         <artifactId>mysql-connector-java</artifactId>
         <version>5.1.29</version>
      </dependency>
   </dependencies>
   <build>
      <finalName>hiberfnate-manytomany</finalName>
      <plugins>
         <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.1</version>
            <configuration>
              <source>1.7</source>
              <target>1.7</target>
            </configuration>
         </plugin>
         <plugin>
            <artifactId>maven-dependency-plugin</artifactId>
            <executions>
              <execution>
                <phase>install</phase>
                <goals>
                  <goal>copy-dependencies</goal>
                </goals>
                <configuration>
                  <outputDirectory>${project.build.directory}/lib</outputDirectory>
                </configuration>
              </execution>
            </executions>
          </plugin>
      </plugins>
   </build>
</project>
```

Spring 框架中的那些用于依赖注入、与 Hibernate 的集成和 DB 事务。来自 Apache Commons 的那些用于日志记录、便利和连接池。我专门使用 Log4J 进行日志记录。其他的是 MySQL JDBC 驱动程序、Hibernate 使用的 JBoss 日志记录、Hibernate 库、JPA 注释和 javax 事务。

构建这个应用程序。您需要做的就是：

复制代码

```
mvn clean install
```

第 3 步——实体类

为了让我的所有场景都与 Hibernate 一起工作，我需要定义我的实体。因为我将`join`表用于多对多关联。我只需要定义两个实体。的实体`Gallery`和 的实体`Image`。该`join`表在两个实体中隐式定义。如您所见，使用该`join`表，我实际上不需要定义三个实体，而只需要定义两个。然后我可以让 Hibernate 和数据库为我完成繁重的工作。

让我向您展示`Image`实体的 Java 代码。这里是：

爪哇 缩小▲ 复制代码

```
package org.hanbo.hibernate.experiment.entities;

import java.util.HashSet;
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;

@Entity
@Table(name = "image")
public class Image
{
   @Id
   @Column(name = "id", nullable = false)
   private int id;

   @Column(name = "filepath", nullable = true, length = 256)
   private String filePath;

   @ManyToMany(fetch = FetchType.LAZY, mappedBy = "associatedImages")
   private Set<Gallery> associatedGalleries;

   public Image()
   {
      associatedGalleries = new HashSet<Gallery>();
   }

   public int getId()
   {
      return id;
   }

   public void setId(int id)
   {
      this.id = id;
   }

   public String getFilePath()
   {
      return filePath;
   }

   public void setFilePath(String filePath)
   {
      this.filePath = filePath;
   }

   public Set<Gallery> getAssociatedGalleries()
   {
      return associatedGalleries;
   }

   public void setAssociatedGalleries(Set<Gallery> associatedGalleries)
   {
      this.associatedGalleries = associatedGalleries;
   }
}
```

这个类最重要的部分是多对多注解：

爪哇 复制代码

```
@ManyToMany(fetch = FetchType.LAZY, mappedBy = "associatedImages")
private Set<Gallery> associatedGalleries;
```

你可能会问，我在哪里可以找到这个“ associatedImages”？它位于Gallery实体类中。的想法mappedBy是让Image实体查找Gallery实体定义并找到图像集合。Gallery实体中的此集合属性称为“ associatedImages”。现在让我分享Gallery实体定义的源代码：

爪哇 缩小▲ 复制代码

```
package org.hanbo.hibernate.experiment.entities;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.JoinColumn;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name = "gallery")
public class Gallery
{
   @Id
   @Column(name = "id", nullable = false)
   private int id;

   @Column(name = "name", nullable = true, length = 128)
   private String name;

   @ManyToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
   @JoinTable(name = "imagetogallery", joinColumns = {
         @JoinColumn(name = "galleryid",
         nullable = false, updatable = false)
      }, inverseJoinColumns = {
         @JoinColumn(name = "imageid",
         nullable = false, updatable = false)
      }
   )
   private Set<Image> associatedImages;

   public Gallery()
   {
      setAssociatedImages(new HashSet<Image>());
   }

   public int getId()
   {
      return id;
   }

   public void setId(int id)
   {
      this.id = id;
   }

   public String getName()
   {
      return name;
   }

   public void setName(String name)
   {
```

```
      this.name = name;
   }

   public Set<Image> getAssociatedImages()
   {
      return associatedImages;
   }

   public void setAssociatedImages(Set<Image> associatedImages)
   {
      this.associatedImages = associatedImages;
   }
}
```

如您所见，该类最复杂的部分如下，它实际上使用join我创建的表定义了多对多关联：

爪哇 复制代码

```
@ManyToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
@JoinTable(name = "imagetogallery", joinColumns = {
      @JoinColumn(name = "galleryid",
      nullable = false, updatable = false)
   }, inverseJoinColumns = {
      @JoinColumn(name = "imageid",
      nullable = false, updatable = false)
   }
)
private Set<Image> associatedImages;
```

您需要了解以下几点：

- 我使用的级联类型是CascadeType.All，这意味着当我对Image实体或Gallery实体进行更改时，所有更改都必须传播（级联效应）到join表。
- 该JoinTable注解定义了join我所创建的表。
- 该JoinTable注释还定义了join列，一个galleryid，一个用于imageid。这些是表中的表列join。
- 第一join列是galleryid在join表中。第二个是反join列imageid。这两个告诉Gallery实体如何在join表中找到自己，以及与图库关联的图像。
- 该join列有两个属性。一个被称为nullable，我将其设置为false表示该列的值不能为null。另一个被称为updatable，我将其设置为false. 这些列中的值是其他表的主键，允许它们更新是不明智的。

第 4 步——存储库类

我还创建了一个存储库类，以便我可以玩这些场景。该repository班是在同一个包的实体。这个类看起来像这样：

爪哇 缩小▲ 复制代码

```
package org.hanbo.hibernate.experiment.entities;

import java.util.List;

import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

@Repository
@SuppressWarnings("unchecked")
public class ImageGalleryRepository
{
   private static Logger _logger = LogManager.getLogger(ImageGalleryRepository.class);

   @Autowired
```

```
    private SessionFactory _sessionFactory;

    @Transactional
    public void deleteAll()
    {
       Session session = _sessionFactory.getCurrentSession();

       Query q = session.createQuery("delete from Gallery");
       q.executeUpdate();

       q = session.createQuery("delete from Image");
       q.executeUpdate();
    }

    @Transactional
    public int testPersistence()
    {
       Session session = _sessionFactory.getCurrentSession();

       Gallery gallery = new Gallery();
       gallery.setId(1);
       gallery.setName("My Test Gallery");

       Image img1 = new Image();
       img1.setId(2);
       img1.setFilePath("C:\\testimages\\img1.jpg");

       gallery.getAssociatedImages().add(img1);

       Image img2 = new Image();
       img2.setId(3);
       img2.setFilePath("C:\\testimages\\img2.jpg");

       gallery.getAssociatedImages().add(img2);

       session.save(gallery);

       return gallery.getId();
    }

    @Transactional
    public void testPersistence2()
    {
       Session session = _sessionFactory.getCurrentSession();

       Query query = session.createQuery(
          "select image from Gallery gallery join gallery.associatedImages image"
          + " where gallery.id = :galId order by image.id desc"
       ).setParameter("galId", 1).
       setMaxResults(2);

       List<Image> imgs = query.list();

       for(Image image : imgs)
       {
          _logger.info(String.format("Image Id: %s", image.getId()));
          _logger.info(String.format("Image File Path: %s", image.getFilePath()));
       }
    }

    @Transactional
    public void testPersistence3()
    {
       Session session = _sessionFactory.getCurrentSession();

       Gallery gal = (Gallery)session.get(Gallery.class, 1);
       Image img = (Image)session.get(Image.class, 3);

       gal.getAssociatedImages().remove(img);
```

```
        session.update(gal);
    }
}
```

这个类相当复杂。

首先，该类被注释为存储库。在里面，有一个名为 的自动装配属性`_sessionFactory`。然后有四种方法，每种方法都有一个目的：

- `deleteAll`: 可以用来清理两个表。这在术语中`join`也会删除表中的行。
- `testPersistence`：用于创建一个画廊和两个图像。然后将两个图像关联到图库。
- `testPersistence2`：用于查找属于特定图库的所有图像。这是由 Hibernate 查询语言完成的。
- `testPersistence3`: 用于从画廊持有的图像集合中删除 id 为 3 的图像。然后会话更新图库。这应该从`join`表中删除一行。

的源代码`deleteAll`：

爪哇 复制代码

```
@Transactional
public void deleteAll()
{
    Session session = _sessionFactory.getCurrentSession();

    Query q = session.createQuery("delete from Gallery");
    q.executeUpdate();

    q = session.createQuery("delete from Image");
    q.executeUpdate();
}
```

此方法为我提供了一种从所有三个表中删除所有行的方法。我这样做的方法是假设我已经正确设置了所有内容，尤其是在代码和数据库表设置中的 CASCADE 设置。当我删除一个表中的所有行时，表中的行将`join`消失。我可以安全地删除另一个表中的所有行，没有问题。

的源代码`testPersistence`：

爪哇 复制代码

```
@Transactional
public int testPersistence()
{
    Session session = _sessionFactory.getCurrentSession();

    Gallery gallery = new Gallery();
    gallery.setId(1);
    gallery.setName("My Test Gallery");

    Image img1 = new Image();
    img1.setId(2);
    img1.setFilePath("C:\\testimages\\img1.jpg");

    gallery.getAssociatedImages().add(img1);

    Image img2 = new Image();
    img2.setId(3);
    img2.setFilePath("C:\\testimages\\img2.jpg");

    gallery.getAssociatedImages().add(img2);

    session.save(gallery);

    return gallery.getId();
}
```

上面的代码应该是不言自明的。我像普通对象一样创建Image和Gallery实体，调用构造函数和设置器。读者应该注意的一件事是实体只有一个保存——Gallery在将两个Image实体添加到GalleryImage 的集合中后保存实体。这是可以做到的，因为 Hibernate 为我们做了繁重的工作。它实际上Image为我们插入了实体。相信你把log4j级别转成DEBUG，就可以看到了。

这是我们可以对这两种类型的实体做的唯一方法吗？当然不是，您可以创建两个Image实体并显式保存它们，然后创建Gallery实体，将两个 Image 实体添加到其集合中，然后保存Gallery. 我已经测试过了。有用。我确定如果你愿意，你可以创建Gallery实体，然后保存它。然后创建两个 Image 实体，也保存它们。最后，在 CRUD 操作的某个地方，您可以将图像添加到Gallery实体的集合中，然后保存Gallery实体。

我向读者保证我会解释为什么我需要AUTO_INCREMENT在join表格中。上面的代码就是原因。如您所见，没有向join表中插入一行的代码。它是由 Hibernate 完成的。它所做的只是插入带有值 (galleryid = 1,imageid = 2或3) 的行。如果您不在主键AUTO_INCREMENT的join表中使用它，则save()通过会话将失败并出现异常。这是我希望有人能告诉我的事情。我花了一段时间才弄明白。现在您知道，对于隐式jointable insert，您**必须**提供一种方法来join自动为表行创建唯一主键。

代码testPersistence2:

爪哇 复制代码

```
@Transactional
public void testPersistence2()
{
   Session session = _sessionFactory.getCurrentSession();

   Query query = session.createQuery(
      "select image from Gallery gallery join gallery.associatedImages image"
      + " where gallery.id = :galId order by image.id desc"
   ).setParameter("galId", 1).
   setMaxResults(2);

   List<Image> imgs = query.list();

   for(Image image : imgs)
   {
      _logger.info(String.format("Image Id: %s", image.getId()));
      _logger.info(String.format("Image File Path: %s", image.getFilePath()));
   }
}
```

对于上述方法，我试图做的是试验，看看如果我只有画廊 ID，我是否可以获得画廊的图像，所有图像。我想我可以用 HQL 轻松做到这一点，我也可以通过从Gallery实体向下遍历到它的Image集合来做到这一点。但是，如果我想Image按日期按降序对集合进行排序，并将图像实体的数量限制为某个数字或将起始索引设置为某个数字，就没有那么容易了。专家可能会证明我错了，但我更喜欢 HQL。再一次，让我告诉你上面的代码是有效的。正如你所看到的，我是做了join的Gallery表，Image表，我甚至没有提到的join表都没有。这不是很棒吗?

我们最后一个方法的代码testPersistence3:

爪哇 复制代码

```
@Transactional
public void testPersistence3()
{
   Session session = _sessionFactory.getCurrentSession();

   Gallery gal = (Gallery)session.get(Gallery.class, 1);
   Image img = (Image)session.get(Image.class, 3);

   gal.getAssociatedImages().remove(img);

   session.update(gal);
}
```

这个方法的作用是，假设我有画廊 id 和图像 id，我想从这个画廊中删除这个图像的关联。同样，您可以看到这种方法有多酷，我查找图库和图像，然后我涉及该getAssociatedImages()图库的 ，然后从集合中删除该图像。最后，我只是保存gallery实体。我可以向你保证这是有效的。这简直太棒了。这一切都是因为我们已将CASCADE设置正确添加到表和实体类中。

接下来，我要运行Main包含该static main方法的类中的所有四个场景。这就是我们接下来要看到的。

第 5 步——主类

该main班只是为我演示在我的仓库类的测试方法途径。这很简单。这里是：

爪哇 复制代码

```
package org.hanbo.hibernate.experiment;

import org.hanbo.hibernate.experiment.entities.ImageGalleryRepository;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main
{
   public static void main(String[] argv)
   {
      ConfigurableApplicationContext context =
         new ClassPathXmlApplicationContext("application-context.xml");

      ImageGalleryRepository repo
         = context.getBean(ImageGalleryRepository.class);

      repo.deleteAll();
      repo.testPersistence();
      repo.testPersistence2();
      repo.testPersistence3();

      context.close();
   }
}
```

上述 Java 类的作用如下：

- 首先，加载应用程序上下文。应用程序上下文包含所有 bean 定义。它必须是要加载的第一件事。
- 一旦应用程序拥有应用程序上下文对象，我就应该能够获得我想要使用的 bean。在这种情况下，我想获取存储库对象，以便我可以运行我的场景。
- 在我的应用程序中，我运行我的场景。
- 关闭应用程序上下文，以便应用程序可以退出。

第 6 步——应用程序上下文 XML 文件

由于这是一个 spring 应用程序，我需要定义 bean，配置 Hibernate、MySQL 和事务管理器等。这是我的应用程序上下文 XML 文件：

XML 缩小▲ 复制代码

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xmlns:tx="http://www.springframework.org/schema/tx"

   xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context-3.0.xsd
         http://www.springframework.org/schema/tx
         http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

   <context:component-scan base-package="org.hanbo.hibernate.experiment.entities" />

   <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
                         destroy-method="close">
     <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
     <property name="url" value="jdbc:mysql://127.0.0.1:3306/hanbotest?autoReconnect=true"/>
     <property name="username" value="hbuser1"/>
```

```
    <property name="password" value="123test321"/>
    <property name="maxActive" value="8"/>
    <property name="maxIdle" value="4"/>
    <property name="maxWait" value="900000"/>
    <property name="validationQuery" value="SELECT 1" />
    <property name="testOnBorrow" value="true" />
  </bean>

  <bean id="sessionFactory"
   class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="packagesToScan">
      <array>
        <value>org.hanbo.hibernate.experiment.entities</value>
      </array>
    </property>
    <property name="hibernateProperties">
      <props>
        <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
        <prop key="hibernate.cache.provider_class">
         org.hibernate.cache.NoCacheProvider</prop>
      </props>
    </property>
  </bean>

  <bean id="transactionManager"
   class="org.springframework.orm.hibernate4.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
  </bean>

  <tx:annotation-driven transaction-manager="transactionManager"/>
</beans>
```

如果你对 Spring 有一点经验，这个文件不会很难阅读。本质上，我结合了注释和应用程序上下文 XML 文件的使用。以下是我的 XML 文件的部分：

- 组件扫描部分告诉 Spring 框架某些包包含可用于依赖注入的类。
- 日期源文件针对 MySQL DB 创建 JDBC 驱动程序的配置。
- 会话工厂部分为 Hibernate 创建会话工厂 bean。
- 最后两部分用于事务管理器。作为读者，您可能已经看到`@Transactional`了存储库中方法的注释。为了使用这个注解，我必须定义这两部分。

第 6 步——Log4J 配置

我还需要配置`log4j`日志记录。这很容易做到，我从`log4j`官方网站获得了一个示例日志属性文件，并针对我的项目进行了修改。内容如下：

复制代码

```
log4j.rootLogger=debug, stdout, R

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n

log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
log4j.appender.R.File=C:/temp/logs/hibernate-manytomany.log
log4j.appender.R.DatePatttern=yyyy-MM-dd

log4j.appender.R.MaxFileSize=10MB
log4j.appender.R.MaxBackupIndex=10

log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

该文件使用每日滚动日志文件。该文件的最大大小为 10MB，最大备份副本为10. 第一行允许您控制日志记录级别。我把它设置为DEBUG. 如果要调低电平，可以更改DEBUG为INFO。其他行，它们不重要。**请确保您已创建日志目录 - C:\temp\logs**

执行结果

该deleteAll()方法将删除所有三个表中的任何行。

该testPersistence()方法将在Gallery表中创建一行，id设置为1。Image表中的两行，ids 是2和3。由于这在 之后运行deleteAll()，如果成功，则证明deleteAll()有效。如果deleteAll()无法工作，我id对所有三行使用相同的s，testPersistence()将失败。

该testPersistence2()方法将输出两行，一行是idof 2，另一行是id 3。这假设tesPersistence()工作正常。并且输出应该是两Image行的详细信息。

最后一个testPersistence3()将删除 image withid 3和 gallery with之间的关联id 1。我没有提供任何输出测试方法=来测试这个。所以你只需要对数据库中的image表和imagetogallery表运行查询。image表中的两行都应该在那里。imagetogallery表中应该只有一行。

兴趣点

使用join表在两个实体之间映射多对多关系可能是您必须执行的最复杂的实体配置。到这里之后，我对使用 Hibernate 和关系数据库非常有信心。这个教程已经够清楚了。有很多信息。我也相信本教程胜过所有其他类似的教程。写这篇文章很有趣。希望你喜欢它!

历史

- 31日 十二月，2015：初稿

执照

分享

关于作者

韩博孙

组长 The Judge Group

美国

手表该会员

没有提供传记

评论和讨论

添加评论或问题 

电子邮件提醒 Search Comments 

-- 本论坛暂无消息 --