Issues:

1. Collection集合类对象无法获得，使用HibernateTemplate. get()时会提示如下错误:

org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: org.fireprotection.domain.InfoItem.images, could not initialize proxy - no Session

@Fetch(FetchMode.Join) //添加初始化模式，等同于@OneToMany中的EARGER

1. DataIntegrityViolationException: not-null property references a null or transient value

小对象参考对象中有nullable = false的对象，小对象保存前没有将大对象列入

1. Mysql 显示General\_Log

Mysql General\_Log 可以用来排错使用, 开启方法如下:

MySQL 控制台>

SET GLOBAL general\_log\_file=’文件位置’ //设定保存文件位置

SET GLOBAL general\_log=on //开启日志功能

SET GLOBAL general\_log=off //关闭日志功能

1. Hibernate 自动创建表格

属性hbm2ddl.auto 值 update

对需要产生表格的实体进行mapping属性的设定

如需要生成外键时，则需要在dialect中选择Mysql57Dialect，可以保证MySQL 存储引擎选择为InnoDB

1. 格式

@Entity //标记为实体类型

@Table(name = “tableName”) //标记表格名称

Public class myData

{

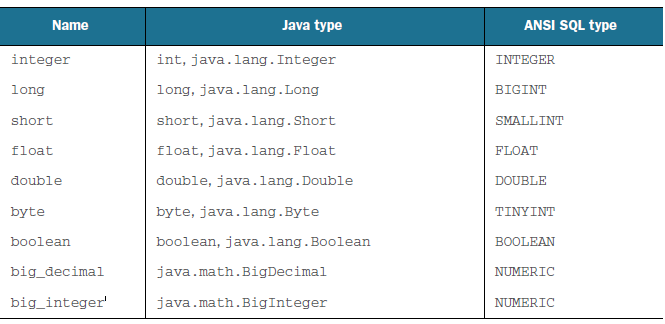
@Id //标记为索引列

@GeneratedValue(strategy = GenerationType.IDENTITY)//自动生成索引序号

Private int id;

}

1. SessionFactory（Hibernate）就是EntityManager（JPA）
2. 通过实现PhysicalNamingStrategy 可以动态映射名称
3. @Subselect(value=”SQLString”) 可用于在不可建立视图的情况下建立视图
4. @Syncronize({“Item1”,”Item2”}) 当Item1，Item2发生变化的时候实体也会跟着变化
5. @Access 表明实体的访问属性，Property 表示听过setter/getter 设定或获得数据,Field表示通过反射获得数据，并可以不设定setter/getter
6. @Formula可实现字段为只读的公式字段
7. @ColumnTransformer(read = “xxxx / 2.2”,write = “? \* 2.2”) 数据读写公式
8. @ColumnDefault(“1.00”) 键的默认值
9. @Generated(GenerationTime.ALWAYS) 表示自动生成字段值
10. @Enumerated(EnumType.String) 默认对于枚举类型的存储结果为转换为数字，如果使用String 则会将枚举类型字符存放于表中
11. 映射类型与数据库类型的相互转换

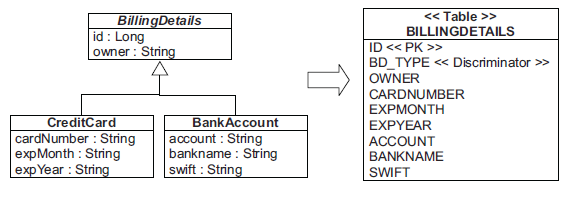


1. 在Hibernate中总共有两种策略,对象继承策略、映射策略
2. @MappedSuperclass映射父类型 用于继承
3. @AttributeOverride (name = “orignalName”,column = @Column(name = “columnName”,nullable = false)) 属性列，名称变更
4. @Inheritance(strategy = InheritanceType.TABLE\_PER\_CLASS) 通过继承策略使类型/表实现继承(每个子类一张表单)
5. @Inheritance(strategy = InheritanceType.Single\_TABLE) 父类与与所有子类型集成于一张表上，该标记只适用于父类(抽象类)

@DiscriminatorColumn 用于指定子类型的识别列表头名称 该标记适用于父类

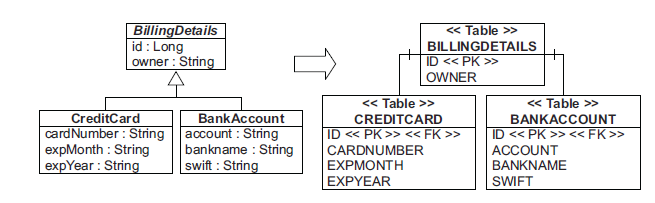
@DiscriminatorValue 用于指定子类型的识别列的数值 该标记适用于子类

(父类将子类的所有属性保存于一张表单)



1. @Inheritance(strategy = InheritanceType.JOINED) 使用外键的方式连接继承类，每个继承类型一个表单(id既是主键亦是外键) 该标记适用于父类

@PrimaryKeyJoinColumn(name = “XXXX\_ID”) 用于指定主键/外键的列名称



1. @SecondaryTable(name = “CREDITCARD”,pkJoinColumns = @PrimaryJoinColumn(name = “CREDIT\_ID”)) 用于切分SINGLE\_TABLE 为单一表，并将此表赋予主键/外键，从而忽略主类的继承策略，以实现多种继承类型的混合，该标记适用于子类
2. @Embeddable 可用于嵌入对象

First、Last、Middle为嵌入的值类型

数据库

create table Contact (

id integer not null,

first varchar(255),

last varchar(255),

middle varchar(255),

notes varchar(255),

starred boolean not null,

website varchar(255),

primary key (id)

)

实体

@Entity(name = "Contact")

public static class Contact {

@Id

private Integer id;

private Name name;

private String notes;

private URL website;

private boolean starred;

//Getters and setters are omitted for brevity

}

实体嵌入对象

@Embeddable

public class Name {

private String first;

private String middle;

private String last;

// getters and setters omitted

}

1. 选择继承策略的方法

如果不需要多态关联/查询，则选用TABLE\_PER\_CLASS (union table)

如果需要多态关联/查询或简单项目，则选用SINGLE\_TABLE (所有类在同一张表中)

如果需要多态关联/查询或复杂项目，然后其子类又有很多属性，则选用JOINED

1. @ManyToOne 指原对象可以是多个，目标对象只能是一个
2. @OneToMany（mappedBy = “xxx”）只原对象只能是一个，目标对象可以是多个,如原对象为集合类,mappedBy用于指定目标对象的标记
3. 映射策略概要原文:

Table per concrete class with implicit polymorphism is the simplest strategy to

map inheritance hierarchies of entities, but it doesn’t support polymorphic

associations very well. In addition, different columns from different tables share

exactly the same semantics, making schema evolution more complex. We recommend

this approach for the top level of your class hierarchy only, where

polymorphism isn’t usually required and when modification of the superclass in

the future is unlikely.

 The table-per-concrete-class-with-unions strategy is optional, and JPA implementations

may not support it, but it does handles polymorphic associations.

 The table-per-class-hierarchy strategy is a winner in terms of both performance

and simplicity; ad hoc reporting is possible without complex joins or unions,

and schema evolution is straightforward. The one major problem is data integrity,

because you must declare some columns as nullable. Another issue is normalization:

this strategy creates functional dependencies between non-key

columns, violating the third normal form.

 The table-per-subclass-with-joins strategy’s primary advantage is that it normalizes

the SQL schema, making schema evolution and integrity constraint definition

straightforward. The disadvantages are that it’s more difficult to

implement by hand, and performance can be unacceptable for complex class hierarchies.

1. 接口及其实现类

Set ⬄ HashSet

SortSet ⬄TreeSet

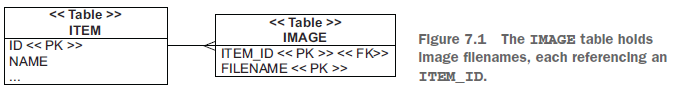
List ⬄ ArrayList

Map ⬄HashMap

SortedMap ⬄ TreeMap

1. 映射 Bag类型

Bag 类型的意思在JAVA中指没有排序的集合，可以用Collection<Object>接口代替，实现类ArrayList<Object>替代



@Entity  
public class Item {

@ElementCollection  
 @CollectionTable(  
 name = "IMAGE", // 变更默认外键集合表的名称,默认表的名称是ITEM\_IMAGES  
 joinColumns = @JoinColumn(name = "ITEM\_ID")) // 外键名称  
 @Column(name = "FILENAME") //外键表所连接列的名称  
 protected Set<String> images = new HashSet<String>(); // Set实现实例

}

1. 无序映射 List类型(一般类型/对象)

@Entity  
public class Item {  
  
  
 @ElementCollection  
 @CollectionTable(name = "IMAGE") // 外键名称  
 @OrderColumn // 表示List是有顺序的List，默认情况下都是无序列  
 @Column(name = "FILENAME")  
 protected List<String> images = new ArrayList<String>(); // List实现实例  
}

1. 无序映射 Map类型(一般类型/对象)

多个实体不能共享Map的key，这样会造成duiplicate key found 错误

@Entity  
public class InfoItem extends BaseItem {  
  
 @ElementCollection  
 @CollectionTable(name = "IMAGE") //Map映射表名称  
 @MapKeyColumn(name = "FILENAME") //Map映射表的次主键，用于映射Map的Key  
 @Column(name = "IMAGENAME") //映射Map的Value  
 private Map<String,String> images = new HashMap<String,String>();  
  
}

1. 有序映射 Map类型(一般类型/对象)

……

…..

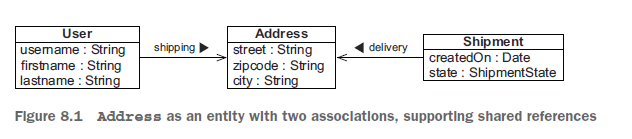
1. 映射对象集合(实体对象)

以下均是用于进行实体对象间映射的标记

@OrderColumn(name = “属性名称\_POSITION”) 表示有序列，列明为”属性名称\_POSITION”,默认名称为”属性名称\_ORDER”

@OneToOne

在不使用集合的情况下晚上下方的视图,就可以用到OneToOne方法，并且User,Shipment共享主键



使用@PrimaryKeyJoinColumn 用于共享其他表的主键，Hibernate会自动根据使用的实体类型进行绑定,主键即为外键

如需要单独设立外键时，则需使用JoinColumn(),如再使用unique=true属性，表示两个实体不能够共享同一个外键

Fetch=Lazy只能工作在optional=false的情况下

Optional=false表示必须指定关联外键

外键生成器，外键在小对象一侧，外键生成器需在大对象一侧，

@Id

@GeneratedValue(generator = "addressKeyGenerator")

@org.hibernate.annotations.GenericGenerator(

name = "addressKeyGenerator",

strategy = "foreign",

parameters =

@org.hibernate.annotations.Parameter(

name = "property", value = "user"

)

)

protected Long id;

外主键生成器(The foreign primary key generator)

…P176

@ManyToOne (小对象->大对象,非绝对)

默认Fetch为EARGER(默认一次性加载),这会影响性能，所以一般操作时需更改为LAZY(延迟加载)

Optional 表示join的方式，true 为Inner Join ,false 为Left Join

保存小对象时需要将大对象reference至小对象

保存小对象前需要先保存大对象

@OneToMany (大对象->小对象,非绝对)

Cascade 级联，级联有多种属性，包含persist(表示存储大对象的同时自动保存小对象),remove(删除大对象的同时自动删除小对象)

orphanRemoval 表示删除大对象的同时是否删除小对象

@OnDelete

删除小对象的同时自动删除大对象，和CascadeType.Remove刚好相反

@ManyToMany

@JoinColumn 与@ManyToOne, @OneToMany配合使用，用于改写外键列(FK)名称, @ManyToOne是使用外键的那一方，@OneToMany是不使用外键的那一方,如不使用@JoinColumn的话默认会已”对象名称\_ID”作为FK列

@JoinColumn 的属性updatable =false，insertable=false 可以使该列处于只读状态

@JoinColumn 和@PrimaryKeyJoinColumn作用是一样的就是用于绑定外键，只是JoinColumn是指主键和外键不是同一字段，而@PrimaryKeyJoinColumn表主键与外键是同一字段

FetchType.Earger 和 FetchType.Lazy，Earger表示立即加载，Lazy表示延迟加载，这两者的实际区别在于Earger是实体，而Lazy使用的是CGLib 实现代理方法,所以使用Lazy加载的对象要使用Session.load()方法而不是Session.get()，Session.get()是不支持延迟加载的

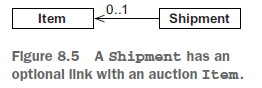
主键通过外键的作为序列生成器

@Id  
@GeneratedValue(generator = “addressKeyGenerator”)

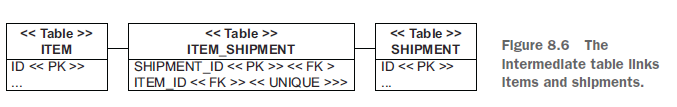
//使用addressKeyGenerator 作为键生成器  
@GenericGenerator(  
 name = "addressKeyGenerator",   
 strategy = "foreign", //策略为外键  
 parameters = @Parameter(name = "property",value = "user") //属性依赖user对象  
)  
private Long id;

@JoinTable 增加中间媒介表，解释nullable 的状态

原状态：



增加中间媒介后的状态



@Entity

public class Shipment {

@OneToOne(fetch = FetchType.LAZY)

@JoinTable(

name = "ITEM\_SHIPMENT",

joinColumns =

@JoinColumn(name = "SHIPMENT\_ID"),

inverseJoinColumns =

@JoinColumn(name = "ITEM\_ID",

nullable = false,

unique = true)

)

protected Item auction;

public Shipment() {

}

public Shipment(Item auction) {

this.auction = auction;

}

// ...

}

@OneToOne 的所有可能功能原文：

This completes our discussion of *one-to-one* association mappings. To summarize, use a

shared primary key association if one of the two entities is always stored before the

other and can act as the primary key source. Use a foreign key association in all other

cases, and a hidden intermediate join table when your *one-to-one* association is

optional.