1. SessionFactory是一个线程安全的、不可更改的，表示数据库和应用domain映射关系。SessionFactory的创建代价非常高，对于给定的数据库，相应的SessionFactory只应该有一个。

2. Session: 一个单线程，短存活的实例。它封装了JDBC。它维护一个“可重复读”的一级缓存。

3. Transaction： 事务

4. 使用hibernate5时，resin4不支持@Table，可以将@Entity的name属性换成表名，删除resin的lib目录下可能有冲突的jar包

# 2. 领域模型（domain model）

## 2.1 映射类型

1. 值类型（value types）：为实体所有，不定义自己的生命周期。

2. 基本类型（basic types）

3. 嵌入类型（Embeddable types）

4. 集合类型（Collection types）

5. 实体类型（Entity types）：是领域类型，和数据库表的列对应。

@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  
}  
  
@Embeddable  
public class Name {  
 private String first;  
 private String middle;  
 private String last;  
 // getters and setters omitted  
}

## 2.2 命名策略

hibernate经过两步处理将对象名映射为数据库的表名：通过用户指定（@Column或@Table）或者hibernate自定的策略来生成一个逻辑名称；将逻辑名称映射到定义的表名，策略为：[PhysicalNamingStrategy](https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html#PhysicalNamingStrategy)

可以通过实现ImplicitNamingStrategy接口或继承ImplicitNamingStrategyJpaCompliantImpl来修改名字对应的策略

## 2.3 基本类型（Basic Types）

基本的值类型一般将数据库中的列映射为一个单一的、非聚合的java类型，可以映射为Java中的装包类型：Integer、BigInteger等

1. @Basic注释：基本类型默认是被javax.persitence.Basic注释的；@Basic定义了两个属性
   * optional(defaults to true):标明该元素是否可以为空；
   * fetch(EAGER,LAZY)：hibernate会忽略该属性。标明是否需要懒加载
2. @Column(name=””): 如果列名和Java中的属性名不一致，可以明确设置列名。其定义的属性名可以参考文档或IDEA的提示
3. BasicTypeRegistry 维护了一个从Java类型到SQL类型的映射表，也可以自己定义BasicType。
4. @Type : 可以修改属性的值映射的类型

### 2.3.1 自定义基本类型

如果想将java.util.BigInteger保存成VARCHAR的列，或者支持全新的类型。有两种方法可以实现：

* 实现一个BasicType并且注册它，因为BasicType接口有许多方法，如果值只是存在一个列中，可以继承AbstractStandarBasicType或者AbstractSingleColumnStandardBasicType.
* 实现UserType，不用注册

两者的代码可参考HibernateDemo com.sogou.umis包

## 2.3.2 枚举类型映射

@Enumerated(EnumType.ORDINAL| EnumType.STRING)可以将枚举类型映射到SQL中。还可以自定义转换函数，将值经过转换后，存入数据库。

public enum PhoneType {  
 *LAND\_LINE*,  
 *MOBILE*;  
}

* EnumType.ORDINAL:将值映射为Integer类型，按顺序排列，对null值，为NULL。对于上例：[INSERT INTO Phone (phone\_number, phone\_type, id) VALUES ('123-456-78990', 2, 1)]
* EnumType.STRING： 将枚举类型的字符串存入数据库，对于上例： [INSERT INTO Phone (phone\_number, phone\_type, id) VALUES ('123-456-78990', 'MOBILE', 1)]

### 2.3.4 映射时间类型

DATE： 代表存储了年月日的时间

TIME：存储了时分秒的时间

TIMESTAMP: 存储DATE和TIME。

[使用@Temporal可以将java.util.Date或者java.util.Calendar映射为DATE](mailto:使用@Temporal可以将java.util.Date或者java.util.Calendar映射为DATE)、TIME、TIMESTAMP

@Column(name = "`timestamp`")  
@Temporal(TemporalType.DATE)  
private Date timestamp;

Java8在java.time包中新增了Date、Time的API

从SQL的Date/Time类型到Java8新增类型的映射为：

DATE ： java.time.LocalDate

TIME : java.time.LocalTime,java.time.OffsetTime

TIMESTAMP: java.time.Instant, java.time.LocalDateTime, java.time.OffsetDateTime, java.time.ZonedDateTime

因为不在是一一对应，所以使用@Temporal将会导致错误。需要指定一个时区，可以在SessionFactory中设定：

settings.put(  
 AvailableSettings.JDBC\_TIME\_ZONE,  
 TimeZone.getTimeZone( "UTC" )  
);

### 2.3.5 属性转换器(AttributeConverter)

对于可以修改的类型，可以使用属性转换器。

public static class MoneyConverter  
 implements AttributeConverter<Money, Long> {  
 @Override  
 public Long convertToDatabaseColumn(Money attribute) {  
 return attribute == null ? null : attribute.getCents();  
 }  
  
 @Override  
 public Money convertToEntityAttribute(Long dbData) {  
 return dbData == null ? null : new Money( dbData );  
 }  
}  
  
@Entity(name = "Account")  
public static class Account {  
 @Id  
 private Long id;  
 private String owner;  
 @Convert(converter = MoneyConverter.class)  
 private Money balance;  
 //Getters and setters are omitted for brevity  
}

一个可以修改的Object运行你修改它内部的结构，Hibernate会将这种修改保存到数据库。

尽管AttributeConverter类型可以被Hibernate脏检查，并被正确处理，但是不可改变的类型的效率更好，应该尽量使用不可修改的类型。

### 2.3.6 列名是SQL关键字

当使用的列名是SQL中的关键字时，可以使用[`或者\”]来转义，Hibernate会根据SQL Dialect来生成合适的SQL语句

@Column(name = "`name`")  
private String name;  
@Column(name = "\"name\"")  
private String name;

【生成的SQL语句】INSERT INTO Product ("name",id) VALUES ('Mobile phone', 1)

还可以配置属性使得全局引用

<property  
 name="hibernate.globally\_quoted\_identifiers"  
 value="true"  
/>

### 2.3.7 @Generated

Hibernate可以在对象实体在更新或者插入后，自动从数据库中获取被@Generated标注的属性。

GenerationTime.ALWAYS：该属性在插入和更新的时候都要生成

GenerationTime.NEVER： 从来不生成

GenerationTime.INSERT: 只在插入的时候生成

@Generated( value = GenerationTime.ALWAYS )  
@Column(columnDefinition =  
 "AS CONCAT(" +  
 " COALESCE(firstName, ''), " +  
 " COALESCE(' ' + middleName1, ''), " +  
 " COALESCE(' ' + middleName2, ''), " +  
 " COALESCE(' ' + middleName3, ''), " +  
 " COALESCE(' ' + middleName4, ''), " +  
 " COALESCE(' ' + middleName5, ''), " +  
 " COALESCE(' ' + lastName, '') " +  
 ")")  
private String fullName;

### 2.3.8 @CreationTimestamp: JVM当前的时间戳

@CreationTimestamp 标记的属性，hibernate在向数据库存储时，会将JVM当前的时间戳的值放入其中。

### 2.3.9 @ValueGenerationType

可以使用@ValueGenerationType来声明生成的属性或者自定义生成器

### 2.3.10 列转换：读写表达式

Hibernate允许你自定义@Basic类型的读写列值的SQL语句。如果数据库提供了加解密的函数，可以像下列一样调用它。

如果使用write表达式，必须用”?”代替其值

@Column(name = "pswd")  
@ColumnTransformer(  
 read = "decrypt( 'AES', '00', pswd )",  
 write = "encrypt('AES', '00', ?)"  
)  
private String password;

如果使用@Columns(cloumns={})则需要使用@Type(type=””)做类型转换，其中type是实现CompositeUserType接口的类。

@ColumnTransformer同@Columns()一起使用时，必须指定forColumn

@Type(type = "org.hibernate.userguide.mapping.basic.MonetaryAmountUserType")  
@Columns(columns = {  
 @Column(name = "money"),  
 @Column(name = "currency")  
})  
@ColumnTransformer(  
 forColumn = "money",  
 read = "money / 100",  
 write = "? \* 100"  
)  
private MonetaryAmount wallet;

### 2.3.11 @Formula

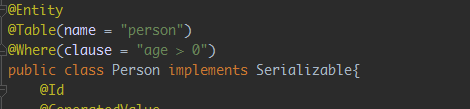
在数据库中计算内容时，可以使用SQL fragment，这种属性是只读的，它的值是由formula计算出来的。SQL fragment可以是任意复杂的，甚至可以使用子查询

@Formula使用的是本地的SQL，可能会有兼容性问题。

@Formula(value = "credit \* rate")  
private Double interest;

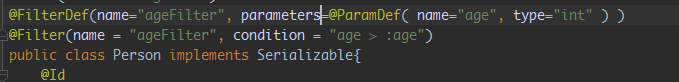
### 2.3.12 @Where

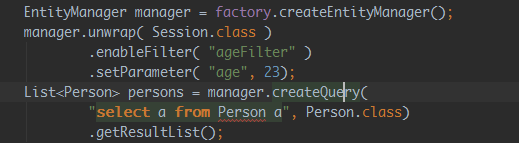
如果要过滤集合，可以使用@Where注释，该注释可以被用于集合和类上



### 2.3.13 @Filter

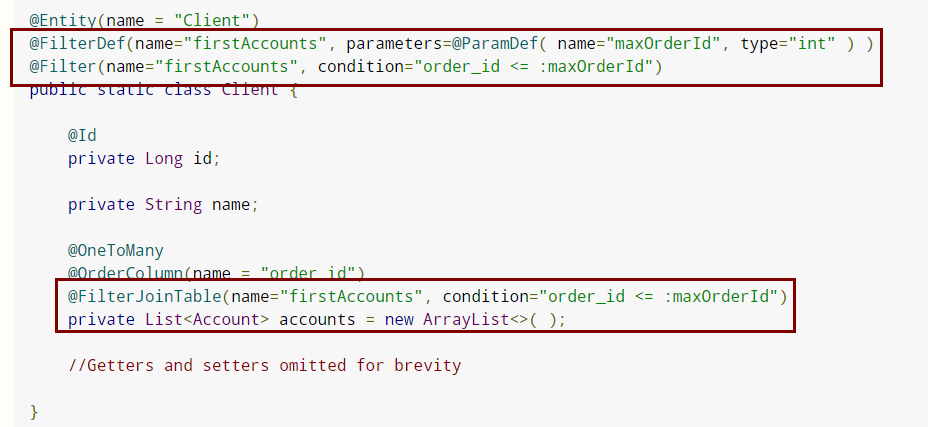
同@Where一样可以过滤集合或者列，不过@Filter允许在运行时指定过滤条件。@Filter在过滤基本类型时，需要将其放置到类上，如果是过滤集合属性，需要将其放到集合定义上

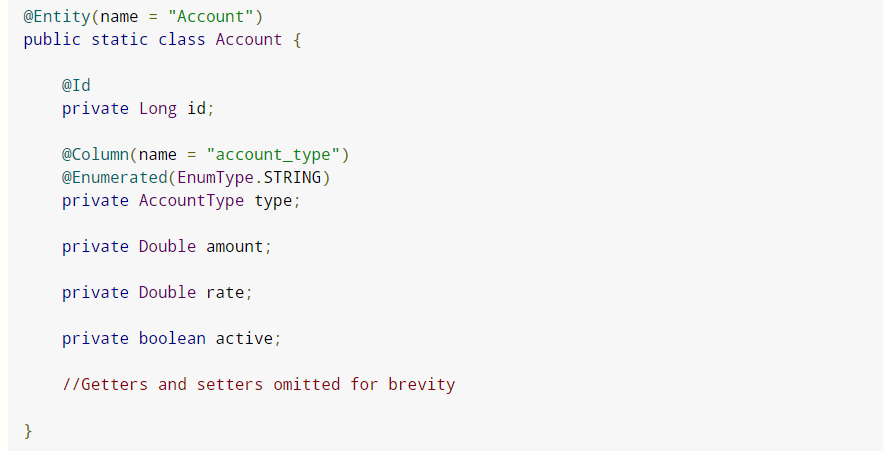




### 2.3.14 @FilterJoinTable

@FilterJoinTable可以被在子表过滤上，使用在@OneToMany集合上





## 2.4 Embeddable类型

在一个Entity中使用了多个相同的Embeddable类型时，需要标明其对应的列。需要使用@AttributeOverride来解决这个问题。



也可以通过修改hibernate的命名策略来解决该问题



## 2.4 Entity类型

hibernate中使用Entity类型必须符合以下条件：

1. Hibernate需要Entity必须有一个无参构造器，可见性最低是package

2. 为保存到数据库的属性声明getters和setters

3. @Entity只有一个name属性，用来指定JPQL queries中的实体名称

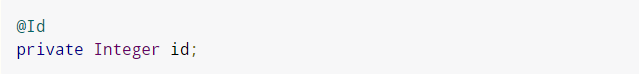
4. @Transient注解的属性，不会被保存到数据库

Hibernate一个核心的特性就是可以通过运行时代理懒加载数据，该特性要求Entity类不能是final或者是实现声明所有getters/setter的接口的类。虽然在hibernate中可以这么做，但是会影响到性能的优化，应该避免将类或者存储的类的属性声明为final。

为持久型属性声明setters和getters，虽然hibernate并不要求，但是最好遵循JavaBean的约定。

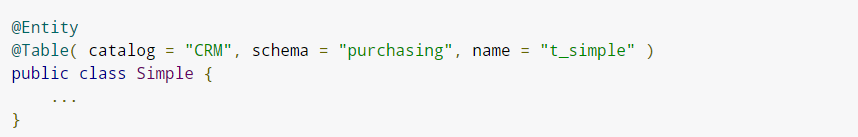
### 2.4.1 identifier

识别符属性不用与数据库定义的主键相对应，但是需要映射到可以唯一标明一列的属性上。Hibernate官方建议将Identifier属性标记在一个可以为空的类型上（非原始类型non-primitive）.



@Entity注解只定义了一个在JPQL查询时用到的实体名的name属性。默认情况下类名就是其实体名。

默认情况下，一个实体映射数据库中的一张表，一个identifier唯一地标识一列，数据库表的名同实体的名一致，也可以使用javax.persitence.Table注解指定表名。不过该方式在resin中不支持。



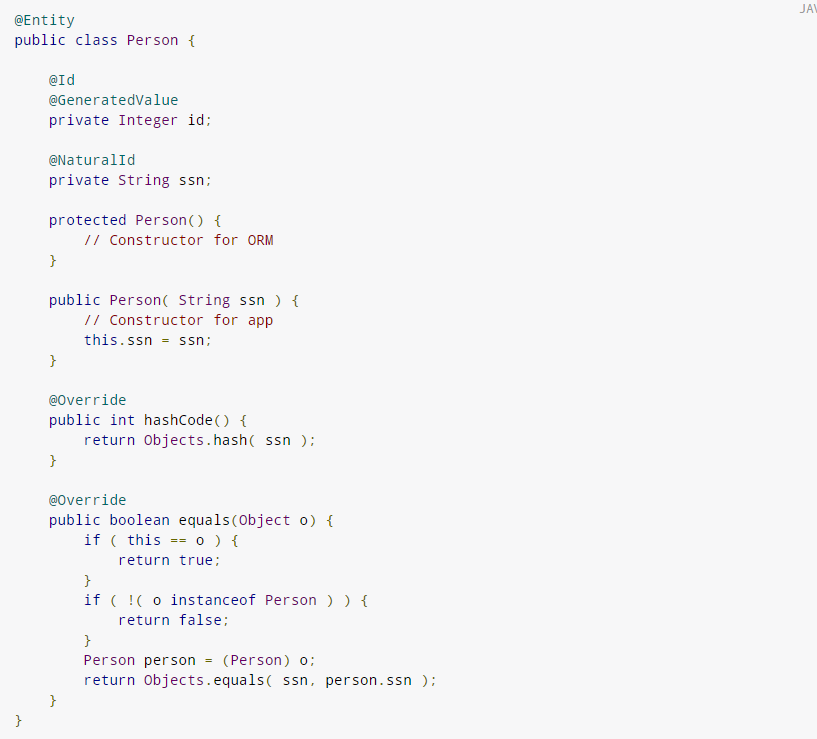
### 2.4.2 equals() 和 hashCode()

1. 当一个类被用作identifier时，必须实现基于id值的equals/hashCode
2. 使用同一个Session多次获取某个Person时，将会返回同一个实例。如果将获取多次的Person都添加到同一个java.util.Set时，将会只添加一个。‘’
3. 使用不同的Session获取Person时，返回的实例是不同的，加到Set中时，取决于是否实现了基于id的equals/hashCode；即使实现了基于id的equals/hashCode也会在 使用generated identifier的类中存在冲突。



上例中，在commit之后p1的id已经被修改。

1. 最终的解决方案是使用一个natural-id或business-key作为equals/hashCode的实现



尽管使用natural-id是equals/hashCode最好的实现，但是或许需要一个唯一约束。

使用identifier来做相等检测时，需要为hashCode提供一个在Entity flushed前后不会变化的量；只用非瞬变的量来比较实体之间是否相等。

### 2.4.3 使用@Version来映射乐观锁

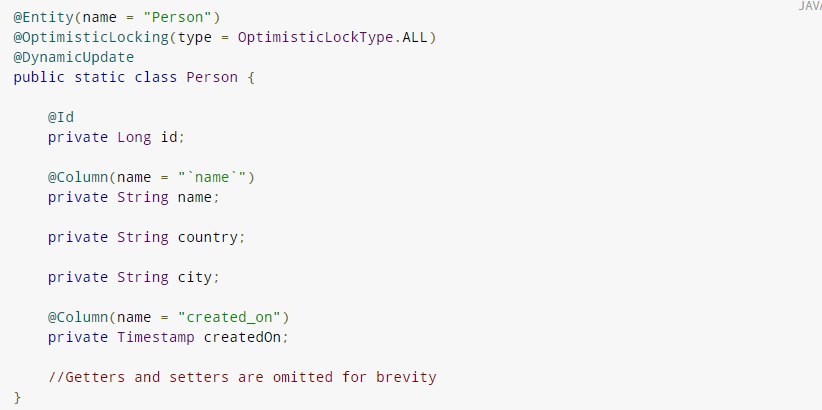
JPA定义了基于version或者timestamp的策略来支持乐观锁。只需要使用javax.persitence.Version注解到持久属性上，就能启用乐观锁。有效的类型有：int、Integer、short、Short、long、Long、java.sql.Timestamp

在许多场合使用@Version属性是有效的，但是你可以依赖实际的列的值来防止丢失更新。

Hibernate支持使用所有的实体属性或者被改变的实体属性作为查询语句，防止丢失更新。使用@OptimisticLocking(type = OptimisticLockType.ALL)注解来使用这种方式。使用OptimisticLockType.ALL时，必须使用@DynamicUpdate

OptimisticLockTypes有四个可以用的值：

* NONE: 乐观锁被禁用，有@Version也不起作用
* VERSION: 基于@Version来启用乐观锁
* ALL：在UPDATE/DELETE时，where条件使用所有的fields
* DIRTY: 在UPDATE/DELETE时，where条件使用dirty filelds，即需要改变的field在where条件中



使用OptimisticLockType.DIRTY时，也要使用@DynamicUpdate和@SelectBeforeUpdate