**树形结构：**

@Entity

@Table(name="t\_types")

**public** **class** Type **implements** Serializable{

**private** **int** id;

**private** String name;

**private** Type parent;

**private** Set<Type> children = **new** HashSet<Type>();

@Id

@GeneratedValue(strategy = GenerationType.*AUTO*)

**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;

}

@ManyToOne

@JoinColumn(name="tid")

**public** Type getParent() {

**return** parent;

}

**public** **void** setParent(Type parent) {

**this**.parent = parent;

}

@OneToMany(mappedBy="parent")

**public** Set<Type> getChildren() {

**return** children;

}

**public** **void** setChildren(Set<Type> children) {

**this**.children = children;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + id;

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Type other = (Type) obj;

**if** (id != other.id)

**return** **false**;

**return** **true**;

}

**注意：**

**每张主键表都需要实现序列化接口，而且每张表重写equals和hashCode方法**

**为什么需要实现序列化接口：**

**序列化就是可以将内存中的类写入硬盘上或者是数据库中；比如将类序列化保存到文件中，下次读取就是反序列化就可以还原到内存中，也可以类序列化为流数据进行转送。类序列化保存类中的所有的变量和状态，其中**

**举个例子，你编写了一款游戏，保存记录时把所有状态一一保存非常麻烦，这时就可以使用Serializable（序列化接口），它的作用是可以将一个对象实例序列化，序列化后你可以选择将它保存在你需要的位置。   
相对的，读取后生成的对象所有属性（除了设置为瞬时值的属性）将和原对象的属性相同（只是内存地址不同）。   
这样可以方便的将一个java对象写入到磁盘中，保存该对象的所有状态！   
值得注意的是序列化的对象中包含的属性和其他对象都需要实现序列化接口，不然无法正常序列化！   
在hibernate里，并非所有的实体类必须实现序列化接口，因为在hibernate中我们通常是将基本类型的数值映射为数据库中的字段。而基础类型都实现了序列化接口（String也实现了）。   
所以，只有在想将一个对象完整存进数据库（存储为二进制码），而不是将对象的属性分别存进数据库，读取时再重新构建的话，就可以不用实现序列化接口。  
实现了Serializable，可以方便保存数据。**

**分布式开发和游戏必须使用序列化：**

hibernate中实体类的序列化就是将类序列化后存入磁盘并释放内存空间。

持久化类是指其实例需要被Hibernate持久化到数据库中的类。持久化类通常都是域模型中的实体域类。持久化类符合JavaBean的规范，包含一些属性

分布式开发：

登录后用序列化来转送数据

朋友网

QQ空间等

QQ客户端

QQ

可能用不同的语言类编写

为什么重写equals、hashCode方法：

只要不是main方法，其中就没有比较等于的规则，而重写equals方法给其规则，使其自定义类有其比较规则，而hashCode是给equals对象的地址，没有重写hashcode也是找不到重写equals对象的地址

注意：myeclipse中有快捷键，重写equals和hashCode

联合主键（学生 课程 分数）：

1. 学生实体表：

@Entity

@Table(name="t\_student")

**public** **class** Student **implements** Serializable{

**private** **int** Sno;

**private** String name;

**private** Set<Course> course = **new** HashSet<Course>();

**private** Set<SC> scores = **new** HashSet<SC>();

@OneToMany(mappedBy="student")

**public** Set<SC> getScores() {

**return** scores;

}

**public** **void** setScores(Set<SC> scores) {

**this**.scores = scores;

}

@Id

// @GeneratedValue(strategy=GenerationType.IDENTITY)

@GeneratedValue

**public** **int** getSno() {

**return** Sno;

}

@ManyToMany

@JoinTable(name="t\_s\_c",joinColumns={@JoinColumn(name="sno")},inverseJoinColumns=@JoinColumn(name="cno"))

**public** Set<Course> getCourse() {

**return** course;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + Sno;

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Student other = (Student) obj;

**if** (Sno != other.Sno)

**return** **false**;

**return** **true**;

}

省略set方法……

1. 课程实体表

@ManyToMany

@JoinTable(name="t\_s\_c",joinColumns={@JoinColumn(name="cno")},inverseJoinColumns=@JoinColumn(name="sno") )

**public** Set<Student> getStudent() {

**return** student;

}

其余类似于学生表

1. 分数实体表

@Entity

@Table(name="t\_s\_c")

**public** **class** SC **implements** Serializable{

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 294979044992880591L;

**private** **int** id;

**private** **float** grade;

**private** Student student;

**private** Course course;

@ManyToOne

@JoinColumn(name="sno")

**public** Student getStudent() {

**return** student;

}

@ManyToOne

@JoinColumn(name="cno")

**public** Course getCourse() {

**return** course;

}

@Id

@GeneratedValue

**public** **int** getId() {

**return** id;

}

省略set和重写equals和hashCode方法

1. 生成的表

create table t\_course (

cno integer not null auto\_increment,

credit float not null,

name varchar(255),

primary key (cno)

)

create table t\_student (

sno integer not null auto\_increment,

age integer not null,

name varchar(255),

sex char(1) not null,

primary key (sno)

)

alter table t\_s\_c

add index FK68F914CE9B6870D (cno),

add constraint FK68F914CE9B6870D

foreign key (cno)

references t\_course (cno)

alter table t\_s\_c

add index FK68F914CA3D2CBE1 (sno),

add constraint FK68F914CA3D2CBE1

foreign key (sno)

references t\_student (sno)

**create table t\_s\_c (**

**id integer not null,**

**grade float not null,**

**cno integer not null auto\_increment,**

**sno integer,**

**primary key (cno, sno)**

**)**

上面使用联合主键：

**联合主键非常难处理，所以需要加一个主键id；但是生成的还是联主键，所以要手动修改其中该表的结构**

解除双方之间的关系：

Student.setCourse(null)；