目录

一、JPA 基础

1.1 JPA 基础

JPA: java persistence api 支持 XML、JDK5.0 注解俩种元数据的形式,是 SUN 公司引入的 JPA ORM 规范

元数据：对象和表之间的映射关系

实体: entity，需要使用 Javax.persistence.Entity 注解或 xml 映射，需要无参构造函数，类和相关字段不能使用 final 关键字

游离状态实体以值方式进行传递，需要 serializable

JPA 是一套规范、有很多框架支持(如 Hibernate3.2 以上、Toplink,一般用 Hibernate 就行 oracle 可以用 toplink)

JPQL

1、与数据库无关的，基于实体的查询语言

2、操作的是抽象持久化模型

3、JPQL 是一种强类型语言，一个 JPQL 语句中每个表达式都有类型

4、EJBQL 的扩展

5、支持 projection(可以查询某个实体的字段而不需要查询整个实体)、批量操作(update、delete)、子查询、join、group by having(group by 聚合后 having 聚合函数 比较 条件)

弱类型语言：没有明显的类型、根据情况变化、容易出错

强类型语言：没个变量都有固定的类型。不容易出错

虽然 JPA 规范中明确表示无法访问一个集合关系字段

抽象模型类型：JPQL 规范将一个实体(属性)中所饮食的各种类型称为抽象模型类型

状态字段

关联字段

查询多个字段查出来的是个对象值数组

1.2JPA 开发过程

JPA 配置文件声明持久化单元 --> 配置文件 persistence.xml
编写带标注的实体类
编写 Dao 类

xml 配置

事务类型分为：RESOURCE_LOCAL
本地事务、JTA(java 事务 API)

注解
@Entity 将 JavaBean 标注为一个实体 name 属性
@Table 数据库中的表，name 名称、catalog 数据库名 @Secondary Table/@Secondary Tables 多个表
@Id 定义了实体的主键信息
@GeneratedValue 逐渐省城策略
@GeneratedValue(Strategy = GenerationType.SEQUENCE)

@SequenceGenerator(name="SEQ_TEST",sequenceName="User_SEQ",allocationSize=25)
@column 属性、字段对应的表字段
@Temporal 属性是时间类型的话可以细分
DATE java.sql.Date
TIME java.sql.Time
TIMESTAMP java.sql.Timestamp
@Lob 标注 CLOB、BLOB
@Base 是否延迟加载@Base(fetch = FETCHTYPE.LAZY/FETCHTYPE.EAGER)

@Transient 实体 bean 中，所有非 static、非 transient 状态变量、字段都要被持久化
如果有字段、变量在数据库中没有对应，标注为 transient 就可以不被持久化
标注方式： 标注在字段上
标注在变量上

实体类写法：
1、必须有无参的构造函数

2、没有 final 类型的变量或方法

3、不可以是 public 类型的，只能通过 get、set 方法读写

管理实体

Persistence

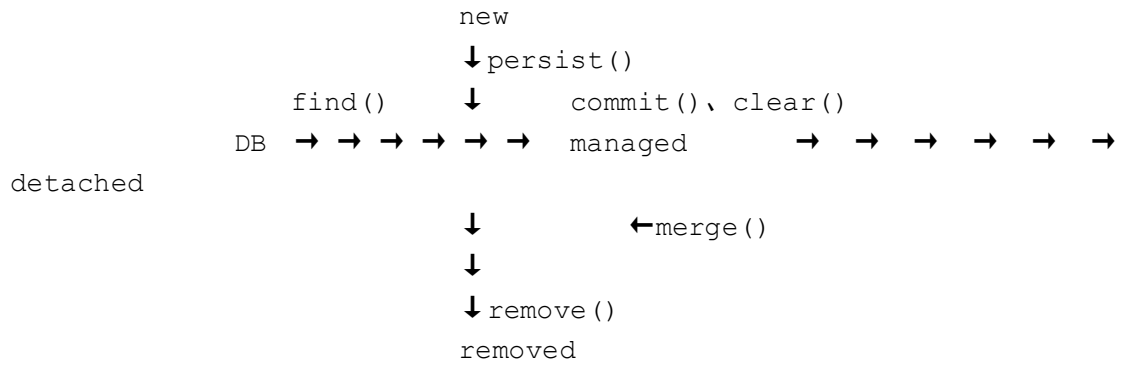
EntityManagerFactory

EntityManager

Persistence.createEntityManagerFactory('persistence.xml 中配置的 persistence unit').createEntityManager()获取 EntityManager

1.3 实体的生命周期及实体管理器常用方法

EntityManager 声明周期	Java 对象	实体管理器数据库
1、 新实体(new)	存在	不存在
2、持久化实体(managed)	存在	存在
3、分离的实体(detached)	不存在	不存在
4、删除的实体(removed)	存在	存在



常用方法

1、persist(Object)	持久化
2、remove(Object)	删除对象
3、find(Class entityClass,Object key)	根据主键查询
4、flush()	实体与底层同步,执行 sql
5、createQuery()	创建 JPQL 查询对象
5、createNativeQuery()	根据普通 SQL 查询
5、createNamedQuery()	命名查询

@NamedQuerie 标注

5、merge(Object)　　将一个 detached 的实体持久化到 EntityManager 中

5、close()　　关闭管理器

---

javax.persistence.Query

int executeUpdate()　　执行更新、删除、添加

Object getSingleResult()　　执行查询(返回一条记录)

List getResultList()　　执行查询(返回结果链表)

Query setParameter(int position,object value)　　给 Query 对象设置参数

Query setMaxResults(int maxResult)　　给 Query 对象设置返回数

Query setFirstResult(int firstResult)　　给 Query 对象设置返回偏移

参数查询(只能用一种)

命名参数查询　"select u from User where id = :uid";

setParameter("uid",value);

位置参数查询

```
Person person = em.find(Person.class,1);
        //相当于 Hibernate 的 get
Person person = em.getReference(Person.class,1);
        //相当于 Hibernate 的 load 返回一个代理对象
        //注意延迟加载时的 事务没关闭的时候才好用
```

find 如果找不到返回的是个 null，这时候下面在调用 null 的方法报 nullpoint 异常

reference 相当于延迟加载　如果找不到，会在第一次使用就报 EntityNotFound 异常

回调函数(相当于拦截器，下面的方法执行前后调用指定的方法)

```
@Prepersist
@PostPersist
@PreRemove
@PostRemove
@PreUpdate
@PostUpdate
@PostLoad
```

载入实体时(find、查询、refresh)

二、环境搭建

2.1 添加 JPA 支持

1、准备 JPA 用到的 jar 包 (JPA 支持包)
2、window → preferences → Java → BuildPath → User Libraries
→new User Library
→Add Jars

3、项目 → 右键 → properties（alt+Enter）
→ Java Build Path → Libraries
→ Add Library → User Library → 自己定义的 JPA 支持包

2.2 添加配置文件

1、项目中 SRC 目录下添加 META-INF 目录 (与 Web 项目下 META-INF 同名)
2、在新添加的 META-INF 中添加配置文件 persistences.xml
persistence.xml 配置信息 (Hibernate)
数据库连接信息查询
主要配置信息：
事务类型：本地事务、JTA 事务
JPA 供应商
数据库驱动、URL、User、Password
3、在 SRC 目录下添加 log4j.properties 文件 (显示数据库操作信息的)

2.3 测试配置

1、MySQL 测试数据库
2、实体注解
3、JUNIT 测试方法

2.4 环境搭建附表

persistence.xml 配置信息

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence                                                    version="1.0"
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
    <!-- name="持久化单元命名"  transaction-type="本地事务/JTA" -->
    <persistence-unit name="JPA" transaction-type="RESOURCE_LOCAL">
        <!-- 供应商 -->
        <provider>org.hibernate.ejb.HibernatePersistence</provider>


        <properties>
            <!-- 参数：数据库驱动名、地址、用户、密码、方言、显示执行 SQL 语句 -->
            <property          name="hibernate.connection.driver_class"
value=""/>
            <property          name="hibernate.connection.driver_class"
value="org.gjt.mm.mysql.Driver"/>
            <property                    name="hibernate.connection.url"
value="jdbc:mysql://127.0.0.1:3306/JPA"/>
            <property              name="hibernate.connection.username"
value="root"/>
            <property              name="hibernate.connection.password"
value="123456"/>
            <property                           name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect"/>
            <property name="hibernate.show_sql" value="true"/>

            <!-- 其他设置 -->
            <property name="minPoolSize" value="5"/>
            <property name="initialPoolSize" value="10"/>
            <property name="idleConnectionTestPeriod" value="120"/>
            <property name="acquireIncrement" value="10"/>
            <property name="checkoutTimeout" value="3600"/>
            <property name="numHelperThreads" value="4"/>
            <property name="maxStatements" value="400"/>
            <property name="maxStatementsPerConnection" value="20"/>
            <property name="maxIdleTime" value="180"/>
            <property name="acquireRetryAttempts" value="30"/>
            <property name="maxPoolSize" value="200"/>

        </properties>
    </persistence-unit>
```

```
</persistence>
```

自动创建|更新|验证数据库表结构。如果不是此方面的需求建议 set value="none"。容易造成数据丢失，一般在测试的时候才用

```
<property name="hibernate.hbm2ddl.auto" value="create"></property>、
validate 验证数据库表结构
create 每次加载 Hibernate 都会删除上一次的表结构，根据 model 重新生成
create-drop 每次加载创建，sessionFactory 关闭表就自动删除
update 加载 Hibernate 就更想你表结构
```

环境测试代码

实体注解

```
@Entity
@Table(name="Person")
public class Person {
    @Id
    @Column(name="pid")
    private Integer id;
    @Column(name="pname")
    private String name;
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}
```

JUNIT 测试方法

```
public EntityManager testGetEM(){
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("JPA");
        EntityManager em = emf.createEntityManager();
        return em;
    }
    public void testAddPerson(){
```

```
        Person p = new Person();
        p.setId(1);
        p.setName("ader");
        EntityManager em = testGetEM();
        EntityTransaction et = em.getTransaction();
        try {
            et.begin();
            em.persist(p);
            et.commit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

MySQL 测试数据库

```
    drop database if exists jpa;
    create database jpa;
    use jpa;
    drop table if exists person;
    drop table if exists  ident;
    drop table if exists  rel;
    create table person(
        pid int primary key auto_increment,
        pname varchar(50)
)
    ;
    create table ident(
        iid int primary key auto_increment,
        iname varchar(50)
    );
    create table rel(
        personid int,
        identid int
    );
```

JPA 支持包

```
    antlr-2.7.6.jar
    cglib-2.1.3.jar
    classes12.jar
    commons-collections-3.1.jar
```

dom4j-1.6.1.jar
ehcache-1.2.3.jar
ejb3-persistence.jar
hibernate3.jar
hibernate-annotations.jar
hibernate-cglib-repack-2.1_3.jar
hibernate-commons-annotations.jar
hibernate-entitymanager.jar
javassist-3.4.GA.jar
jta-1.1.jar
log4j-1.2.15.jar
persistence-api-1.0.jar
slf4j-api-1.5.2.jar
slf4j-log4j12.jar

Hibernate 核心包(8 个文件)：hibernate-distribution-3.3.1.GA.ZIP
---
hibernate3.jar
lib\bytecode\cglib\hibernate-cglib-repack-2.1_3.jar (CGLIB 库，Hibernate 用它来实现 PO 字节码的动态
生成，非常核心的库，必须使用的 jar 包)
lib\required*.jar
Hibernate 注解包(3 个文件)：hibernate-annotations-3.4.0.GA.ZIP
---
hibernate-annotations.jar
lib\ejb3-persistence.jar, hibernate-commons-annotations.jar
Hibernate 针对 JPA 的实现包(3 个文件)：
hibernate-entitymanager-3.4.0.GA.ZIP
---
hibernate-entitymanager.jar
lib\test\log4j.jar, slf4j-log4j12.jar

数据库连接信息查询

1、Hibernate JDBC 属性

属性名	用途
hibernate.connection.driver_class	jdbc 驱动类
hibernate.connection.url	jdbc URL
hibernate.connection.username	数据库用户
hibernate.connection.password	数据库用户密码

属性名	用途
hibernate.dialect	数据库方言

2、驱动包

Db2：db2java.jar(JDBC 直连)

：db2jcc.jar(Hibernate 要用到此驱动 jar 文件和上面的驱动 jar 文件)

sybase：jconn3d.jar

MSSQL：msbase.jar+mssqlserver.jar+msutil.jar

MySQL：mysql-connector-java-3.1.12-bin.jar

Oracle10g：ojdbc14.jar

3、连接字符串(可以先添加驱动包然后到包里找 Driver.class)

a、MSSQL

驱动：com.microsoft.jdbc.sqlserver.SQLServerDriver

地址：jdbc:microsoft:sqlserver://127.0.0.1:1433;DatabaseName=数据库

b、Oracle10g

驱动：oracle.jdbc.driver.OracleDriver

地址：jdbc:oracle:thin:@127.0.0.1:1521:全局标识符

c、MySQL

驱动：org.gjt.mm.mysql.Driver

地址：jdbc:mysql://127.0.0.1:3306/数据库

d、Access

驱动：sun.jdbc.odbc.JdbcOdbcDriver

地址：jdbc:odbc:Driver={MicroSoft Access Driver (*.mdb)};DBQ=c:\\demodb.mdb

e、DB2

驱动：COM.ibm.db2.jdbc.net.DB2Driver

地址：jdbc:db2://127.0.0.1:6789/demodb

4、Hibernate SQL 方言 (hibernate.dialect)

RDBMS	方言
DB2	org.hibernate.dialect.DB2Dialect
DB2 AS/400	org.hibernate.dialect.DB2400Dialect
DB2 OS390	org.hibernate.dialect.DB2390Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
MySQL	org.hibernate.dialect.MySQLDialect
MySQL with InnoDB	org.hibernate.dialect.MySQLInnoDBDialect

RDBMS	方言
MySQL with MyISAM	org.hibernate.dialect.MySQLMyISAMDialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle 9i/10g	org.hibernate.dialect.Oracle9Dialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Ingres	org.hibernate.dialect.IngresDialect
Progress	org.hibernate.dialect.ProgressDialect
Mckoi SQL	org.hibernate.dialect.MckoiDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Pointbase	org.hibernate.dialect.PointbaseDialect
FrontBase	org.hibernate.dialect.FrontbaseDialect
Firebird	org.hibernate.dialect.FirebirdDialect

三、常用注解

3.1 批注完全参考

@Entity 要将 Java 类指定为 JPA 实体，请使用批注
详细信息


1 JPA 批注参考.htm

3.2 ID 相关的

复合主键需要

1、 实现序列话
2、 重写 hascode、equal 方法
3、 有构造方法

@Embeddable 复合主键设置可以被引用
@EmbeddedId 引用独立复合主键 ID

3.3 主键生成策略

使用 Hibernate 的主键生成策略生成字符串主键

```
@Id
@GenericGenerator(name="generator",strategy="uuid")
@GeneratedValue(generator="generator")
@Column(name="id")
```

使用 Hibernate 的主键生成策略与其他类共享主键

```
@Id
    @GenericGenerator(name = "generator",
                      strategy = "foreign",
                      parameters = {
              @Parameter(name = "property", value = "person")
            })
  @GeneratedValue(generator = "generator")
  @Column(name="cid")
```

3.4 字段、添加字段、添加表关联

@Column 持久化字段
可添加、可更新、可为空
长度、表名、字段名、unique 是否唯一

@JoinColumn
name 列名
referencedColumnName 指向对象的列名
unique 约束唯一
@JoinColumns 多个连接的列

```
    @JoinColumns({
    @JoinColumn(name="ADDR_ID",
    referencedColumnName="ID"),
    @JoinColumn(name="ADDR_ZIP",
    referencedColumnName="ZIP")
        })
```

@JoinTable

```
    @JoinTable(
    name="EJB_PROJ_EMP",
    joinColumns=@JoinColumn(name="EMP_ID",
    referencedColumnName="ID"),
```

```
inverseJoinColumns=@JoinColumn(name="PROJ_ID",
referencedColumnName="ID")
)
```

3.5 映射相关

@OneToOne
@ManyToOne
@ManyToMany

cascade 级联、CURD
fetch 一次性全部读取相关对象，还是 lazy 加载
optional 关联对象是否允许为空
targetEntity 关联对象

3.6 其他

排序
@OrderBy("lastname ASC", "seniority DESC")

共享主键
@PrimaryKeyJoinColumn

标注为非持久话对象
@Transient
时间类型
@Temporal（TemporalType.……）
枚举类型
@Enumerated(EnumType.……)

@Lob //声明属性对应的是一个大文件数据字段。
@Basic(fetch = FetchType.LAZY) //设置为延迟加载，当我们在数据库中取这条记录的时候，不会去取

四、JPA 映射

4.1 一对一映射

4.1.1 共享主键映射

1、一端提供主键、一端共享主键
设置生成策略 generator，主键值 gereratedValue(generator="")
uuid 字符串 ID
foreign 引用别人 ID 作为自己的主键(需要设置引用对象参数)
2、oneToOne
targetEntity 关联的目标对象，是类名.class 形式
fetch 抓去策略，有关联的一起抓去、还是 lazy 加载
cascade 级联
mappedBy 本对象被映射为*(在 PrimaryKeyJoinColumn 的另一端)
3、PrimaryKeyJoinColumn 设置在一端即可
name 自身字段
referenceColumnName 指向对象的字段
注意：
只要一个 PrimaryKeyJoinColumn，另一端的 oneToOne 设置 mappedBy
都只有一个
注意：
1、向共享主键的对象设置提供主键的对象，然后持久化共享主键对象
2、需要设置级联
3、共享主键端维护关系、提供主键端被维护使用 mappedBy

Person 提供主键

```
@Id
@GenericGenerator(name = "generator", strategy = "uuid")
@GeneratedValue(generator = "generator")
@Column(name="pid")
private String id;

@Column(name="pname",length=2)
private String name;

@OneToOne(mappedBy="person",fetch=FetchType.EAGER,targetEntity=Id
card.class)
private Idcard idcard;
```

Idcard 共享主键

```
@Id
@GenericGenerator(name = "generator",
        strategy = "foreign",
```

```
            parameters = {
              @Parameter(name = "property", value = "person")
            })
    @GeneratedValue(generator = "generator")
    @Column(name="cid")
    private String id;
    @Column(name="cno")
    private String no;
    @OneToOne(targetEntity=Person.class,fetch=FetchType.EAGER,cascade
=CascadeType.ALL)
    @PrimaryKeyJoinColumn(name="id",referencedColumnName="id")
    private Person person;

    @Id
    @GenericGenerator(name = "generator",
            strategy = "foreign",
            parameters = {
              @Parameter(name = "property", value = "person")
            })
            @GeneratedValue(generator = "generator")
    @Column(name="cid")
    private String id;
    @Column(name="cno")
    private String no;
    @OneToOne(mappedBy="idcard",fetch=FetchType.EAGER,targetEntity=Pe
rson.class)
    private Person person;
```

使用共享主键关联

```
            Person p = new Person();
            p.setName("ader");
            Idcard idcard = new Idcard();
            idcard.setNo("321321");
            idcard.setPerson(p);
```

4.1.2 关联外键映射

关系的维护端
@OneToOne(级联)
@JoinColumn(name="本表中关联字段",referencedColumnName="指向字段")
被维护端
@OneToOne(mappedBy="")

关系维护端添加一个字段作为外键指向被维护段。被维护端声明 mappedBy

```
@Entity
    @Table(name="Test_Trousers")
    public class Trousers {
       @Id
       public Integer id;
       @OneToOne
       @JoinColumn(name = "zip_id")
       public TrousersZip zip;
    }

    @Entity
    @Table(name="Test_TrousersZip")
    public class TrousersZip {
       @Id
       public Integer id;
       @OneToOne(mappedBy = "zip")
       public Trousers trousers;
        }
```

4.1.3 添加表关联

添加关联表的一一关联

```
@OneToOne
@JoinTable(name ="关联表名称",
 joinColumns = @JoinColumn(name="关联本表的字段"),
 inverseJoinColumns = @JoinColumn(name="要关联的表的字段")
)
@Entity
@Table(name="Test_People")
public class People {
   @Id
   public Integer id;
   @OneToOne
   @JoinTable(name ="TestPeoplePassports",
```

```
    joinColumns = @JoinColumn(name="perple_fk"),
    inverseJoinColumns = @JoinColumn(name="passport_fk")
  )
  public Passport passport;
}

@Entity
@Table(name="Test_Passport")
public class Passport {
  @Id
  public Integer id;
  @OneToOne(mappedBy = "passport")
  public People people;
}
```

4.2 一对多关联

我们维护的多是他们的关系 而不是实体所以建议使用添加关系表
在多的一端：添加字段、或者添加表 设置级联添加
在一端： mappedBy
用法：new 一个一的一端对象 set 到多的对象中，在持久化多的一端

4.2.1 添加字段的一对多、多对一关联

manyToOne 多对一
oneToMany 用一个 set 存放对象
添加字段关联，一般是在多的一端维护关系，设置级联。一的一端 mappedBy

```
@Id
@GenericGenerator(name="t1",strategy="uuid")
@GeneratedValue(generator="t1")
@Column(name="cid")
private String cid;
@Column(name="cname")
```

```
    private String cname;
    @OneToMany(fetch=FetchType.EAGER,mappedBy="cls")
    private Set<Student> studentSet;


    @Id
    @GenericGenerator(name="t2",strategy="uuid")
    @GeneratedValue(generator="t2")
    @Column(name="id")
    private String id;
    @Column(name="name")
    private String name;
    @ManyToOne(fetch=FetchType.EAGER,cascade=CascadeType.PERSIST,targ
etEntity=Clas.class)
    @JoinColumn(name="classid",referencedColumnName="cid")
    private Clas cls;
```

4.2.2 添加表的一对多、多对一关联

多的一端

```
    @Id
    @GenericGenerator(name="generator",strategy="uuid")
    @GeneratedValue(generator="generator")
    @Column(name="id")
    private String id;
    @Column(name="name")
    private String name;
    @ManyToOne(targetEntity=C.class,cascade = CascadeType.PERSIST)
    @JoinTable(name="csrel",

    joinColumns=@JoinColumn(name="sid",referencedColumnName="id"),

    inverseJoinColumns=@JoinColumn(name="cid",referencedColumnName="c
id")
    )
    private C c;
```

一的一端

```
    @Id
    @GenericGenerator(name="generator",strategy="uuid")
    @GeneratedValue(generator="generator")
    @Column(name="cid")
    private String id;
```

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

@OneToMany(targetEntity=S.class,mappedBy="c")
private Set<S> sset;
```

4.3 多对多关联

添加表关联

假设Teacher 和 Student是多对多的关系，具体元数据声明如下：

```
pubic class Teacher{
@ManyToMany(targetEntity = Student.class, cascade =
CascadeType.PERSIST)
@JoinTable(table = @Table(name = "M2M_TEACHER_STUDENT"),
joinColumns = @JoinColumn(name = "TEACHER_ID", referencedColumnName
= "ID"),
inverseJoinColumns = @JoinColumn(name = "STUDENT_ID",
referencedColumnName = "ID"))
public List<Student> getStudents() {
return students;
}
}

public class Student{
@ManyToMany(targetEntity = Teacher.class, mappedBy = "students")
public List<Teacher> getTeachers() {
return teachers;       }
}
**********************************************
Student student = em.find(Student.class, 1);
Teacher teacher = em.getReference(Teacher.class, 1);
student.removeTeacher(teacher);
```

在一个事务中

```
**********************************************
```

4.4 继承映射

继承关系映射在同一张表中

```
@Entity
@Table(name = "customer")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "flag", discriminatorType =
DiscriminatorType.STRING)
public class Customer{
}

@Entity
@DiscriminatorValue(value = "A")
public class CustomerA extends Customer{
}

@Entity
@DiscriminatorValue(value = "B")
public class CustomerB extends Customer{
}
```

SINGLE_TABLE：父子类都保存到同一个表中，通过字段值进行区分。
JOINED：父子类相同的部分保存在同一个表中，不同的部分分开存放，通过表连 接获取完整数据；

TABLE_PER_CLASS：每一个类对应自己的表，一般不推荐采用这种方式。

五、JPQL

JPQL 是一种强类型语言，一个 JPQL 语句中每个表达式都有类型。
JPQL 类似 Hibernate 的 HQL
单字段 返回具体类型
多字段 返回多个具体类型的数组

查询单个字段，返回 List List<String>
查询单个字段，返回 List 只有一条记录 还是 List<String>

查询多个字段，返回 List List<Object[]> 对象数组由多个字段值组成
查询多个字段，返回 List 只有一条记录 还是 List<Object[]>

	一条记录 getSingleResult	多条记录
单一字段	单一值对象	List<Object>
多个字段	值对象组成的数组	List<Object[]>
实体对象	一个对象 entity	List<entity>

JPA 操作的实体

需要@Entity name 注解

或者是限定词（com.sunyard.User 为全称，User 为限定词）

标识变量

select u from User; //这里 u 就是标识变量

路径表达式

表示福报 + 访问操作符(.) + 状态字段/关联字段

状态字段：包括一个实体不代表关联关系的任何字段

关联字段：包括了实体中任何有关联关系的字段或者属性

参数（select u from User u where uid=?1 and uname=:name）

? 1 位置参数 setParameter('name','value')

: name 命名参数 setParameter('name','value')

不建议在 SQL 语句中直接写参数，防止注入。

六、常见异常

1、异常信息：org.hibernate.hql.ast.QuerySyntaxException: person is not mapped

异常环境：查询

异常原因：查询语句中 Person 类没有大写

2、java.lang.ClassCastException: [Ljava.lang.Object; cannot be cast to java.lang.String

异常环境：查询、遍历显示

异常原因：转型出错

3、javax.persistence.NonUniqueResultException: result returns more than one elements

异常环境：查询、getSingleResult

异常原因：getSingleResult 只能获取一条数据，而查询语句返回的是多条数据

4、org.hibernate.PropertyValueException: not-null property references a null or transient value: com.sunyard.entities.Person.name

异常环境：数据插入

异常原因：JPA 的 Entity 中一个属性定义为 nullable=false，插入数据该字段为 null

5、 执行添加没反应、没异常

异常原因：没有开启事务、没有提交事务

6 、 javax.persistence.PersistenceException: org.hibernate.PersistentObjectException: detached entity passed to persist: com.sunyard.entities.Person

异常环境：OneToOne 共享主键关联

异常原因：一对一中，一个提供主键、另一个共享其主键，共享主键的对象可以 set 提供主键的对象 然后添加到数据库中

方向弄反了 后果就是没人提供主键

7 、 org.hibernate.TransientObjectException: object references an unsaved transient instance - save the transient instance before flushing:

异常环境：多对一添加

异常原因：在多的一端维护 ，没有添加级联

8、javax.persistence.PersistenceException: [PersistenceUnit: JPA] Unable to configure EntityManagerFactory

异常原因：很多、实体管理器 Factory 没有成功创建，是注解的问题

9、org.hibernate.MappingException: Unable to find column with logical name: sid in org.hibernate.mapping.

异常环境：添加表做多对一关联映射

异常原因：表字段写反了，name 添加表字段名 referencedColumnName 指向本表字段名