

# Hibernate的学习记录总结

## 一、简介

用来记录我在学习hibernate中的知识点内容。

## 用到的数据库

单表操作

- people

id	name	money
1	张三	1000

一对多级联

- customer

id	name
6	张三

- orders

id	name	c_id
8	订单1	6

多对多级联

- account

id	name
2	张三

- course

id	name
2	Java

- account\_course

id	a_id	cid
1	2	2

## 二、Hibernate工程创建的步骤

### 1、创建工程，导入相关依赖（pom.xml）

```
<?xml version="1.0" encoding="UTF-8"?>
<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/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>hibernate</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.20</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.4.20.Final</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.12</version>
        </dependency>
    </dependencies>
    <!-- 用于读取java里的xml文件-->
    <build>
        <resources>
            <resource>
```

```

        <directory>src/main/java</directory>
        <includes>
            <include>**/*.xml</include>
        </includes>
    </resource>
</resources>
</build>
</project>

```

## 2、创建Hibernate配置文件

文件位置：src/main/resources/hibernate.cfg.xml

```

<?xml- version="1.0".encoding="UTF-8"?>
    <!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">
    <hibernate-configuration>
        <session-factory>
            <!-- 数据源配置 -->
            <property
name="connection.username">admin</property>
            <property
name="connection.password">123</property>
            <property
name="connection.driver_class">com.mysql.cj.jdbc.Driver</p
roperty>
            <property
name="connection.url">jdbc:mysql://localhost:3305/test?
useUnicode=true&amp;characterEncoding=utf8&amp;serverTimez
one=UTC&amp;allowMultiQueries=true</property>

            <!-- c3p0数据库连接池 -->
            <property
name="hibernate.c3p0.acquire_increment">10</property>
            <property
name="hibernate.c3p0.idle_test_period">10000</property>
            <property
name="hibernate.c3p0.timeout">5000</property>

```

```

        <property
name="hibernate.c3p0.max_size">30</property>
        <property
name="hibernate.c3p0.min_size">5</property>
        <property
name="hibernate.c3p0.max_statements">10</property>

        <!-- 数据库方言：根据数据库类型（mysql还是
oracle）识别sql语句类型 -->
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</propert
y>

        <!-- 打印SQL语句 -->
        <property name="show_sql">true</property>

        <!-- 格式化SQL语句 -->
        <property
name="format_sql">true</property>

        <!-- 是否自动生成数据库 -->
        <property name="hibernate.hbm2ddl.auto">
</property>

        <!-- 注册实体关系映射文件 -->
        <mapping
resource="com/gloryh/entity/People.hbm.xml"></mapping>
        <mapping
resource="com/gloryh/entity/Customer.hbm.xml"></mapping>
        <mapping
resource="com/gloryh/entity/Orders.hbm.xml"></mapping>
        <mapping
resource="com/gloryh/entity/Account.hbm.xml"></mapping>
        <mapping
resource="com/gloryh/entity/Course.hbm.xml"></mapping>
    </session-factory>
</hibernate-configuration>

```

核心配置：session—factory

Session Factory：针对单个数据库映射 经过编译的内存镜像文件，将数据库转化成一个Java可以识别的镜像文件。

构建SessionFactory非常耗费资源，所以通常一个工程只需要创建一个SessionFactory，Mybatis也是如此。

### 3、创建实体类

```
package com.gloryh.entity;

import lombok.Data;

/**
 * People实体类
 *
 * @author 黄光辉
 * @since 2020/8/21
 */
@Data
public class People {
    private Integer id;
    private String name;
    private Double money;
}
```

```
package com.gloryh.entity;

import lombok.Data;
import lombok.Getter;
import lombok.Setter;

import java.util.Set;

/**
 * Customer实体类
 *
 * @author 黄光辉
 * @since 2020/8/21
 */
@Getter
@Setter
public class Customer {
```

```

    private Integer id;
    private String name;
    /**
     * 外键，采用集合方式映射 一对多
     */
    private Set<Orders> orders;
}

```

```

package com.gloryh.entity;

import lombok.Getter;
import lombok.Setter;

/**
 * Orders实体类
 *
 * @author 黄光辉
 * @since 2020/8/21
 */
@Getter
@Setter
public class Orders {
    private Integer id;
    private String name;
    /**
     * 外键 多对一，映射customer
     */
    private Customer customer;
}

```

```

package com.gloryh.entity;

import lombok.Getter;
import lombok.Setter;

import java.util.Set;

/**
 * Account实体类

```

```

*
* @author 黄光辉
* @since 2020/8/21
**/
@Getter
@Setter
public class Account {
    private Integer id;
    private String name;
    private Set<Course> courses;
}

```

```

package com.gloryh.entity;

import lombok.Getter;
import lombok.Setter;

import java.util.Set;

/**
 * Course实体类
 *
 * @author 黄光辉
 * @since 2020/8/21
 **/
@Getter
@Setter
public class Course {
    private Integer id;
    private String name;
    private Set<Account> accounts;
}

```

## 4、创建实体类-关系映射文件

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping>

```

```

<class name="com.gloryh.entity.People" table="people">
  <id name="id" type="java.lang.Integer">
    <column name="id"></column>
    <generator class="identity"></generator>
  </id>
  <property name="name" type="java.lang.String">
    <column name="name"></column>
  </property>
  <property name="money" type="java.lang.Double">
    <column name="money"></column>
  </property>
</class>
</hibernate-mapping>

```

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping>
  <class name="com.gloryh.entity.Customer"
table="customer">
    <id name="id" type="java.lang.Integer">
      <column name="id"></column>
      <generator class="identity"></generator>
    </id>
    <property name="name" type="java.lang.String">
      <column name="name"></column>
    </property>
    <!-- 设置延迟加载（默认为开启状态）添加字段 lazy="true"
会只加载我们需要的数据而不加载级联数据 -->
    <!-- 若未开启延迟加载，添加字段lazy="false" 在执行SQL时
会默认进行级联查询，获取所有相关数据 -->
    <!-- 当添加的字段为 lazy="extra" 时，会使用一种比延迟加
载更懒（智能）的加载方式 -->
    <set name="orders" table="orders" lazy="true">
      <key column="c_id"></key>
      <one-to-many class="com.gloryh.entity.Orders">
</one-to-many>
    </set>

```



```
    </class>
</hibernate-mapping>
```

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.gloryh.entity.Orders" table="orders">
        <id name="id" type="java.lang.Integer">
            <column name="id"></column>
            <generator class="identity"></generator>
        </id>
        <property name="name" type="java.lang.String">
            <column name="name"></column>
        </property>
        <!-- 在many-to-one此标签下使用延迟加载用proxy 和 no-
proxy, 不开启延迟加载使用false -->
        <!-- lazy="false" 不开启延迟加载, 级联出相关联的表 -->
        <!-- lazy="proxy" 无论调用方法是否需要访问customer的成
员变量, 都会发送SQL语句查询Customer -->
        <!-- lazy="no-proxy" 当调用方法需要访问customer的成员
变量时, 发送SQL语句查询Customer, 否则不查询 -->
        <many-to-one name="customer"
class="com.gloryh.entity.Customer" column="c_id">

        </many-to-one>

    </class>
</hibernate-mapping>
```

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.gloryh.entity.Account"
table="account">
```

```

        <id name="id" type="java.lang.Integer">
            <column name="id"></column>
            <generator class="identity"></generator>
        </id>
        <property name="name" type="java.lang.String">
            <column name="name"></column>
        </property>
        <set name="courses" table="account_course">
            <key column="a_id"></key>
            <many-to-many class="com.gloryh.entity.Course"
column="cid"></many-to-many>
        </set>

    </class>
</hibernate-mapping>

```

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.gloryh.entity.Course" table="course">
        <id name="id" type="java.lang.Integer">
            <column name="id"></column>
            <generator class="identity"></generator>
        </id>
        <property name="name" type="java.lang.String">
            <column name="name"></column>
        </property>
        <set name="accounts" table="account_course"
lazy="true">
            <key column="cid"></key>
            <many-to-many
class="com.gloryh.entity.Account" column="a_id"></many-to-
many>
        </set>

    </class>
</hibernate-mapping>

```

## 5、调用Hibernate API 完成操作

### 1.实体类映射文件注册到Hibernate的配置文件中

```
<!-- 注册实体关系映射文件 -->
<mapping resource="com/gloryh/entity/People.hbm.xml">
</mapping>
<mapping resource="com/gloryh/entity/Customer.hbm.xml">
</mapping>
<mapping resource="com/gloryh/entity/Orders.hbm.xml">
</mapping>
<mapping resource="com/gloryh/entity/Account.hbm.xml">
</mapping>
<mapping resource="com/gloryh/entity/Course.hbm.xml">
</mapping>
```

注：即hibernate.cfg.xml文件（可参照第二步中的Hibernate.cfg.xml看一下具体操作）中

### 2.使用Hibernate API 完成数据操作（示例为单表）

```
package com.gloryh.test;

import com.gloryh.entity.People;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

/**
 * 测试类
 *
 * @author 黄光辉
 * @since 2020/8/21
 */
public class Test {
    public static void main(String[] args) {
        //创建Configuration
        Configuration configuration = new
Configuration().configure();
        System.out.println(configuration);
        //获取SessionFactory
```

```
SessionFactory
sessionFactory=configuration.buildSessionFactory();
//获取Session
Session session = sessionFactory.openSession();
People people = new People();
people.setName("张三");
people.setMoney(1000.0);
session.save(peopple);
session.beginTransaction().commit();
session.close();

}
}
```

注：这里我使用了默认的Hibernate配置文件名（Hibernate.cfg.xml），如果并没有使用默认的配置文件名(如Hibernate.xml)，则需要在加载配置文件时输入文件名（即 Configuration configuration = new Configuration().configure("Hibernate.xml");）。

## 三、Hibernate的级联操作

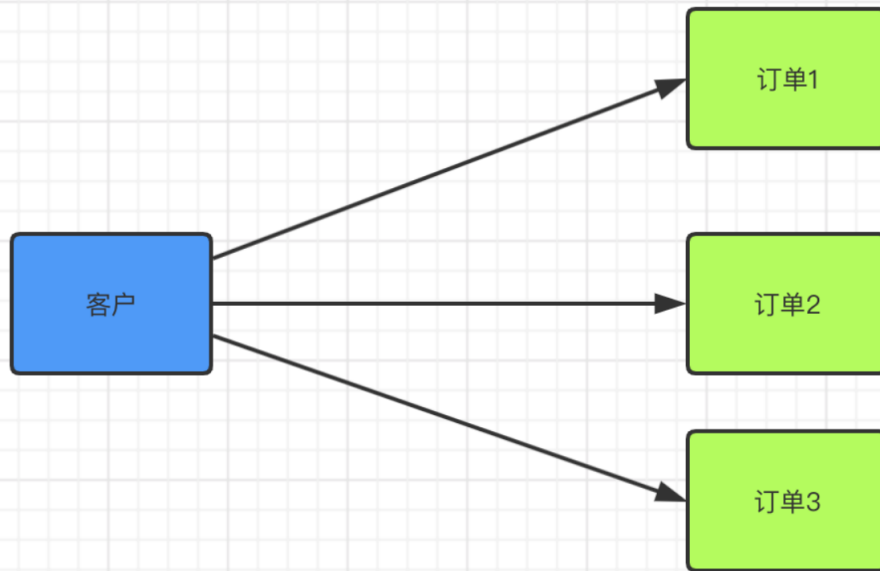
Java和数据库对于这一对多关系和多对多关系的体现完全是两种不同的方式，Hibernate框架的作用就是将这两种方式进行转换和映射。

### 1、一对多关系

#### 1.Java的体现

客户和订单：每个客户可以购买多个产品，生成多个订单，但是一个订单只能属于一个客户，所以客户为一，订单为多。

使用的表：客户表（Customer）和订单表（Order），其中主表为客户表，次表为订单表，使用主外键进行二表级联。



在面向对象中，使用成员变量来维护，即在主表中将次表的实体类以Set集合的方式添加进来，在次表中则调用主表的实体类对应，代码如下：

```
package com.gloryh.entity;

import lombok.Getter;
import lombok.Setter;

import java.util.Set;

/**
 * Customer实体类
 *
 * @author 黄光辉
 * @since 2020/8/21
 */
@Getter
@Setter
public class Customer {
    private Integer id;
    private String name;
    /**
     * 外键，采用集合方式映射 一对多
     */
    private Set<Orders> orders;

    /**
```

\* 重写toString方法，不加载Set<Orders>，防止出现相互调用形成死循环，造成内存溢出

```
* @return
*/
@Override
public String toString() {
    return "Customer{" +
        "id=" + id +
        ", name='" + name + '\'' +
        '}';
}
}
```

```
package com.gloryh.entity;
```

```
import lombok.Getter;
import lombok.Setter;
```

```
/**
```

```
* orders实体类
```

```
*
```

```
* @author 黄光辉
```

```
* @since 2020/8/21
```

```
**/
```

```
@Getter
```

```
@Setter
```

```
public class Orders {
```

```
    private Integer id;
```

```
    private String name;
```

```
    /**
```

```
     * 外键    多对一，映射customer
```

```
    */
```

```
    private Customer customer;
```

```
    /**
```

\* 重写toString方法，不加载Customer，防止出现相互调用形成死循环，造成内存溢出

```
    * @return
```

```
    */
```

```
@Override
```

```
public String toString() {
```

```

        return "Orders{" +
            "id=" + id +
            ", name='" + name + '\'' +
            '}';
    }
}

```

## 2.Hibernate的实现

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.gloryh.entity.Customer"
table="customer">
        <id name="id" type="java.lang.Integer">
            <column name="id"></column>
            <generator class="identity"></generator>
        </id>
        <property name="name" type="java.lang.String">
            <column name="name"></column>
        </property>
        <!-- 一对多用set one-to-many -->
        <set name="orders" table="orders" >
            <key column="c_id"></key>
            <one-to-many class="com.gloryh.entity.Orders">
</one-to-many>
        </set>

    </class>
</hibernate-mapping>

```

- set标签用来配置实体类中的集合属性orders
- name标签用来配置实体类的属性名
- table用来配置数据库中对应的表名
- key用来配置外键
- one-to-many与集合泛型的实体类对应

```

<?xml version="1.0"?>

```

```

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.gloryh.entity.Orders" table="orders">
        <id name="id" type="java.lang.Integer">
            <column name="id"></column>
            <generator class="identity"></generator>
        </id>
        <property name="name" type="java.lang.String">
            <column name="name"></column>
        </property>
        <!-- 多对一用many-to-one -->
        <many-to-one name="customer"
class="com.gloryh.entity.Customer" column="c_id">

            </many-to-one>

        </class>
</hibernate-mapping>

```

- many-to-one标签配置实体类对应的对象属性
- name 属性名
- class 属性对应的类
- column 外键

### 3.Hibernate API实现

```

//创建Configuration
Configuration configuration = new
Configuration().configure();
//获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
//获取session
Session session =sessionFactory.openSession();
//创建Customer对象
Customer customer = new Customer();
customer.setName("李四");
//创建Orders

```



```

Orders orders = new Orders();
orders.setName("订单2");
//建立关联关系
orders.setCustomer(customer);

//保存
session.save(customer);
session.save(orders);

//提交事务
session.beginTransaction().commit();

//关闭session
session.close();

```

```

INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.
Hibernate:
insert
into
    customer
    (name)
values
    (?)
Hibernate:
insert
into
    orders
    (name, c_id)
values
    (?, ?)
Process finished with exit code 0

```

对象	customer @test (8.0) - 表	orders @test (8.0) - 表
开始事务	文本 ▾	筛选 ▾ 排序 ▾ 导入 导出
id	name	
▶ 6	张三	
7	李四	

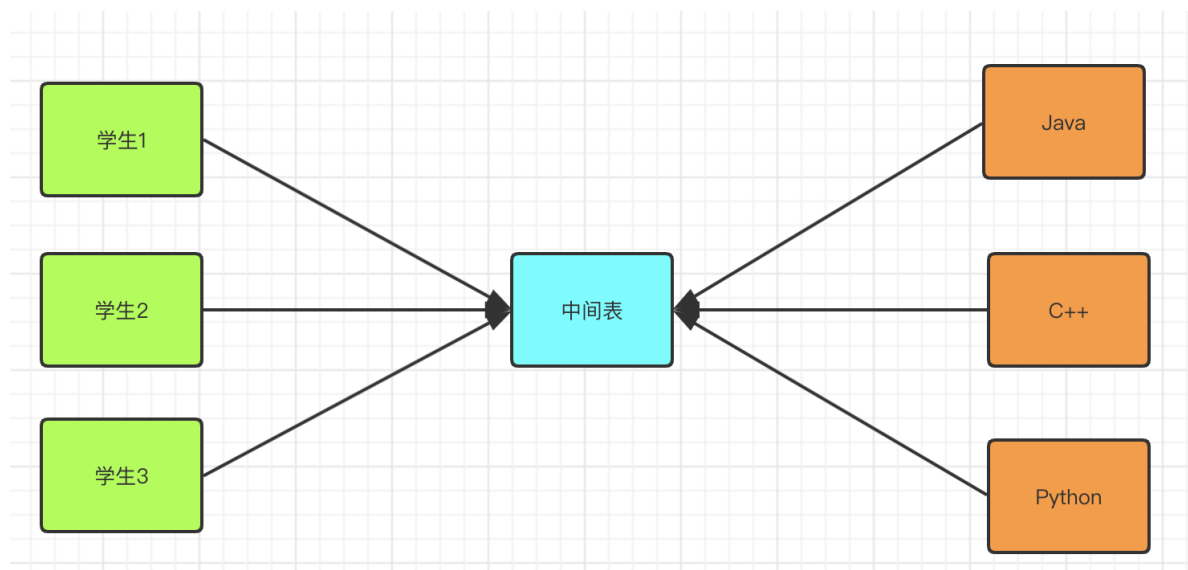
对象	customer @test (8.0) - 表	orders @test (8.0) - 表
开始事务	文本 ▾	筛选 ▾ 排序 ▾ 导入 导出
id	name	c_id
▶ 8	订单1	6
9	订单2	7

## 2、多对多关系

### 1.Java的体现

学生选课：一门课程可以被多个学生选择，一个学生又可以选择多门课程，所以学生是多，课程也是多。

使用的表：数据库中采用两个一对多来维护。所以学生表（account）和课程表（course）都是主表，额外增加一张中间表（account\_course）作为次表，两张表和中间表都是一对多关系，采用主外键来进行级联。



在面向对象中，两个主表都以Set集合的形式将对方的实体类文件以成员变量方式添加进来，代码如下：

```
package com.gloryh.entity;

import lombok.Getter;
import lombok.Setter;

import java.util.Set;

/**
 * Account实体类
 *
 * @author 黄光辉
 * @since 2020/8/21
 */
@Getter
@Setter
```

```

public class Account {
    private Integer id;
    private String name;
    private Set<Course> courses;

    @Override
    public String toString() {
        return "Account{" +
            "id=" + id +
            ", name='" + name + '\'' +
            '}';
    }
}

package com.gloryh.entity;

import lombok.Getter;
import lombok.Setter;

import java.util.Set;

/**
 * Course实体类
 *
 * @author 黄光辉
 * @since 2020/8/21
 */
@Getter
@Setter
public class Course {
    private Integer id;
    private String name;
    private Set<Account> accounts;

    @Override
    public String toString() {
        return "Course{" +
            "id=" + id +
            ", name='" + name + '\'' +
            '}';
    }
}

```

## 2.Hibernate的实现

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.gloryh.entity.Course" table="course">
        <id name="id" type="java.lang.Integer">
            <column name="id"></column>
            <generator class="identity"></generator>
        </id>
        <property name="name" type="java.lang.String">
            <column name="name"></column>
        </property>
        <set name="accounts" table="account_course"
lazy="true">
            <key column="cid"></key>
            <many-to-many
class="com.gloryh.entity.Account" column="a_id"></many-to-
many>
            </set>

        </class>
</hibernate-mapping>
```

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.gloryh.entity.Account"
table="account">
        <id name="id" type="java.lang.Integer">
            <column name="id"></column>
            <generator class="identity"></generator>
        </id>
        <property name="name" type="java.lang.String">
            <column name="name"></column>
```

```

        </property>
        <set name="courses" table="account_course">
            <key column="a_id"></key>
            <many-to-many class="com.gloryh.entity.Course"
column="cid"></many-to-many>
        </set>

    </class>
</hibernate-mapping>

```

- name 实体类对应集合的属性名
- table 中间表名
- key 外键
- many-to-many 与集合泛型的实体类对应
- column 属性与中间表的外键字段名相对应

### 3.Hibernate API实现

```

// 创建Configuration
Configuration configuration = new
Configuration().configure();
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取Session
Session session = sessionFactory.openSession();

// 创建Course对象
Course course = new Course();
course.setName("Python");

// 创建Account对象
Account account = new Account();
account.setName("李四");

// 创建Course集合
Set<Course> courses = new HashSet<>();
courses.add(course);

// 将Course集合赋值给Account
account.setCourses(courses);

```

```
// 保存
session.save(account);
session.save(course);

// 提交事务
session.beginTransaction().commit();

// 关闭连接
session.close();
```

```
INFO: HHH0000490: Using JtaPlatform implementation: [org.hib
Hibernate:
    insert
    into
        account
        (name)
    values
        (?)
Hibernate:
    insert
    into
        course
        (name)
    values
        (?)
Hibernate:
    insert
    into
        account_course
        (a_id, cid)
    values
        (?, ?)
```

对象	course @test (8.0) - 表	account_course @test (8.0) ...	account @test (8.0) - 表
开始事务	文本 ▾	筛选	排序
	导入	导出	
id	name		
2	张三		
3	李四		

对象	course @test (8.0) - 表	account_course @test (8.0) ...	account @test (8.0) - 表
开始事务	文本 ▾	筛选	排序
	导入	导出	
id	name		
2	Java		
3	Python		

对象			
course @test (8.0) - 表			
account_course @test (8.0) ...			
account @test (8.0) - 表			
开始事务 文本 筛选 排序 导入 导出			
id	a_id	cid	
1	2	2	
2	3	3	

## 四、Hibernate的延迟加载

Hibernate的加载方式分为延迟加载、惰性加载和懒加载。

使用延迟加载可以降低Java程序与数据库的交互频次，从而提高程序的运行效率。

思路：当我们只需要一个表的数据时，因为级联的原因，我们可能要执行两个数据库查询语句，造成资源的浪费。例如：我们在查询客户表信息而不需要订单表信息（Customer和Orders），不使用延迟加载的话，因为二者级联的原因，需要执行查询客户表的信息和订单表的对应信息，就需要两条SQL语句。使用延迟加载的话，就只执行查询客户表信息的操作，只执行一条SQL语句。

延迟加载可以看作一种优化机制，根据具体的需求，自动选择要执行的SQL语句数量。

可以使用 lazy 字段来控制延迟加载的开启和方式。

注：延迟加载默认是开启的。

### 1、一对多的延迟加载

测试用到的方法：

1.只查询一个主表

```
// 创建Configuration
Configuration configuration = new
Configuration().configure();
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取session
Session session = sessionFactory.openSession();

// 返回查询结果(id=6)
Customer customer = session.get(Customer.class, 6);
// 打印结果,此时打印并无orders数据
System.out.println(customer);
// 关闭连接
session.close();
```

## 2.查询主表和次表内对应的信息

```
// 创建Configuration
Configuration configuration = new
Configuration().configure();
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取session
Session session = sessionFactory.openSession();

// 返回查询结果(id=6)
Customer customer = session.get(Customer.class, 6);
// 打印结果,此时打印调用获取orders
System.out.println(customer.getOrders());
// 关闭连接
session.close();
```

## 3.查询一个主表和次表对应信息的数量



```

// 创建Configuration
Configuration configuration = new
Configuration().configure();
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取session
Session session = sessionFactory.openSession();

// 返回查询结果(id=6)
Customer customer = session.get(Customer.class, 6);
// 打印结果,此时打印对应Orders的数量
System.out.println(customer.getOrders().size());
// 关闭连接
session.close();

```

#### 4.只查询一个次表

```

// 创建Configuration
Configuration configuration = new
Configuration().configure();
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取session
Session session = sessionFactory.openSession();

// 返回查询结果(id=6)
Orders orders = session.get(Orders.class, 8);
// 打印结果,此时打印并无customer数据
System.out.println(orders);
// 关闭连接
session.close();

```

#### 5.查询次表和主表内对应的信息

```
// 创建Configuration
Configuration configuration = new
Configuration().configure();
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取session
Session session = sessionFactory.openSession();

// 返回查询结果(id=6)
Orders orders = session.get(Orders.class, 8);
// 打印结果,此时打印对应customer数据
System.out.println(orders.getCustomer());
// 关闭连接
session.close();
```

## 1.查询Customer, 对Orders不进行延迟加载设置, 在customer.hbm.xml中进行设置 (lazy="false")

```
<set name="orders" table="orders" lazy="false">
    <key column="c_id"></key>
    <one-to-many class="com.gloryh.entity.Orders"></one-
to-many>
</set>
```

运行结果:

1.只查询一个主表

```

Hibernate:
  select
    customer0_.id as id1_3_0_,
    customer0_.name as name2_3_0_
  from
    customer customer0_
  where
    customer0_.id=?

```

```

Hibernate:
  select
    orders0_.c_id as c_id3_4_0_,
    orders0_.id as id1_4_0_,
    orders0_.id as id1_4_1_,
    orders0_.name as name2_4_1_,
    orders0_.c_id as c_id3_4_1_
  from
    orders orders0_
  where
    orders0_.c_id=?

```

Customer{id=6, name='张三'}

## 2.查询主表和次表内对应的信息

```

Hibernate:
  select
    customer0_.id as id1_3_0_,
    customer0_.name as name2_3_0_
  from
    customer customer0_
  where
    customer0_.id=?

```

```

Hibernate:
  select
    orders0_.c_id as c_id3_4_0_,
    orders0_.id as id1_4_0_,
    orders0_.id as id1_4_1_,
    orders0_.name as name2_4_1_,
    orders0_.c_id as c_id3_4_1_
  from
    orders orders0_
  where
    orders0_.c_id=?
[Orders{id=8, name='订单1'}]

```

## 3.查询一个主表和次表对应信息的数量

```

Hibernate:
  select
    customer0_.id as id1_3_0_,
    customer0_.name as name2_3_0_
  from
    customer customer0_
  where
    customer0_.id=?
Hibernate:
  select
    orders0_.c_id as c_id3_4_0_,
    orders0_.id as id1_4_0_,
    orders0_.id as id1_4_1_,
    orders0_.name as name2_4_1_,
    orders0_.c_id as c_id3_4_1_
  from
    orders orders0_
  where
    orders0_.c_id=?
1

```

## 2.查询Customer，对Orders进行延迟加载设置，在customer.hbm.xml中进行设置（lazy="true"或不添加）

```

<set name="orders" table="orders" lazy="true">
  <key column="c_id"></key>
  <one-to-many class="com.gloryh.entity.Orders"></one-
to-many>
</set>

```

运行结果：

1.只查询一个主表

```

Hibernate:
  select
    customer0_.id as id1_3_0_,
    customer0_.name as name2_3_0_
  from
    customer customer0_
  where
    customer0_.id=?
Customer{id=6, name='张三'}

```

## 2.查询主表和次表内对应的信息

```

Hibernate:
  select
    customer0_.id as id1_3_0_,
    customer0_.name as name2_3_0_
  from
    customer customer0_
  where
    customer0_.id=?
Hibernate:
  select
    orders0_.c_id as c_id3_4_0_,
    orders0_.id as id1_4_0_,
    orders0_.id as id1_4_1_,
    orders0_.name as name2_4_1_,
    orders0_.c_id as c_id3_4_1_
  from
    orders orders0_
  where
    orders0_.c_id=?
[Orders{id=8, name='订单1'}]

```

## 3.查询一个主表和次表对应信息的数量

```

Hibernate:
  select
    customer0_.id as id1_3_0_,
    customer0_.name as name2_3_0_
  from
    customer customer0_
  where
    customer0_.id=?
Hibernate:
  select
    | orders0_.c_id as c_id3_4_0_,
    orders0_.id as id1_4_0_,
    orders0_.id as id1_4_1_,
    orders0_.name as name2_4_1_,
    orders0_.c_id as c_id3_4_1_
  from
    orders orders0_
  where
    orders0_.c_id=?

```

1

### 3.查询Customer，对Orders进行惰性加载设置，在customer.hbm.xml中进行设置（lazy="extra"）

```

<set name="orders" table="orders" lazy="extra">
  <key column="c_id"></key>
  <one-to-many class="com.gloryh.entity.Orders"></one-
to-many>
</set>

```

运行结果：

1.只查询一个主表

```

INFO: HHH0000490: Using JtaPlatform Impleme
Hibernate:
  select
    customer0_.id as id1_3_0_,
    customer0_.name as name2_3_0_
  from
    customer customer0_
  where
    customer0_.id=?
Customer{id=6, name='张三'}
进程已结束,退出代码0

```

## 2.查询主表和次表内对应的信息

```
Hibernate:
  select
    customer0_.id as id1_3_0_,
    customer0_.name as name2_3_0_
  from
    customer customer0_
  where
    customer0_.id=?
Hibernate:
  select
    orders0_.c_id as c_id3_4_0_,
    orders0_.id as id1_4_0_,
    orders0_.id as id1_4_1_,
    orders0_.name as name2_4_1_,
    orders0_.c_id as c_id3_4_1_
  from
    orders orders0_
  where
    orders0_.c_id=?
[Orders{id=8, name='订单1'}]
```

## 3.查询一个主表和次表对应信息的数量

```
Hibernate:
  select
    customer0_.id as id1_3_0_,
    customer0_.name as name2_3_0_
  from
    customer customer0_
  where
    customer0_.id=?
Hibernate:
  select
    count(id)
  from
    orders
  where
    c_id =?
1
```

#### 4.查询Orders, 对Customer不进行延迟加载的设置, 在Orders.hbm.xml中设置 (lazy="false")

```
<many-to-one name="customer"
class="com.gloryh.entity.Customer" column="c_id"
lazy="false">
</many-to-one>
```

运行结果:

4.只查询一个次表

```
Hibernate:
  select
    orders0_.id as id1_4_0_,
    orders0_.name as name2_4_0_,
    orders0_.c_id as c_id3_4_0_
  from
    orders orders0_
  where
    orders0_.id=?
Hibernate:
  select
    customer0_.id as id1_3_0_,
    customer0_.name as name2_3_0_
  from
    customer customer0_
  where
    customer0_.id=?
Orders{id=8, name='订单1'}
```

5.查询次表和主表内对应的信息



```

Hibernate:
  select
    orders0_.id as id1_4_0_,
    orders0_.name as name2_4_0_,
    orders0_.c_id as c_id3_4_0_
  from
    orders orders0_
  where
    orders0_.id=?
Hibernate:
  select
    customer0_.id as id1_3_0_,
    customer0_.name as name2_3_0_
  from
    customer customer0_
  where
    customer0_.id=?
Customer{id=6, name='张三'}

```

5.查询Orders, 对Customer不进行延迟加载的设置, 在Orders.hbm.xml中设置 (lazy="proxy"或不设置)

```

<many-to-one name="customer"
class="com.gloryh.entity.Customer" column="c_id"
lazy="proxy">
</many-to-one>

```

运行结果:

4.只查询一个次表

```
2020-03-24, 2020-03-27:30:17 / org.hibernate.  
INFO: HHH000490: Using JtaPlatform imple  
Hibernate:  
    select  
        orders0_.id as id1_4_0_,  
        orders0_.name as name2_4_0_,  
        orders0_.c_id as c_id3_4_0_  
    from  
        orders orders0_  
    where  
        orders0_.id=?  
Orders{id=8, name='订单1'}  
  
进程已结束,退出代码0
```

## 5.查询次表和主表内对应的信息

```
Hibernate:  
    select  
        orders0_.id as id1_4_0_,  
        orders0_.name as name2_4_0_,  
        orders0_.c_id as c_id3_4_0_  
    from  
        orders orders0_  
    where  
        orders0_.id=?  
Hibernate:  
    select  
        customer0_.id as id1_3_0_,  
        customer0_.name as name2_3_0_  
    from  
        customer customer0_  
    where  
        customer0_.id=?  
Customer{id=6, name='张三'}
```

## 总结

1.当我们只查询主表而不需要次表时, lazy状态为ture和extra时并没有执行查询orders表的操作, 而是指只执行了一条SQL语句用于查询Customer表, 而lazy状态为false时则不仅执行了查询Customer表的操作, 还执行了查询Orders中对应信息的操作, 共两条SQL语句。

所以此时, lazy状态为true和extra更加的节省资源, 更有效率。

2.当我们查询主表和次表内对应的信息时，三种情况执行情况一样。

3.当我们查询主表和次表对应信息的数量时，三种没有区别，都执行了两次SQL语句的操作，但是状态为extra的第二条查询语句返回的是count查询的结果，其余两种都是使用的普通查询语句。

所以此时，lazy的状态为extra更加的节省资源，更有效率

通过观察运行结果，我们可以发现，三种情况的运行结果中，智能度排序为：**extra>true>false**

4.当我们只查询次表而不需要主表时，lazy状态为proxy时并没有执行查询orders表的操作，而是指只执行了一条SQL语句用于查询Orders表，而lazy状态为false时则不仅执行了查询Orders表的操作，还执行了查询Customer中对应信息的操作，共两条SQL语句。

所以此时，lazy状态为proxy更加的节省资源，更有效率。

5.当我们查询次表和主表内对应的信息时，两种情况执行情况一样。

其实在对次表的设置中还存在no-proxy的字段（类似于extra），在使用该字段时需要编译时字节码增强，否则与proxy无区别。

次表操作中，智能度排序为：**no-proxy>proxy>false**

## 2、多对多关系的延迟加载

多对多的延迟加载与多对一中前三条差不多，具体可以自己测试。

# 五、Hibernate配置文件详解

## 1、hibernate.cfg.xml

hibernate.cfg.xml文件（可以更改名字，更改后加载文件需要输入文件的名称），该文件用于配置Hibernate的全局环境。

```
<?xml- version="1.0".encoding="UTF-8"?>
    <!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-
        configuration-3.0.dtd">
```

```
<hibernate-configuration>
    <session-factory>
        <!-- 数据源配置 -->
        <property
name="connection.username">admin</property>
        <property
name="connection.password">123</property>
        <property
name="connection.driver_class">com.mysql.cj.jdbc.Driver</p
roperty>
        <property
name="connection.url">jdbc:mysql://localhost:3305/test?
useUnicode=true&characterEncoding=utf8&serverTimez
one=UTC&allowMultiQueries=true</property>

        <!-- c3p0数据库连接池 -->
        <property
name="hibernate.c3p0.acquire_increment">10</property>
        <property
name="hibernate.c3p0.idle_test_period">10000</property>
        <property
name="hibernate.c3p0.timeout">5000</property>
        <property
name="hibernate.c3p0.max_size">30</property>
        <property
name="hibernate.c3p0.min_size">5</property>
        <property
name="hibernate.c3p0.max_statements">10</property>

        <!-- 数据库方言：根据数据库类型（mysql还是
oracle）识别sql语句类型 -->
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</propert
y>

        <!-- 打印SQL语句 -->
        <property name="show_sql">true</property>

        <!-- 格式化SQL语句 -->
        <property
name="format_sql">true</property>
```

```

        <!-- 是否自动生成数据库 -->
        <property name="hibernate.hbm2ddl.auto">
</property>

        <!-- 注册实体关系映射文件 -->
        <mapping
resource="com/gloryh/entity/People.hbm.xml"></mapping>
        <mapping
resource="com/gloryh/entity/Customer.hbm.xml"></mapping>
        <mapping
resource="com/gloryh/entity/Orders.hbm.xml"></mapping>
        <mapping
resource="com/gloryh/entity/Account.hbm.xml"></mapping>
        <mapping
resource="com/gloryh/entity/Course.hbm.xml"></mapping>
    </session-factory>
</hibernate-configuration>

```

## 数据库基本配置信息

```

<!-- 数据源配置 -->
    <property
name="connection.username">admin</property>
    <property
name="connection.password">123</property>
    <property
name="connection.driver_class">com.mysql.cj.jdbc.Driver</p
roperty>
    <property
name="connection.url">jdbc:mysql://localhost:3305/test?
useUnicode=true&characterEncoding=utf8&serverTimez
one=UTC&allowMultiQueries=true</property>

```

## 集成C3P0，设置数据库连接池信息

```
<!-- c3p0数据库连接池 -->
<property
name="hibernate.c3p0.acquire_increment">10</property>
<property
name="hibernate.c3p0.idle_test_period">10000</property>
<property name="hibernate.c3p0.timeout">5000</property>
<property name="hibernate.c3p0.max_size">30</property>
<property name="hibernate.c3p0.min_size">5</property>
<property
name="hibernate.c3p0.max_statements">10</property>
```

## Hibernate的基本信息

```
<!-- 数据库方言：根据数据库类型（mysql还是oracle）识别sql语句类型
-->
<property
name="dialect">org.hibernate.dialect.MySQLDialect</property>

<!-- 打印SQL语句 -->
<property name="show_sql">true</property>

<!-- 格式化SQL语句 -->
<property name="format_sql">true</property>

<!-- 是否自动生成数据库 -->
<property name="hibernate.hbm2ddl.auto"></property>
```

- update：动态创建表，表存在，则直接使用，不存在则直接创建
- create：无论表是否存在，都创建
- create-drop：初始化创建表，程序结束时删除表
- validate：检验实体关系映射文件和数据表是否对应，不能对应直接报错

## 注册实体类关系映射文件

```
<!-- 注册实体关系映射文件 -->
<mapping resource="com/gloryh/entity/People.hbm.xml">
</mapping>
<mapping resource="com/gloryh/entity/Customer.hbm.xml">
</mapping>
<mapping resource="com/gloryh/entity/Orders.hbm.xml">
</mapping>
<mapping resource="com/gloryh/entity/Account.hbm.xml">
</mapping>
<mapping resource="com/gloryh/entity/Course.hbm.xml">
</mapping>
```

## 2、实体关系映射文件

以Customer.hbm.xml文件为例：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.gloryh.entity.Customer"
table="customer">
        <id name="id" type="java.lang.Integer">
            <column name="id"></column>
            <generator class="identity"></generator>
        </id>
        <property name="name" type="java.lang.String">
            <column name="name"></column>
        </property>
        <set name="orders" table="orders" lazy="extra">
            <key column="c_id"></key>
            <one-to-many class="com.gloryh.entity.Orders">
</one-to-many>
        </set>
    </class>
</hibernate-mapping>
```

## hibernate-mapping 属性

- package: 给class节点对应的实体类统一设置包名, 设置包名后, 在给class的name属性进行定义时接可以省略包名。

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping package="com.gloryh.entity">
    <class name="Customer" table="customer">
        <id name="id" type="java.lang.Integer">
            <column name="id"></column>
            <generator class="identity"></generator>
        </id>
        <property name="name" type="java.lang.String">
            <column name="name"></column>
        </property>
        <!-- 设置延迟加载（默认为开启状态）添加字段 lazy="true"
        会只加载我们需要的数据而不加载级联数据 -->
        <!-- 若未开启延迟加载，添加字段lazy="false" 在执行SQL时
        会默认进行级联查询，获取所有相关数据 -->
        <!-- 当添加的字段为 lazy="extra" 时，会使用一种比延迟加
        载更懒（智能）的加载方式 -->
        <set name="orders" table="orders" lazy="extra">
            <key column="c_id"></key>
            <one-to-many class="Orders"></one-to-many>
        </set>

    </class>
</hibernate-mapping>
```

- schema: 数据库 schema 的名称
- catalog: 数据库 catalog 的名称
- default-cascade: 默认的级联关系, 默认为none
- default-access: Hibernate用来访问属性的策略
- default-lazy: 指定未明确标注的lazy属性的Java属性和集合类, Hibernate会采用什么样的加载风格, 默认为true。
- auto-import: 指定我们是否可以在查询语句中使用非全限定类名, 默认为true。**如果项目有两个同名的持久化类, 最好在这两个类的对应映射文件中都设置为false**



## class属性

- name: 实体类名
- table: 数据表明
- schema: 数据库 schema 的名称, 会覆盖hibernate-mapping的 schema
- catalog: 数据库 catalog 的名称, 会覆盖hibernate-mapping的 catalog
- proxy: 指定一个接口, 在延迟加载时作为代理使用
- dynamic-insert: 动态添加

测试方法:

```
package com.gloryh.test;

import com.gloryh.entity.People;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

/**
 * 测试类
 *
 * @author 黄光辉
 * @since 2020/8/21
 */
public class Test {
    public static void main(String[] args) {
        //创建Configuration
        Configuration configuration = new
Configuration().configure();
        //获取SessionFactory
        SessionFactory
sessionFactory=configuration.buildSessionFactory();
        //获取Session
        Session session
=sessionFactory.openSession();
        People people = new People();
        people.setName("李四");
    }
}
```

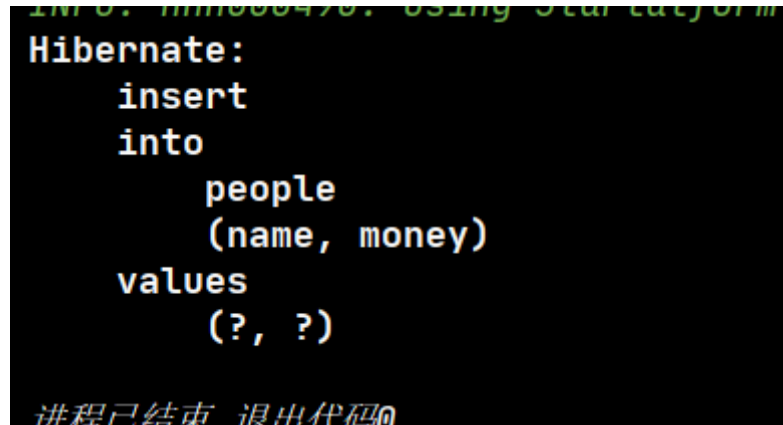
```

        session.save(people);
        session.beginTransaction().commit();
        session.close();

    }
}

```

使用前控制台信息：



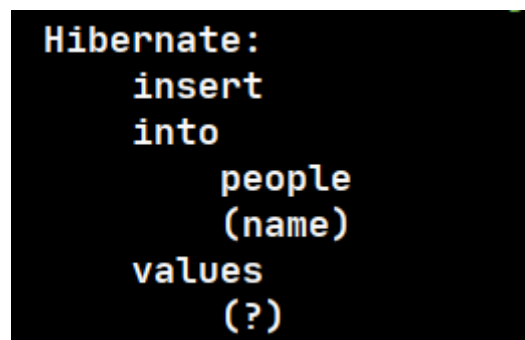
```

INFO: 1111000470: Using StandardJdbc
Hibernate:
insert
into
    people
    (name, money)
values
    (?, ?)

进程已结束 退出代码0

```

使用后控制台信息：



```

Hibernate:
insert
into
    people
    (name)
values
    (?)

```

优点：避免多余的数据库操作，提高效率

使用方法：

```

<class name="com.gloryh.entity.People"
table="people" dynamic-insert="true">

```

- dynamic-update：动态更新，与动态添加类似，使用方法也类似
- where：查询时给SQL添加where条件

使用方法：

```

<class name="com.gloryh.entity.People"
table="people" dynamic-insert="true" where="id =
1">

```

测试方法:

```
package com.gloryh.test;

import com.gloryh.entity.People;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

import java.util.List;

/**
 * @author 黄光辉
 * @since 2020/8/24
 */
public class Test1 {
    public static void main(String[] args) {
        // 创建Configuration
        Configuration configuration = new
Configuration().configure();
        // 获取SessionFactory
        SessionFactory sessionFactory =
configuration.buildSessionFactory();
        // 获取Session
        Session session = sessionFactory.openSession();
        String hql = "from People";
        Query query = session.createQuery(hql);
        List<People> list = query.list();
        for (People people :list) {
            System.out.println(people);
        }
        session.beginTransaction().commit();
        session.close();
    }
}
```

使用前运行结果:

```
INFO: HHH000490: Using JtaPlatform implementation:
Hibernate:
    select
        people0_.id as id1_5_,
        people0_.name as name2_5_,
        people0_.money as money3_5_
    from
        people people0_
People(id=1, name=张三, money=1000.0)
People(id=2, name=李四, money=null)
People(id=3, name=李四, money=null)
```

使用后运行结果：

```
Hibernate:
    select
        people0_.id as id1_5_,
        people0_.name as name2_5_,
        people0_.money as money3_5_
    from
        people people0_
    where
        (
            people0_.id = 1
        )
People(id=1, name=张三, money=1000.0)
```

## id属性

- name: 实体类属性名
- type: 实体类属性数据类型

此处可以设置两种数据类型的数据：Java数据类型或者Hibernate映射类型。

实体类的属性数据类型必须与数据库数据表相对应的字段数据类型一致（int 对应 int，String 对应 varchar）。

Q：如何进行映射？

A：Java数据类型映射到Hibernate映射类型，再由Hibernate映射类型映射到SQL数据类型，即Java ---> Hibernate ---> SQL

- column: 数据表的主键字段名

- generator: 主键生成策略
  - 1.hilo算法
  - 2.increment: Hibernate自增
  - 3.identity: 数据库自增
  - 4.native: 本地策略, 根据底层数据库自动选择主键的生成策略
  - 5.uuid.hex算法
  - 6.select算法

## property属性

- name: 实体类的属性名
- column: 数据表字段名
- type: 数据类型
- update: 该字段是否可以修改, 默认为true
- insert: 该字段是否可以添加, 默认为true
- lazy: 延迟加载策略

## 实体类关系映射文件属性

- inverse: 设置级联表值之间的维护关系, 防止重复操作导致资源浪费, 效率变低。

测试方法:

```
package com.gloryh.test;

import com.gloryh.entity.Customer;
import com.gloryh.entity.Orders;
import com.gloryh.entity.People;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

import java.util.HashSet;
import java.util.List;
import java.util.Set;
```

```
/**
 *
 * @author 黄光辉
 * @since 2020/8/24
 */
public class Test2 {
    public static void main(String[] args) {
        // 创建Configuration
        Configuration configuration = new
Configuration().configure();
        // 获取SessionFactory
        SessionFactory sessionFactory =
configuration.buildSessionFactory();
        // 获取Session
        Session session =
sessionFactory.openSession();
        Customer customer = new Customer();
        customer.setName("王五");

        Orders orders1=new Orders();
        orders1.setName("订单1");
        orders1.setCustomer(customer);

        Orders orders2=new Orders();
        orders2.setCustomer(customer);
        orders2.setName("订单2");

        Set<Orders> orders =new HashSet<>();
        orders.add(orders1);
        orders.add(orders2);
        customer.setOrders(orders);

        session.save(customer);
        session.save(orders1);
        session.save(orders2);
        session.beginTransaction().commit();
        session.close();
    }
}
```

进行级联表之间的维护关系限制前运行结果：

```
INFO: HHH0000490: Using JtaPlatform implementation
Hibernate:
    insert
    into
        customer
        (name)
    values
        (?)
Hibernate:
    insert
    into
        orders
        (name, c_id)
    values
        (?, ?)
Hibernate:
    insert
    into
        orders
        (name, c_id)
    values
        (?, ?)
Hibernate:
    update
        orders
    set
        c_id=?
    where
        id=?
Hibernate:
    update
        orders
    set
        c_id=?
    where
        id=?
```

进行级联表之间的维护关系限制后运行结果：

```

Hibernate:
insert
into
    customer
    (name)
values
    (?)
Hibernate:
insert
into
    orders
    (name, c_id)
values
    (?, ?)
Hibernate:
insert
into
    orders
    (name, c_id)
values
    (?, ?)

```

原因：Customer和Orders都在维护一对多关系，所以会重复设置主外键约束关系。

解决办法：

- 1.在Java代码中去掉其中一方的维护关系代码
- 2.通过配置解决，如：

```

<set name="orders" table="orders"
lazy="extra" inverse="true">
    <key column="c_id"></key>
    <one-to-many class="Orders"></one-to-
many>
</set>

```

inverse属性是用来设置是否将维护权交给对方，默认为false，不交出维护权。由于双方都在维护，就会造成重复操作，造成资源浪费。将其中乙方的属性设置为true，即可解决。



## cascade: 用来设置级联操作

使用该属性前进行主表的删除需要使用迭代器首先获取然后删除相关联表内的对应信息才能执行删除操作，代码如下：

```
package com.gloryh.test;

import com.gloryh.entity.Customer;
import com.gloryh.entity.Orders;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import java.util.Iterator;

/**
 *
 * @author 黄光辉
 * @since 2020/8/24
 */
public class Test3 {
    public static void main(String[] args) {
        // 创建Configuration
        Configuration configuration = new
Configuration().configure();
        // 获取SessionFactory
        SessionFactory sessionFactory =
configuration.buildSessionFactory();
        // 获取Session
        Session session =
sessionFactory.openSession();
        Customer customer =
session.get(Customer.class,8);

        Iterator<Orders> iterator
=customer.getOrders().iterator();
        while (iterator.hasNext()){
            session.delete(iterator.next());
        }
    }
}
```

```

        session.delete(customer);

        session.beginTransaction().commit();

        session.close();
    }
}

```

在实体关系映射文件中设置cascade值就可以不使用迭代器完成级联删除：

```

<set name="orders" table="orders"
lazy="extra" inverse="true"
cascade="delete">
    <key column="c_id"></key>
    <one-to-many class="Orders"></one-to-
many>
</set>

```

代码如下：

```

package com.gloryh.test;

import com.gloryh.entity.Customer;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

/**
 *
 * @author 黄光辉
 * @since 2020/8/24
 */
public class Test3 {
    public static void main(String[] args) {
        // 创建Configuration
        Configuration configuration = new
Configuration().configure();
        // 获取SessionFactory
        SessionFactory sessionFactory =
configuration.buildSessionFactory();
    }
}

```

```
// 获取Session
Session session =
SessionFactory.openSession();
Customer customer =
session.get(Customer.class,9);

session.delete(customer);

session.beginTransaction().commit();

session.close();
}
}
```

## 六、Hibernate HQL 语句

HQL (Hibernate Query Language)：这是Hibernate框架提供的一种查询机制，它和SQL类似，不同的是HQL是面向对象的查询语句，能够让开发者以面向对象的思想来编写查询语句，对Java编程是一种很友好的方式。

HQL不能直接参与数据库的交互，是一种中间层语言。Java调用HQL，HQL反馈Hibernate，Hibernate将HQL解析成SQL语句，然后对DB进行操作。

HQL只能完成查询、修改和删除，无法进行新增操作。

### 查询相关操作

#### 1.查询对象

查询表中所有数据，自动完成对象的封装，并返回list集合。

HQL进行查询，from关键字后面不能写表名，而要写表对应的实体类名。

测试方法：

```
// 创建Configuration
Configuration configuration = new
Configuration().configure();
```

```

System.out.println(configuration);
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取Session
Session session = sessionFactory.openSession();
//定义HQL语句
String hql = "from People";
Query query =session.createQuery(hql);
//遍历查询结果
List<People> list =query.list();
for (People people :list ) {
System.out.println(people);
}
session.close();

```

查询结果：

```

INFO: HHH0000490: Using JtaPlatform implementation
Hibernate:
    select
        people0_.id as id1_5_,
        people0_.name as name2_5_,
        people0_.money as money3_5_
    from
        people people0_
People(id=1, name=张三, money=1000.0)
People(id=2, name=李四, money=null)
People(id=3, name=李四, money=null)

Process finished with exit code 0

```

## 2.分页查询

HQL分页查询可以通过调用query的方法来完成。

- setFirstResult() 设置起始下标
- setMaxResults() 设置截取长度

测试方法：

```

// 创建Configuration

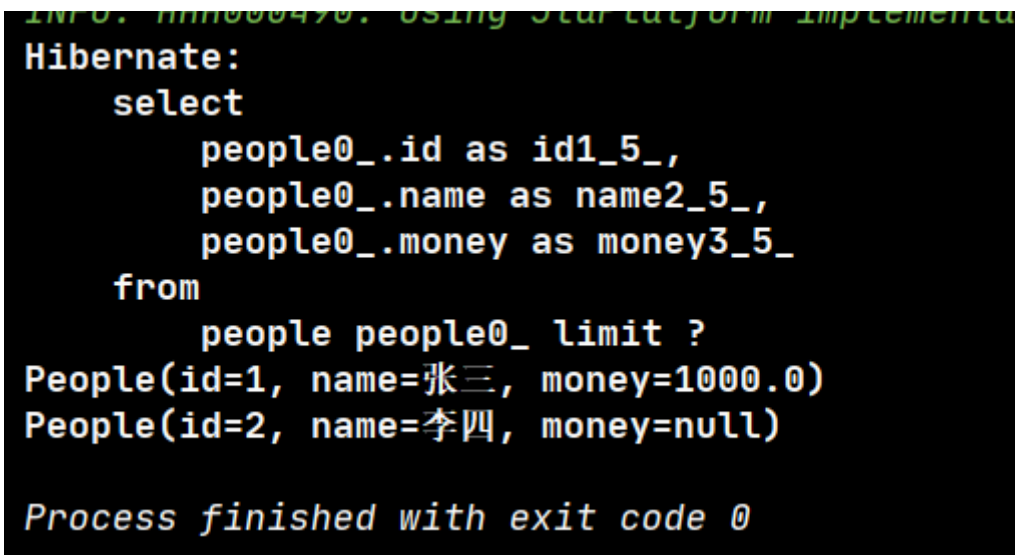
```

```

Configuration configuration = new
Configuration().configure();
System.out.println(configuration);
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取Session
Session session = sessionFactory.openSession();
//定义HQL语句
String hql = "from People";
Query query =session.createQuery(hql);
query.setFirstResult(0);
query.setMaxResults(2);
//遍历查询结果
List<People> list =query.list();
for (People people :list ) {
System.out.println(people);
}
session.close();

```

查询结果：



```

INFO: main:00470: Using standard JPA implementation
Hibernate:
    select
        people0_.id as id1_5_,
        people0_.name as name2_5_,
        people0_.money as money3_5_
    from
        people people0_ limit ?
People(id=1, name=张三, money=1000.0)
People(id=2, name=李四, money=null)

Process finished with exit code 0

```

### 3.where 条件查询

HQL可以直接追加where关键字作为查询条件，与SQL用法一致。

测试方法：

```

// 创建Configuration

```

```

Configuration configuration = new
Configuration().configure();
System.out.println(configuration);
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取Session
Session session = sessionFactory.openSession();
//定义HQL语句
String hql = "from People where id = 1";
Query query =session.createQuery(hql);
//打印查询结果
People people =(People) query.list().get(0);

System.out.println(people);

session.close();

```

查询结果：

```

Hibernate:
  select
    people0_.id as id1_5_,
    people0_.name as name2_5_,
    people0_.money as money3_5_
  from
    people people0_
  where
    people0_.id=1
People(id=1, name=张三, money=1000.0)

Process finished with exit code 0

```

注：此时query.list() 集合内只有一个People对象可以通过 query.list().get(0) 直接获取People对象。但是如果查询结果为空，则无法获取到下标为0的对象，会抛出异常。如 (id = 0)：

```

Hibernate:
select
  people0_.id as id1_5_,
  people0_.name as name2_5_,
  people0_.money as money3_5_
from
  people people0_
where
  people0_.id=0
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
    at java.util.ArrayList.rangeCheck(ArrayList.java:657)
    at java.util.ArrayList.get(ArrayList.java:433)
    at com.gloryh.test.HQLTest.select(HQLTest.java:34)
    at com.gloryh.test.HQLTest.main(HQLTest.java:19)

```

可以通过query的另一个方法uniqueResult() 来替换。如下：

```

//定义HQL语句
String hql = "from People where id = 0";
Query query = session.createQuery(hql);
//打印查询结果
People people = (People) query.uniqueResult();
System.out.println(people);

```

此时若查询结果为空，不会抛出异常，而会返回null。

```

Hibernate:
select
  people0_.id as id1_5_,
  people0_.name as name2_5_,
  people0_.money as money3_5_
from
  people people0_
where
  people0_.id=0
null

```

## 4.模糊查询

HQL可以使用 like 进行模糊查询，与SQL用法一致。

测试方法：

```

// 创建Configuration
Configuration configuration = new
Configuration().configure();
System.out.println(configuration);
// 获取SessionFactory

```

```

SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取Session
Session session = sessionFactory.openSession();
//定义HQL语句
String hql = "from People where name like '%三%'";
Query query =session.createQuery(hql);
//遍历查询结果
List<People> list =query.list();
for (People people :list ) {
System.out.println(people);
}
session.close();

```

查询结果：

```

Hibernate:
  select
    people0_.id as id1_5_,
    people0_.name as name2_5_,
    people0_.money as money3_5_
  from
    people people0_
  where
    people0_.name like '%三%'
People(id=1, name=张三, money=1000.0)

```

## 5.查询结果排序

HQL可以直接追加order by关键字实现排序，与SQL用法一致。

测试方法：

```

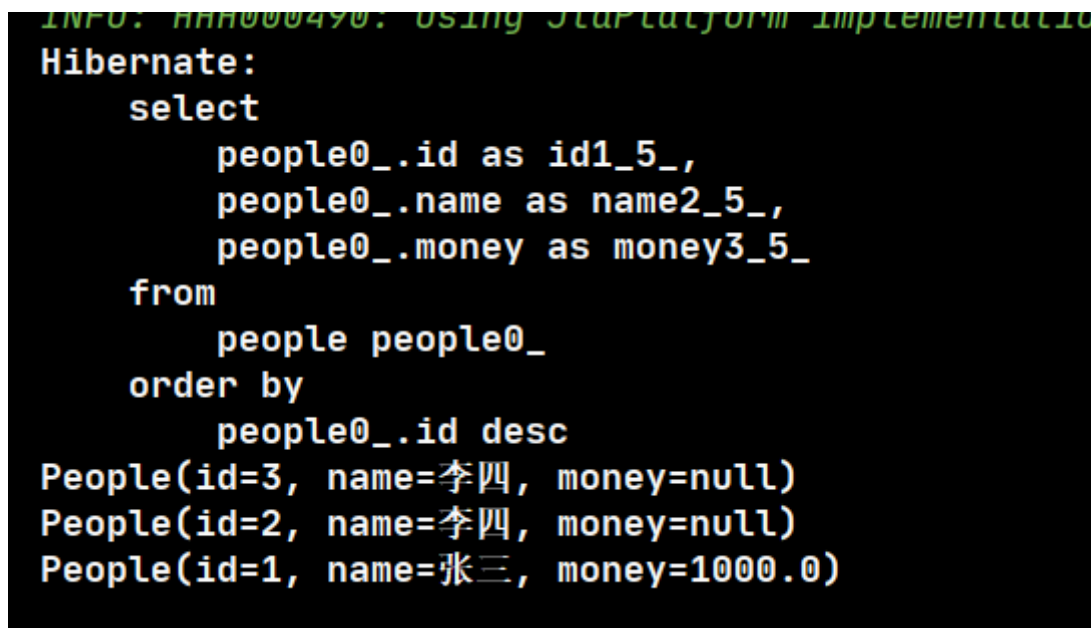
// 创建Configuration
Configuration configuration = new
Configuration().configure();
System.out.println(configuration);
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取Session
Session session = sessionFactory.openSession();

```



```
//定义HQL语句
String hql = "from People order by id desc ";
Query query =session.createQuery(hql);
//遍历查询结果
List<People> list =query.list();
for (People people :list ) {
    System.out.println(people);
}
session.close();
```

查询结果:



```
INFO: HH0000490: Using StandardJdbcImplementation
Hibernate:
    select
        people0_.id as id1_5_,
        people0_.name as name2_5_,
        people0_.money as money3_5_
    from
        people people0_
    order by
        people0_.id desc
People(id=3, name=李四, money=null)
People(id=2, name=李四, money=null)
People(id=1, name=张三, money=1000.0)
```

注: desc 为降序排列, asc (或不写) 为升序排列。

## 6.查询实体对象属性

查询具体某个属性课直接使用select字段, 与SQL用法一致。

测试方法:

```
// 创建Configuration
Configuration configuration = new
Configuration().configure();
System.out.println(configuration);
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取Session
Session session = sessionFactory.openSession();
```

```
//定义HQL语句
String hql = "select name from People order by id desc ";
Query query =session.createQuery(hql);
//遍历查询结果
List<String > list =query.list();
for (String name :list ) {
    System.out.println(name);
}
session.close();
```

查询结果:

```
Hibernate:
  select
    people0_.name as col_0_0_
  from
    people people0_
  order by
    people0_.id desc
李四
李四
张三
```

## 7.占位符的使用

HQL中使用占位符要在字段前加:(冒号), 例如 where name = :name,然后通过query.setString()方法【实体类占位符要用 query.setEntity()方法】, 匹配占位符代表的内容。

测试方法:

```
// 创建Configuration
Configuration configuration = new
Configuration().configure();
System.out.println(configuration);
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取Session
Session session = sessionFactory.openSession();
//定义HQL语句
String hql = "from People where name = :name";
```

```

Query query =session.createQuery(hql);
query.setString("name","李四");
//遍历查询结果
List<People > list =query.list();
for (People people :list ) {
System.out.println(people);
}
session.close();

```

查询结果:

```

Hibernate:
  select
    people0_.id as id1_5_,
    people0_.name as name2_5_,
    people0_.money as money3_5_
  from
    people people0_
  where
    people0_.name=?
People(id=2, name=李四, money=null)
People(id=3, name=李四, money=null)

```

## 8.级联查询

级联查询可以通过已知对象查询级联对象，可以采用占位符方式创建级联对象查询的HQL语句。

测试方法:

```

// 创建Configuration
Configuration configuration = new
Configuration().configure();
System.out.println(configuration);
// 获取SessionFactory
SessionFactory sessionFactory =
configuration.buildSessionFactory();
// 获取Session
Session session = sessionFactory.openSession();
// 定义HQL语句
String hql1 = "from Customer where name = :name";
Query query1 = session.createQuery(hql1);

```

```

query1.setString("name", "张三");
Customer customer = (Customer) query1.uniqueResult();
String hql2 = "from Orders where customer = :customer";
Query query2 = session.createQuery(hql2);
query2.setEntity("customer", customer);
List<Orders> list = query2.list();
for (Orders order : list) {
    System.out.println(order);
}
session.close();

```

查询结果：

```

Hibernate:
    select
        customer0_.id as id1_3_,
        customer0_.name as name2_3_
    from
        customer customer0_
    where
        customer0_.name=?
Hibernate:
    select
        orders0_.id as id1_4_,
        orders0_.name as name2_4_,
        orders0_.c_id as c_id3_4_
    from
        orders orders0_
    where
        orders0_.c_id=?
Orders{id=8, name='订单1'}

Process finished with exit code 0

```

## 删除和更新操作

删除和更新操作类似与查询操作，通常可以先查询，然后通过session进行更新，或者直接使用delete关键字或者update关键字进行操作。