Hibernate 入门第三讲——Hibernate 的常见配置

在《Hibernate 入门第一讲——Hibernate 框架的快速入门》一讲中，我有讲到 Hibernate 的两个配置文件，今天就来详细地介绍这两个配置文件。在 Hibernate 中，我们主要使用两种配置文件：

- 核心配置文件——hibernate.cfg.xml(主要描述 Hibernate 的相关配置)；
- 映射配置文件——xxx.hbm.xml。

映射配置文件

映射配置文件的名称是`类名.hbm.xml`，它一般放置在实体类所在的包下。这个配置文件的主要作用是建立表与类之间的映射关系。下面我来粗略地介绍一下该映射配置文件，当然你可以在以后的 Hibernate 学习中逐渐地补全一些细枝末节。

- 如果统一声明包名，那么在`<class>`标签的 name 属性的值中就不需要写类的全名了；

```
<hibernate-mapping package="com.meimeixia.hibernate.demo01">
    <!-- 建立类与表的映射 -->
    <class name="Customer" table="cst_customer" catalog="hibernate_demo01">

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

- 关于`<class>`标签配置的详细介绍：
  - 该标签用来建立类与表的映射关系。
  - 该标签中有如下这些属性：
    - name 属性：类的全路径

- table 属性：映射到数据库里面的那个表的名称，如果表的名称与类名一致，那么 table 属性可以省略
- catalog 属性：数据库名称，可以省略，如果省略，则参考核心配置文件中 url 路径中的库名称

- 关于<id>标签配置的详细介绍：

```
<hibernate-mapping>
    <!-- 建立类与表的映射 -->
    <class name="com.meimeixia.hibernate.demo01.Customer" table="cst_customer">
        <!-- 建立类中的属性与表中的主键相对应 -->
        <id name="cust_id" column="cust_id">
            <!-- 主键的生成策略，后面会讲，现在使用的是本地生成策略 -->
            <generator class="native" />
        </id>

        <!-- 建立类中的普通属性和表中的字段相对应 -->
        <property name="cust_name" column="cust_name" />
        <property name="cust_source" column="cust_source" />
        <property name="cust_industry" column="cust_industry" />
        <property name="cust_level" column="cust_level" />
        <property name="cust_phone" column="cust_phone" />
        <property name="cust_mobile" column="cust_mobile" />
    </class>
</hibernate-mapping>
```


首先该标签必须存在，该标签用来建立类中的 id 属性与表中的主键的对应关系。该标签中有如下这些属性：

- name：类中的属性名称
- column：表中的主键名称，类中的属性名和表中的字段名（主键名）如果一致，column 可以省略
- length：字段长度，如果 length 忽略不写，且你的表是自动创建这种方案，那么 length 的默认长度是 255（可以根据你的映射文件自动建表，如果现在数据库里面是没有表的，那么只要一运行咱们的程序，它就可以

帮你把表建起来。如果你没有给定长度，那么它便会使用默认长度，像字符串的长度默认就是 255）

- type：指定类型，你可以不用写，Hibernate 会帮你自动转换

该标签中的`<generator>`子标签主要是描述主键生成策略的，这里就不做篇幅来介绍了，请看后面的文章。

- 关于`<property>`标签
  - 该标签用来建立类中的普通属性与表中非主键字段的对应关系。
  - 该标签中有如下这些属性：
    - name：类中的属性名
    - column：表中的字段名
    - length：长度
    - type：类型
    - not-null：设置是否非空
    - unique：设置唯一

关于 Hibernate 映射配置文件中的类型问题

对于 type 属性它的取值可以有三种：

1. Java 中的数据类型；
2. Hibernate 中的数据类型；
3. SQL 的数据类型。

可参考下表：

java类型	Hibernate映射类型	SQL类型
java.math.BigDecimal	big_decimal	numeric
byte[]	binary	varbinary(blob)
boolean(java.lang.Boolean)	boolean	bit
byte(java.lang.Byte)	byte	tinyint
java.util.Calendar	calendar	timestamp
java.sql.Clob	clob	clob
java.util.Date 或java.sql.Date	date	date
double(java.lang.Double)	double	double
float(java.lang.Float)	float	float
int (java.lang.Integer)	integer	integer
java.util.Local	local	varchar
long(java.lang.Long)	long	bigint
java.io.Serializable的某个实例	serializable	varbinary(或blob)
java.lang.String	string	varchar
java.lang.String	text	clob
java.util.Date 或 java.sql.Timestamp	time	timestamp

这样看来，我实体类(Customer.java)的映射配置文件可以写为：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!-- 建立类与表的映射 -->
    <class name="com.meimeixia.hibernate.demo01.Customer" table="cst_customer">
        <!-- 建立类中的属性与表中的主键相对应 -->
        <id name="cust_id" column="cust_id">
            <!-- 主键的生成策略，后面会讲，现在使用的是本地生成策略 -->
            <generator class="native" />
        </id>

        <!-- 建立类中的普通属性和表中的字段相对应 -->
        <property name="cust_name" column="cust_name" length="32" type="string" /><!-- Hibernate数据类型 -->

        <property name="cust_source" column="cust_source" />
        <property name="cust_industry" column="cust_industry" />
        <property name="cust_level" column="cust_level" />
        <property name="cust_phone" column="cust_phone" />
        <property name="cust_mobile" column="cust_mobile" />
    </class>
</hibernate-mapping>
```

或者

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!-- 建立类与表的映射 -->
    <class name="com.meimeixia.hibernate.demo01.Customer" table="cst_customer">
        <!-- 建立类中的属性与表中的主键相对应 -->
        <id name="cust_id" column="cust_id">
            <!-- 主键的生成策略，后面会讲，现在使用的是本地生成策略 -->
            <generator class="native" />
        </id>

        <!-- 建立类中的普通属性和表中的字段相对应 -->
        <property name="cust_name" length="32">
            <column name="cust_name" sql-type="varchar"></column> <!-- SQL的数据类型 -->
        </property>

        <property name="cust_source" column="cust_source" />
        <property name="cust_industry" column="cust_industry" />
        <property name="cust_level" column="cust_level" />
        <property name="cust_phone" column="cust_phone" />
        <property name="cust_mobile" column="cust_mobile" />
    </class>
</hibernate-mapping>
```


核心配置文件

Hibernate 的核心配置文件，即 hibernate.cfg.xml，主要用来描述 Hibernate 的相关配置。对于 Hibernate 的核心配置文件它有两种方式：

- 第一种方式：属性文件，即 hibernate.properties，其内容应该是这样子的：

```
hibernate.connection.driver_class=com.mysql.jdbc.Driver
...
hibernate.show_sql=true
```

温馨提示：这种属性文件的方式不能引入映射文件，须手动编写代码加载映射文件。

- 第二种方式：XML 文件，即 hibernate.cfg.xml。

我们在开发中使用比较多的是 hibernate.cfg.xml 这种方式，原因是它的配置能力更强，并且易于修改。所以我主要讲解的是 hibernate.cfg.xml 这种配置方式。我就以《Hibernate 入门第一讲——Hibernate 框架的快速入门》一文案例中的 hibernate.cfg.xml 核心配置文件为例进行讲解。

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

<hibernate-configuration>
        <session-factory>
                <!-- 下面是三个必须要有的配置 -->
                <!-- 配置连接 MySQL 数据库的基本参数 -->
                <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
                <property
name="hibernate.connection.url">jdbc:mysql:///hibernate_demo01</property>
                <property name="hibernate.connection.username">root</property>
                <property name="hibernate.connection.password">liayun</property>

                <!-- 配置 Hibernate 的方言 -->
                <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

                <!-- 下面两个是可选的配置哟！ -->
                <!-- 打印 sql 语句 -->
                <property name="hibernate.show_sql">true</property>
                <!-- 格式化 sql 语句 -->
                <property name="hibernate.format_sql">true</property>

                <!-- 告诉 Hibernate 的核心配置文件加载哪个映射文件 -->
                <mapping resource="com/meimeixia/hibernate/demo01/Customer.hbm.xml"/>
```

```
        </session-factory>
</hibernate-configuration>
```

可将以上配置文件的内容分为 3 部分来看待：

- 加载数据库相关信息

```
<!-- 配置连接MySQL数据库的基本参数 -->
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql:///hibernate_demo01</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">liayun</property>
```

- Hibernate 的相关配置

```
<!-- 配置Hibernate的方言 -->
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<!-- 下面两个是可选的配置哟！ -->
<!-- 打印sql语句 -->
<property name="hibernate.show_sql">true</property>
<!-- 格式化sql语句 -->
<property name="hibernate.format_sql">true</property>
```


- 加载映射配置文件

```
<!-- 告诉Hibernate的核心配置文件加载哪个映射文件 -->
<mapping resource="com/meimeixia/hibernate/demo01/Customer.hbm.xml"/>
```

温馨提示：对于 hibernate.cfg.xml 配置文件中的要配置的内容可以参考 project/etc/hibernate.properties 文件中的配置。如果你查阅 hibernate.properties 文件，便可发现有如下内容：

```
#hibernate.hbm2ddl.auto create-drop
#hibernate.hbm2ddl.auto create
#hibernate.hbm2ddl.auto update
#hibernate.hbm2ddl.auto validate
```

那么 hibernate.hbm2ddl.auto 这个玩意到底是个什么东东呢？这儿我就来详解讲讲，先说结论：配置这个玩意之后，我们就可以进行表的自动创建。这个玩意有如下 5 个取值：

- none：不使用 Hibernate 帮我们自动建表；

- create：如果数据库中已经有了表，则删除原有表，重新创建；如果没有表，则新建表。即每次都会创建一个新的表，但不删除，一般在测试中使用。下面我来举例说明该属性值，要知道我也是在《Hibernate 入门第一讲——Hibernate 框架的快速入门》一文案例的基础上来讲解的。

  首先在 hibernate.cfg.xml 配置文件中加入如下内容：

```
<!-- 自动创建表 -->
<property name="hibernate.hbm2ddl.auto">create</property>
```

```
<!-- 下面三个是可选的配置哟！ -->
<!-- 打印sql语句 -->
<property name="hibernate.show_sql">true</property>
<!-- 格式化sql语句 -->
<property name="hibernate.format_sql">true</property>
<!-- 自动创建表 -->
<property name="hibernate.hbm2ddl.auto">create</property>
```

然后执行单元测试类——HibernateDemo1.java 中的 demo1()方法：

```
package com.meimeixia.hibernate.demo01;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.junit.Test;

/**
 * Hibernate 的入门案例
 * @author liayun
 *
 */
public class HibernateDemo1 {

    //保存用户的案例
    @Test
```

```
    public void demo1() {
            //1. 加载 Hibernate 的核心配置文件
            Configuration configuration = new Configuration().configure();
            //如果在 Hibernate 的核心配置文件没有设置加载哪个映射文件，则可手动加载映射
文件

    //configuration.addResource("com/meimeixia/hibernate/demo01/Customer.hbm.xml");

            //2. 创建 SessionFactory 对象，类似于 JDBC 中的连接池
            SessionFactory sessionFactory = configuration.buildSessionFactory();

            //3. 通过 SessionFactory 获取到 Session 对象，类似于 JDBC 中的 Connection
            Session session = sessionFactory.openSession();

            //4. 手动开启事务，（最好是手动开启事务）
            Transaction transaction = session.beginTransaction();

            //5. 编写代码
            Customer customer = new Customer();
            customer.setCust_name("叶子");

            session.save(customer);//保存一个用户

            //6. 事务提交
            transaction.commit();

            //7. 释放资源
            session.close();
            sessionFactory.close();
    }

}
```

此刻，hibernate_demo01 数据库里面应该是有 cst_customer 表的，那么一旦运行以上测试方法，不出意外，我们就能在 Eclipse 的控制台上看到删表和建表的 sql 语句：

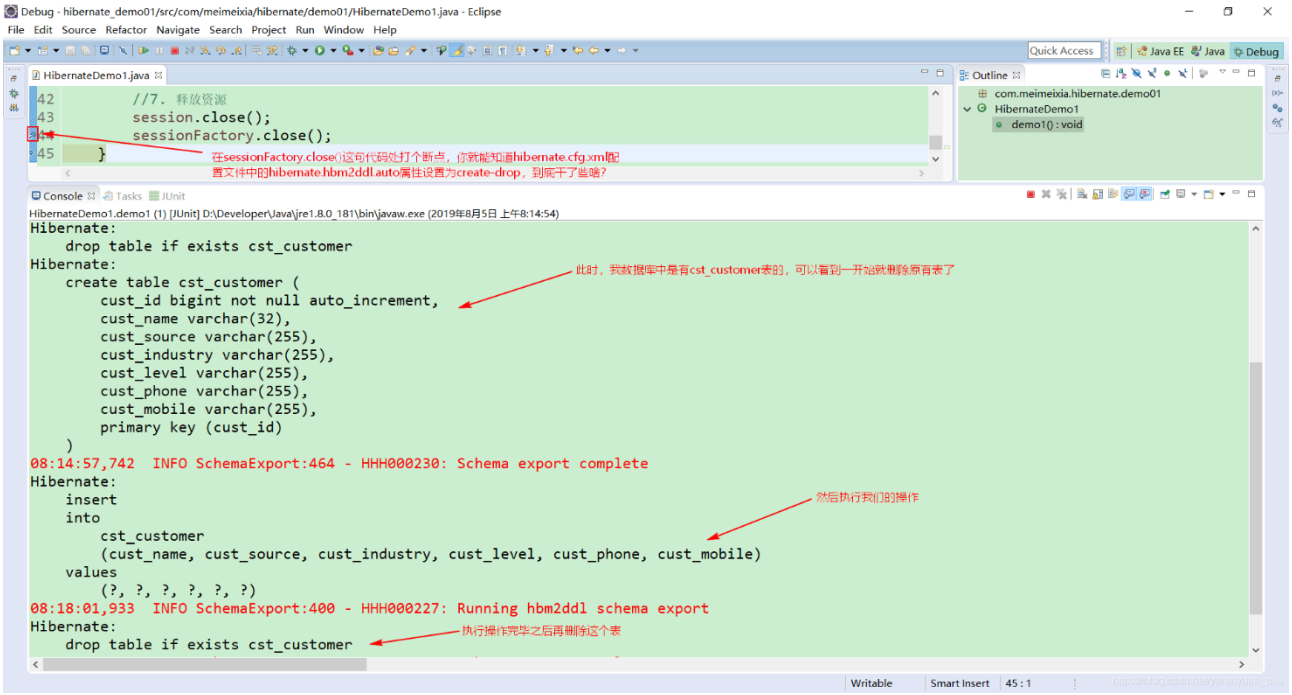
```
Hibernate:
    drop table if exists cst_customer
Hibernate:
    create table cst_customer (
        cust_id bigint not null auto_increment,
        cust_name varchar(32),
        cust_source varchar(255),
        cust_industry varchar(255),
        cust_level varchar(255),
        cust_phone varchar(255),
        cust_mobile varchar(255),
        primary key (cust_id)
    )
17:32:53,335  INFO SchemaExport:464 - HHH000230: Schema export complete
Hibernate:
    insert
    into
        cst_customer
        (cust_name, cust_source, cust_industry, cust_level, cust_phone, cust_mobile)
    values
        (?, ?, ?, ?, ?, ?)
```


如果要是 hibernate_demo01 数据库里面没有 cst_customer 表，那么一旦运行以上测试方法，我们就只能在 Eclipse 的控制台上看到建表的 sql 语句了。

- create-drop

  如果数据库中已经有表，则删除原有表，再新建一个新表，然后执行操作，执行操作完毕之后再删除这个表（妈的，这可真是毛病啊！幸好，你以后不会用到它）；如果没有表，则新建一个，使用完了删除该表，一般也是做测试时用。如果你要对 create-drop 该属性值进行测试，也很简单，可以像下

面这样子做。



- update

  如果数据库中有表，则不创建，使用原有表；如果没有表则创建新表，并且如果映射不匹配，会自动更新表结构。下面我来说道说道该属性值，首先在 hibernate.cfg.xml 配置文件中加入如下内容：

```
<!-- 自动创建表 -->
<property name="hibernate.hbm2ddl.auto">update</property>
```

```
<!-- 下面三个是可选的配置哟！ -->
<!-- 打印sql语句 -->
<property name="hibernate.show_sql">true</property>
<!-- 格式化sql语句 -->
<property name="hibernate.format_sql">true</property>
<!-- 自动创建表 -->
<property name="hibernate.hbm2ddl.auto">update</property>
```

然后在实体类——Customer.java 中增加一个属性，比如说 `private String cust_sex`。

```
package com.meimeixia.hibernate.demo01;

public class Customer {

	private Long cust_id;
	private String cust_name;
	private String cust_source;
	private String cust_industry;
	private String cust_level;
	private String cust_phone;
	private String cust_mobile;
	private String cust_sex;

	public Long getCust_id() {
		return cust_id;
	}
	public void setCust_id(Long cust_id) {
		this.cust_id = cust_id;
	}
	public String getCust_name() {
		return cust_name;
	}
	public void setCust_name(String cust_name) {
		this.cust_name = cust_name;
	}
	public String getCust_source() {
		return cust_source;
	}
	public void setCust_source(String cust_source) {
		this.cust_source = cust_source;
	}
	public String getCust_industry() {
		return cust_industry;
	}
	public void setCust_industry(String cust_industry) {
		this.cust_industry = cust_industry;
	}
	public String getCust_level() {
		return cust_level;
	}
	public void setCust_level(String cust_level) {
		this.cust_level = cust_level;
	}
```

```
	public String getCust_phone() {
		return cust_phone;
	}
	public void setCust_phone(String cust_phone) {
		this.cust_phone = cust_phone;
	}
	public String getCust_mobile() {
		return cust_mobile;
	}
	public void setCust_mobile(String cust_mobile) {
		this.cust_mobile = cust_mobile;
	}
	public String getCust_sex() {
		return cust_sex;
	}
	public void setCust_sex(String cust_sex) {
		this.cust_sex = cust_sex;
	}
	@Override
	public String toString() {
		return "Customer [cust_id=" + cust_id + ", cust_name=" + cust_name + ",
cust_source=" + cust_source
				+ ", cust_industry=" + cust_industry + ", cust_level=" +
cust_level + ", cust_phone=" + cust_phone
				+ ", cust_mobile=" + cust_mobile + "]";
	}

}
```

接着修改实体类的映射配置文件(Customer.hbm.xml)的内容为：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
	<!-- 建立类与表的映射 -->
	<class name="com.meimeixia.hibernate.demo01.Customer" table="cst_customer">
		<!-- 建立类中的属性与表中的主键相对应 -->
		<id name="cust_id" column="cust_id">
			<!-- 主键的生成策略，后面会讲，现在使用的是本地生成策略 -->
			<generator class="native" />
```

```
            </id>

            <!-- 建立类中的普通属性和表中的字段相对应 -->
            <property name="cust_name" column="cust_name" length="32" />
            <property name="cust_source" column="cust_source" />
            <property name="cust_industry" column="cust_industry" />
            <property name="cust_level" column="cust_level" />
            <property name="cust_phone" column="cust_phone" />
            <property name="cust_mobile" column="cust_mobile" />
            <property name="cust_sex" column="cust_sex" />
    </class>
</hibernate-mapping>
```

最后运行单元测试类（HibernateDemo1.java）中的 demo1()方法：

```
package com.meimeixia.hibernate.demo01;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.junit.Test;

/**
 * Hibernate 的入门案例
 * @author liayun
 *
 */
public class HibernateDemo1 {

    //保存用户的案例
    @Test
    public void demo1() {
            //1. 加载 Hibernate 的核心配置文件
            Configuration configuration = new Configuration().configure();
            //如果在 Hibernate 的核心配置文件没有设置加载哪个映射文件，则可手动加载映射文件

    //configuration.addResource("com/meimeixia/hibernate/demo01/Customer.hbm.xml");

            //2. 创建 SessionFactory 对象，类似于 JDBC 中的连接池
            SessionFactory sessionFactory = configuration.buildSessionFactory();
```

```
            //3. 通过 SessionFactory 获取到 Session 对象，类似于 JDBC 中的 Connection
            Session session = sessionFactory.openSession();

            //4. 手动开启事务，（最好是手动开启事务）
            Transaction transaction = session.beginTransaction();

            //5. 编写代码
            Customer customer = new Customer();
            customer.setCust_name("叶美美");

            session.save(customer);//保存一个用户

            //6. 事务提交
            transaction.commit();

            //7. 释放资源
            session.close();
            sessionFactory.close();
        }

}
```

此时，我们只能在 Eclipse 的控制台上看到向 cst_customer 表中插入记录的 sql 语句了。

```
Hibernate:
    insert
    into
        cst_customer
        (cust_name, cust_source, cust_industry, cust_level, cust_phone, cust_mobile, cust_sex)
    values
        (?, ?, ?, ?, ?, ?, ?)
07:22:31,922  INFO connections:235 - HHH10001008: Cleaning up connection pool [jdbc:mysql:///hibernate_demo01]
```

说明hibernate.hbm2ddl.auto的值要是设置为update，如果数据库中有表，那么并不会创建新表，而是使用原有表，而且如果映射不匹配，会自动更新表结构

你这个时候再去查看 cst_customer 表可发现该表多出了一个 cust_sex 字段。

cust_id	cust_name	cust_source	cust_industry	cust_level	cust_phone	cust_mobile	cust_sex
1	叶子	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)
2	叶美美	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)

多出来了一列

这就已说明了如果映射不匹配，会自动更新表结构。但是注意：只能添加，不能说我这个表里面有 3 个字段，我映射 2 个了，然后它就帮我删了，这是不行的！

- validate

  如果没有表，不会创建表，只会使用数据库中原有的表。它的作用主要是校验映射关系和表结构。为了便于测试 validate，先做这样子的准备工作：首先将实体类（Customer.java）中的 cust_sex 属性去掉，然后再在映射配置文件（Customer.hbm.xml）去掉该 cust_sex 属性和表中字段的映射，接着将 hibernate.cfg.xml 配置文件的 hibernate.hbm2ddl.auto 属性设置为 create，最后运行单元测试类（HibernateDemo1.java）中的 demo1()方法，这样数据库中就创建好了一个新的 cst_customer 表，并且已经插入一条记录。

cust_id	cust_name	cust_source	cust_industry	cust_level	cust_phone	cust_mobile
1	叶美美	(Null)	(Null)	(Null)	(Null)	(Null)

  这样，一切都是新的，会更方便我们测试 validate。这里再次在实体类（Customer.java）中增加 cust_sex 属性，

```
package com.meimeixia.hibernate.demo01;

public    class Customer {

        private Long cust_id;
        private String cust_name;
        private String cust_source;
        private String cust_industry;
        private String cust_level;
        private String cust_phone;
        private String cust_mobile;
        private String cust_sex;

        public Long getCust_id() {
                    return cust_id;
        }
        public void setCust_id(Long cust_id) {
                    this.cust_id = cust_id;
        }
```

```
	public String getCust_name() {
		return cust_name;
	}
	public void setCust_name(String cust_name) {
		this.cust_name = cust_name;
	}
	public String getCust_source() {
		return cust_source;
	}
	public void setCust_source(String cust_source) {
		this.cust_source = cust_source;
	}
	public String getCust_industry() {
		return cust_industry;
	}
	public void setCust_industry(String cust_industry) {
		this.cust_industry = cust_industry;
	}
	public String getCust_level() {
		return cust_level;
	}
	public void setCust_level(String cust_level) {
		this.cust_level = cust_level;
	}
	public String getCust_phone() {
		return cust_phone;
	}
	public void setCust_phone(String cust_phone) {
		this.cust_phone = cust_phone;
	}
	public String getCust_mobile() {
		return cust_mobile;
	}
	public void setCust_mobile(String cust_mobile) {
		this.cust_mobile = cust_mobile;
	}
	public String getCust_sex() {
		return cust_sex;
	}
	public void setCust_sex(String cust_sex) {
		this.cust_sex = cust_sex;
	}
	@Override
```

```
    public String toString() {
        return "Customer [cust_id=" + cust_id + ", cust_name=" + cust_name + ", cust_source=" + cust_source
                + ", cust_industry=" + cust_industry + ", cust_level=" + cust_level + ", cust_phone=" + cust_phone
                + ", cust_mobile=" + cust_mobile + "]";
    }

}
```

- 接着在实体类的映射配置文件(Customer.hbm.xml)中加上该 cust_sex 属性和表中字段的映射。

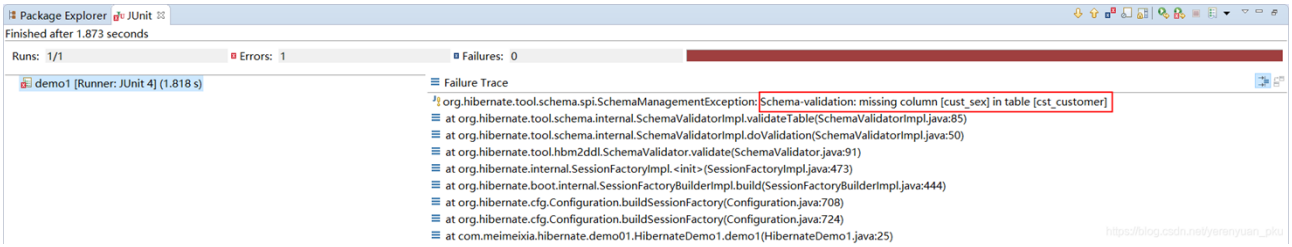
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!-- 建立类与表的映射 -->
    <class name="com.meimeixia.hibernate.demo01.Customer" table="cst_customer">
        <!-- 建立类中的属性与表中的主键相对应 -->
        <id name="cust_id" column="cust_id">
            <!-- 主键的生成策略，后面会讲，现在使用的是本地生成策略 -->
            <generator class="native" />
        </id>

        <!-- 建立类中的普通属性和表中的字段相对应 -->
        <property name="cust_name" column="cust_name" length="32" />
        <property name="cust_source" column="cust_source" />
        <property name="cust_industry" column="cust_industry" />
        <property name="cust_level" column="cust_level" />
        <property name="cust_phone" column="cust_phone" />
        <property name="cust_mobile" column="cust_mobile" />
        <property name="cust_sex" column="cust_sex" />
    </class>
</hibernate-mapping>
```

从上可以看出表结构与映射文件已经不匹配了。下面将 hibernate.cfg.xml 配置文件的 hibernate.hbm2ddl.auto 属性设置为 validate。

```
<!-- 自动创建表 -->
<property name="hibernate.hbm2ddl.auto">validate</property>
```

```
<!-- 下面三个是可选的配置哟！ -->
<!-- 打印sql语句 -->
<property name="hibernate.show_sql">true</property>
<!-- 格式化sql语句 -->
<property name="hibernate.format_sql">true</property>
<!-- 自动创建表 -->
<property name="hibernate.hbm2ddl.auto">validate</property>
```

再次运行单元测试类（HibernateDemo1.java）中的 demo1()方法，这时就能看到报如下异常：



也即说明了如果表结构与映射文件不匹配，会报异常。