Hibernate入门这一篇就够了

前言

本博文主要讲解**介绍Hibernate框架，ORM的概念和Hibernate入门**，相信你们看了就会使用Hibernate了!

什么是Hibernate框架？

Hibernate是一种ORM框架，全称为 Object_Relative DateBase-Mapping，在Java对象与关系数据库之间**建立某种映射，以实现直接存取Java对象**！

为什么要使用Hibernate？

既然Hibernate是关于Java对象和关系数据库之间的联系的话，也就是**我们MVC中的数据持久层->在编写程序中的DAO层...**

首先，我们来回顾一下我们在DAO层写程序的历程吧：

1. **在DAO层操作XML，将数据封装到XML文件上，读写XML文件数据实现CRUD**
2. **在DAO层使用原生JDBC连接数据库，实现CRUD**
3. **嫌弃JDBC的Connection\Statement\ResultSet等对象太繁琐，使用对原生JDBC的封装组件-->DbUtils组件**

我们来看看使用DbUtils之后，程序的代码是怎么样的：

```
public class CategoryDAOImpl implements zhongfucheng.dao.CategoryDao{

    @Override
    publicvoidaddCategory(Category category) {

        QueryRunner queryRunner = new QueryRunner(Utils2DB.getDataSource());

        String sql = "INSERT INTO category (id, name, description) VALUES(?,?,?)";
        try {
            queryRunner.update(sql, new Object[]{category.getId(), category.getName(), category.getDescription()});

        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    public Category findCategory(String id) {
        QueryRunner queryRunner = new QueryRunner(Utils2DB.getDataSource());
        String sql = "SELECT * FROM category WHERE id=?";

        try {
            Category category = (Category) queryRunner.query(sql, id, new BeanHandler(Category.class));

            return category;

        } catch (SQLException e) {
            throw new RuntimeException(e);
        }

    }

    @Override
    public List<Category> getAllCategory() {
        QueryRunner queryRunner = new QueryRunner(Utils2DB.getDataSource());
        String sql = "SELECT * FROM category";

        try {
            List<Category> categories = (List<Category>) queryRunner.query(sql, new BeanListHandler(Category.class));

            return categories;
        } catch (SQLException e) {
            throw  new RuntimeException(e);
        }
```

```
    }
}
```

其实使用DbUtils时，DAO层中的代码编写是很**有规律的。**

- **当插入数据的时候，就将JavaBean对象拆分，拼装成SQL语句**
- **当查询数据的时候，用SQL把数据库表中的列组合，拼装成JavaBean对象**

也就是说：**javaBean对象和数据表中的列存在映射关系!**如果程序能够自动生成SQL语句就好了....那么Hibernate就实现了这个功能！

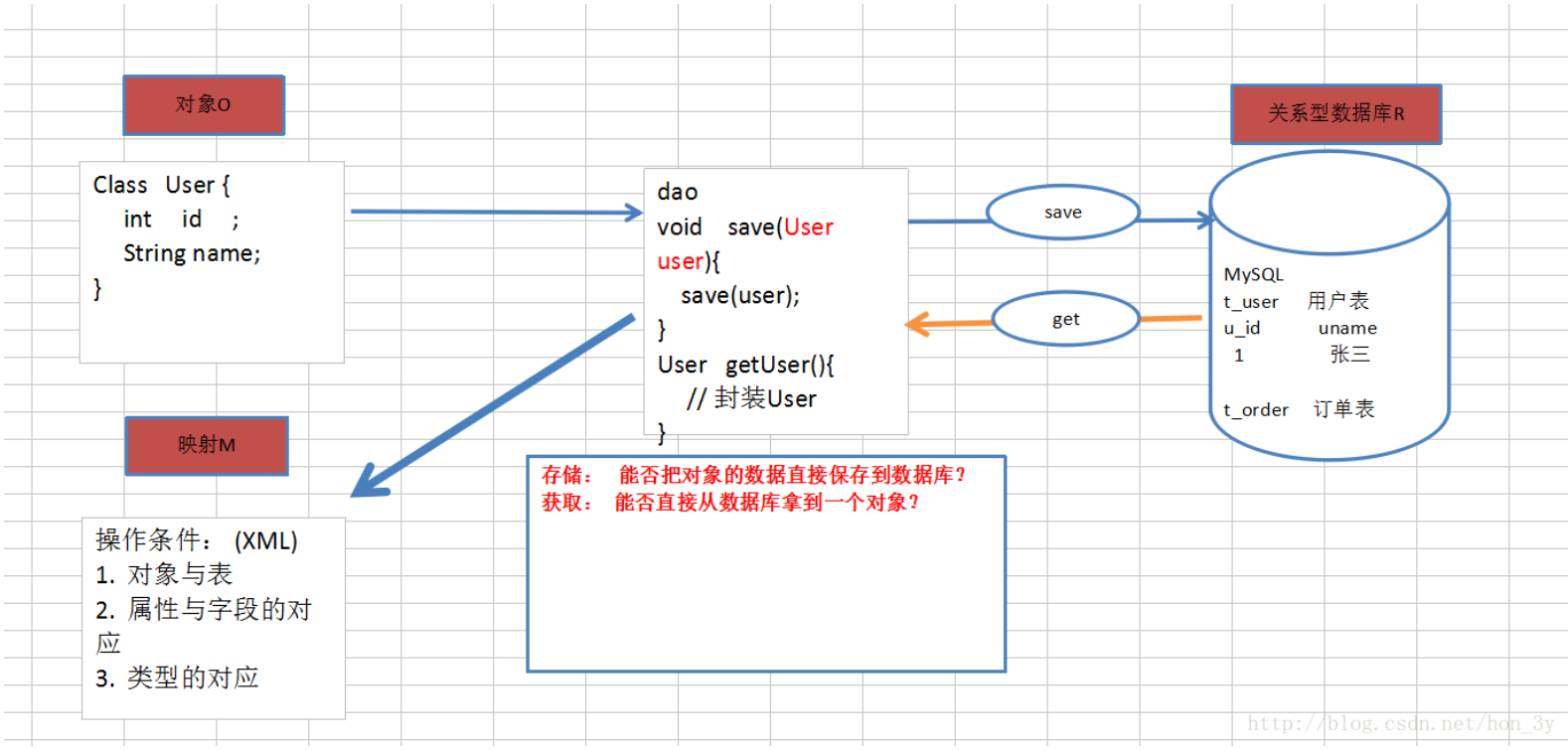
简单来说：**我们使用Hibernate框架就不用我们写很多繁琐的SQL语句，从而简化我们的开发！**

ORM概述

在介绍Hibernate的时候，说了**Hibernate是一种ORM的框架。**那什么是ORM呢？**ORM是一种思想**

- O代表的是Objcet
- R代表的是Relative
- M代表的是Mapping

ORM->对象关系映射....ORM关注是**对象与数据库中的列的关系**



Hibernate快速入门

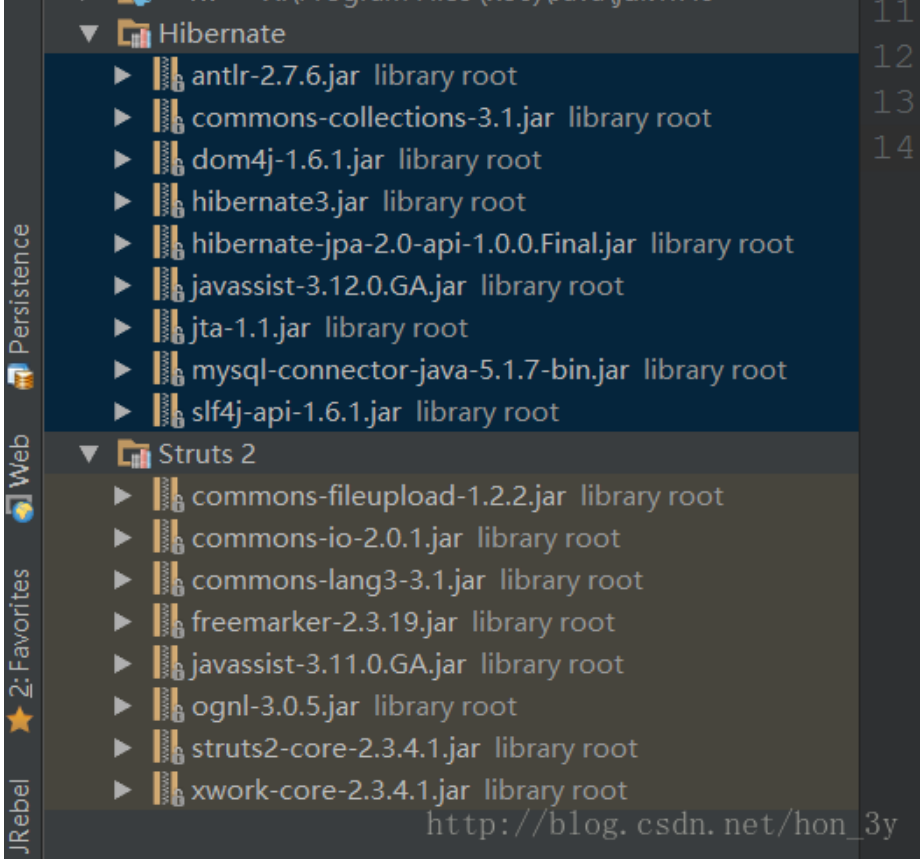
学习一个框架无非就是三个步骤：

- **引入jar开发包**
- **配置相关的XML文件**
- **熟悉API**

引入相关jar包

我们使用的是Hibernate3.6的版本

hibernate3.jar核心 + required 必须引入的(6个) + jpa 目录 + 数据库驱动包



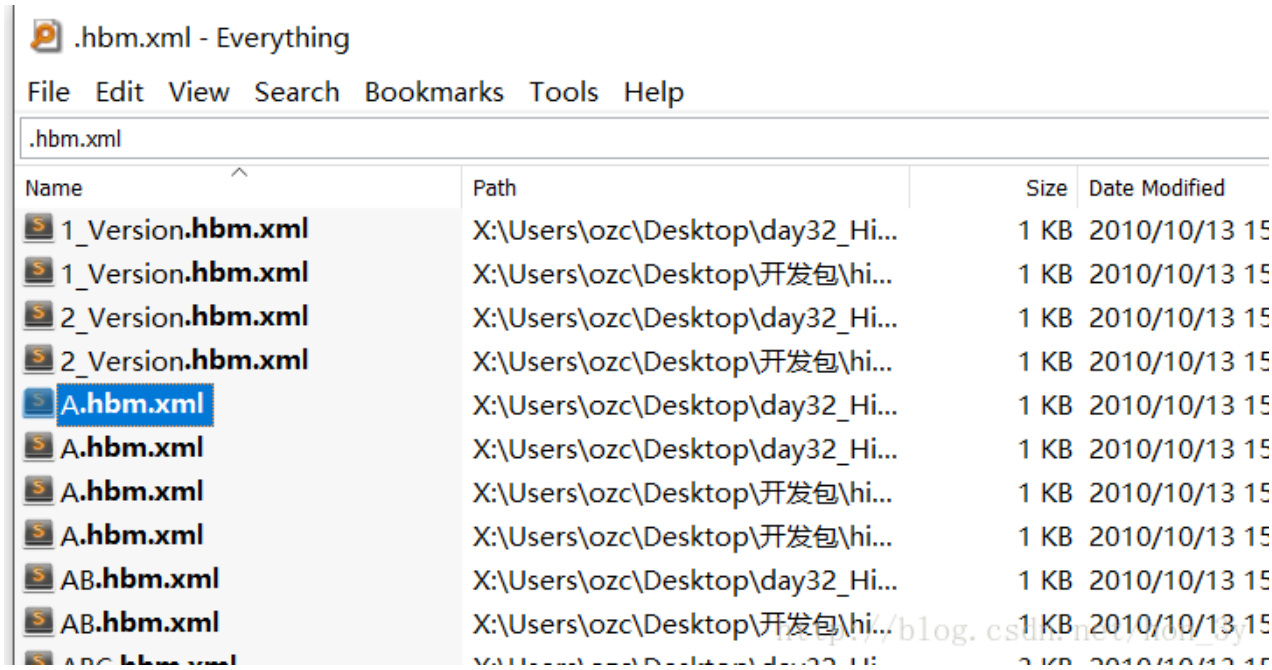
编写对象和对象映射

编写一个User对象->**User.java**

```
public class User {

    private int id;
    private String username;
    private String password;
    private String cellphone;

    //各种setter和getter
}
```

编写对象映射->**User.hbm.xml。一般它和JavaBean对象放在同一目录下**

我们是不知道该XML是怎么写的，可以搜索一下Hibernate文件夹中后缀为 `.hbm.xml` 。看看它们是怎么写的。然后复制一份过来




```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
```

```
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<!--   This mapping demonstrates content-based discrimination for the  table-per-hierarchy mapping strategy, using a formula  discriminator.-->

<hibernate-mapping    package="org.hibernate.test.array">

    <class name="A" lazy="true" table="aaa">

        <id name="id">
            <generator class="native"/>
        </id>

        <array name="bs" cascade="all" fetch="join">
            <key column="a_id"/>
            <list-index column="idx"/>
            <one-to-many class="B"/>
        </array>

    </class>

    <class name="B" lazy="true" table="bbb">
        <id name="id">
            <generator class="native"/>
        </id>
    </class>


</hibernate-mapping>
```

- 在上面的模板上修改～下面会具体讲解这个配置文件!

```
<!--在domain包下-->
<hibernate-mapping package="zhongfucheng.domain">

    <!--类名为User，表名也为User-->
    <class name="User"  table="user">

        <!--主键映射，属性名为id，列名也为id-->
        <id name="id" column="id">
            <!--根据底层数据库主键自动增长-->
            <generator class="native"/>

        </id>

        <!--非主键映射，属性和列名一一对应-->
        <property name="username" column="username"/>
        <property name="cellphone" column="cellphone"/>
        <property name="password" column="password"/>
    </class>
</hibernate-mapping>
```

主配置文件

hibernate.cfg.xml

如果使用Intellij Idea生成的Hibernate可以指定生成出主配置文件 `hibernate.cfg.xml` ，它是要**放在src目录下的**

如果不是自动生成的，我们可以在Hibernate的 `hibernate-distribution-3.6.0.Final\project\etc` 这个目录下可以找到

它长得这个样子：

```

<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.url."/>
        <property name="connection.driver_class"/>
        <property name="connection.username"/>
```

```
        <property name="connection.password"/>
        <!-- DB schema will be updated if needed -->
        <!-- <property name="hbm2ddl.auto">update</property> -->
    </session-factory>
</hibernate-configuration>
```

通过上面的模板进行修改，后面会有对该配置文件进行讲解！

```
<hibernate-configuration>
    <!-- 通常，一个session-factory节点代表一个数据库 -->
    <session-factory>

        <!-- 1. 数据库连接配置 -->
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql:///zhongfucheng</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root</property>
        <!--             数据库方法配置， hibernate在运行的时候，会根据不同的方言生成符合当前数据库语法的sql          -->
        <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>


        <!-- 2. 其他相关配置 -->
        <!-- 2.1 显示hibernate在运行时候执行的sql语句 -->
        <property name="hibernate.show_sql">true</property>
        <!-- 2.2 格式化sql -->
        <property name="hibernate.format_sql">true</property>
        <!-- 2.3 自动建表  -->
        <property name="hibernate.hbm2ddl.auto">create</property>

        <!--3. 加载所有映射-->
        <mapping resource="zhongfucheng/domain/User.hbm.xml"/>

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

测试

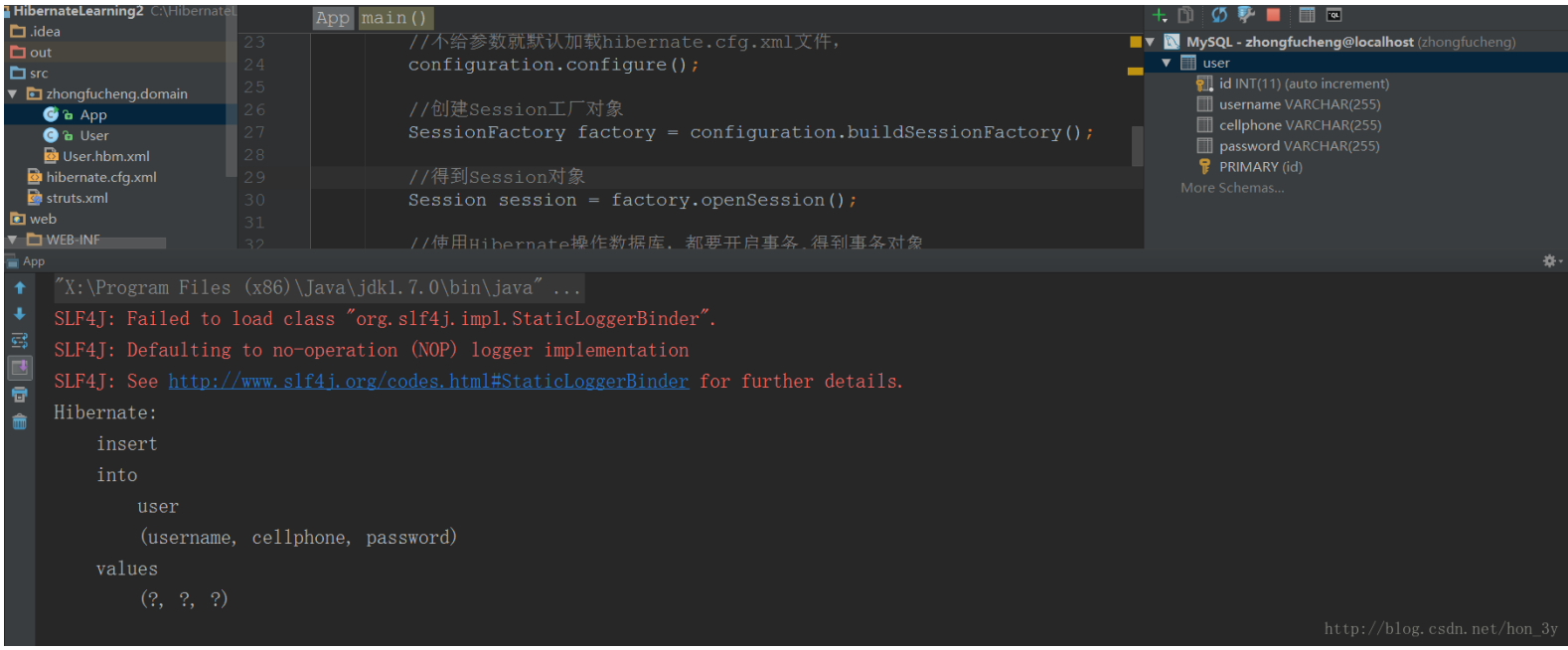
```
package zhongfucheng.domain;

import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.classic.Session;

/** * Created by ozc on 2017/5/6. */
public class App {
    publicstaticvoidmain(String[] args) {

        //创建对象
        User user = new User();
        user.setPassword("123");
        user.setCellphone("122222");
        user.setUsername("nihao");

        //获取加载配置管理类
        Configuration configuration = new Configuration();

        //不给参数就默认加载hibernate.cfg.xml文件，
        configuration.configure();

        //创建Session工厂对象
        SessionFactory factory = configuration.buildSessionFactory();

        //得到Session对象
        Session session = factory.openSession();

        //使用Hibernate操作数据库，都要开启事务,得到事务对象
        Transaction transaction = session.getTransaction();
```

```
        //开启事务
        transaction.begin();

        //把对象添加到数据库中
        session.save(user);

        //提交事务
        transaction.commit();

        //关闭Session
        session.close();
    }
}
```

值得注意的是：**JavaBean的主键类型只能是int类型，因为在映射关系中配置是自动增长的，String类型是不能自动增长的。如果是你设置了String类型，又使用了自动增长，那么就会报出下面的错误！**

```
Caused by: com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Table 'zhongfucheng.user' does
```

执行完程序后，Hibernate就为我们创建对应的表，并把数据存进了数据库了



我们看看快速入门案例的代码用到了什么对象吧，然后一个一个讲解

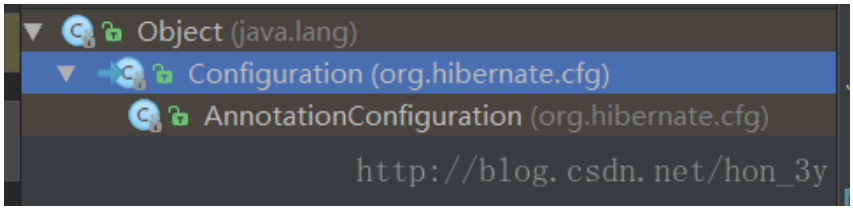
```
publicstaticvoidmain(String[] args) {

        //创建对象
        User user = new User();
        user.setPassword("123");
        user.setCellphone("122222");
        user.setUsername("nihao");

        //获取加载配置管理类
        Configuration configuration = new Configuration();

        //不给参数就默认加载hibernate.cfg.xml文件，
        configuration.configure();

        //创建Session工厂对象
        SessionFactory factory = configuration.buildSessionFactory();

        //得到Session对象
        Session session = factory.openSession();

        //使用Hibernate操作数据库，都要开启事务,得到事务对象
        Transaction transaction = session.getTransaction();

        //开启事务
        transaction.begin();
```

```
        //把对象添加到数据库中
        session.save(user);

        //提交事务
        transaction.commit();

        //关闭Session
        session.close();
    }
```

Configuration

配置管理类：主要管理配置文件的一个类

它**拥有一个子类AnnotationConfiguration**，也就是说：**我们可以使用注解来代替XML配置文件来配置相对应的信息**



configure方法

configure()方法用于加载配置文件

- **加载主配置文件的方法**
  - 如果指定参数，那么加载参数的路径配置文件
  - **如果不指定参数，默认加载src/目录下的hibernate.cfg.xml**

---

buildSessionFactory方法

buildSessionFactory()用于创建Session工厂

SessionFactory

SessionFactory-->Session的工厂，也可以说代表了hibernate.cfg.xml这个文件...hibernate.cfg.xml的就有 `<session-factory>` 这么一个节点

openSession方法

创建一个Session对象

getCurrentSession方法

创建Session对象或取出Session对象

Session

Session是Hibernate最重要的对象，Session维护了一个连接（Connection），只要使用Hibernate操作数据库，都需要用到Session对象

通常我们在DAO层中都会有以下的方法，**Session也为我们提供了对应的方法来实现**！

```
public interface IEmployeeDao {

    voidsave(Employee emp);
    voidupdate(Employee emp);
    Employee findById(Serializable id);
    List<Employee> getAll();
    List<Employee> getAll(String employeeName);
    List<Employee> getAll(int index, int count);
    voiddelete(Serializable id);

}
```

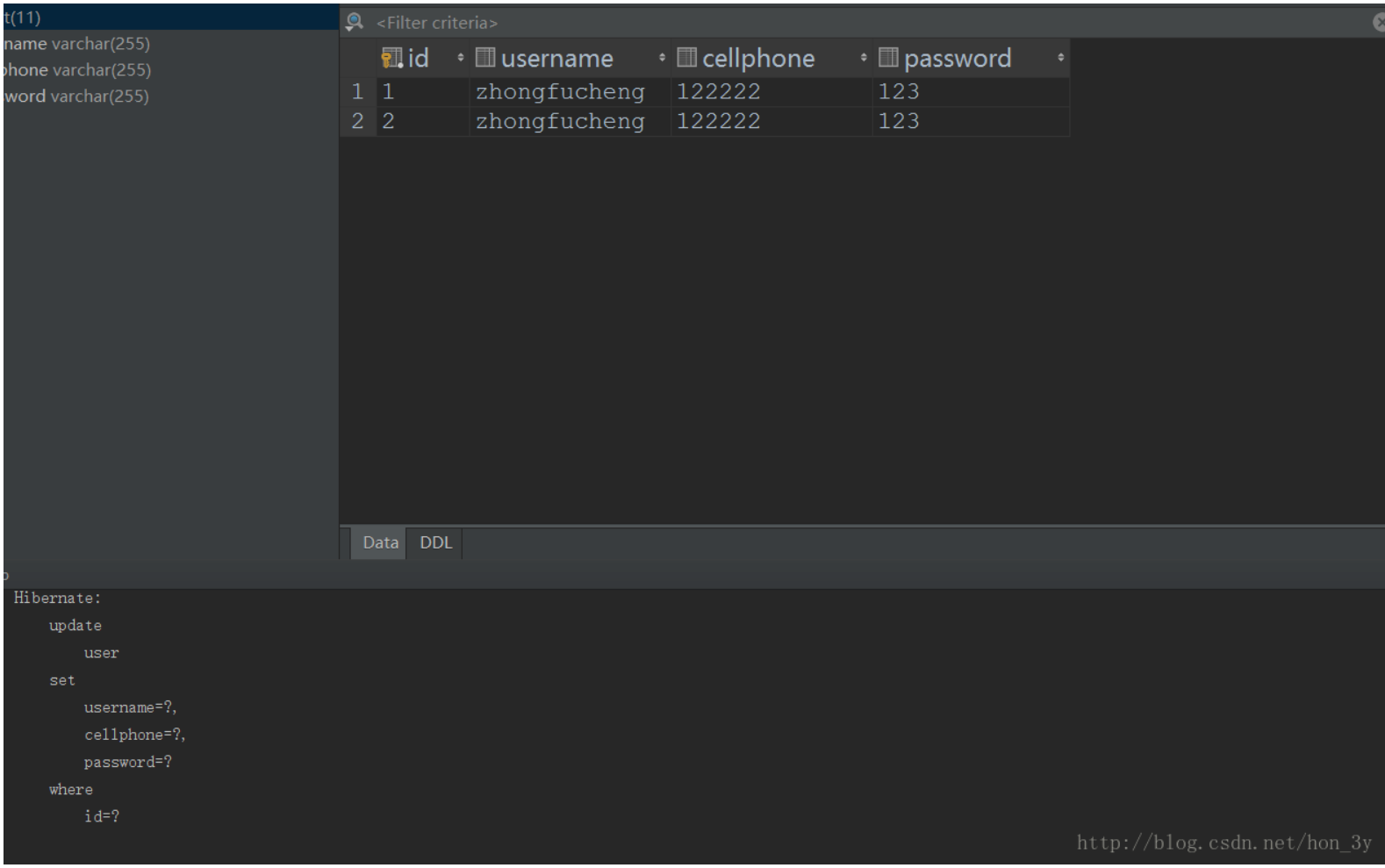
更新操作

我们在快速入门中使用到了save(Objcet o)方法，调用了这个方法就把对象保存在数据库之中了。Session对象还提供着其他的方法来进行对数据库的更新

- **session.save(obj);【保存一个对象】**
- **session.update(obj);【更新一个对象】**
- **session.saveOrUpdate(obj);【保存或者更新的方法】**
  - **没有设置主键，执行保存；**
  - **有设置主键，执行更新操作;**
  - **如果设置主键不存在报错！**

我们来使用一下update()方法吧....**既然是更新操作了，那么肯定需要设置主键的**，不设置主键，数据库怎么知道你要更新什么。将id为1的记录修改成如下：

```
        user.setId(1);
        user.setPassword("qwer");
        user.setCellphone("1111");
        user.setUsername("zhongfucheng");
```



主键查询

通过主键来查询数据库的记录，从而返回一个JavaBean对象

- **session.get(javaBean.class, int id);【传入对应的class和id就可以查询】**
- **session.load(javaBean.class, int id);【支持懒加载】**

User重写toString()来看一下效果：

```
        User user1 = (User) session.get(User.class, 1);
         System.out.println(user1);
```

HQL查询

HQL:hibernate query language 即hibernate提供的面向对象的查询语言

- **查询的是对象以及对象的属性【它查询的是对象以及属性，因此是区分大小写的！】。**

SQL：Struct query language 结构化查询语言

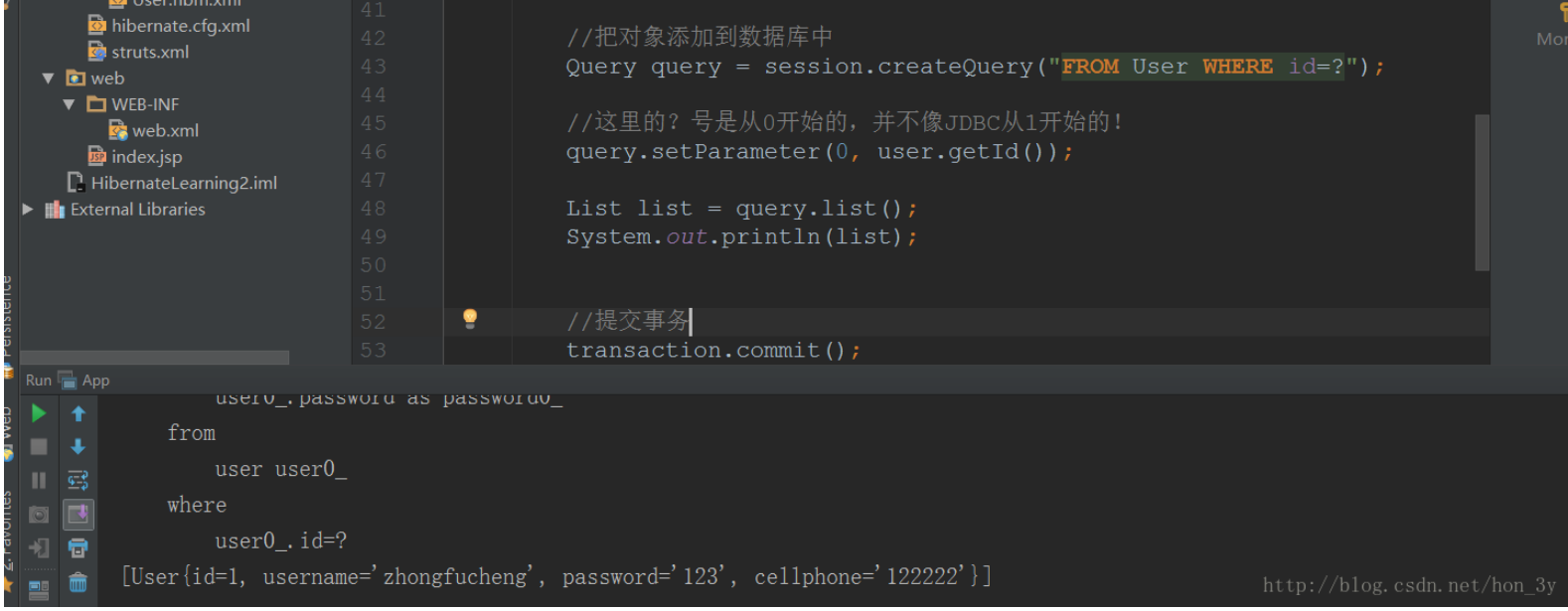
- **查询的是表以及列【不区分大小写】**

HQL是面向对象的查询语言，可以用来查询全部的数据！

```
Query query = session.createQuery("FROM User");

List list = query.list();
System.out.println(list);
```

当然啦，它也可以**传递参数进去查询**

```
Query query = session.createQuery("FROM User WHERE id=?");

//这里的？号是从0开始的，并不像JDBC从1开始的！
query.setParameter(0, user.getId());

List list = query.list();
System.out.println(list);
```



QBC查询

QBC查询: query by criteria 完全面向对象的查询

从上面的HQL查询，我们就可以发现：**HQL查询是需要SQL的基础的，因为还是要写少部分的SQL代码**....**QBC查询就是完全的面向对象查询**...但是呢，我们用得比较少

我们来看一下怎么使用吧：

```
//创建关于user对象的criteria对象
Criteria criteria = session.createCriteria(User.class);

//添加条件
criteria.add(Restrictions.eq("id", 1));

//查询全部数据
List list = criteria.list();
System.out.println(list);
```

本地SQL查询

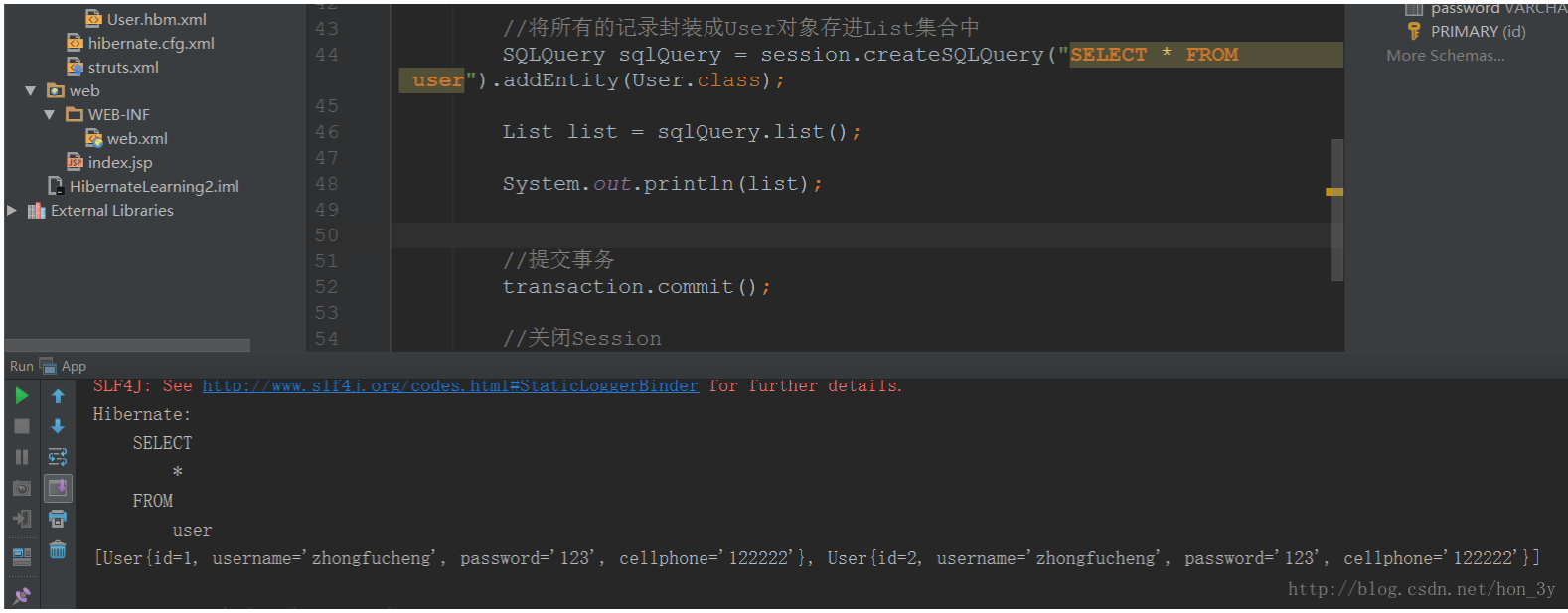
有的时候，如果SQL是非常复杂的，我们**不能靠HQL查询来实现功能的话，我们就需要使用原生的SQL来进行复杂查询了！**

但是呢，它有一个缺陷：**它是不能跨平台的...因此我们在主配置文件中已经配置了数据库的"方言"了。**

我们来简单使用一下把：

```
//将所有的记录封装成User对象存进List集合中
```

```
SQLQuery sqlQuery = session.createSQLQuery("SELECT * FROM user").addEntity(User.class);

List list = sqlQuery.list();

System.out.println(list);
```



beginTransaction方法

开启事务，返回的是一个事务对象....**Hibernate规定所有的数据库操作都必须在事务环境下进行，否则报错！**

主配置文件

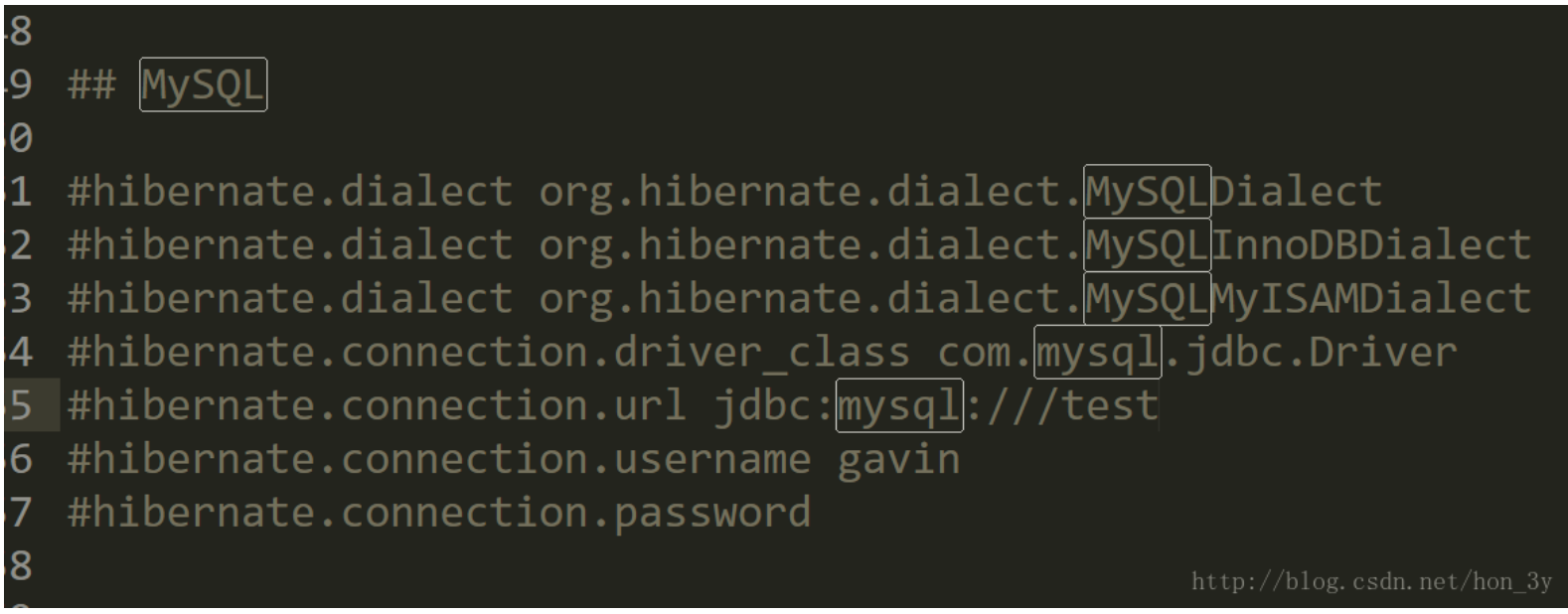
主配置文件主要配置：

- **数据库的信息**
- **其他参数**
- **加载映射文件**

常用的配置信息都可以在 `hibernate-distribution-3.6.0.Final\project\etc\hibernate.properties` 目录下可以找到..

数据库信息

常用的配置信息都可以在hibernate.properties文件中找到，因此，我们来搜索一下：




```
<!-- 1. 数据库连接配置 -->
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql:///hib_demo</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">root</property>
<!--            数据库方法配置， hibernate在运行的时候，会根据不同的方言生成符合当前数据库语法的sql【大致可以理解成：不同的版
```

```
本对应的SQL不同】          -->
        <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
```

其他参数信息

常用的有那么三个：

```
        <!-- 2. 其他相关配置 -->
        <!-- 2.1 显示hibernate在运行时候执行的sql语句 -->
        <property name="hibernate.show_sql">true</property>
        <!-- 2.2 格式化sql -->
        <property name="hibernate.format_sql">true</property>
        <!-- 2.3 自动建表  -->
        <property name="hibernate.hbm2ddl.auto">update</property>
```

需要我们注意的是自动建表，其中它有几个参数：

```
61
62  ## auto schema export
63
64  #hibernate.hbm2ddl.auto create-drop
65  #hibernate.hbm2ddl.auto create
66  #hibernate.hbm2ddl.auto update
67  #hibernate.hbm2ddl.auto validate
68                          http://blog.csdn.net/hon_3y
```

- create-drop 每次在创建sessionFactory时候执行创建表。当调用sesisonFactory的close方法的时候，删除表！
- **create 每次都重新建表； 如果表已经存在就先删除再创建**
- **update 如果表不存在就创建； 表存在就不创建；**
- validate (生成环境时候) 执行验证： 当映射文件的内容与数据库表结构不一样的时候就报错！

加载映射文件

值得注意的是：**mapping的属性使用的是resource!**

```
        <!--3. 加载映射文件-->
        <mapping resource="zhongfucheng/domain/User.hbm.xml"/>
```

加载映射文件其实我们可以在程序中加载，不一定在配置文件中配置....**一般地，我们在测试的时候一般使用程序的方式去加载映射文件【方便】**

那么怎么在程序中加载映射文件呢？

在Configuration对象中提供了**addClass()的方法。**

一般地我们的**映射配置文件和JavaBean对象是放在同一个包下**的。并且映射文件的**命名是有规范的。**因此Hibernate是可以通过**提供的JavaBean对象从而找到相对应的映射文件**！

```
        //获取加载配置管理类
        Configuration configuration = new Configuration();

        //加载User的映射文件！
        configuration.configure().addClass(User.class);
```

映射配置文件

映射文件: 映射一个实体类对象； 描述一个对象最终实现可以直接保存对象数据到数据库中

通常地，我们都是一个JavaBean对象对应一个映射配置文件，并且配置文件和JavaBean对象是放在同一个目录下的

我们按照快速入门的映射配置文件一步一步来讲解：

```xml
<!--在domain包下-->
<hibernate-mapping package="zhongfucheng.domain">

    <!--类名为User，表名也为User-->
    <class name="User"  table="user">

        <!--主键映射，属性名为id，列名也为id-->
        <id name="id" column="id">
            <!--根据底层数据库主键自动增长-->
            <generator class="native"/>

        </id>

        <!--非主键映射，属性和列名一一对应-->
        <property name="username" column="username"/>
        <property name="cellphone" column="cellphone"/>
        <property name="password" column="password"/>
    </class>
</hibernate-mapping>
```

`hibernate-mapping` 节点

常用的属性：

- **package【要映射的对象所在的包(可选,如果不指定,此文件所有的类都要指定全路径)】**
- **auto-import**
  - **默认为true， 在写hql的时候自动导入包名**
  - **如果指定为false, 再写hql的时候必须要写上类的全名；**

`class` 节点

class 映射某一个对象的(一般情况，一个对象写一个映射文件，即一个class节点)

常用的属性：

- **name【指定要映射的对象的类型】**
- **table【指定对象对应的表】**
  - **如果没有指定，默认与对象名称一样**

`property` 节点

property是普通属性的映射，即**JavaBean普通的成员变量属性就使用property来描述**！

常用的属性：

- **name 指定对象的属性名称**
- **column 指定对象属性对应的表的字段名称**
  - **如果不写默认与对象属性一致。**
- **length 指定字符的长度, 默认为255**
- **type 指定映射表的字段的类型，如果不指定会匹配属性的类型**
  - **java类型： 必须写全名【例：java.lang.String】**
  - ** hibernate类型： 直接写类型，都是小写**

值得注意的是：**如果列名称为数据库关键字，需要用反引号或改列名。**当然啦，我们一般不使用关键字来作为列名

`id` 节点

id是主键映射....

- **name 指定对象的属性名**
- **column 指定对象属性对应的表的字段名称**

`<id>` 节点下还有子节点 `<generator class=""/>`

主键的自动生成策略

- identity 自增长(mysql,db2)
- sequence 自增长(序列)， oracle中自增长是以序列方法实现**
- **native 自增长【会根据底层数据库自增长的方式选择identity或sequence】**
  - **如果是mysql数据库, 采用的自增长方式是identity**

    - **如果是oracle数据库， 使用sequence序列的方式实现自增长**
- increment 自增长(会有并发访问的问题，一般在服务器集群环境使用会存在问题。)

指定主键生成策略为**手动指定主键的值**

- **assigned**

指定主键生成策略为**UUID生成的值**

- **uuid**

foreign(外键的方式， one-to-one讲)

`composite-id`

主键一般分为两种：

- 单列主键
- 多列复合主键

单列主键就是上面那种，那么**如果要使用多列复合主键就需要使用** `<composite-id>` **节点来配置了**

- 现在我有这么下面的一个对象，我想**使用username和password作为复合主键**

```
public class User2 {

    private String username;
    private String password;
    private String cellphone;

    //各种setter和getter方法
}
```

- **将username和password抽取成一个类---->CompositeKey....必须实现Serializable接口**

```

package zhongfucheng.domain;

/** * Created by ozc on 2017/5/6. */
public class CompositeKey  implements Serializable{

    private String username;
    private String password;

    public String getUsername() {
        return username;
    }

    publicvoidsetUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    publicvoidsetPassword(String password) {
        this.password = password;
    }
}
```

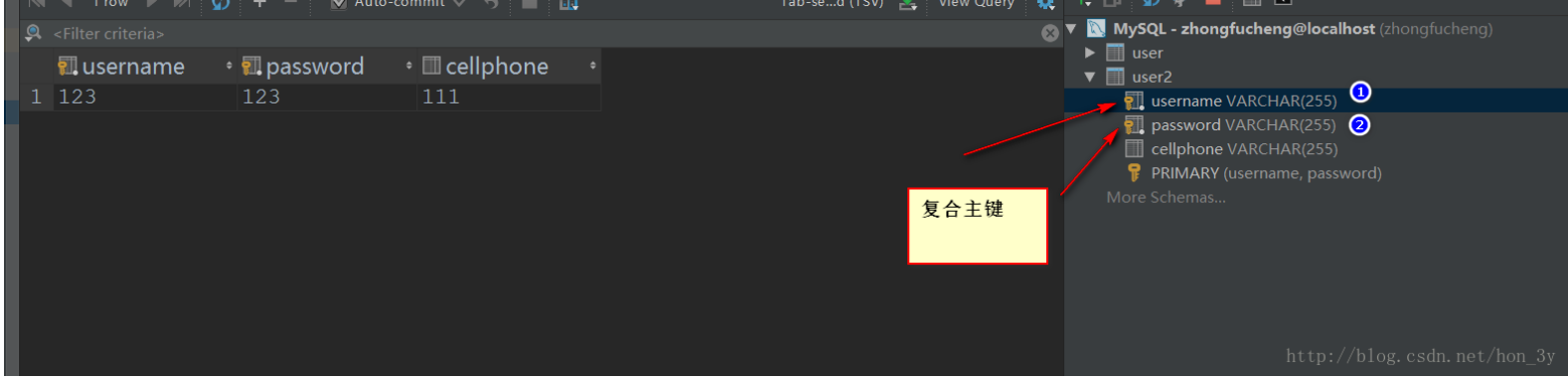
- 在User2中需要**指定一个变量来维护这个主键对象**

```

package zhongfucheng.domain;

/** * Created by ozc on 2017/5/6. */
public class User2 {

    //在User对象中维护这个主键对象
    private CompositeKey key;
```

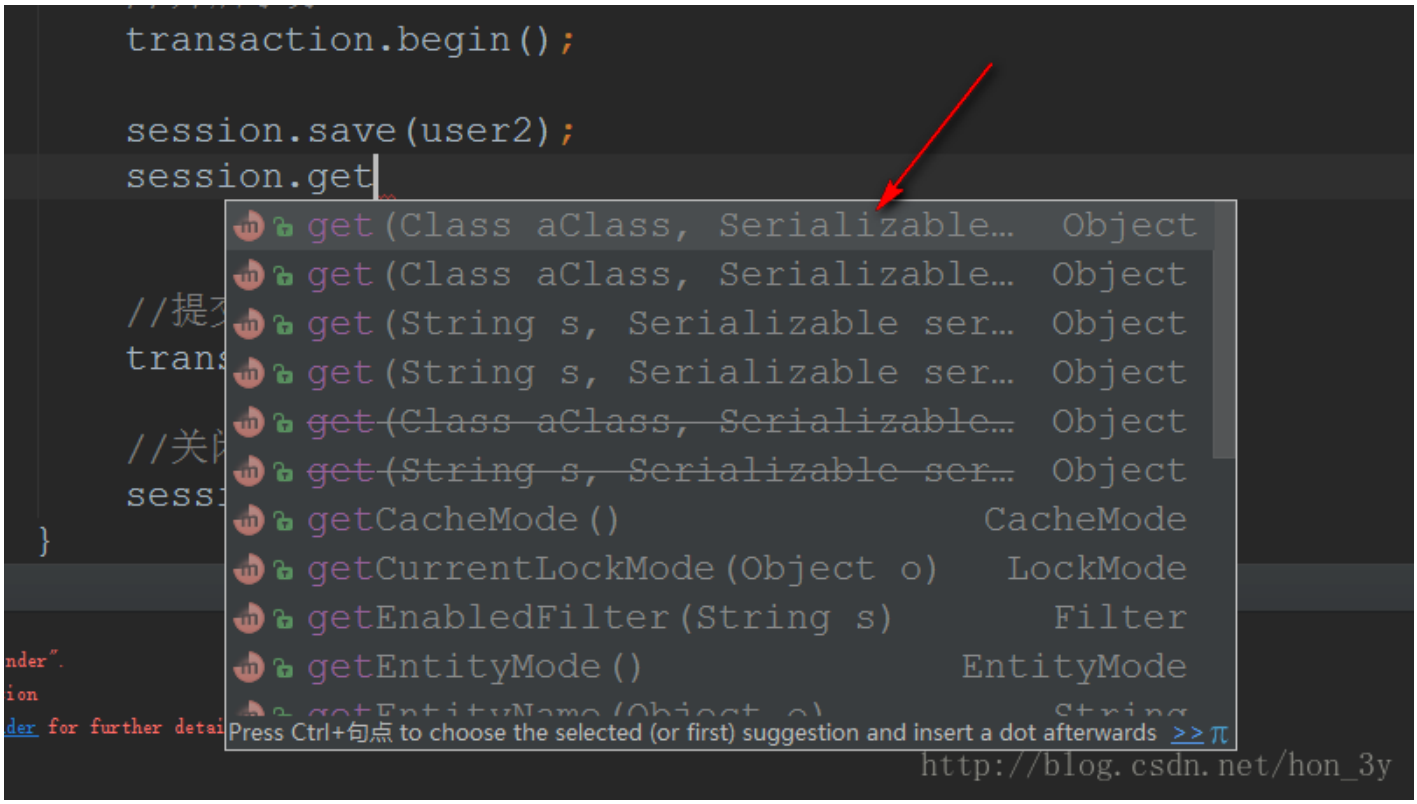
```
    private String cellphone;

    public CompositeKey getKey() {
        return key;
    }

    publicvoidsetKey(CompositeKey key) {
        this.key = key;
    }

    public String getCellphone() {
        return cellphone;
    }

    publicvoidsetCellphone(String cellphone) {
        this.cellphone = cellphone;
    }
}
```

测试

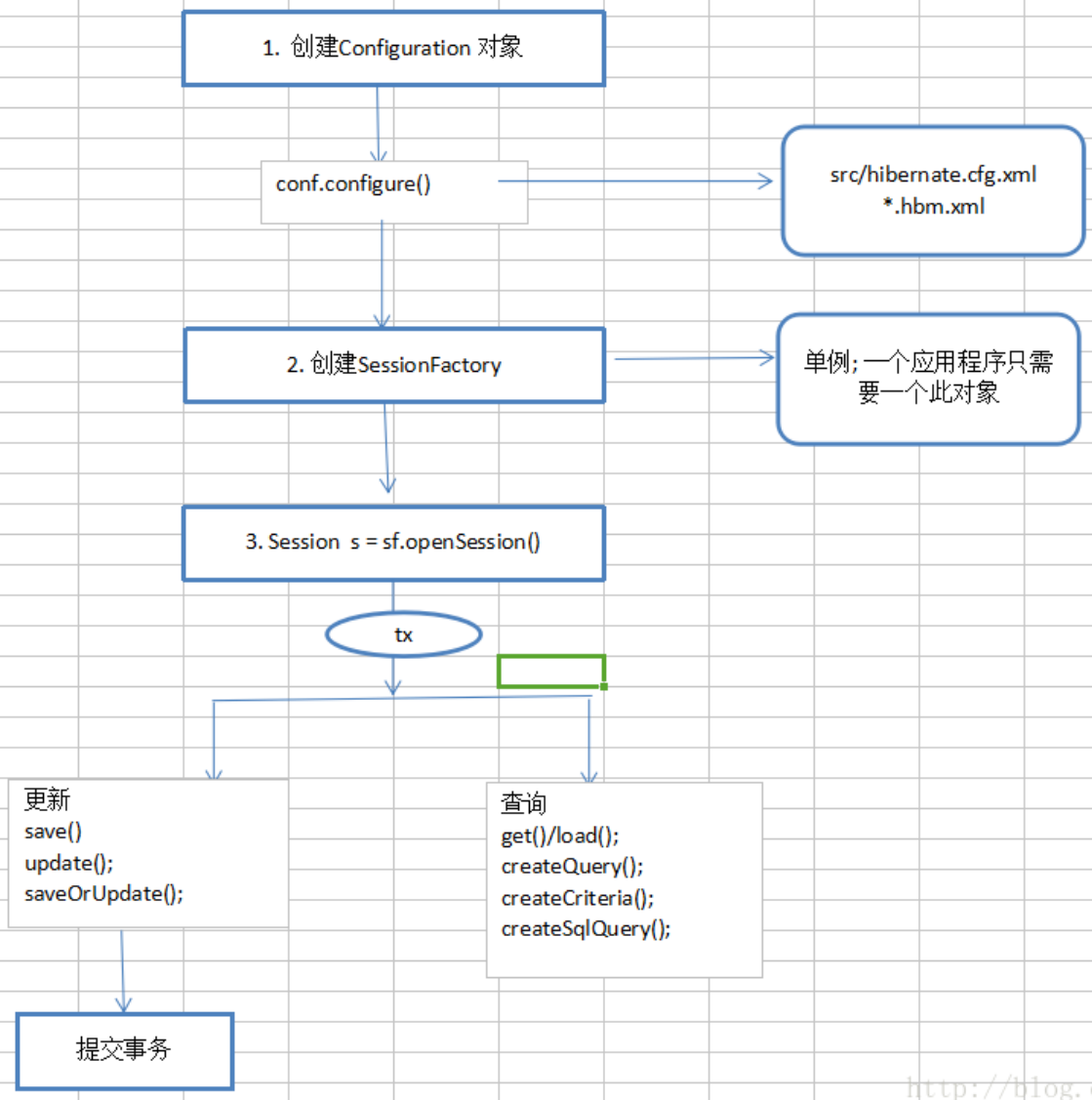
```
    publicstaticvoidmain(String[] args) {

        //创建对象
        User2 user2 = new User2();
        CompositeKey compositeKey = new CompositeKey();
        compositeKey.setUsername("123");
        compositeKey.setPassword("123");
        user2.setCellphone("111");
        user2.setKey(compositeKey);

        //获取加载配置管理类
        Configuration configuration = new Configuration();

        //加载User的映射文件！
        configuration.configure().addClass(User2.class);

        //创建Session工厂对象
        SessionFactory factory = configuration.buildSessionFactory();

        //得到Session对象
        Session session = factory.openSession();

        //使用Hibernate操作数据库，都要开启事务,得到事务对象
        Transaction transaction = session.getTransaction();

        //开启事务
        transaction.begin();

        //添加User2对象到数据库
        session.save(user2);


        //提交事务
        transaction.commit();

        //关闭Session
        session.close();
    }
```



因为**Hibernate在取得的时候是需要实现Serializable这个接口的对象的**...因此**compositeKey必须实现Serializable接口**



Hibernate执行流程图



如果文章有错的地方欢迎指正，大家互相交流。习惯在微信看技术文章，想要获取更多的Java资源的同学，可以**关注微信公众号:Java3y**

出处：http://www.cnblogs.com/Java3y/p/8520601.html