Hibernate

官网:hibernate.org

属于ORM框架 - Object Relationship Mapping - 对象关系映射.

最流行的全自动的ORM框架[采用的是hql语句],Mybatis是半自动的ORM框架[需要对SQL掌握的].

底层都是使用的JDBC技术[Java DataBase Connectivity] - Java数据库互连.

jdbc[由SUN公司]作用:它本质上提供了大量的接口,程序员只需要面向接口编程.

关于db的实现是由数据库厂商去实现[数据库的驱动].

DB厂商

Java app1 -> | J sqlserver[MS]

java app2 -> | D mysql[瑞典的...,现在是oracle]

java app3 -> | BC oracle[oracle]

```
//mysql厂商...打成一个jar包文件
public class MysqlDriver{
    //和数据库进行连接操作....
    public MysqlConnection getMysqlConnection(){
        //mysql厂商提供的数据库的连接的一些配置...

    }
}

//oracle厂商...打成一个jar包文件
```

```
public class OracleDriver{
    //提供了和数据库进行交互的操作
    public OracleConnection getOracleConnection(){
        //oracle厂商提供的数据库的连接的一些配置信息...
    }
}

//开发人员如果想要和mysql进行连接的.
//首先导入mysql的jar - 数据库的驱动.
//哪天需要更改数据库了,重新导入oracle.jar包文件
public class TestMysql{

    public void testConnection(){
        //mysql连接
        //MysqlDriver driver01 = new MysqlDriver();
        //调用连接的方法
        //driver01.getMysqlConnection();

        OracleDriver driver01 = new OracleDriver();
        driver01.getOracleConnection();

        //重新执行mvn package - war - > 扔到[部署]到服务器.
    }
}

//JDBC出现 - 大量的都是接口,实现是由各个DB厂商去实现的
//接口 - 契约[制定者,遵守],摒弃底层
public interface Driver{//定义了驱动的接口
    //指定了关于数据库操作的方法
    Connection getConnection();
}

//契约扔到市场上,如果DB想要和java合作的.
//mysql,oracle开始"骚动"...
public class MyDriver implements Driver{
    @Override
    public Conection getConnection(){
        //具体的实现...  DB厂商实现的
    }
}
//oracle
public class OracleDriver implements Driver{
```

```
    @Override
    public Conection getConnection(){
        //具体的实现...  DB厂商实现的
    }
}

//整合,面向JDBC编程的时候
public class BookDao{
    public void testDao(){
        //获取连接
        //可能导入的是mysql的驱动
        //导入的是oracle驱动
        //面向接口编程
        //对象的编译时类型可以写成接口,对象的运行时类型写成实现类
        Driver driver =
Class.forName("xxx.xxx.xxx.MysqlDriver"),newInstance();
        //最终将这些信息全部写入到.properties文件或者.xml文件
        Driver driver =
Class.forName("xxx.xxx.xxx.OracleDriver"),newInstance();

        //获取连接
        Connection conn = driver.getConnection();
    }
}
```

操作

pom.xml

```
  <properties>
    <!--web工程推荐utf-8-->
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <hibernate.version>4.3.8.Final</hibernate.version>
  </properties>
<!--ojdbc6.jar-->

<!--hibernate框架-->
<dependency>
     <groupId>org.hibernate</groupId>
     <artifactId>hibernate-core</artifactId>
     <version>${hibernate.version}</version>
</dependency>
```

关于hibernate.cfg.xml文件

约定:

- 如果是普通工程,则默认放入在src的根目录
- 如果是maven工程,所有资源文件全部放入到src/main/resources目录 - 类路径.

其实hibernate默认的配置文件是hibernate.properties文件.

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-
3.0.dtd">
<!--hibernate配置-->
<hibernate-configuration>
    <!--session-factory - 数据源工厂-->
    <session-factory>
        <!--数据库的相关配置-->
        <!--数据库的驱动-->
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</proper
ty>
        <!--数据库的url-->
        <!--主协议:次协议:客户端类型:@主机ip:端口号:版本-->
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:XE</property>
```

```
        <!--用户名和密码-->
        <property name="connection.username">aistar</property>
        <property name="connection.password">aistar</property>

        <!--SQL的方言-->
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</prope
rty>

        <!--是否显示sql,在生产环境下需要设置成false-->
        <property name="show_sql">true</property>
        <!--是否格式化sql语句-->
        <property name="format_sql">true</property>

        <!--
            create
            update
        -->
        <property name="hbm2ddl.auto">update</property>
    </session-factory>
</hibernate-configuration>
```

重要的API

1. Configuration[C] - 读取hibernate.cfg.xml文件.
2. SessionFactory
3. Session - 提供crud操作.

步骤

- hibernate.cfg.xml
- tech.aistar.util - HibernateUtil
- tech.aistar.entity - BookType实体类

```
package tech.aistar.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
```

```java
import javax.persistence.Table;
import java.io.Serializable;

/**
 * 图书类型...
 */
@Entity
@Table(name = "book_type")
public class BookType implements Serializable{

    //对象标识
    private Integer id;

    //类型名称
    private String typeName;

    //空参构造
    public BookType(){

    }

    public BookType(String typeName){
        this.typeName = typeName
    }

    /**主键的生成策略...**/
    @Id
    @GeneratedValue
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getTypeName() {
        return typeName;
    }

    public void setTypeName(String typeName) {
        this.typeName = typeName;
```

```
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("BookType{");
        sb.append("id=").append(id);
        sb.append(", typeName='").append(typeName).append('\'');
        sb.append('}');
        return sb.toString();
    }
}
```

- 需要在hiberante.cfg.xml中添加类的映射

```
<mapping class="tech.aistar.entity.BookType"></mapping>
```

- 指定DAO - 持久层接口[和数据库进行直接的交互的那一层]

  tech.aistar.dao - IBookTypeDao

```
package tech.aistar.dao;

import tech.aistar.entity.BookType;

import java.util.List;

/**
 * Created by Administrator on 2019/1/22 0022.
 */
public interface IBookTypeDao {

    /**
     * 保存一个图书类型
     * @param bookType
     */
    void save(BookType bookType);

    /**
     * 查询所有的图书类型
     * @return
     */
    List<BookType> findAll();
```

```
}
```

- 编写接口的实现类 - 完成hibernate的编程步骤

```
package tech.aistar.dao.impl;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import tech.aistar.dao.IBookTypeDao;
import tech.aistar.entity.BookType;
import tech.aistar.util.HibernateUtil;

import java.util.List;

/**
 * Created by Administrator on 2019/1/22 0022.
 */
public class BookTypeDaoImpl implements IBookTypeDao{
    @Override
    public void save(BookType bookType) {
        //hibernate编程步骤
        //1. 获取session = Connection + Cache[一级缓存]
        Session session = HibernateUtil.getSession();

        //2.开始事务
        Transaction tx = session.beginTransaction();

        //3. 执行CRUD操作
        session.save(bookType);

        //4. 提交事务
        tx.commit();

        //5. 关闭session
        session.close();//释放资源
    }

    @Override
    public List<BookType> findAll() {
```

```
        //hibernate编程步骤
        //1. 获取session = Connection + Cache[一级缓存]
        Session session = HibernateUtil.getSession();

        //2.开始事务
        Transaction tx = session.beginTransaction();

        //3. 执行CRUD操作
        //session.save(bookType);

        //定义一个hql语句
        String hql = "from BookType";//面向的是对象的查询语句
        //String sql = "select * from book_type";//sql语句

        //获取   Query查询语句对象
        Query query = session.createQuery(hql);

        List<BookType> bookTypes = query.list();

        //4. 提交事务
        tx.commit();

        //5. 关闭session
        session.close();//释放资源

        return bookTypes;
    }
}
```

- 单元测试

```
package tech.aistar.dao;

import org.junit.Test;
import tech.aistar.dao.impl.BookTypeDaoImpl;
import tech.aistar.entity.BookType;

/**
 * Created by Administrator on 2019/1/22 0022.
 */
public class TestBookTypeDao {
```

```
    private IBookTypeDao bookTypeDao = new BookTypeDaoImpl();

    @Test
    public void testSave(){
        //模拟一条图书类型的数据...
        BookType bookType = new BookType("热卖图书2");
        bookTypeDao.save(bookType);
    }
   @Test
    public void testFindAll(){
        List<BookType> bookTypeList = bookTypeDao.findAll();
        if(null!=bookTypeList && bookTypeList.size()>0){
            for(BookType b:bookTypeList){
                System.out.println(b);
            }
        }
    }
}
```