JAVA反射机制

博客阅读：http://www.cnblogs.com/jqyp/archive/2012/03/29/2423112.html

1.什么是反射机制？

反射机制是指程序在运行时能够获取自身的信息。

在java中，只要给定类的名字，就可以通过反射机制来获得类的所有信息

2.反射机制的优点与缺点

- 这涉及到了动态与静态的概念
  - 静态编译：在编译时确定类型，绑定对象
  - 动态编译：运行时确定类型，绑定对象。 动态编译最大限度的发挥了java的灵活性，体现了多态的运用，有效降低类的耦合性
- 反射机制的优点：可以实现动态创建对象和编译
  - 比如，一个大型的软件，不可能一次就把把它设计的很完美，当这个程序编

译后，发布了，当发现需要更新某些功能时，我们不可能要用户把以前的卸载，再重新安装新的版本，假如这样的话，这个软件肯定是没有多少人用的。采用静态的话，需要吧整个程序重新编译一次才可以实现功能的更新，而采用反射机制，就可以不用卸载，只需要在运行时才动态的创建和编译

- 缺点：对性能有影响，使用反射基本上是一种解释操作，慢于直接执行相同的操作

3.API

首先得根据传入的类的全名来创建Class对象。

Class c=Class.forName("className");注明：className必须为全限定名

Object obj=c.newInstance();//创建对象的实例

OK，有了对象就什么都好办了，想要什么信息就有什么信息了。

获得构造函数的方法

- Constructor getConstructor(Class[] params)//根据指定参数获得public构造器
- Constructor[] getConstructors()//获得public的所有构造器
- Constructor getDeclaredConstructor(Class[] params)//根据指定参数获得

public和非public的构造器

- Constructor[] getDeclaredConstructors()//获得public和非public构造器

获得类方法的方法

- Method getMethod(String name, Class[] params),根据方法名，参数类型获得方法
- Method[] getMethods()//获得所有的public方法
- Method getDeclaredMethod(String name, Class[] params)//根据方法名和参数类型，

获得public和非public的方法

- Method[] getDeclaredMethods()//获得所以的public和非public方法

获得类中属性的方法

Field getField(String name)//根据变量名得到相应的public变量

Field[] getFields()//获得类中所有public的变量

Field getDeclaredField(String name)//根据方法名获得public和非public变量

Field[] getDeclaredFields()//获得类中所有的public和非public方法

常用的就这些，知道这些，其他的都好办……

4.反射机制应用

、用反射机制能干什么事

刚开始在使用jdbc时侯，在编写访问数据库时写到想吐，有八个表，每个表都有增删改查中操作那时候还不知道有反射机制这个概念，所以就对不同的表创建不同的dao类,这样不仅开发速率地，而且代码冗余的厉害，最要命的是看着差不多的，然后直接复制修改，由于容易犯各种低级的错误（大小写啊，多一个或少一个字母啊……），一个错误就可以让你找半天。

有了java反射机制，什么都好办了，只需要写一个dao类，四个方法，增删改查，传入不同的对象，就OK啦，无需为每一个表都创建dao类，反射机制会自动帮我们完成剩下的事情，这就是它的好处。说白了，反射机制就是专门帮我们做那些重复的有规则的事情，所以现在很多的自动生成代码的软件就是运用反射机制来完成的，只要你按照规则

```
public class NetJavaSession {
    /**
     * 解析出保存对象的sql语句
     *
     * @param object
     *            ：需要保存的对象
     * @return: 保存对象的sql语句
     */
    public static String getSaveObjectSql(Object object) {
        // 定义一个sql字符串
        String sql = "insert into ";
        // 得到对象的类
        Class c = object.getClass();
        // 得到对象中所有的方法
        Method[] methods = c.getMethods();
        // 得到对象中所有的属性
        Field[] fields = c.getFields();
        // 得到对象类的名字
        String cName = c.getName();
        // 从类的名字中解析出表名
        String tableName = cName.substring(cName.lastIndexOf(".") + 1,
                cName.length());
```

```
sql += tableName + " ( ";
List<String> mList = new ArrayList<String>();
List vList = new ArrayList();
for (Method method : methods) {
    String mName = method.getName();
    if (mName.startsWith("get") && !mName.startsWith("getClass")) {
        String fieldName = mName.substring(3, mName.length());
        mList.add(fieldName);
        System.out.println("字段名字----->" + fieldName);
        try {
            Object value = method.invoke(object, null);
            System.out.println("执行方法返回的值: " + value);
            if (value instanceof String) {
                vList.add("\"" + value + "\"");
                System.out.println("字段值------>" + value);
            } else {
                vList.add(value);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
public class NetJavaSession {
    /**
```

```
    * 解析出保存对象的sql语句
    *
    * @param object
    *            ：需要保存的对象
    * @return：保存对象的sql语句
    */
    public static String getSaveObjectSql(Object object) {
        // 定义一个sql字符串
        String sql = "insert into ";
        // 得到对象的类
        Class c = object.getClass();
        // 得到对象中所有的方法
        Method[] methods = c.getMethods();
        // 得到对象中所有的属性
        Field[] fields = c.getFields();
        // 得到对象类的名字
        String cName = c.getName();
        // 从类的名字中解析出表名
        String tableName = cName.substring(cName.lastIndexOf(".") + 1,
                cName.length());
        sql += tableName + "(";
        List<String> mList = new ArrayList<String>();
        List vList = new ArrayList();
        for (Method method : methods) {
            String mName = method.getName();
            if (mName.startsWith("get") && !mName.startsWith("getClass")) {
                String fieldName = mName.substring(3, mName.length());
                mList.add(fieldName);
                System.out.println("字段名字----->" + fieldName);
                try {
                    Object value = method.invoke(object, null);
                    System.out.println("执行方法返回的值：" + value);
                    if (value instanceof String) {
                        vList.add("\"" + value + "\"");
                        System.out.println("字段值------>" + value);
                    } else {
                        vList.add(value);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
        for (int i = 0; i < mList.size(); i++) {
            if (i < mList.size() - 1) {
                sql += mList.get(i) + ",";
            } else {
                sql += mList.get(i) + ") values(";
            }
        }
        for (int i = 0; i < vList.size(); i++) {
            if (i < vList.size() - 1) {
                sql += vList.get(i) + ",";
            } else {
                sql += vList.get(i) + ")";
            }
        }

        return sql;
    }

    public static List getDatasFromDB(String tableName, int Id) {

        return null;
```

```
    }

    /**
     * 将对象保存到数据库中
     *
     * @param object
     *            : 需要保存的对象
     * @return: 方法执行的结果;1:表示成功, 0:表示失败
     */
    public int saveObject(Object object) {
        Connection con = Connect2DBFactory.getDBConnection();
        String sql = getSaveObjectSql(object);
        try {
            // Statement statement=(Statement) con.createStatement();
            PreparedStatement psmt = con.prepareStatement(sql);
            psmt.executeUpdate();
            return 1;
        } catch (SQLException e) {
            e.printStackTrace();
            return 0;
        }
    }

    /**
     * 从数据库中取得对象
     *
     * @param arg0
     *            : 对象所属的类
     * @param id
     *            : 对象的id
     * @return:需要查找的对象
     */
    public Object getObject(String className, int Id) {
        // 得到表名字
        String tableName = className.substring(className.lastIndexOf(".") + 1,
                className.length());
        // 根据类名来创建Class对象
        Class c = null;
        try {
            c = Class.forName(className);

        } catch (ClassNotFoundException e1) {

            e1.printStackTrace();
        }
        // 拼凑查询sql语句
        String sql = "select * from " + tableName + " where Id=" + Id;
        System.out.println("查找sql语句: " + sql);
        // 获得数据库链接
        Connection con = Connect2DBFactory.getDBConnection();
        // 创建类的实例
        Object obj = null;
        try {

            Statement stm = con.createStatement();
            // 得到执行查寻语句返回的结果集
            ResultSet set = stm.executeQuery(sql);
            // 得到对象的方法数组
            Method[] methods = c.getMethods();
            // 遍历结果集
            while (set.next()) {
                obj = c.newInstance();
                // 遍历对象的方法
                for (Method method : methods) {
```

```
                    String methodName = method.getName();
                    // 如果对象的方法以set开头
                    if (methodName.startsWith("set")) {
                        // 根据方法名字得到数据表格中字段的名字
                        String columnName = methodName.substring(3,
                                methodName.length());
                        // 得到方法的参数类型
                        Class[] parmts = method.getParameterTypes();
                        if (parmts[0] == String.class) {
                            // 如果参数为String类型，则从结果集中按照列名取得对应的值，并且执行改set方法
                            method.invoke(obj, set.getString(columnName));
                        }
                        if (parmts[0] == int.class) {
                            method.invoke(obj, set.getInt(columnName));
                        }
                    }

                }
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
        return obj;
    }
```