使用Cglib动态创建Java类，解决你意想不到的问题，让复杂变简单

2014-12-10 且看且珍惜 文章来源 阅 4827 转 37

转藏到我的图书馆

微信分享：

在日常Java数据库开发中我们很少会用到自动创建Java类及类中的变量和方法，目前最常见的第三方数据库操作框架基本都要手动创建一个和数据库表一模一样的javabean，并提供每个字段对于的set，get方法，鉴于这种方式能否用一种方法只要一个数据库表名称，就可以返回一个对应该表的javabean的List列表或者其他集合，而完全不用创建一系列的Javabean呢？ 答案是有，Java本身自带的类库目前好像还没有类似的功能，但是开源组织开发出一个jar包，就是cglib，目前的版本已经到cglib-nodep-2.1\_3.jar，其底层采用asm动态创建字节码，对外只提供了很少的几个方法，而且也没有相应的ＡＰＩ文档。所以学习起来比较难以下手。

下面我简单的介绍下如何使用cglib动态创建类。（例子是参考网上的）

package com.javase.util;

import java.util.Iterator;

import java.util.Map;

import java.util.Set;

import net.sf.cglib.beans.BeanGenerator;

import net.sf.cglib.beans.BeanMap;

public class CglibBeanUtil {

public Object object = null;

/\*\*

\* 属性map

\*/

public BeanMap beanMap = null;

public CglibBeanUtil() {

super();

}

@SuppressWarnings("unchecked")

public CglibBeanUtil(Map propertyMap) {

this.object = generateBean(propertyMap);

this.beanMap = BeanMap.create(this.object);

}

/\*\*

\* 给bean属性赋值

\*

\* @param property

\* 属性名

\* @param value

\* 值

\*/

public void setValue(String property, Object value) {

beanMap.put(property, value);

}

/\*\*

\* 通过属性名得到属性值

\*

\* @param property

\* 属性名

\* @return 值

\*/

public Object getValue(String property) {

return beanMap.get(property);

}

/\*\*

\* 得到该实体bean对象

\*

\* @return

\*/

public Object getObject() {

return this.object;

}

@SuppressWarnings("unchecked")

private Object generateBean(Map propertyMap) {

BeanGenerator generator = new BeanGenerator();

Set keySet = propertyMap.keySet();

for (Iterator i = keySet.iterator(); i.hasNext();) {

String key = (String) i.next();

generator.addProperty(key, (Class) propertyMap.get(key));

}

return generator.create();

}

}

//-----------------------------------------------------------------------------------------------------

//下面是测试类

package com.cglib.test;

import java.lang.reflect.Method;

import java.util.HashMap;

import com.cglib.util.CglibBeanUtil;

public class CglibTest {

@SuppressWarnings("unchecked")

public static void main(String[] args) throws ClassNotFoundException {

// 设置类成员属性

HashMap propertyMap = new HashMap();

propertyMap.put("id", Class.forName("java.lang.Integer"));

propertyMap.put("name", Class.forName("java.lang.String"));

propertyMap.put("address", Class.forName("java.lang.String"));

// 生成动态 Bean

CglibBeanUtil bean = new CglibBeanUtil(propertyMap);

// 给 Bean 设置值

bean.setValue("id", new Integer(123));

bean.setValue("name", "454");

bean.setValue("address", "789");

// 从 Bean 中获取值，当然了获得值的类型是 Object

System.out.println(" >> id = " + bean.getValue("id"));

System.out.println(" >> name = " + bean.getValue("name"));

System.out.println(" >> address = " + bean.getValue("address"));

// 获得bean的实体

Object object = bean.getObject();

// 通过反射查看所有方法名

Class clazz = object.getClass();

System.out.println(clazz.getSimpleName());

Method[] methods = clazz.getDeclaredMethods();

for (int i = 0; i < methods.length; i++) {

System.out.println(methods[i].getName());

}

}

}

通过以上例子我们可以发现，只要提供一个Map型的属性键值对（键是变量名，值是变量的类型），就可以创建出对应的类的所以字段（注意，cglib在给创建的字段前面都有前缀“$cglib\_prop\_”，使用时要注意，而类名是$cglib开头的如“net.sf.cglib.empty.Object$$BeanGeneratorByCGLIB$$b5c0bff6”，这个类是唯一的），并自动提供了每个字段的set，get方法。并且该类是生成在内存中，并没有生成在本地。

现在我们要做一种用底层jdbc操作数据只需根据表名返回的List<Object>型的集合，完全不用手动创建Javabean。

思路是：

1：创建一个数据库连接工具类DBUtil

2：定义一个接口：ObjectDao，该接口包含方法

List<Object> getObjectList(Connection conn, String tableName);//通过表名和前台传的连接返回对象

//如果是一个完整的项目，一般数据库连接是固定的不会每次都是一个新的数据库连接。所以上面方法中的连接可以根据实际情况省掉，而只需一个全局的连接。

现在有个问题是，通过jdbc获取到该表的数据后，如果动态的将不同的表的字段和值能组装成一个类实例并放入List中，那么只能有一种方法那就是反射。要用反射那么我们就要用Class，所以上面接口的方法看着不能满足，

所以我们在上面接口再定义个方法

List<Object> getObjectList(Connection conn, Object ob, String tableName); //和上面的方法一样，只是剥离开重新调用了下

我们最终目的是用jdbc查询后返回的ResultSet和Class来组装成一个List<Object>，所以单独定义一个工具类

GenerateUtil，其中有个方法public static List<Object> generateObjectListFromDB(Class cls, ResultSet rs)，这样在接口的实现类中最终调用这个工具类的方法就可以返回我们所要的集合了。

3：下面就要实现这个接口：ObjectDaoImpl

下面直接贴上代码：

package com.javase.dao.impl;

import java.lang.reflect.Field;

import java.lang.reflect.Method;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.ResultSetMetaData;

import java.sql.SQLException;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import com.javase.dao.ObjectDao;

import com.javase.db.DBUtil;

import com.javase.util.CglibBeanUtil;

import com.javase.util.GenerateUtil;

public class ObjectDaoImpl implements ObjectDao

{

@Override

public List<Object> getObjectList(Connection conn, String tableName)

{

Map<String, Class> columnMap = new HashMap<String, Class>();

CglibBeanUtil beanUtil = null;

String sql = "select \* from " + tableName;

Object ob = null;

try

{

ResultSetMetaData rs = conn.prepareStatement(sql).executeQuery().getMetaData();

for(int i = 1; i <= rs.getColumnCount(); i++)

{

columnMap.put(rs.getColumnName(i), Class.forName(rs.getColumnClassName(i)));

}

ob = new CglibBeanUtil(columnMap).getObject();

} catch (SQLException e)

{

e.printStackTrace();

} catch (ClassNotFoundException e)

{

e.printStackTrace();

}

return this.getObjectList(conn, ob, tableName);

}

public List<Object> getObjectList(Connection conn, Object ob, String tableName)

{

String sql = "select \* from " + tableName;

ResultSet rs = null;

List<Object> list = null;

try

{

rs = conn.prepareStatement(sql).executeQuery();

list = GenerateUtil.generateObjectListFromDB(ob.getClass(), rs);

} catch (SQLException e)

{

e.printStackTrace();

} finally

{

DBUtil.closeConnection(conn);

}

return list;

}

}

其中调用的工具生成类有2个，一个是CglibBeanUtil，和最上面的cglib例子一样，

另一个工具类是GenerateUtil

代码为：

package com.javase.util;

import java.lang.reflect.Field;

import java.lang.reflect.Method;

import java.sql.ResultSet;

import java.util.LinkedList;

import java.util.List;

public class GenerateUtil

{

public static List<Object> generateObjectListFromDB(Class cls, ResultSet rs)

{

System.out.println(cls.getName());

List<Object> list = new LinkedList<Object>();

Field[] fields = cls.getDeclaredFields();

try

{

while(rs.next())

{

Object ob = cls.newInstance();

for(int i = 0; i < fields.length; i++)

{

String fieldName = fields[i].getName().replace("$cglib\_prop\_", "");

String setMethodName = "set" + fieldName.substring(0, 1).toUpperCase() + fieldName.substring(1);

Method setMethod = cls.getDeclaredMethod(setMethodName, new Class[]{fields[i].getType()});

setMethod.invoke(ob, rs.getObject(fieldName));

}

list.add(ob);

}

} catch(Exception e)

{

e.printStackTrace();

}

return list;

}

}

所以我们对外提供的方法就是ObjectDaoImpl类的 getObjectList(Connection conn, String tableName)，其中的数据库连接可以根据自己需要删掉换成全局的，再往外还可以包装对应的业务层Service及其ServiceImpl，完全根据自己实际需要。

最后我们用一个测试类：

package com.javase.dao;

import java.sql.Connection;

import java.util.List;

import com.javase.dao.impl.ObjectDaoImpl;

import com.javase.db.DBUtil;

public class test

{

public static void main(String[] args)

{

Connection conn = DBUtil.getConnection('O', "138.138.2.103", 1522, "SJPT", "TPS", "SJPTTPS");

ObjectDaoImpl odi = new ObjectDaoImpl();

List<Object> list = odi.getObjectList(conn, "TP\_CD00101");

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

}

}

运行后完全正常

还有点是对事务的支持我没做，还有就是获得List<Object>后如何使用也没写，既然在daoImpl类中可以得到Class，那么再定义一个获取对于的Class的工具类，构造一个公共的方法获取List中对象的数据的方法即可，在这里我简单提下，方法参数就为(List<Object> list, Class class)，在其中运用反射即可获取所有数据，这个由各位自己完成。

cglib的实例运用就说到这，谢谢