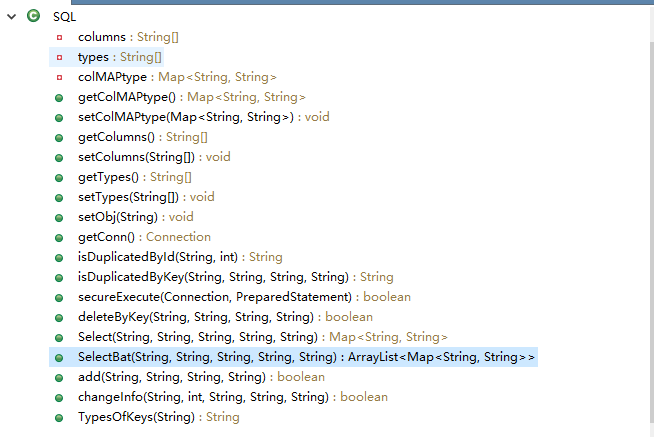
数据库进行封装操作，使得基本操作可以简单地使用基本操作。

现有的基本操作如下图所示：



使用SQL类能够快速地写出一个entityDao类，一般来说，所需的仅仅是粘贴复制（容我吹一下23333）。

SQL类使用效果简介

假设我们要写一个entityDao，举例子，比如userCollection类，用来表示用户的收藏记录。

userCollection有如下的私有属性：

1. id

2.userid

3.courseid

4.collecteddate

然后构造setter和getter方法。

我们的userCollectionDao使用SQL类来构造会十分的简单，但是我们需要对原先的entity也就是userCollection做一点点处理：

1. 重写toString方法。
2. 添加一个toSQL的静态方法。

{

**public** **static** String toSQL(){

String attrs="id,userid,courseid,collecteddate";

String types="int,int,int,Timestamp";

**return** attrs+"#"+types;

}

**public** String toString(){

**return** String.*format*(GraceJava.*repeat*("%s", , 4),id,userid,courseid,collecteddate);

}

}

这里的GraceJava,repeat的作用是将"%s" 重复数次，并使用"," 连接，然后格式化字符串。我们做这些工作是为了让SQL类帮助我们自动推导类型。

然后我们的userCollectionDao就可以轻易的使用SQL类来开发。如下：

**public** **class** userCollectionDao {

**private** SQL javaSQL;

**private** **final** String tableName="userCollection";

**public** userCollectionDao (){

javaSQL=**new** SQL();

javaSQL.setObj(userCollection.*toSQL*());

}

定义一个final变量 tableName 表示mysql的对应表名。

定义一个私有变量javaSQL并在类初始化时使用一个setObj方法，参数就是

userCollection.*toSQL*()这个东西。

你可以发现，换成任何一个entityDao,这里的代码只需要修改 数据库的表名 和

entity.toSQL()里的具体的entity。

然后以下给出一些entityDao里的数据库基本操作（依然用userCollectionDao举例）：

**public** **boolean** deleteById(**int** id){

**boolean** ret=javaSQL.deleteByKey(tableName, "id", GraceJava.*int2str*(id), "int");

**return** ret;

}

deleteById方法对所有entityDao都是如此。

不变的还有以下两种查询方法：

**public** Map<String , String> SelectById(String SelectedKeys,**int** id){

**return** javaSQL.Select(tableName, SelectedKeys, "id", GraceJava.*int2str*(id), "int");

}

**public** ArrayList<Map<String,String>> SelectByKeys(String SelectedKeys,String MatchKeys,String MatchValues){

String MatchValueTypes=javaSQL.TypesOfKeys(MatchKeys);

**return** javaSQL.SelectBat(tableName, SelectedKeys, MatchKeys, MatchValues, MatchValueTypes);

}

他们分别返回一个单例查询和一个批量查询结果。

而对于数据库的增和改，不同的entity需要改动大概20个字符串。

约定一些记号：

Entity: 一个实例类

entity: Entity 的一个实例对象。

下面代码的红色部分表示建立不同entityDao需要改动的地方。

数据库增添记录：

**public** **boolean** add(Entity entity){

String judge=javaSQL.isDuplicatedById(tableName, entity.getId());

**boolean** added=**false**;

**if** (judge.equals("No")){

String columns= String.*join*(",",javaSQL.getColumns());

String types=String.*join*(",", javaSQL.getTypes());

added=javaSQL.add(tableName,columns , entity.toString(), types);

}

**return** added;

}

数据库通过id修改记录(用现在的新entity来更新数据库中对应的entity，这显然也是一种面向对象的方法)：

**public** **boolean** updateById(**int** id, Entity entity){

String MatchValues=entity.toString();

String columns= String.*join*(",",javaSQL.getColumns());

String types=String.*join*(",", javaSQL.getTypes());

**boolean** ret=javaSQL.changeInfo(tableName, id,columns , MatchValues, types);

**return** ret;

}

如何调用SQL类构造的entityDao方法

entityDao{

boolean add(Entity entity);

// add 方法。参数：对应的entity对象。会通过id来判断是否重复，重复则不执行添加操作。

boolean deleteById(int id);

//根据id删除记录

boolean updateById(int id, Entity entity);

//根据一个新的entity去修改数据库中对应id上旧entity。

Map<String , String> SelectById(String SelectedKeys,int id);

//根据id返回对应的单例记录的对应字段值（字符串格式）。

// SelectedKeys应该是这样的 “username, password, level”这样用”,”分割开的字符串。

ArrayList < Map<String , String>> SelectByKeys(String SelectedKeys,String MatchKeys,String MatchValues);

//根据MatchKeys和对应的值MatchValues选择 对应的字段值SelectedKeys，会返回一个ArrayList包含一条或多条查询结果。

/\*

例如下面这段代码：

ArrayList<Map<String,String>> mappings= ud.SelectByKeys("username","level" ,"1");

Map<String,String> map =ud.SelectById("username", 1);

System.***out***.println(mappings);

System.***out***.println(map);

Console的结果如下：

[{username=SaberLily}, {username=Archer}]

{username=SaberLily}

\*/

}