DbUtils(二) 结果集实例

单行数据处理：ScalarHandler　ArrayHandler　MapHandler　BeanHandler

多行数据处理：BeanListHandler　AbstractListHandler（ArrayListHandler MapListHandler ColumnListHandler）

AbstractKeyedHandler（KeyedHandler BeanMapHandler）

可供扩展的类：BaseResultSetHandler

Dbutils使用结果集的方法有query、insert、insertBatch三个。这些方法都在QueryRunner类中，需要注意的是insert和update方法都能执行 "insert"开头的sql语句，但是返回值有区别。insert 执行后返回的是表中的插入行生成的主键值，update 返回的是受语句影响的行数。所以，如果目标表中有主键且需要返回插入行的主键值就用 insert 方法，如果表没有主键或者不需要返回主键值可使用 update 方法。

先建立测试用数据表[users]：

id	userName	loginName	userPassword	userLevel	userLock
1	测试用户1	test1	jiseflwes	10	0
2	知道什么	hello	2556sefsfs	10	1
3	编程就编程	cjava	sfsfsef254sefs	2	0

字段类型，id 为主键：

```
[id] [int] IDENTITY(1,1) NOT NULL,
[userName] [nchar](20) NOT NULL,
[loginName] [nchar](20) NOT NULL,
[userPassword] [nchar](100) NOT NULL,
[userLevel] [int] NOT NULL,
[userLock] [bit] NOT NULL,
```

1、ScalarHandler<T>

用于获取结果集中第一行某列的数据并转换成 T 表示的实际对象。

该类对结果集的处理直接在 handle 方法中进行，不涉及 dbutils 库的其他类。

```
String sql = "select * from users";
// ---- query 语句 ----
// ScalarHandler 的参数为空或null时，返回第一行第一列的数据
int rs = runner.query(sql, new ScalarHandler<Integer>());
System.out.println("ScalarHandler: " + rs); // Print:[ScalarHandler: 1]

// ScalarHandler 的参数可以是列的索引（从1开始）或列名
String rs = runner.query(sql, new ScalarHandler<String>(2));
// 或者 String rs = runner.query(sql, new ScalarHandler<String>("userName"));
System.out.println("ScalarHandler: " + rs); // Print:[ScalarHandler: 测试用户1]

// ---- insert 语句 ----
// 因为我使用的是mssql数据库，QueryRunner的insert获取插入数据的主键其实调用的是select SCOPE_IDENTITY()
// 数据库执行后返回的类型是numeric，映射到 java 类型就是 java.math.BigDecimal
String inSql = "insert users (userName, loginName, userPassword, userLevel, userLock) values (?, ?, ?, ?, ?)";
BigDecimal insertRs = runner.insert(inSql,new ScalarHandler(),  "java程序编写", "javahello", "sefsfsfwew", "15", false);
System.out.println("ScalarHandler: " + insertRs); // Print:[ScalarHandler: 4]
```

使用的时候一定要保证提供正确的列索引或列名，并且结果类型也要正确可转换。

2、ArrayHandler

用于获取结果集中的第一行数据，并将其封装到一个数组中，一列值对应一个数组元素。

handle 源码：

```
public Object[] handle(ResultSet rs) throws SQLException {
    // convert = new BasicRowProcessor()
    // 如果有数据，将调用 BasicRowProcessor 的 toArray(rs) 方法处理
    return rs.next() ? this.convert.toArray(rs) : EMPTY_ARRAY;
}
```

```
// ---- query 语句 ----
String sql = "select * from users";
Object[] rs = runner.query(sql, new ArrayHandler());
// Print: ArrayHandler: [1, 测试用户1, test1, jiseflwes, 10, false]
System.out.println("ArrayHandler: " + Arrays.toString(rs));

// ---- insert 语句 ----
String inSql = "insert users_t (userName, loginName, userPassword, userLevel, userLock) values (?, ?, ?, ?, ?)";
Object[] insertRs = runner.insert(inSql,new ArrayHandler(),  "java程序编写", "javahello", "sefsfsfwew", "15", false);
// Print: ArrayHandler: [5]
System.out.println("ArrayHandler: " + Arrays.toString(insertRs));
```

3、MapHandler

用于获取结果集中的第一行数据，并将其封装到一个Map中，Map 中 key 是数据的列别名（as label）,如果没有就是列的实际名称，Map 中 value 就是列的值，注意代表列的 key 不区分大小写。

handle 源码：

```
public Map<String, Object> handle(ResultSet rs) throws SQLException {
    // convert = new BasicRowProcessor()
    // 如果有数据，将调用 BasicRowProcessor 的 toMap(rs) 方法处理
    return rs.next() ? this.convert.toMap(rs) : null;
}
```

通过查看 BasicRowProcessor 代码，可以知道封装结果集的 Map 其实是一个 LinkedHashMap 对象。

```
// ---- query 语句 ----
String sql = "select userName, loginName, userPassword as password, userLevel, userLock from users";
Map<String, Object> rs = runner.query(sql, new MapHandler());
// Print: MapHandler: {userName=测试用户1, loginName=test1, password=jiseflwes, userLevel=10, userLock=false}
System.out.println("MapHandler: " + rs);
// 列名小写 Print: username: 测试用户1
System.out.println("username: " + rs.get("username"));
// 列名大写 Print: USERNAME: 测试用户1
System.out.println("USERNAME: " + rs.get("USERNAME"));
// 使用了as指定别名，那么取数据的时候一定要用别名，否则返回null。
// Print: userPassword as password: jiseflwes
System.out.println("userPassword as password: " + rs.get("password"));
// Print: userPassword as password: null
System.out.println("userPassword as password: " + rs.get("userPassword"));

// ---- insert 语句 ----
String inSql = "insert users (userName, loginName, userPassword, userLevel, userLock) values (?, ?, ?, ?, ?)";
Map<String, Object> insertRs = runner.insert(inSql,new MapHandler(),  "java程序编写", "javahello", "sefsfsfwew", "15", false);
// 注意这个键（key）是由数据库驱动定义的。
// Print: MapHandler:{GENERATED_KEYS=6}
System.out.println("MapHandler: " + insertRs);
// 我用的是微软提供的驱动，所以是GENERATED_KEYS，如果是其他驱动就又不同了（比如jtds驱动key是ID）
// jtds驱动使用 insertRs.get("ID")
// Print: MapHandler: 6
System.out.println("MapHandler: " + insertRs.get("GENERATED_KEYS"));
```

4、BeanHandler<T>

用于获取结果集中的第一行数据，并将其封装到JavaBean对象。

整个转换过程最终会在 BeanProcessor 类中完成。

+ View Code

执行代码：

```
//---- query 语句 ----
String sql = "select * from users";
Users rs = runner.query(sql, new BeanHandler<Users>(Users.class));
// Print: BeanHandler: Users{id=1, userName='测试用户1', loginName='test1', userPassword='jiseflwes', userLevel=10, userLock=true}
System.out.println("BeanHandler: " + rs);
// Print: BeanHandler: test1
System.out.println("BeanHandler: " + rs.getLoginName());
```

需要注意的是，默认的情况下要保证表的字段和javabean的属性一致（字符一致即可，对大小写不敏感），比如字段是userLock，那么javabean中属性必须是userLock这几个字母（可以是userlock，userLock，userLOCK，不过还是建议按照规范来定义）。

但有个问题，数据表中的字段可能已经定下来了，而且名称可能不太规范，比如用下划线（login_name），或者加了一个类型前缀（chrLoginName），如果修改表字段，那么涉及到的修改地方太多了，其实查看源码可以知道BeanHandler有两个构造方法：

```
// BeanHandler 构造方法
public BeanHandler(Class<T> type) {
    this(type, ArrayHandler.ROW_PROCESSOR);
}
public BeanHandler(Class<T> type, RowProcessor convert) {
    this.type = type;
    this.convert = convert;
}
```

可以看出其实都是调用的第二个方法。

```
runner.query(sql, new BeanHandler<Users>(Users.class));
// 等价于
runner.query(sql, new BeanHandler<Users>(Users.class, new BasicRowProcessor()));
// 等价于
runner.query(sql, new BeanHandler<Users>(Users.class, new BasicRowProcessor(new BeanProcessor())));
// 所以关键的地方在 new BeanProcessor() 这个具体处理结果的对象
```

情况一：只涉及到下划线，表字段名用下划线间隔（如表users_t字段：[id]，[user_name]，[login_name]，[user_password]，[user_level]，[user_lock]），现在要封装到Javabean中Users类（代码见上），其中属性使用驼峰命名。可以用dbutils1.6提供的BeanProcessor类的子类GenerousBeanProcessor。

```
String sql = "select id,user_name,login_name,user_password,user_level,user_lock from users_t";
// 创建一个BeanProcessor对象
// GenerousBeanProcessor 仅仅重写了父类BeanProcessor的mapColumnsToProperties方法
BeanProcessor bean = new GenerousBeanProcessor();
// 将GenerousBeanProcessor对象传递给BasicRowProcessor
RowProcessor processor = new BasicRowProcessor(bean);
// 最后使用GenerousBeanProcessor的mapColumnsToProperties处理表字段到javabean的属性映射
Users rs = runner.query(sql, new BeanHandler<Users>(Users.class, processor));
// Print: BeanHandler: Users{id=1, userName='测试用户1', loginName='test1', userPassword='jiseflwes', userLevel=10, userLock=true}
System.out.println("BeanHandler: " + rs);
```

情况二：完全改变表字段到Javabean属性的映射（如表users_m字段：[yhmid]，[charUsername]，[charLoginName]，[charPassword]，[intLevel]，[boolLock]）映射到Users类（代码同上）：

```
// BeanProcessor 有两个构造方法，可以传入一个HashMap集合
// HashMap 规定了表字段映射到Javabean的哪个属性，即key为字段名称，value为对应的javabean属性
// map.put(表字段名称, Javabean属性名称)
Map<String, String> map = new HashMap<String, String>();
map.put("yhmid", "id");
map.put("charUsername", "userName");
map.put("charLoginName", "loginName");
```

```
map.put("charPassword", "userPassword");
map.put("intLevel", "userLevel");
map.put("boolLock", "userLock");
// 用构建好的HashMap建立一个BeanProcessor对象
BeanProcessor bean = new BeanProcessor(map);
RowProcessor processor = new BasicRowProcessor(bean);
Users rs = runner.query(sql, new BeanHandler<Users>(Users.class, processor));
// Print: BeanHandler: Users{id=1, userName='测试用户1', loginName='test1', userPassword='jiseflwes', userLevel=10, userLock=true}
System.out.println("BeanHandler: " + rs);
```

5、BeanListHandler<T>

用于将结果集的每一行数据转换为Javabean，再将这个Javabean添加到ArrayList中。可以简单的看着是BeanHandler的高级版，只不过是多了一步，就是将生成的Javabean添加到ArrayList中，其他的处理都和BeanHandler一样。

```
String sql = "select * from users";
List<Users> rs = runner.query(sql, new BeanListHandler<Users>(Users.class));
// Print: BeanListHandler: [
// Users{id=1, userName='测试用户1', loginName='test1', userPassword='jiseflwes', userLevel=10, userLock=true},
// Users{id=1, userName='知道什么', loginName='hello', userPassword='2556sefsfs', userLevel=10, userLock=true},
// Users{id=1, userName='编程就编程', loginName='cjava', userPassword='sfsfsef254sefs', userLevel=2, userLock=false}]
System.out.println("BeanListHandler: " + rs);
```

6、AbstractListHandler<T>

```
// AbstractListHandler 类实现了handle方法
@Override
public List<T> handle(ResultSet rs) throws SQLException {
    List<T> rows = new ArrayList<T>();
    while (rs.next()) {
        rows.add(this.handleRow(rs)); // 每个子类实现自己的handleRow方法
    }
    return rows;
}
```

AbstractListHandler抽象类已经实现handle方法，该方法其实只是起到一个包装作用，将处理好的每行数据添加到ArrayList中。每行的数据处理通过调用handleRow方法实现，所有它的3个子类都必须实现这个方法。

6.1 ArrayListHandler (extends AbstractListHandler<Object[]>)

用于将结果集每行数据转换为Object数组（处理过程等同与ArrayHandler），再将该数组添加到ArrayList中。简单点，就是将每行数据经过ArrayHandler处理后添加到ArrayList中。

```
String sql = "select * from users";
List rs = runner.query(sql, new ArrayListHandler());
// Print:
// [1, 测试用户1, test1, jiseflwes, 10, true]
// [2, 知道什么, hello, 2556sefsfs, 10, true]
// [3, 编程就编程, cjava, sfsfsef254sefs, 2, false]
for(Object user : rs) {
    System.out.println(Arrays.toString((Object[])user));
}
```

6.2 MapListHandler (extends AbstractListHandler<Map<String, Object>>)

用于将结果集每行数据转换为Map（处理过程等同与MapHandler），再将Map添加到ArrayList中。简单点，就是将每行数据经过MapHandler处理后添加到ArrayList中。

```
String sql = "select * from users";
List rs = runner.query(sql, new MapListHandler());
// Print:
// {1, 测试用户1, test1, jiseflwes, 10, true}
// {2, 知道什么, hello, 2556sefsfs, 10, true}
// {3, 编程就编程, cjava, sfsfsef254sefs, 2, false}
for(Object user : rs) {
    System.out.println(Arrays.toString((Map<String, Object>)user));
}
```

6.3 ColumnListHandler<T> (extends AbstractListHandler<T>)

根据列索引或列名获取结果集中某列的所有数据，并添加到ArrayList中。可以理解为ScalarHandler<T>的加强版。

```
String sql = "select * from users";
List<String> rs = runner.query(sql, new ColumnListHandler<String>(2));
// 等同 List<String> rs = runner.query(sql, new ColumnListHandler<String>("userName"));
// Print:
// 测试用户1
// 知道什么
// 编程就编程
for(String user : rs) {
    System.out.println(user);
}
```

7、AbstractKeyedHandler<K, V>

AbstractKeyedHandler是一个抽象类，已经实现了handle方法，其子类必须实现createKey(ResultSet rs)和createRow(ResultSet rs)方法，以便handle()的调用。

```
/**
  * 返回一个HashMap<K, V>
  * createKey(rs) 将某列的值作为HashMap的key
  * createRow(rs) 将结果集转换后作为HashMap的value
  */
 @Override
 public Map<K, V> handle(ResultSet rs) throws SQLException {
     Map<K, V> result = createMap();
     while (rs.next()) {
         result.put(createKey(rs), createRow(rs)); // 需要子类自己实现
     }
     return result;
 }
```

7.1 KeyedHandler<K> (extends AbstractKeyedHandler<K, Map<String, Object>>)

用于获取所有结果集，将每行结果集转换为Map<String, Object>，并指定某列为key。可以简单的认为是一个双层Map，相当于先对每行数据执行MapHandler，再为其指定key添加到一个HaspMap中。KeyedHandler<K> 中的<K>是指定的列值的类型。

```
String sql = "select * from users";
// 在这儿指定主键id为结果key，也可以传入列名 new KeyedHandler<Integer>("id")
Map<Integer, Map<String, Object>> rs = runner.query(sql, new KeyedHandler<Integer>(1));
// Print: KeyedHandler: {
//         1={id=1, userName=测试用户1, loginName=test1, userPassword=jiseflwes, userLevel=10, userLock=true},
//         2={id=2, userName=知道什么, loginName=hello, userPassword=2556sefsfs, userLevel=10, userLock=true},
//         3={id=3, userName=编程就编程, loginName=cjava, userPassword=sfsfsef254sefs, userLevel=2, userLock=false}}
System.out.println("KeyedHandler: " + rs);

// 也可以指定其他列作为key，但是需要注意如果指定的列值存在重复值，那么后面的值将覆盖前面的，最终HashMap中key都是唯一的。
// 如指定列userLevel为key，最终只有两个结果，因为前两行userLevel值都是10。
Map<Integer, Map<String, Object>> rs = runner.query(sql, new KeyedHandler<Integer>("userLevel"));
// Print: KeyedHandler: {
//         2={id=3, userName=编程就编程, loginName=cjava, userPassword=sfsfsef254sefs, userLevel=2, userLock=false},
//         10={id=2, userName=知道什么, loginName=hello, userPassword=2556sefsfs, userLevel=10, userLock=true}}
System.out.println("KeyedHandler: " + rs);
```

7.2 BeanMapHandler<K, V> (extends AbstractKeyedHandler<K, V>)

用于获取所有结果集，将每行结果集转换为Javabean作为value，并指定某列为key，封装到HashMap中。相当于对每行数据的做BeanHandler一样的处理后，再指定列值为Key封装到HashMap中。

```
String sql = "select * from users";
// new BeanMapHandler<Integer, Users>(Users.class,"id")
Map<Integer, Users> rs = runner.query(sql, new BeanMapHandler<Integer, Users>(Users.class,1));
```

```
// Print: BeanMapHandler: {
// 1=Users{id=1, userName='测试用户1', loginName='test1', userPassword='jiseflwes', userLevel=10, userLock=true},
// 2=Users{id=2, userName='知道什么', loginName='hello', userPassword='2556sefsfs', userLevel=10, userLock=true},
// 3=Users{id=3, userName='编程就编程', loginName='cjava', userPassword='sfsfsef254sefs', userLevel=2, userLock=false}}
System.out.println("BeanMapHandler: " + rs);
```

需要注意这个结果转换类也可以像BeanHandler的情况一和情况二介绍的那样定义一个processor，但默认情况下这么做了就会以每行的第一列作为Key，不能指定其他列为Key。

```
// 这种情况下，以每行第一列为key
Map<Integer, Users> rs = runner.query(sql, new BeanMapHandler<Integer, Users>(Users.class,processor));
```

8、BaseResultSetHandler<T>

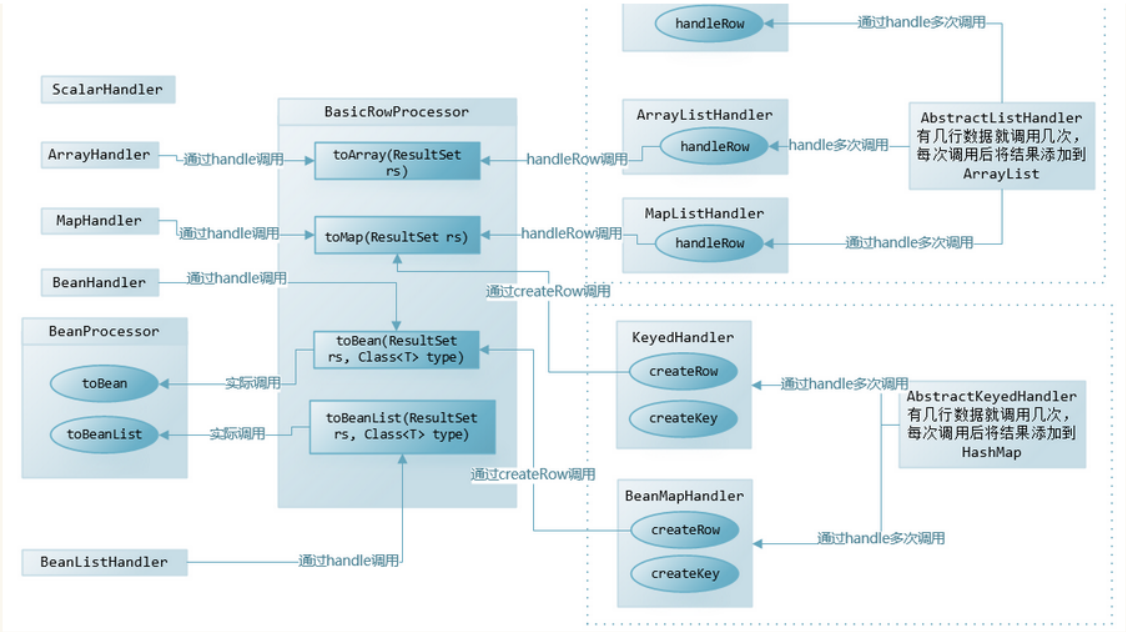
根据文档介绍，如果上面的结果集处理类都不能满足你的要求，可以通过继承这个抽象类定义自己的结果处理类，子类必须实现无参方法handle()。

做个简单的例子，比如要将指定列值加一个前缀"class-"后添加到ArrayList中：

```
//------------- 定义类 MeResultHandler.java -------------
/**
 * 自定义的结果处理类，对结果集的操作直接调用父类已经封装好的方法。
 * 这儿只是对取到的结果包装加工。
 */
public class MeResultHandler extends BaseResultSetHandler<List<String>> {

    private final int columnIndex;

    // 指定要获取值的列索引
    public MeResultHandler(int columnIndex) {
        this.columnIndex = columnIndex;
    }

    // 重写父类的方法，封装每行数据
    @Override
    protected List<String> handle() throws SQLException {
        List<String> rows = new ArrayList<String>();
        // 因为父类已经封装好了对ResultSet各种操作，直接调用父类方法 next()
        while(this.next()) {
            rows.add(handleRow());
        }
        return rows;
    }

    // 自定义的数据处理方法
    @SuppressWarnings("unchecked")
    private String handleRow() throws SQLException {
        // 直接调用父类方法 getObject()
        return "class-" + String.valueOf(this.getObject(this.columnIndex));
    }
}
//------------- 使用类 -------------
List<String> rs = runner.query(sql, new MeResultHandler(1));
// Print: MeResultHandler: [class-1, class-2, class-3]
System.out.println("MeResultHandler: " + rs);
```

==========

总的来说，最终的数据处理是在 BasicRowProcessor 的四个方法中进行，涉及到JavaBean的话会通过 BasicRowProcessor 调用 BeanProcessor 的两个方法。其他的都是对每行数据转换后的结果的封装。


handleRow
通过handle多次调用
ScalarHandler
BasicRowProcessor
ArrayListHandler
AbstractListHandler
有几行数据就调用几次，
每次调用后将结果添加到
ArrayList
ArrayHandler
通过handle调用
toArray(ResultSet
rs)
handleRow调用
handleRow
handle多次调用
MapHandler
toMap(ResultSet rs)
MapListHandler
BeanHandler
通过createRow调用
BeanProcessor
toBean(ResultSet
rs, Class<T> type)
KeyedHandler
createRow
createKey
toBean
实际调用
toBeanList
toBeanList(ResultSet
rs, Class<T> type)
AbstractKeyedHandler
有几行数据就调用几次，
每次调用后将结果添加到
HashMap
BeanMapHandler
BeanListHandler

喜欢简洁，远离繁琐