REX DB 1.0

目录

开发/运行运行环境

Rexdb需要如下运行环境：

- **JDK 5.0及以上版本**

在开始前，请检查环境变量中的如下jar包：

- **JDBC驱动**
- **rexdb-1.0.0.jar（或其它版本）**
- javassist-3.20.0-GA.jar（可选）
- logger4j/logger4j2/slf4j（可选其一，也可以都不使用）
- dbcp/C3P0/BoneCP等（可选其一，也可以都不使用）

全局配置 *rexdb.xml*

Rexdb依赖全局配置文件**rexdb.xml**，用于配置数据源、日志、异常信息语言等。该文件默认存放于classpath环境变量中（例如，在Java EE应用中，应将其放置于**WEB-INF/classes**目录中）。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//rex-soft.org//REXDB DTD 1.0//EN" "http://www.rex-soft.org/dtd/rexdb-1-config.dtd">
<configuration>
    <!-- 默认数据源，Oracle数据库，使用框架内置的连接池 -->
    <dataSource>
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
        <property name="url" value="jdbc:oracle:thin:@127.0.0.1:1521:rexdb" />
        <property name="username" value="rexdb" />
        <property name="password" value="12345678" />
    </dataSource>
    <!-- student数据源，Mysql数据库，使用了Apache DBCP连接池 -->
    <dataSource id="student" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="jdbcUrl" value="jdbc:mysql://127.0.0.1:3306/rexdb?characterEncoding=utf8" />
        <property name="username" value="root" />
        <property name="password" value="12345678" />
    </dataSource>
</configuration>
```

更多选项请参考Rexdb用户手册。

查询单条记录 *DB.get()*

`org.rex.DB.get()`方法用于查询单条记录，并返回指定的java对象实例（无记录时返回null），格式如下：

> T DB.get([String dataSourceId,] String sql, [Object[] | org.rex.db.Ps | Map | Object parameter,] Class clazz)

- dataSourceId：可选，配置文件中的数据源id，不设置时使用默认数据源；
- sql：必填，待执行的SQL语句；
- parameter：可选，执行SQL时的预编译参数。根据该参数的类型不同，SQL中使用`?`或者`#{}`标记预编译参数；
- class：必填，需要转换的结果集类型。

例1：执行SQL，并获取结果

```
Student stu = DB.get("select * from t_student where class='3年1班' and num=1", Student.class);
```

例2：执行带有预编译参数的SQL，当parameter参数为`Object数组`、`org.rex.db.Ps`时，SQL中使用`?`标记预编译参数，例如：

```
Student stu = DB.get("select * from t_student where class=? and num=?", new Object[]{"3年1班", 1}, Student.class);
Student stu = DB.get("select * from t_student where class=? and num=?", new Ps("3年1班", 1), Student.class);
```

例3：执行带有预编译参数的SQL，当parameter参数为`java.util.Map`、`Java对象`时，SQL中使用`#{}`标记预编译参数，例如：

```
//map为java.util.Map类型的实例，包含名为“clazz”和“num”的键；obj为普通的java对象，包含名为“clazz”和“num”的成员变量
Student stu = DB.get("select * from t_student where class=#{clazz} and num=#{num}", map, Student.class);
Student stu = DB.get("select * from t_student where class=#{clazz} and num=#{num}", obj, Student.class);
```

例4：在指定数据源中执行SQL

```
//配置文件rexdb.xml中有id为student的数据源
Student stu = DB.get("student", "select * from t_student where class='3年1班' and num=1", Student.class);
```

查询单条Map记录 *DB.getMap()*

`org.rex.DB.get()`方法用于查询单条记录，并返回一个org.rex.RMap实例（无记录时返回null），org.rex.RMap是java.util.HashMap的子类，提供了数据类型转换等功能。格式如下：

> RMap DB.getMap([String dataSourceId,] String sql, [Object[] | Ps | Map | Object parameter])

- dataSourceId：可选，配置文件中的数据源id，不设置时使用默认数据源；
- sql：必填，待执行的SQL语句；
- parameter：可选，执行SQL时的预编译参数。根据该参数的类型不同，SQL中使用`?`或者`#{}`标记预编译参数。

例1：执行SQL，并获取结果

```
RMap stu = DB.getMap("select * from t_student where class='3年1班' and num=1");
```

例2：执行带有预编译参数的SQL，当parameter参数为`Object数组`、`org.rex.db.Ps`时，SQL中使用`?`标记预编译参数

```
RMap stu = DB.get("select * from t_student where class=? and num=?", new Object[]{"3年1班", 1});
RMap stu = DB.get("select * from t_student where class=? and num=?", new Ps("3年1班", 1));
```

例3：执行带有预编译参数的SQL，当parameter参数为`java.util.Map`、`Java对象`时，SQL中使用`#{}`标记预编译参数

```
//map为java.util.Map类型的实例，包含名为“class”和“num”的键；obj为普通的java对象，包含名为“clazz”和“num”的成员变量
RMap stu = DB.get("select * from t_student where class=#{clazz} and num=#{num}", map);
RMap stu = DB.get("select * from t_student where class=#{clazz} and num=#{num}", obj);
```

例4：在指定数据源中执行SQL

```
//配置文件rexdb.xml中有id为student的数据源
RMap stu = DB.get("student", "select * from t_student where class='3年1班' and num=1");
```

查询多条记录 *DB.getList()*

`org.rex.DB.getList()`方法用于查询多条记录，并返回一个java.util.List实例（无记录时返回空的List实例）。格式如下：

> List DB.getList([String dataSourceId,] String sql, [Object[] | Ps | Map | Object parameter,] Class clazz [, int offset, int rows])

- dataSourceId：可选，配置文件中的数据源id，不设置时使用默认数据源；
- sql：必填，待执行的SQL语句；
- parameter：可选，执行SQL时的预编译参数。根据该参数的类型不同，SQL中使用`?`或者`#{}`标记预编译参数。
- class：必填，需要转换的结果集类型；
- offset：可选，分页查询的起始行号；
- rows：可选，分页查询待获取的结果集条目。

例1：执行SQL，并获取结果

```
List<Student> list = DB.getList("select * from t_student where class='3年1班'", Student.class);
```

例2：执行带有预编译参数的SQL，当parameter参数为`Object数组`、`org.rex.db.Ps`时，SQL中使用`?`标记预编译参数

```
List<Student> list = DB.getList("select * from t_student where class=?", new Object[]{"3年1班"}, Student.class);
List<Student> list = DB.getList("select * from t_student where class=?", new Ps("3年1班"), Student.class);
```

例3：执行带有预编译参数的SQL，当parameter参数为`java.util.Map`、`Java对象`时，SQL中使用`#{}`标记预编译参数

```
//map为java.util.Map类型的实例，包含名为“class”的键；obj为普通的java对象，包含名为“clazz”的成员变量
List<Student> list = DB.getList("select * from t_student where class=#{clazz}", map, Student.class);
List<Student> list = DB.getList("select * from t_student where class=#{clazz}", obj, Student.class);
```

例4：执行分页查询，查询第100～110条记录

```
List<Student> list = DB.getList("select * from t_student where class='3年1班'", Student.class, 100, 10);
```

例5：在指定数据源中执行SQL

```
//配置文件rexdb.xml中有id为student的数据源
List<Student> list = DB.getList("student", "select * from t_student where class='3年1班'", Student.class);
```

查询多条Map记录 *DB.getMapList()*

`org.rex.DB.getMapList()`方法用于查询多条记录，并返回一个java.util.List实例（无记录时返回空的List实例）。格式如下：

> List DB.getList([String dataSourceId,] String sql, [Object[] | Ps | Map | Object parameter] [, int offset, int rows])

- dataSourceId：可选，配置文件中的数据源id，不设置时使用默认数据源；
- sql：必填，待执行的SQL语句；
- parameter：可选，执行SQL时的预编译参数。根据该参数的类型不同，SQL中使用`?`或者`#{}`标记预编译参数。
- offset：可选，分页查询的起始行号；
- rows：可选，分页查询待获取的结果集条目。

例1：执行SQL，并获取结果

```
List<RMap> list = DB.getMapList("select * from t_student where class='3年1班'");
```

例2：执行带有预编译参数的SQL，当parameter参数为`Object数组`、`org.rex.db.Ps`时，SQL中使用`?`标记预编译参数

```
List<RMap> list = DB.getMapList("select * from t_student where class=?", new Object[]{"3年1班"});
List<RMap> list = DB.getMapList("select * from t_student where class=?", new Ps("3年1班"));
```

例3：执行带有预编译参数的SQL，当parameter参数为`java.util.Map`、`Java对象`时，SQL中使用`#{}`标记预编译参数

```
//map为java.util.Map类型的实例，包含名为“class”的键；obj为普通的java对象，包含名为“clazz”的成员变量
```

```
List<RMap> list = DB.getMapList("select * from t_student where class=#{clazz}", map);
List<RMap> list = DB.getMapList("select * from t_student where class=#{clazz}", obj);
```

例4：执行分页查询，查询第100～110条记录

```
List<RMap> list = DB.getMapList("select * from t_student where class='3年1班'", 100, 10);
```

例5：在指定数据源中执行SQL

```
//配置文件rexdb.xml中有id为student的数据源
List<RMap> list = DB.getMapList("student", "select * from t_student where class='3年1班'");
```

插入/更新/删除 *DB.update()*

`org.rex.DB.update()`方法用于执行插入/更新/删除操作，该接口将返回受影响的记录条数。格式如下：

> int DB.update([String dataSourceId,] String sql [, Object[] | Ps | Map | Object parameter])

例1：执行SQL

```
DB.update("delete from t_student where num = 1");
```

例2：执行带有预编译参数的SQL，当parameter参数为`Object数组`、`org.rex.db.Ps`时，SQL中使用`?`标记预编译参数

```
string sql = "insert into t_student(num, student_name, student_class,create_time) values (?, ?, ?, ?)";
DB.update(sql, new Object[]{1, "钟小强","3年1班", new Date()});
DB.update(sql, new Ps(2, "王小五", "3年1班",new Date()));
```

例3：执行带有预编译参数的SQL，当parameter参数为`java.util.Map`、`Java对象`时，SQL中使用`#{}`标记预编译参数

```
String sql = "update t_student set student_name = #{studentName} where num = #{num}";
DB.update(sql,map);//map为java.util.Map类型的实例，包含名为“studentName”和“num”的键
DB.update(sql,new Students(1, "钟小强", null, null));//obj为普通的java对象，包含名为“studentName”和“num”的成员变量
```

例4：在指定数据源中执行SQL

```
//配置文件rexdb.xml中有id为student的数据源
List<RMap> list = DB.getMapList("student", "delete from t_student where num = 1");
```

批量处理 *DB.batchUpdate()*

`DB.batchUpdate()`方法用于执行批处理操作，该接口可以有效提升执行大量数据变更时的执行性能，格式如下：

> int[] DB.batchUpdate([String datasource,] String[] sqls)

例1：执行多个SQL

```
String[] sqls = new String[]{"delete from t_student where num=1", "delete from t_student where num=2"};
DB.batchUpdate(sqls);
```

例2：执行带有预编译参数的SQL，当parameter参数元素类型为`Object数组`、`org.rex.db.Ps`时，SQL中使用`?`标记预编译参数

```
string sql = "insert into t_student(num, student_name, student_class,create_time) values (?, ?, ?, ?)";
DB.batchUpdate(sql, new Object[][]{{1, "钟小强","3年1班", new Date()}, {2, "王小五","3年1班", new Date()}});
DB.batchUpdate(sql, new Ps[]{new Ps(3, "李小华", "3年1班", new Date()), new Ps(4, "赵小明", "3年1班", new Date())});
```

例3：执行带有预编译参数的SQL，当parameter参数元素类型为`java.util.Map`、`Java对象`时，SQL中使用`#{}`标记预编译参数

```
String sql = "update t_student set student_name = #{studentName} where num = #{num}";
DB.batchUpdate(sql, maps);//maps为java.util.Map数组实例，数组中每个元素都包含名为“studentName”和“num”的键
DB.batchUpdate(sql, objs);//objs为Student类型的java对象实例数组，Student对象包含名为“studentName”和“num”的成员变量
```

例4：在指定数据源中执行SQL

```
String[] sqls = new String[]{"delete from t_student where num=1", "delete from t_student where num=2"};
DB.batchUpdate("student", sqls);
```

事务

Rexdb使用编程的方式处理事务，以下接口用于事务处理：

> void DB.beginTransaction([String dataSourceId] [,DefaultDefinition definition]) //开启事物
> void DB.commit([String dataSourceId]) //提交事务
> void DB.rollback([String dataSourceId]) //回滚事务

JTA事物接口如下：

> void DB.beginJta([DefaultDefinition definition]) //开启JTA事物
> void DB.commitJta() //提交JTA事务
> void DB.rollbackJta() //回滚JTA事务

例：

```
DB.beginTransaction();
try{
    DB.update("delete from t_student where num = 1");
    DB.update("delete from t_student where num = 2");
    DB.commit();
}catch(Exception e){
    DB.rollback();
}
```

调用

`DB.call()`方法用于执行调用操作，可用于调用存储过程和函数，支持返输入、输出参数和返回值。格式如下：

> RMap DB.call([String dataSourceId,] String sql [, Object[] | Ps | Map | Object parameter])

例1：调用存储过程/函数

```
DB.call("test_proc()");
```

例2：调用有输入参数的存储过程/函数

```
DB.call("{call test_proc_in(?)", new Ps(200));
```

例3：调用有输出参数的存储过程/函数时，必须使用org.rex.db.Ps对象声明输出参数

```
Ps ps = new Ps();
ps.addOutInt("age");
RMap result = DB.call("{call test_proc_out(?)}", ps);
int age = result.getInt("age")
```

例4：调用同时有输入输出参数的存储过程/函数，必须使用org.rex.db.Ps对象，并按照SQL中标记的顺序声明

```
Ps ps = new Ps();
ps.add(200);
ps.addOutInt("major");
RMap result = DB.call("{call test_proc_in_out(?, ?)}", ps);
int major = result.getInt("major");
```

例5：调用即是输入参数也是输出参数的存储过程/函数，必须使用org.rex.db.Ps对象

```
Ps ps = new Ps();
ps.addInOut("count", 10);
RMap result = DB.call("{call test_proc_inout(?)}", ps);
int count = result.getInt("count");
```

例6：调用带有返回值的存储过程/函数，返回值将按照return_1、return_2的顺序命名

```
RMap result = DB.call("{call exdb_test_proc_return()}");
List<RMap> return1 = result.getList("return_1");
```

关于调用的其它用法请参见用户手册。

更多

Rexdb还有更多功能，例如：

- 设置异常信息为中文/英文；
- 开启/关闭日志；
- 执行SQL前的语法检查；
- 自动检查连接/状态中的警告；
- 设置查询超时时间；
- 设置事物超时时间/隔离级别/自动回滚/自动的批处理事务；
- 启动动态字节码编译/反射缓存；
- 自动转换日期类型的参数；

详情请参见用户手册。