JDBC 系列课程

时间: 2017/11/22
作者:袁毅雄
邮箱: 896778954@qq.com

目录

数据库访问技术 JDBC

什么是 JDBC

JDBC（Java DataBase Connectivity,java 数据库连接）是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。JDBC 提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序

JDBC 的定义

有了 JDBC，向各种关系数据发送 SQL 语句就是一件很容易的事。换言之，有了 JDBC API，就不必为访问 Sybase 数据库专门写一个程序，为访问 Oracle 数据库又专门写一个程序，或为访问 Informix 数据库又编写另一个程序等等，程序员只需用 JDBC API 写一个程序就够了，

它可向相应数据库发送 SQL 调用。同时，将 Java 语言和 JDBC 结合起来使程序员不必为不同的平台编写不同的应用程序，只须写一遍程序就可以让它在任何平台上运行，这也是 Java 语言“编写一次，处处运行”的优势。

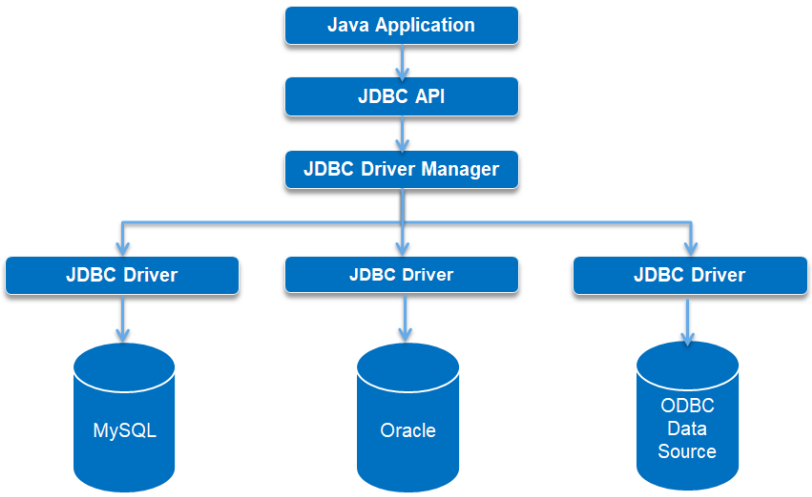
Java 数据库连接体系结构是用于 Java 应用程序连接数据库的标准方法。JDBC 对 Java 程序员而言是 API，对实现与数据库连接的服务提供商而言是接口模型。作为 API，JDBC 为程序开发提供标准的接口，并为数据库厂商及第三方中间件厂商实现与数据库的连接提供了标准方法。JDBC 使用已有的 SQL 标准并支持与其它数据库连接标准，如 ODBC 之间的桥接。JDBC 实现了所有这些面向标准的目标并且具有简单、严格类型定义且高性能实现的接口。

Java 具有坚固、安全、易于使用、易于理解和可从网络上自动下载等特性，是编写数据库应用程序的杰出语言。所需要的只是 Java 应用程序与各种不同数据库之间进行对话的方法。而 JDBC 正是作为此种用途的机制。

JDBC 扩展了 Java 的功能。例如，用 Java 和 JDBC API 可以发布含有 applet 的网页，而该 applet 使用的信息可能来自远程数据库。企业也可以用 JDBC 通过 Intranet 将所有职员连到一个或多个内部数据库中（即使这些职员所用的计算机有 Windows、 Macintosh 和 UNIX 等各种不同的操作系统）。随着越来越多的程序员开始使用 Java 编程语言，对从 Java 中便捷地访问数据库的要求也在日益增加。

MIS 管理员们都喜欢 Java 和 JDBC 的结合，因为它使信息传播变得容易和经济。企业可继续使用它们安装好的数据库，并能便捷地存取信息，即使这些信息是储存在不同数据库管理系统上。新程序的开发期很短。安装和版本控制将大为简化。程序员可只编写一遍应用程序或只更新一次，然后将它放到服务器上，随后任何人就都可得到最新版本的应用程序。对于商务上的销售信息服务， Java 和 JDBC 可为外部客户提供获取信息更新的更好方法。

JDBC 执行流程图



JDBC 六步走

```
// 第一步:导入驱动
// 第二步:构建数据库连接
// 第三步:预编译SQL
// 第四步:执行SQL
// 第五步:处理结果
// 第六步:释放资源
```

JDBC CURD

Reade

```
        // 第一步:导入驱动
        Class.forName("com.mysql.jdbc.Driver");

        // 第二步:构建数据库连接
        String url = "jdbc:mysql://127.0.0.1:3306/test?characterEncoding=UTF-8";
        String user = "root";
        String password = "root";
        Connection connection = DriverManager.getConnection(url, user, password);

        // 第三步:预编译SQL
        String sql = "select gradeid,gradename from grade";
        PreparedStatement ps = connection.prepareStatement(sql);

        // 第四步:执行SQL
        ResultSet resultSet = ps.executeQuery();

        // 第五步:处理结果
        while (resultSet.next()) {
            int gradeid = resultSet.getInt("gradeid");
            String gradename = resultSet.getString("gradename");

            System.out.println(gradeid + " " + gradename);
```

```
}

// 第六步:释放资源
resultSet.close();
ps.close();
connection.close();
```

Create

```
// 第一步:导入驱动
Class.forName("com.mysql.jdbc.Driver");

// 第二步:构建数据库连接
String url = "jdbc:mysql://127.0.0.1:3306/test?characterEncoding=UTF-8";
String user = "root";
String password = "root";
Connection connection = DriverManager.getConnection(url, user, password);

// 第三步:预编译SQL
String sql = "insert into grade(gradeid,gradename)values(?,?)";
PreparedStatement ps = connection.prepareStatement(sql);

ps.setInt(1, 20);
ps.setString(2,"java4");

// 第四步:执行SQL
int row= ps.executeUpdate();

// 第五步:处理结果
System.out.println(row>0?"添加成功":"添加失败");

// 第六步:释放资源
ps.close();
connection.close();
```

Update

```
// 第一步:导入驱动
Class.forName("com.mysql.jdbc.Driver");
```

```
// 第二步:构建数据库连接
String url = "jdbc:mysql://127.0.0.1:3306/test?characterEncoding=UTF-8";
String user = "root";
String password = "root";
Connection connection = DriverManager.getConnection(url, user, password);

// 第三步:预编译SQL
String sql = "update grade set gradename=? where gradeid=?";
PreparedStatement ps = connection.prepareStatement(sql);

ps.setString(1,"java5");
ps.setInt(2, 20);

// 第四步:执行SQL
int row= ps.executeUpdate();

// 第五步:处理结果
System.out.println(row>0?"修改成功":"修改失败");

// 第六步:释放资源
ps.close();
connection.close();
```

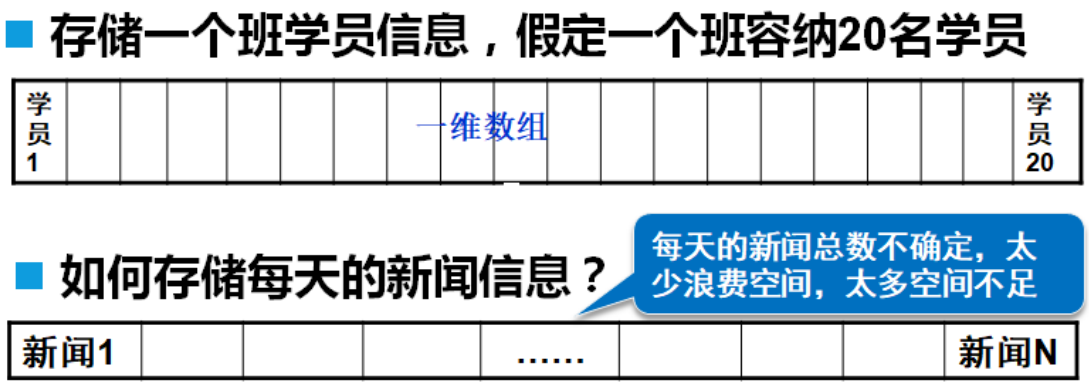
Delete

```
// 第一步:导入驱动
Class.forName("com.mysql.jdbc.Driver");

// 第二步:构建数据库连接
String url = "jdbc:mysql://127.0.0.1:3306/test?characterEncoding=UTF-8";
String user = "root";
String password = "root";
Connection connection = DriverManager.getConnection(url, user, password);

// 第三步:预编译SQL
String sql = "delete from grade where gradeid=?";
PreparedStatement ps = connection.prepareStatement(sql);
ps.setInt(1, 20);

// 第四步:执行SQL
int row= ps.executeUpdate();
```

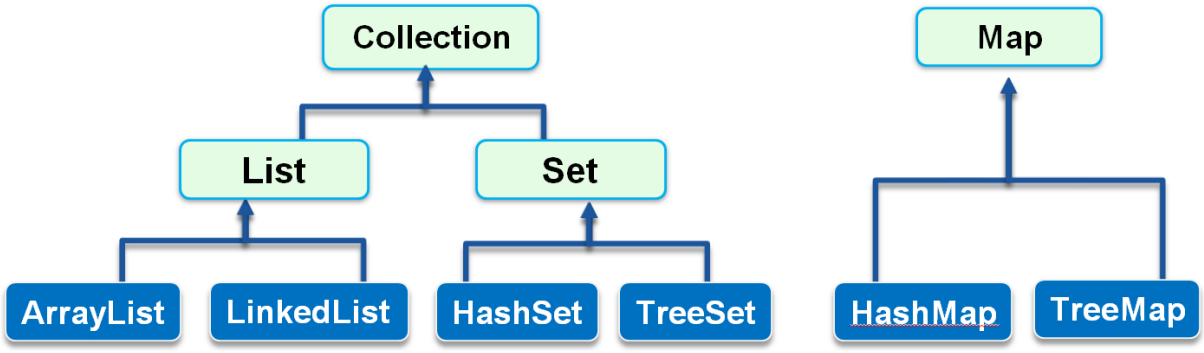
```
        // 第五步:处理结果
        System.out.println(row>0?"删除成功":"删除失败");

        // 第六步:释放资源
        ps.close();
        connection.close();
```

集合框架

为什么使用集合



集合的结构体系



Collection

特点:无序,不唯一

List

特点:有序,不唯一

ArrayList

特点：有序,不唯一,遍历速度比较快

```
ArrayList arrayList=new ArrayList();
//List arrayList=new ArrayList();

 arrayList.add("java"); //0
 arrayList.add("c");       //1
 arrayList.add("c++");    //2
 arrayList.add("java"); //3

 for (int i = 0; i < arrayList.size(); i++) {
        System.out.println(arrayList.get(i));
 }
```

LinkedList

```
  //LinkedList linkedList=new LinkedList();
List=new LinkedList();

linkedList.add("java");
linkedList.add("c");
linkedList.add("c++");

//linkedList.addFirst("c");
//linkedList.addLast("php");

//linkedList.removeLast();
```

```
        //linkedList.removeFirst();

        for (int i = 0; i < linkedList.size(); i++) {
            System.out.println(linkedList.get(i));
        }
```

特点：有序,不唯一,修改数据比较快

Set

特点:无序,唯一

HashSet 可扩展唯一性

```
public class TestSet01 {
    public static void main(String[] args) {
        Set set = new HashSet();
        set.add(new Text("哈哈", 1));
        set.add(new Text("哈哈", 1));
        set.add(new Text("呵呵", 1));
        set.add(new Text("嘻嘻", 1));
        set.add(new Text("你好", 5));
        set.add(new Text("不好", 6));

        Iterator iterator = set.iterator();
        while (iterator.hasNext()) {
            Object object = iterator.next();
            System.out.println(object.toString());
        }
    }
}
```

```
class Text {
    private String content;
    private int index;

    public Text(String content, int index) {
        super();
        this.content = content;
        this.index = index;
```

```
    }

    @Override
    public String toString() {
        return "Text [content=" + content + ", index=" + index + "]";
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + index;
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Text other = (Text) obj;
        if (other.getIndex() != this.getIndex())
            return false;

        return true;
    }

}
```

TreeSet 可以扩展排序功能

```
public class TestSet01 {
    public static void main(String[] args) {
        Set set = new TreeSet();
        set.add(new Text("哈哈",5));
        set.add(new Text("呵呵",2));
        set.add(new Text("嘻嘻",3));
        set.add(new Text("你好",4));
        set.add(new Text("不好",1));
```

```
            Iterator iterator = set.iterator();
            while (iterator.hasNext()) {
                Object object = iterator.next();
                System.out.println(object.toString());
            }

        }
}
```

```
class Text implements Comparable {
    private String content;
    private int index;

    @Override
    public int compareTo(Object o) {
        if (o instanceof Text) {
            Text temp = (Text) o;

            if(this.getIndex()>temp.getIndex()){
                return 1;
            }else if(this.getIndex()<temp.getIndex()){
                return -1;
            }else{
                return 0; //相等
            }
        }
        return 0;
    }

    @Override
    public String toString() {
        return "Text [content=" + content + ", index=" + index + "]";
    }
}
```

```
        Map map = new HashMap();
        map.put("1001", "旺旺雪饼");
        map.put("1002", "绿茶");
        map.put("1003", "红茶");
        map.put("1004", "雪花啤酒");
        map.put("1005", "王老吉");

        Set set = map.entrySet();
```

```
Iterator iterator = set.iterator();
while (iterator.hasNext()) {
    Map.Entry row = (Map.Entry) iterator.next();
    System.out.println(row.getKey() + ":" + row.getValue());
}
```

Map

HashMap

```
Map map = new HashMap();

map.put("1001", "旺旺雪饼");
map.put("1002", null);
map.put("1003", "红茶");
map.put("1004", null);
map.put("1005", "王老吉");
map.put(null, "王老吉");

System.out.println(map.get("1003"));

Set keys = map.keySet();
System.out.println(keys);

Collection collection = map.values();
System.out.println(collection);

Iterator iterator = map.keySet().iterator();

while (iterator.hasNext()) {
    Object key = iterator.next();
    Object value = map.get(key);

    System.out.println(key + ":" + value);
}
```

泛型

什么是泛型

泛型是程序设计语言的一种特性。允许程序员在强类型程序设计语言中编写代码时定义一些可变部分，那些部分在使用前必须作出指明。各种程序设计语言和其编译器、运行环境对泛型的支持均不一样。将类型参数化以达到代码复用提高软件开发工作效率的一种数据类型。泛型类是引用类型，是堆对象，主要是引入了类型参数这个概念。

泛型的分类

泛型的定义主要有以下两种：

1.在程序编码中一些包含类型参数的类型，也就是说泛型的参数只可以代表类，不能代表个别对象。（这是当今较常见的定义）

2.在程序编码中一些包含参数的类。其参数可以代表类或对象等等。（人们大多把这称作模板）不论使用哪个定义，泛型的参数在真正使用泛型时都必须作出指明。

一些强类型编程语言支持泛型，其主要目的是加强类型安全及减少类转换的次数，但一些支持泛型的编程语言只能达到部分目的。

泛型类

```
class Demo<T>{}
class Demo<A,B>{}
class Demo<T extends Object> {}
class Demo<A extends Object, B extends Object> {}


Demo<Integer> demo = new Demo<Integer>();
Demo<A1> demo = new Demo<A1>();
```

泛型方法/泛型参数/泛型的返回值/泛型的成员变量

```
class Demo<T> {
        T obj;
        public T get(T t){
                return t;
        }
}
```

泛型的集合

List/ArrayList/LinkedList

```
List<News> list = new ArrayList<News>();
```

Set/HashSet/TreeSet

```
Set<News> set=new HashSet<News>();
```

Map/HashMap

```
Map<String, News> map=new HashMap<String, News>();
```

Collections 集合的操作工具

```
Collections.sort(list);//排序
Collections.binarySearch(list, 10); //找存在与否
Collections.min(list); //找最小
Collections.max(list); //找最大
```

Iterator 集合的迭代器

迭代 List

```
Iterator iterator = list.iterator();
while (iterator.hasNext()) {
    Object object = iterator.next();
    System.out.println(object.toString());
}
```

迭代 Set

```
Iterator iterator = set.iterator();
 while (iterator.hasNext()) {
        Object object = iterator.next();
        System.out.println(object.toString());
 }
```

迭代 Map

```
Iterator iterator = map.keySet().iterator();

 while (iterator.hasNext()) {
        Object key = iterator.next();
        Object value = map.get(key);

        System.out.println(key + ":" + value);
 }
```

foreach 优化集合的迭代

迭代 List

```
List<Integer> list = new ArrayList<Integer>();
for (Integer i : list) {
        System.out.println(i);
}
```

迭代 Set

```
Set<Integer> set = new HashSet<Integer>();
for (Integer i : set) {
        System.out.println(i);
}
```

迭代 Map

```
Map<String, Integer> map=new HashMap<String, Integer> ();
for (String key : map.keySet()) {
    System.out.println(key+":"+map.get(key));
}
```

高级 JDBC

什么是 DbUtils

Commons DbUtils 是 Apache 组织提供的一个对 JDBC 进行简单封装的开源工具类库，使用它能够简化 JDBC 应用程序的开发，同时也不会影响程序的性能。

DbUtiles 特征

DBUtils 是 java 编程中的数据库操作实用工具，小巧简单实用，
1.对于数据表的读操作，他可以把结果转换成 List，Array，Set 等 java 集合，便于程序员操作；
2.对于数据表的写操作，也变得很简单（只需写 sql 语句）
3.可以使用数据源，使用 JNDI，数据库连接池等技术来优化性能--重用已经构建好的数据库连接对象，而不像 php，asp 那样，费时费力的不断重复的构建和析构这样的对象。

DbUtiles 两步走

前置条件

Jar	来源	作用
mysql-connector-java-5.0.8-bin.jar	Mysql	数据库连接
c3p0-0.9.1.2.jar	Jbosss	连接池
commons-dbutils-1.4.jar	apache DbUtils	数据库 CURD
commons-beanutils-1.8.3.jar	apache DbUtils	实体映射
commons-logging-1.1.1.jar	apache DbUtils	日志

数据源

什么是数据源

数据源是指数据库应用程序所使用的数据库或者数据库服务器。

数据源（Data Source）顾名思义，数据的来源，是提供某种所需要数据的器件或原始媒体。在数据源中存储了所有建立数据库连接的信息。就像通过指定文件名称可以在文件系统中找到文件一样，通过提供正确的数据源名称，你可以找到相应的数据库连接。

常见的数据源分类

数据源名称	所属机构
DBCP	Apache
C3P0	JBoss Hibernate
ODBC	Microsoft
Druid	Alibaba

C3P0 数据源基础配置

1.最常用配置

initialPoolSize:连接池初始化时创建的连接数,default : 3（建议使用）

minPoolSize:连接池保持的最小连接数,default : 3（建议使用）

maxPoolSize:连接池中拥有的最大连接数，如果获得新连接时会使连接总数超过这个值则不会再获取新连接，而是等待其他连接释放，所以这个值有可能会设计地很大,default : 15（建议使用）

acquireIncrement:连接池在无空闲连接可用时一次性创建的新数据库连接数,default : 3（建议使用）

2.管理连接池的大小和连接的生存时间

maxConnectionAge:配置连接的生存时间，超过这个时间的连接将由连接池自动断开丢弃掉。当然正在使用的连接不会马上断开，而是等待它 close 再断开。配置为 0 的时候则不会对连接的生存时间进行限制。default : 0 单位 s（不建议使用）

maxIdleTime:连接的最大空闲时间，如果超过这个时间，某个数据库连接还没有被使用，则会断开掉这个连接。如果为 0，则永远不会断开连接,即回收此连接。default : 0 单位 s（建议使用）

maxIdleTimeExcessConnections:这个配置主要是为了快速减轻连接池的负载，比如连接池中连接数因为某次数据访问高峰导致创建了很多数据连接，但是后面的时间段需要的数据库连接数很少，需要快速释放，必须小于 maxIdleTime。其实这个没必要配置，maxIdleTime 已经配置了。default : 0 单位 s（不建议使用）

3.配置连接测试：

automaticTestTable:配置一个表名，连接池根据这个表名用自己的测试 sql 语句在这个空表上测试数据库连接,这个表只能由 c3p0 来使用，用户不能操作。default : null（不建议使用）

preferredTestQuery:与上面的 automaticTestTable 二者只能选一。自己实现一条 SQL 检测语句。default : null（建议使用）

idleConnectionTestPeriod:用来配置测试空闲连接的间隔时间。测试方式还是上面的两种之一，可以用来解决 MySQL8 小时断开连接的问题。因为它保证连接池会每隔一定时间对空闲连接进行一次测试，从而保证有效的空闲连接能每隔一定时间访问一次数据库，将于 MySQL8 小时无会话的状态打破。为 0 则不测试。default : 0(建议使用)

testConnectionOnCheckin:如果为 true，则在 close 的时候测试连接的有效性。default : false（不建议使用）

testConnectionOnCheckout:性能消耗大。如果为 true，在每次 getConnection 的时候都会测试，为了提高性能,尽量不要用。default : false（不建议使用）

4.配置 PreparedStatement 缓存：

maxStatements:连接池为数据源缓存的 PreparedStatement 的总数。由于 PreparedStatement 属于单个 Connection,所以这个数量应该根据应用中平均连接数乘以每个连接的平均 PreparedStatement 来计算。同时 maxStatementsPerConnection 的配置无效。default : 0（不建议使用）

maxStatementsPerConnection:连接池为数据源单个 Connection 缓存的 PreparedStatement 数，这个配置比 maxStatements 更有意义，因为它缓存的服务对象是单个数据连接，如果设置的好，肯定是可以提高性能的。为 0 的时候不缓存。default : 0（看情况而论）

5.重连相关配置

acquireRetryAttempts:连接池在获得新连接失败时重试的次数，如果小于等于 0 则无限重试直至连接获得成功。default : 30（建议使用）

acquireRetryDelay:连接池在获得新连接时的间隔时间。default : 1000 单位 ms（建议使用）
breakAfterAcquireFailure:如果为 true，则当连接获取失败时自动关闭数据源，除非重新启动应用程序。所以一般不用。default : false（不建议使用）
checkoutTimeout:配置当连接池所有连接用完时应用程序 getConnection 的等待时间。为 0 则无限等待直至有其他连接释放或者创建新的连接，不为 0 则当时间到的时候如果仍没有获得连接，则会抛出 SQLException。其实就是 acquireRetryAttempts*acquireRetryDelay。default : 0（与上面两个，有重复，选择其中两个都行）

6.定制管理 Connection 的生命周期

connectionCustomizerClassName:用来定制 Connection 的管理，比如在 Connection acquire 的时候设定 Connection 的隔离级别，或者在 Connection 丢弃的时候进行资源关闭，
就可以通过继承一个 AbstractConnectionCustomizer 来实现相关方法，配置的时候使用全类名。有点类似监听器的作用。default : null（不建议使用）

7.配置未提交的事务处理

autoCommitOnClose:连接池在回收数据库连接时是否自动提交事务。如果为 false，则会回滚未提交的事务，如果为 true，则会自动提交事务。default : false（不建议使用）
forceIgnoreUnresolvedTransactions：这个配置强烈不建议为 true。default : false（不建议使用）
一般来说事务当然由自己关闭了，为什么要让连接池来处理这种不细心问题呢？

8.配置 debug 和回收 Connection

unreturnedConnectionTimeout:为 0 的时候要求所有的 Connection 在应用程序中必须关闭。如果不为 0，则强制在设定的时间到达后回收 Connection，所以必须小心设置，保证在回收之前所有数据库操作都能够完成。这种限制减少 Connection 未关闭情况的不是很适用。建议手动关闭。default : 0 单位 s（不建议使用）
debugUnreturnedConnectionStackTraces:如果为 true 并且 unreturnedConnectionTimeout 设为大于 0 的值，当所有被 getConnection 出去的连接 unreturnedConnectionTimeout 时间到的时候，就会打印出堆栈信息。只能在 debug 模式下适用，因为打印堆栈信息会减慢 getConnection 的速度 default : false（不建议使用）

C3P0 配置模板

整合其他框架属性文件配置 c3p0.properties

```
#驱动
c3p0.driverClass=com.mysql.jdbc.Driver
#地址
c3p0.jdbcUrl=jdbc:mysql://localhost:3306/jdbc
#用户名
c3p0.user=root
#密码
c3p0.password=lovejava
#-------------------------------
#连接池初始化时创建的连接数
c3p0.initialPoolSize=3
#连接池保持的最小连接数
c3p0.minPoolSize=3
#连接池在无空闲连接可用时一次性创建的新数据库连接数,default:3
c3p0.acquireIncrement=3
#连接池中拥有的最大连接数，如果获得新连接时会使连接总数超过这个值则不会再获取新连接，而是等待其他连接释放，所以这个值有可能会设计地很大,default : 15
c3p0.maxPoolSize=15
#连接的最大空闲时间，如果超过这个时间，某个数据库连接还没有被使用，则会断开掉这个连接,单位秒
c3p0.maxIdleTime=100
#连接池在获得新连接失败时重试的次数，如果小于等于 0 则无限重试直至连接获得成功
c3p0.acquireRetryAttempts=30
#连接池在获得新连接时的间隔时间
c3p0.acquireRetryDelay=1000
```

单独使用自定义数据源 XML 文件配置 c3p0-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<c3p0-config>
    <default-config>
        <property name="user">root</property>
        <property name="password">root</property>
        <property name="driverClass">com.mysql.jdbc.Driver</property>
        <property name="jdbcUrl">jdbc:mysql://localhost:3306/news</property>
    </default-config>
</c3p0-config>
```

使用 DbUtile 实现快速开发 CURD 操作

Create

```
DataSource ds = new ComboPooledDataSource();
QueryRunner ruanner = new QueryRunner(ds);
public void save() throws SQLException {
        int row=ruanner.update("insert into student(studentno,phone,address,email)values(?,?,?,?)",3000,"11","11","11");
        System.out.println(row>0?"添加成功":"添加失败");
}
```

Update

```
DataSource ds = new ComboPooledDataSource();
QueryRunner ruanner = new QueryRunner(ds);
public void edit() throws SQLException {
        int row=ruanner.update("update student set studentname=?,sex=? where studentNo=?", "arvin",2,1000);
        System.out.println(row>0?"修改成功":"修改失败");
}
```

Delete

```
DataSource ds = new ComboPooledDataSource();
QueryRunner ruanner = new QueryRunner(ds);
public void remove() throws SQLException {
        int row=ruanner.update("delete from student where studentno=?", 3000);
        System.out.println(row>0?"删除成功":"删除失败");
}
```

Reade

ReadeObject

```
DataSource ds = new ComboPooledDataSource();
QueryRunner ruanner = new QueryRunner(ds);
public void findObjectById() throws SQLException {
        Student stu = ruanner.query("select * from student where studentNo=?", new BeanHandler<>(Student.class),
```

```
1000);
            System.out.println(stu.toString());
    }
```

ReadeObjects

```
    DataSource ds = new ComboPooledDataSource();
    QueryRunner ruanner = new QueryRunner(ds);
    public void findAll() throws SQLException {
        List<Student> list = ruanner.query("select * from student", new BeanListHandler<>(Student.class));
        for (Student student : list) {
            System.out.println(student.toString());
        }
    }
```