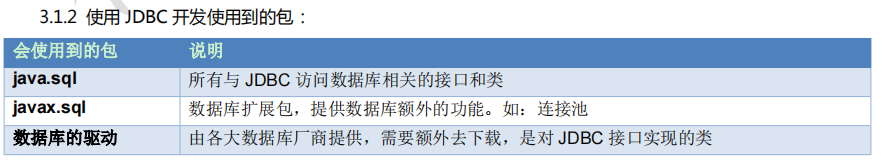
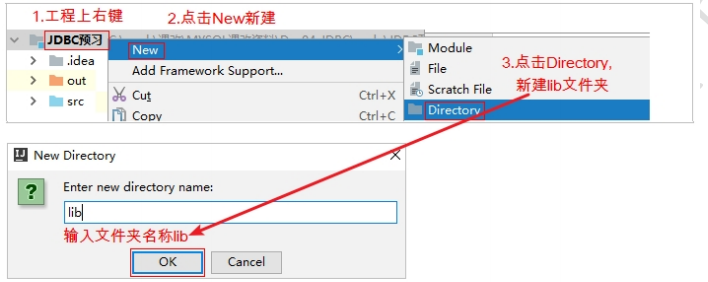
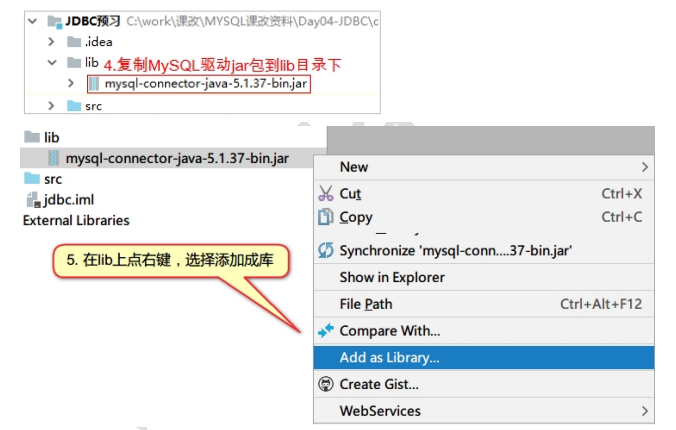
# JDBC jar包

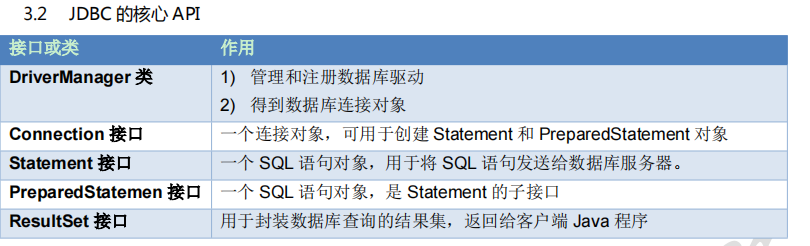


## 驱动jar包的导入





# JDBC核心API



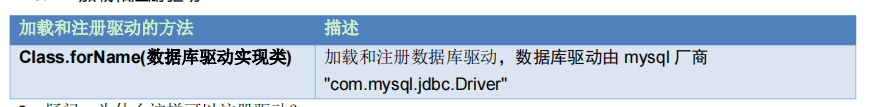
## (1)DriverManager类

DriverManager 作用：

管理和注册驱动

创建数据库的连接

### 加载和注册驱动



疑问：为什么这样可以注册驱动？

|  |
| --- |
| **public class** Demo1 {  **public static void** main(String[] args) **throws** ClassNotFoundException {**4** / **21**  *//抛出类找不到的异常，注册数据库驱动*  Class.*forName*(**"com.mysql.jdbc.Driver"**);  }  } |

 com.mysql.jdbc.Driver 源代码：

// Driver 接口，所有数据库厂商必须实现的接口，表示这是一个驱动类。

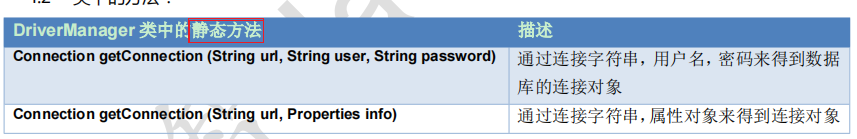
|  |
| --- |
| **public class** Driver **implements** java.sql.Driver {  **public** Driver() **throws** SQLException {  }  **static** {  **try** {  DriverManager.registerDriver(**new** Driver()); //注册数据库驱动  } **catch** (SQLException var1) {  **throw new** RuntimeException(**"Can't register driver!"**);  }  }  } |

说明：Class.forName() 加载某个类会触发其初始化，初始化会执行类静态代码块。

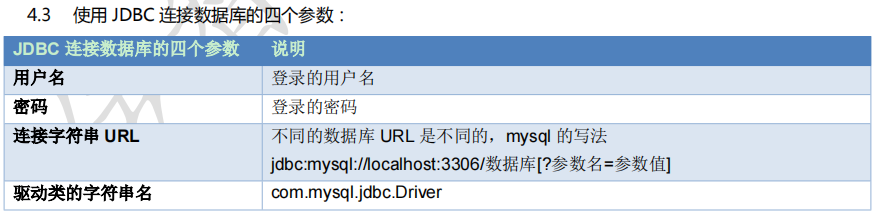
注：从 JDBC3 开始，目前已经普遍使用的版本。可以不用注册驱动而直接使用。Class.forName 这句话可以省略

### 获取数据库连接对象

1. DriverManager的静态方法：

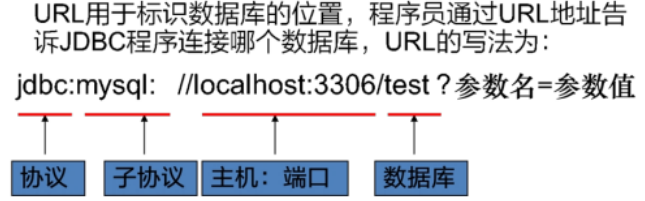


1. 参数说明：



* 连接数据库的 URL 地址格式：

协议名:子协议://服务器名或 IP 地址:端口号/数据库名?参数=参数值



* MySQL 中可以简写：

前提：必须是本地服务器，端口号是 3306

jdbc:mysql:///数据库名

1. 连接Mysql的乱码的处理

如果数据库出现乱码，可以指定参数: ?characterEncoding=utf8，表示让数据库以 UTF-8 编码来处理数据。

|  |
| --- |
| jdbc:mysql://localhost:3306/数据库?characterEncoding=utf8 |

### 案例：得到 MySQL 的数据库连接对象

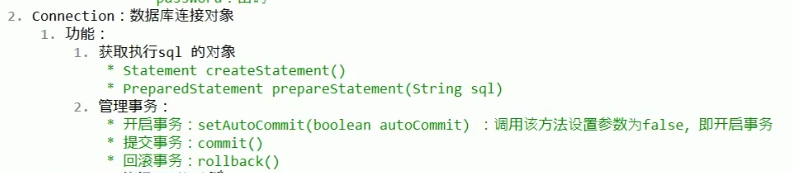
#### 1) 使用用户名、密码、URL 得到连接对象

|  |
| --- |
| **package** com.itheima;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.SQLException;  */\*\**  *\* 得到连接对象*  *\*/*  **public class** Demo2 {  **public static void** main(String[] args) **throws** SQLException {  String url = **"jdbc:mysql://localhost:3306/day24"**;  *//1) 使用用户名、密码、URL 得到连接对象*  Connection connection = DriverManager.*getConnection*(url, **"root"**, **"root"**);  *//com.mysql.jdbc.JDBC4Connection@68de145*  System.***out***.println(connection);  }  } |

#### **2) 使用属性文件和 url 得到连接对象**

|  |
| --- |
| package com.itheima;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.SQLException;  import java.util.Properties;  public class Demo3 {  public static void main(String[] args) throws SQLException {  //url 连接字符串  String url = "jdbc:mysql://localhost:3306/day24";  //属性对象  Properties info = new Properties();  //把用户名和密码放在 info 对象中6 / 21  info.setProperty("user","root");  info.setProperty("password","root");  Connection connection = DriverManager.getConnection(url, info);  //com.mysql.jdbc.JDBC4Connection@68de145  System.out.println(connection);  }  } |

## (2)Connection类



## (3)Statement类

* Statement 作用

代表一条语句对象，用于发送 SQL 语句给服务器，用于执行静态 SQL 语句并返回它所生成结果的对象。

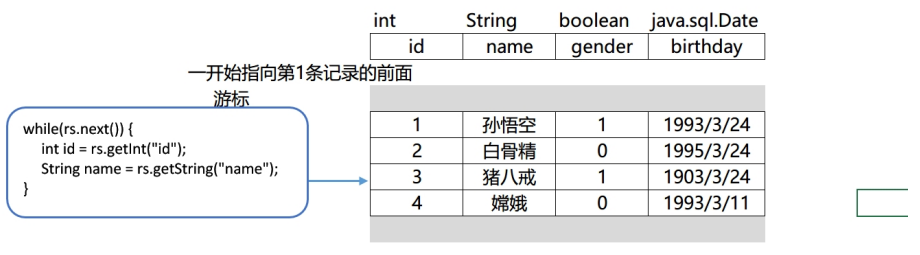
* Statement 中的方法

executeUpdate(String sql) 还可以执行DDL,但是不常用;

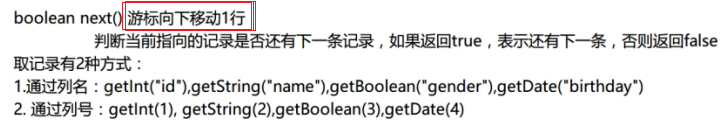
返回值>0表示执行成功，反之失败。

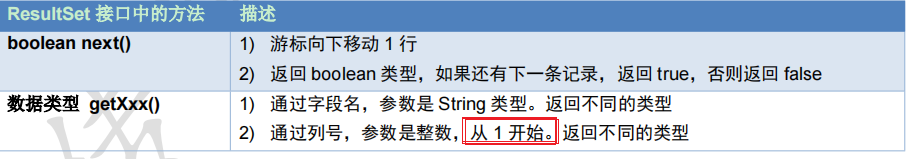
## ResultSet类

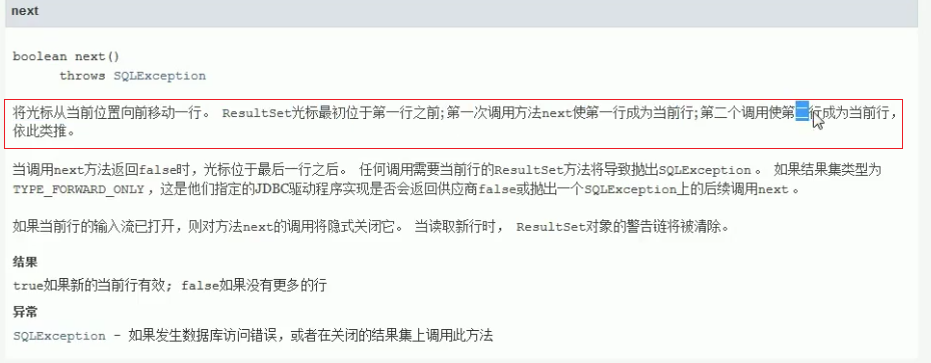
作用：封装数据库查询的结果集，对结果集进行遍历，取出每一条记录



### 核心方法







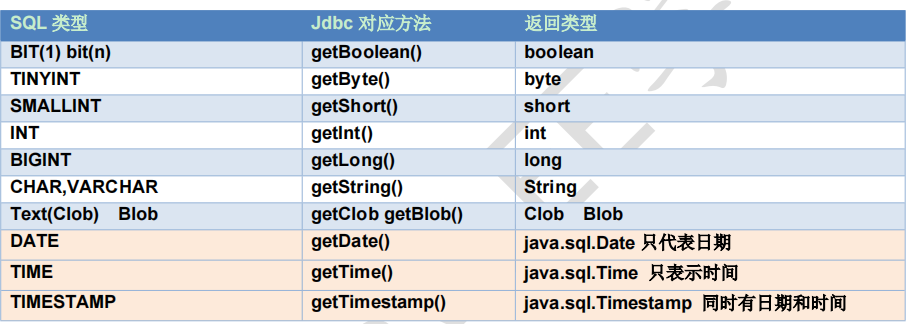
关于 ResultSet 接口中的注意事项：

1) 如果光标在第一行之前，使用 rs.getXX()获取列值，报错：Before start of result set

2) 如果光标在最后一行之后，使用 rs.getXX()获取列值，报错：After end of result set

3) 使用完毕以后要关闭结果集 ResultSet，再关闭 Statement，再关闭 Connection

### 常用数据类型转换表



java.sql.Date、Time、Timestamp(时间戳)，三个共同父类是：java.util.Date

|  |
| --- |
| public class Demo6DQL {  public static void main(String[] args) throws SQLException {  //1) 得到连接对象  Connection connection =  DriverManager.getConnection("jdbc:mysql://localhost:3306/day24","root","root");  //2) 得到语句对象  Statement statement = connection.createStatement();  //3) 执行 SQL 语句得到结果集 ResultSet 对象  ResultSet rs = statement.executeQuery("select \* from student");  //4) 循环遍历取出每一条记录  while(rs.next()) {  int id = rs.getInt("id");  String name = rs.getString("name");  boolean gender = rs.getBoolean("gender");  Date birthday = rs.getDate("birthday");  //5) 输出的控制台上  System.out.println("编号：" + id + ", 姓名：" + name + ", 性别：" + gender + ", 生日：" +  birthday);  }  //6) 释放资源  rs.close();  statement.close();  connection.close();  }  } |

# 数据库工具类 JdbcUtils

* 什么时候自己创建工具类？

如果一个功能经常要用到，我们建议把这个功能做成一个工具类，可以在不同的地方重用

需求：

上面写的代码中出现了很多重复的代码，可以把这些公共代码抽取出来

* 创建类 JdbcUtil 包含 3 个方法：

1) 可以把几个字符串定义成常量：用户名，密码，URL，驱动类

2) 得到数据库的连接：getConnection()

3) 关闭所有打开的资源：

close(Connection conn, Statement stmt)，close(Connection conn, Statement stmt, ResultSet rs)

* Mysql8.0版本的链接注意事项：

1. jar包

<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.11</version>  
</dependency>

1. 链接参数

drivername=com.mysql.cj.jdbc.Driver  
url=jdbc:mysql://localhost:3306/100p?characterEncoding=utf8&useSSL=false&serverTimezone=UTC&rewriteBatchedStatements=true  
username=root  
password=root

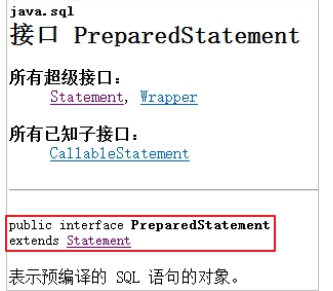
|  |
| --- |
| package main.JDBC;  import java.io.IOException; import java.io.InputStream; import java.sql.\*; import java.util.Properties;  public class JDBCUtils {  private static String *url*;  private static String *username*;  private static String *password*;  private static String *driver*;   static {   try {  InputStream resourceAsStream = JDBCUtils.class.getClassLoader().getResourceAsStream("jdbc.properties");  Properties properties = new Properties();  properties.load(resourceAsStream);   *url* = (String) properties.get("url");  *username* = (String) properties.get("username");  *password* = (String) properties.get("password");  *driver* = (String) properties.get("drivername");   Class.*forName*(*driver*);    } catch (IOException e) {  e.printStackTrace();  } catch (ClassNotFoundException e) {  e.printStackTrace();  }   }    public static Connection getConnection() throws SQLException {  return DriverManager.*getConnection*(*url*, *username*, *password*);  *//return DriverManager.getConnection("jdbc:mysql://localhost:3306/100p", "root", "root");* }   public static void close(Connection con, Statement st, ResultSet rs) {  if (con != null) {  try {  con.close();  } catch (SQLException e) {  e.printStackTrace();  }  }  if (rs != null) {  try {  rs.close();  } catch (SQLException e) {  e.printStackTrace();  }  }  if (st != null) {  try {  st.close();  } catch (SQLException e) {  e.printStackTrace();  }  }    }   public static void close(Connection con, Statement st) {  if (con != null) {  try {  con.close();  } catch (SQLException e) {  e.printStackTrace();  }  }  if (st != null) {  try {  st.close();  } catch (SQLException e) {  e.printStackTrace();  }  }   } } |

# SQL注入问题

如果配置password为：a' or '1'='1

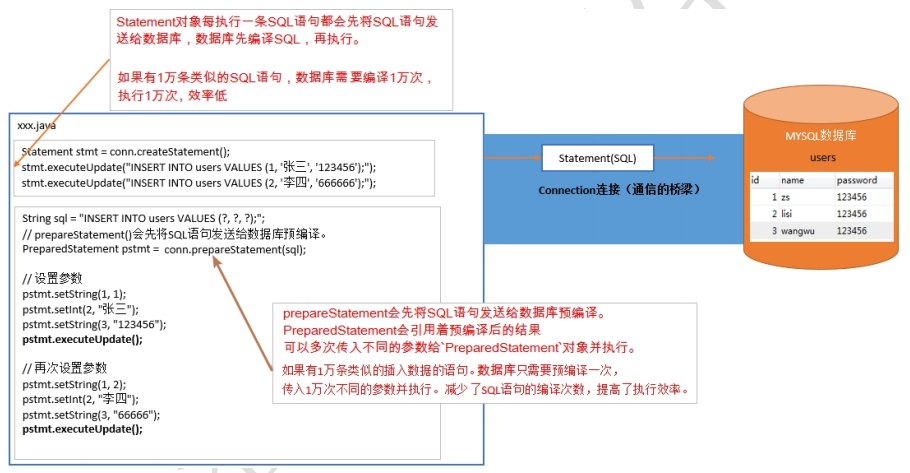
# Prestatement

## 继承结构与作用



PreparedStatement 是 Statement 接口的子接口，继承于父接口中所有的方法。它是一个预编译的 SQL 语句

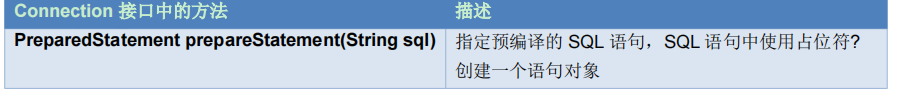
## PreparedSatement 的执行原理



## 功能和优点

1. prepareStatement()会先将 SQL 语句发送给数据库预编译。PreparedStatement 会引用着预编译后的结果。 可以多次传入不同的参数给 PreparedStatement 对象并执行。减少 SQL 编译次数，提高效率。
2. 安全性更高，没有 SQL 注入的隐患。
3. 提高了程序的可读性

## Connection 创建 PreparedStatement 对象



* **PreparedStatement 接口中的方法**



* **使用 PreparedStatement 的步骤**

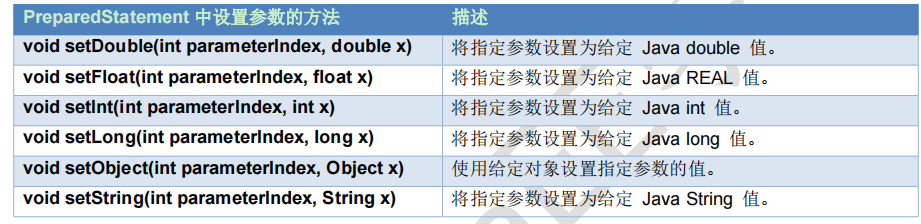
1) 编写 SQL 语句，未知内容使用?占位："SELECT \* FROM user WHERE name=? AND password=?";

2) 获得 PreparedStatement 对象

3) 设置实际参数：setXxx(占位符的位置, 真实的值)

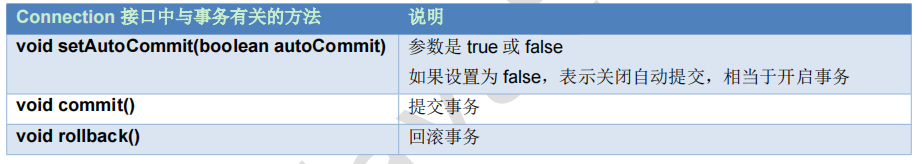
4) 执行参数化 SQL 语句

5) 关闭资源



# JDBC 事务的处理

## API介绍



## 开发步骤

1) 获取连接

2) 开启事务

3) 获取到 PreparedStatement

4) 使用 PreparedStatement 执行两次更新操作

5) 正常情况下提交事务

6) 出现异常回滚事务

7) 最后关闭资源

# 数据库连接池

## 1.JDBC传统模式开发存在的主要问题

### 1.1>时间和内存资源消耗巨大

普通的JDBC数据库连接使用DriverManager来获取，每次向数据库建立连接的时候都要将Connection加载到内存中，再根据JDBC代码（或配置文件）中的用户名和密码进行验证其正确性。这一过程一般会花费0.05~1s,一旦需要数据库连接的时候就必须向数据库请求一个，执行完后再断开连接。显然，如果同一个数据库在同一时间有数十人甚至上百人请求连接势必会占用大量的系统资源，严重的会导致服务器崩溃。

### 1.2>有内存泄漏的风险

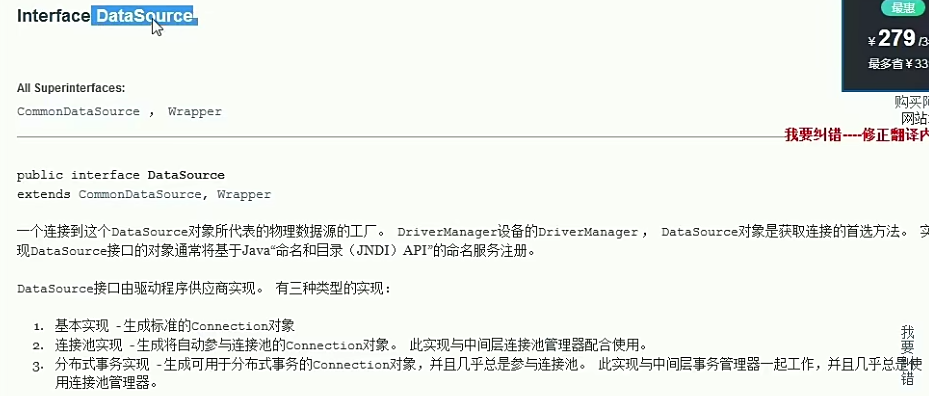
因为每一次数据库连接使用完后都需要断开连接，但如果程序出现异常致使连接未能及时关闭，这样就可能导致内存泄漏，最终只能以重启数据库的方法来解决；

另外使用传统JDBC模式开发不能控制需要创建的连接数，系统一般会将资源大量分出给连接以防止资源不够用，如果连接数超出一定数量也会有极大的可能导致内存泄漏。

连接池的概念：就是一个容器，存放数据库链接的容器

当系统初始化好以后，容器就被创建，容器中会申请一些链接对象，当用户来访问数据库的时候，从容器中获取链接对象，用户访问完以后，将链接对象归还到容器中。

## 2.Interface DataSource



1. getConnection()

由数据库厂商实现

1. 常用的数据库连接池技术
   1. C3P0
   2. Druid:数据库连接池实现技术，由阿里巴巴提供

## C3P0

C3P0是一个开源的JDBC连接池，支持JDBC3规范,实现了JDBC2的标准扩展说明的Connection和Statement池的DataSources对象。

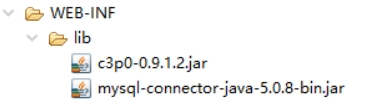
使用C3P0（数据库连接池）的必要性

　　当我们在进行基于数据库的web程序开发时，我们可以先在主程序（如Servlet、Bean）中通过JDBC中的DriverManager建立数据库连接，然后将要对数据库进行操作的sql语句封装到Statement中，最后在返回结果集后断开数据库连接。以上是较为传统的开发模式，然而用这种模式开发会埋下严重的安全隐患。

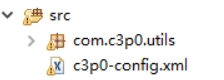
### C3P0实操

#### 1.导入jar包

主要是c3p0和mysql的jar包必须导入,其他根据需求添加



#### 2.配置xml文件



配置文件名必须为c3p0-config.xml或者c3p0-config.properties

* 一般c3p0-config.xml模板

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <c3p0-config>    <default-config>      <!--mysql数据库连接的各项参数-->      <property name="driverClass">com.mysql.jdbc.Driver</property>      <property name="jdbcUrl">jdbc:mysql://localhost:3306/day06</property>      <property name="user">root</property>      <property name="password">root</property>      <!--配置数据库连接池的初始连接数、最小链接数、获取连接数、最大连接数、最大空闲时间-->      <property name="initialPoolSize">10</property>      <property name="minPoolSize">10</property>      <property name="acquireIncrement">5</property>      <property name="maxPoolSize">100</property>      <property name="maxIdleTime">30</property>    </default-config>  </c3p0-config> |

当然，除了以上这几种常用的参数设置以外，这里还有一份有关c3p0-config.xml参数的详细清单，如有需要可自行增加。

* c3p0-config.xml参数清单

|  |
| --- |
| <c3p0-config>      <default-config>      <!--当连接池中的连接耗尽的时候c3p0一次同时获取的连接数。Default: 3 -->      <property name="acquireIncrement">3</property>        <!--定义在从数据库获取新连接失败后重复尝试的次数。Default: 30 -->      <property name="acquireRetryAttempts">30</property>        <!--两次连接中间隔时间，单位毫秒。Default: 1000 -->      <property name="acquireRetryDelay">1000</property>        <!--连接关闭时默认将所有未提交的操作回滚。Default: false -->      <property name="autoCommitOnClose">false</property>        <!--c3p0将建一张名为Test的空表，并使用其自带的查询语句进行测试。如果定义了这个参数那么      属性preferredTestQuery将被忽略。你不能在这张Test表上进行任何操作，它将只供c3p0测试      使用。Default: null-->      <property name="automaticTestTable">Test</property>        <!--获取连接失败将会引起所有等待连接池来获取连接的线程抛出异常。但是数据源仍有效      保留，并在下次调用getConnection()的时候继续尝试获取连接。如果设为true，那么在尝试      获取连接失败后该数据源将申明已断开并永久关闭。Default: false-->      <property name="breakAfterAcquireFailure">false</property>        <!--当连接池用完时客户端调用getConnection()后等待获取新连接的时间，超时后将抛出      SQLException,如设为0则无限期等待。单位毫秒。Default: 0 -->      <property name="checkoutTimeout">100</property>        <!--通过实现ConnectionTester或QueryConnectionTester的类来测试连接。类名需制定全路径。      Default: com.mchange.v2.c3p0.impl.DefaultConnectionTester-->      <property name="connectionTesterClassName"></property>        <!--指定c3p0 libraries的路径，如果（通常都是这样）在本地即可获得那么无需设置，默认null即可      Default: null-->      <property name="factoryClassLocation">null</property>        <!--强烈不建议使用该方法，将这个设置为true可能会导致一些微妙而奇怪的bug-->      <property name="forceIgnoreUnresolvedTransactions">false</property>        <!--每60秒检查所有连接池中的空闲连接。Default: 0 -->      <property name="idleConnectionTestPeriod">60</property>        <!--初始化时获取三个连接，取值应在minPoolSize与maxPoolSize之间。Default: 3 -->      <property name="initialPoolSize">3</property>        <!--最大空闲时间,60秒内未使用则连接被丢弃。若为0则永不丢弃。Default: 0 -->      <property name="maxIdleTime">60</property>        <!--连接池中保留的最大连接数。Default: 15 -->      <property name="maxPoolSize">15</property>        <!--JDBC的标准参数，用以控制数据源内加载的PreparedStatements数量。但由于预缓存的statements      属于单个connection而不是整个连接池。所以设置这个参数需要考虑到多方面的因素。      如果maxStatements与maxStatementsPerConnection均为0，则缓存被关闭。Default: 0-->      <property name="maxStatements">100</property>        <!--maxStatementsPerConnection定义了连接池内单个连接所拥有的最大缓存statements数。Default: 0 -->      <property name="maxStatementsPerConnection"></property>        <!--c3p0是异步操作的，缓慢的JDBC操作通过帮助进程完成。扩展这些操作可以有效的提升性能      通过多线程实现多个操作同时被执行。Default: 3-->      <property name="numHelperThreads">3</property>        <!--当用户调用getConnection()时使root用户成为去获取连接的用户。主要用于连接池连接非c3p0      的数据源时。Default: null-->      <property name="overrideDefaultUser">root</property>        <!--与overrideDefaultUser参数对应使用的一个参数。Default: null-->      <property name="overrideDefaultPassword">password</property>        <!--密码。Default: null-->      <property name="password"></property>        <!--定义所有连接测试都执行的测试语句。在使用连接测试的情况下这个一显著提高测试速度。注意：      测试的表必须在初始数据源的时候就存在。Default: null-->      <property name="preferredTestQuery">select id from test where id=1</property>        <!--用户修改系统配置参数执行前最多等待300秒。Default: 300 -->      <property name="propertyCycle">300</property>        <!--因性能消耗大请只在需要的时候使用它。如果设为true那么在每个connection提交的      时候都将校验其有效性。建议使用idleConnectionTestPeriod或automaticTestTable      等方法来提升连接测试的性能。Default: false -->      <property name="testConnectionOnCheckout">false</property>        <!--如果设为true那么在取得连接的同时将校验连接的有效性。Default: false -->      <property name="testConnectionOnCheckin">true</property>        <!--用户名。Default: null-->      <property name="user">root</property>        <!--早期的c3p0版本对JDBC接口采用动态反射代理。在早期版本用途广泛的情况下这个参数      允许用户恢复到动态反射代理以解决不稳定的故障。最新的非反射代理更快并且已经开始      广泛的被使用，所以这个参数未必有用。现在原先的动态反射与新的非反射代理同时受到      支持，但今后可能的版本可能不支持动态反射代理。Default: false-->      <property name="usesTraditionalReflectiveProxies">false</property>      </default-config>  </c3p0-config> |