连接池&JdbcTemplate

# 回顾

## 使用JDBC连接数据库的四个参数：

|  |  |
| --- | --- |
| **JDBC连接数据库的四个参数** | **说明** |
| **用户名** | root |
| **密码** | root |
| **URL** | jdbc:mysql://localhost:3306/数据库 |
| **驱动类** | com.mysql.jdbc.Driver |

## JDBC的核心API

### 核心API的功能：

|  |  |
| --- | --- |
| **类名** | **作用** |
| **DriverManager实现类** | 1. 注册和管理驱动 2. 得到连接对象 |
| **Connection接口** | 代表一个连接对象，可以创建语句对象 |
| **Statement接口** | 代表SQL语句 |
| **ResultSet接口** | 代表查询的结果集 |

### DriverManager类得到连接的方法

|  |  |
| --- | --- |
| **DriverManager类中的静态方法** | **描述** |
| **Connection getConnection(String url, String user, String password)** | 指定URL，用户名，密码得到一个连接 |
| **Connection getConnection(String url, Properties info)** | 指定URL和属性集合得到一个连接 |

### Connection接口中的方法：

|  |  |
| --- | --- |
| **Connection接口中的方法** | **方法名** |
| **得到Statement** | createStatement() |
| **得到PreparedStatement** | prepareStatement(String sql) |

## ResultSet接口的方法：

|  |  |
| --- | --- |
| **ResultSet接口中的方法** | **描述** |
| **boolean next()** | 1. 向下移动一行 2. 判断是否是在最后一行的后面 |
| **数据类型 getXxx(参数)** | 得到结果集中的数据   1. 通过列名 2. 通过列号 |

## PreparedStatement接口中的方法

|  |  |
| --- | --- |
| **PreparedStatement接口中的方法** | **描述** |
| **int executeUpdate()** | 实现增删改 |
| **ResultSet executeQuery()** | 实现查询 |

## Connection中处理事务的方法

|  |  |
| --- | --- |
| **Connection接口中与事务有关的方法** | **说明** |
| **void setAutoCommit(boolean autoCommit)** | 开启事务 |
| **void commit()** | 提交事务 |
| **void rollback()** | 回滚事务 |

# 学习目标

1. 能够理解连接池解决现状问题的原理
2. 能够使用C3P0连接池
3. 能够编写连接池工具类
4. 能够使用DRUID连接池
5. 能够使用JdbcTemplate执行SQL语句
6. 能够理解JdbcTemplate的原理

# 连接池概述

## 什么是连接池

### JDBC访问数据库时操作Connection对象：

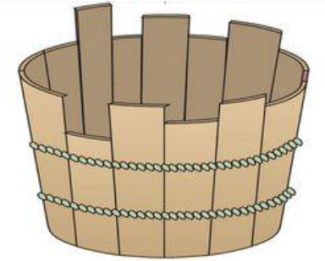
每次访问数据库的一条SQL语句都需要先创建连接，而使用一次以后，就关闭连接。整个访问数据库的过程中，速度最慢的部分就是创建连接。



### 连接对象的使用问题：

1. 获取数据库连接需要消耗比较多的资源，而每次操作都要重新获取新的连接对象，执行一次操作就把连接关闭，而数据库创建连接通常需要消耗相对较多的资源，创建时间也较长。这样数据库连接对象的使用率低。
2. 假设网站一天10万访问量，数据库服务器就需要创建10万次连接，极大的浪费数据库的资源，并且极易造成数据库服务器内存溢出

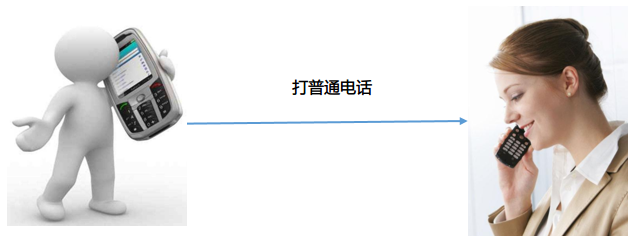
### 解决思路

根据木桶理论，一只桶能装多少水由最短的木板决定。所以我们如果需要提升数据库的访问速度，就应该提高数据库的连接速度。因此我们需要解决两个问题：

1. 提高得到连接对象的速度
2. 提高连接对象的使用率

### 生活中的连接池：

* 普通电话



* 连接池

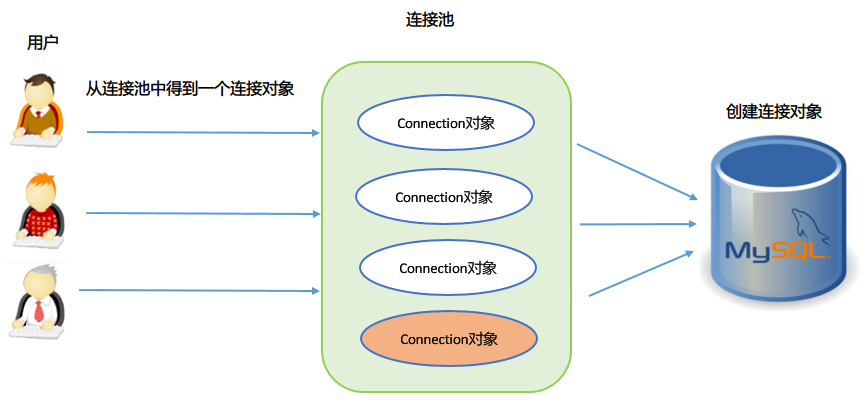


### 解决方案：

|  |  |
| --- | --- |
| **连接对象** | **操作特点** |
| **创建时** | 服务器或程序启动的时候就在内存中创建了一定数量的连接对象，这个内存就是连接池。 |
| **使用时** | 不用自己再去创建连接对象，而是直接从连接池中得到一个空闲的创建好的连接对象。 |
| **关闭时** | 不是真的关闭连接，而是将连接对象放回到连接池中，给其他的用户使用。 |

### 连接池的原理图

当用户访问数据库的时候，直接从已经创建好的连接池中得到一个连接对象使用即可。连接池由程序一开始启动的时候就创建好，并且创建了一定数量的连接对象在内存中。这个内存空间称为连接池，也叫数据源。



## 数据库连接池API

### 数据源接口：

|  |
| --- |
| **javax.sql.DataSource 数据源连接池接口，自己没有实现** |

### 数据源接口中的方法：

|  |  |
| --- | --- |
| **DataSource接口中的方法** | **描述** |
| **Connection getConnection()** | 从连接池中得到一个连接对象 |

### 常用连接池参数

就算不设置任何参数，连接池也是可以直接使用的，因为每个连接池都有默认的参数。

|  |  |
| --- | --- |
| **常用参数** | **描述** |
| **初始连接数 initialPoolSize** | 连接池一开始已经创建好的连接数 |
| **最大连接数**  **maxPoolSize** | 连接池中最大的连接个数 |
| **最长等待时间 checkoutTimeout** | 如果连接池中所有的连接对象都在使用，设置等待多久的时间，单位是毫秒。如果超时会抛出异常。 |
| **最长空闲等待时间**  **maxIdleTime** | 如果一个连接对象在连接池中长时间没有使用，多久以后被服务器回收。 |

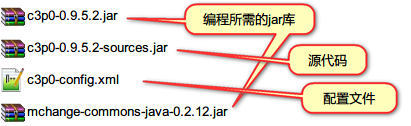
## 常用连接池的介绍

DataSource本身只是Oracle公司提供的一个接口，没有具体的实现，它的实现由连接池的数据库厂商去实现。我们只需要学习这个工具如何使用即可。常用的连接池实现组件有这些：

* 阿里巴巴-德鲁伊druid连接池：Druid是阿里巴巴开源平台上的一个项目
* DBCP(DataBase Connection Pool)数据库连接池，是Apache上的一个Java连接池项目，也是Tomcat使用的连接池组件。
* C3P0是一个开源的JDBC连接池，目前使用它的开源项目有Hibernate，Spring等。C3P0有自动回收空闲连接功能。
* Proxool数据库连接池技术，它是sourceforge下的一个开源项目，这个项目提供一个健壮、易用的连接池，最为关键的是这个连接池提供监控的功能。

# C3P0连接池的使用

## 使用C3P0开发



### 开发步骤：

1. 导入jar包c3p0-0.9.5.2.jar 和 mchange-commons-java-0.2.12.jar，  
   如果要连接mysql，还要数据库驱动程序：mysql-connector-java-5.1.37-bin.jar
2. 在src目录下创建配置文件c3p0-config.xml，配置对应参数
3. 设置数据库的连接参数：用户名、密码、URL、驱动名
4. 设置连接池配置参数：初始连接数、最大连接数、最长等待时间、最长空闲等待时间
5. 创建数据源的实现类ComboPooledDataSource，使用默认配置或命名配置
6. 从连接池中得到10个连接对象
7. 使用命名的配置再进行测试

### 配置文件的要求：

1. 文件名：c3p0-config.xml
2. 路径：放在src目录下
3. 默认配置：没有指定配置名的情况下，默认使用的连接池配置
4. 命名配置： 指定配置名，使用，可以有多个命名配置
5. 多个配置方案的好处：
   1. 可以连接不同的数据库，如：day23,day24
   2. 可以指定不同的连接池的配置参数
   3. 可以连接不同厂商的数据库，如：Oracle，mysql

### 配置文件c3p0-config.xml

<?xml version="1.0" encoding="utf-8" ?>

<c3p0-config>

<!-- 使用默认的配置读取连接池对象 -->

<default-config>

<!-- 连接数据库参数 -->

<property name="user">root</property>

<property name="password">root</property>

<property name="jdbcUrl">jdbc:mysql://localhost:3306/day25</property>

<property name="driverClass">com.mysql.jdbc.Driver</property>

<!-- 连接池的参数，所有的连接池的参数都有默认值 -->

<!-- initialPoolSize 连接池一开始创建的时候就创建了多少个连接数 -->

<property name="initialPoolSize">5</property>

<!-- maxPoolSize 连接池最大连接数 -->

<property name="maxPoolSize">10</property>

<!-- checkoutTimeout 如果连接池中连接对象不够，用户等待多久，如果超过这个时间，抛出异常，单位：毫秒 -->

<property name="checkoutTimeout">3000</property>

</default-config>

<!-- 命名配置 ，可以有多个命名配置 -->

<named-config name="otherc3p0">

<!-- 连接参数 -->

<property name="driverClass">com.mysql.jdbc.Driver</property>

<property name="jdbcUrl">jdbc:mysql://localhost:3306/day25</property>

<property name="user">root</property>

<property name="password">root</property>

<!-- 连接池参数 -->

<property name="initialPoolSize">5</property>

<property name="maxPoolSize">8</property>

<property name="checkoutTimeout">2000</property>

</named-config>

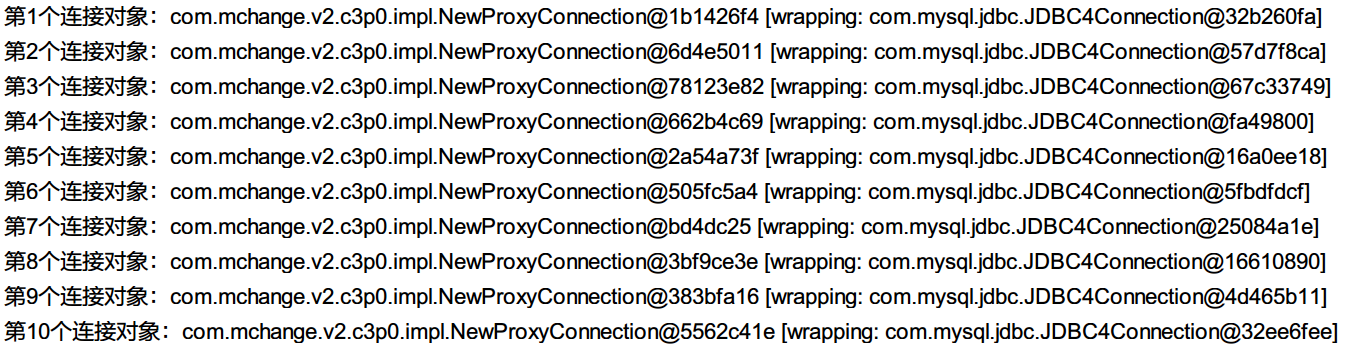
</c3p0-config>

### API介绍

|  |  |
| --- | --- |
| **构造方法** | **描述** |
| **ComboPooledDataSource()** | 使用默认的配置信息 |
| **ComboPooledDataSource(命名配置)** | 使用命名的配置信息  参数：配置信息的名字, otherc3p0  xml: <named-config name="otherc3p0"> |

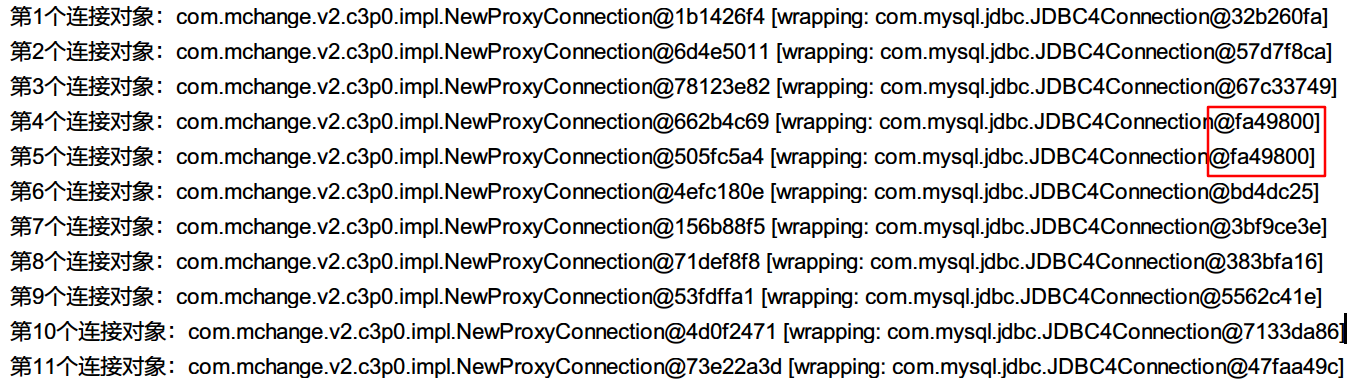
### 运行效果：

* 从连接池中得到10个连接对象



* 创建11个连接对象，出现异常；释放其中一个，发现有重用的连接对象





### 使用默认的配置名：

package com.itheima;

import com.mchange.v2.c3p0.ComboPooledDataSource;

import java.sql.Connection;

import java.sql.SQLException;

public class Demo01C3p0 {

public static void main(String[] args) throws SQLException {

//创建ComboPooledDataSource类, 使用默认配置

ComboPooledDataSource ds = new ComboPooledDataSource();

//从连接池中得到10个连接对象

for (int i = 1 ; i <=11 ; i++) {

Connection connection = ds.getConnection();

System.out.println("第" + i + "个连接对象：" + connection);

//在第4个连接对象释放

if (i==4) {

connection.close(); //将连接对象放回到池中

}

}

}

}

### 使用命名的配置名：

package com.itheima;

import com.mchange.v2.c3p0.ComboPooledDataSource;

import java.sql.Connection;

import java.sql.SQLException;

public class Demo01C3p0 {

public static void main(String[] args) throws SQLException {

//创建ComboPooledDataSource类, 使用默认配置

// ComboPooledDataSource ds = new ComboPooledDataSource();

//使用命名配置

ComboPooledDataSource ds = new ComboPooledDataSource("otherc3p0");

//从连接池中得到10个连接对象

for (int i = 1 ; i <=9 ; i++) {

Connection connection = ds.getConnection();

System.out.println("第" + i + "个连接对象：" + connection);

//在第4个连接对象释放

if (i==4) {

connection.close(); //将连接对象放回到池中

}

}

}

}

# DRUID连接池

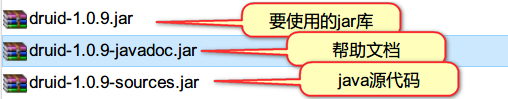
## DRUID简介

Druid是阿里巴巴开发的号称为监控而生的数据库连接池，在功能、性能、扩展性方面，都超过其他数据库连接池。Druid已经在阿里巴巴部署了超过600个应用，经过一年多生产环境大规模部署的严苛考验。如：一年一度的双十一活动，每年春运的抢火车票。

Druid的下载地址：<https://github.com/alibaba/druid>

DRUID连接池使用的jar包：druid-1.0.9.jar

## 下载的包结构如下



## DRUID常用的配置参数

|  |  |
| --- | --- |
| **参数** | **说明** |
| **url** | **数据库连接字符串 jdbc:mysql://localhost:3306/数据库名** |
| **username** | **数据库的用户名** |
| **password** | **数据库的密码** |
| **driverClassName** | **驱动类名。根据url自动识别，这一项可配可不配，如果不配置druid会根据url自动识别数据库的类型，然后选择相应的数据库驱动名** |
| **initialSize** | **初始化时建立的物理连接的个数。初始化发生在显式调用init方法，或者第一次获取连接对象时** |
| **maxActive** | **连接池中最大连接数** |
| **maxWait** | **获取连接时最长等待时间，单位是毫秒。** |

## DRUID连接池基本使用

### API介绍

|  |  |
| --- | --- |
| **DruidDataSourceFactory的方法** | **方法** |
| **public static DataSource createDataSource(Properties properties)** | 通过属性集合中指定的连接池参数创建一个数据源对象 |

### 开发步骤

1. 在src目录下创建一个properties文件，文件名随意，设置上面的参数
2. 加载properties文件的内容到Properties对象中
3. 创建DRUID连接池，使用配置文件中的参数
4. 从DRUID连接池中取出10个连接输出
5. 再从连接池中取出11个连接，并且释放其中的一个。

### 案例代码

* 属性文件：在src目录下新建一个DRUID配置文件，命名为：druid.properties
  + 参数：初始连接数3个，最大连接池10，最长等待时间2秒

url=jdbc:mysql://localhost:3306/day25

username=root

# 数据库的密码

password=root

driverClassName=com.mysql.jdbc.Driver

# 初始连接数

initialSize=5

# 连接池中最大连接数

maxActive=10

# 获取连接时最长等待时间，单位是毫秒。

maxWait=2000

* Java代码

package com.itheima;

import com.alibaba.druid.pool.DruidDataSourceFactory;

import javax.sql.DataSource;

import java.io.IOException;

import java.io.InputStream;

import java.sql.Connection;

import java.util.Properties;

public class Demo02Druid {

public static void main(String[] args) throws Exception {

//2) 加载properties文件的内容到Properties对象中

Properties info = new Properties();

//从类路径下加载一个资源文件，转成输入流

InputStream inputStream = Demo02Druid.class.getResourceAsStream("/druid.properties");

info.load(inputStream);

//System.out.println(info);

//3) 创建DRUID连接池，使用配置文件中的参数

DataSource ds = DruidDataSourceFactory.createDataSource(info);

//4) 从DRUID连接池中取出10个连接输出

for (int i = 1; i <=11 ; i++) {

Connection connection = ds.getConnection();

System.out.println("第"+ i + "个连接对象：" + connection);

if (i==5) {

connection.close(); //释放其中一个

}

}

//5) 再从连接池中取出11个连接，并且释放其中的一个。

}

}

### 案例效果

1. 正常得到10个连接



1. 多出的连接超过2秒，抛出异常





# 连接池工具类

## 创建工具类DataSourceUtils.java

1. 创建私有静态数据源成员变量DataSource ds
2. 在静态代码块中创建连接池
   1. 创建属性对象
   2. 从类路径下加载属性文件，得到输入流对象
   3. 通过工厂类创建一个数据源
3. 创建公有的得到数据源的方法getDataSource()
4. 创建得到连接对象的方法 getConnection()
5. 创建释放资源的方法 close()

package com.itheima.utils;

import com.alibaba.druid.pool.DruidDataSourceFactory;

import javax.sql.DataSource;

import java.io.InputStream;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Properties;

/\*\*

\* 数据源的工具类

\*/

public class DataSourceUtils {

//1) 创建私有静态数据源成员变量DataSource ds

private static DataSource ds;

//2) 在静态代码块中创建连接池

static {

//a) 创建属性对象

Properties info = new Properties();

//b) 从类路径下加载属性文件，得到输入流对象

InputStream in = DataSourceUtils.class.getResourceAsStream("/druid.properties");

try {

info.load(in);

//c) 通过工厂类创建一个数据源

ds = DruidDataSourceFactory.createDataSource(info);

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\*创建公有的得到数据源的方法

\*/

public static DataSource getDataSource() {

return ds;

}

/\*\*

\* 创建得到连接对象的方法

\*/

public static Connection getConnection() {

try {

return ds.getConnection();

} catch (SQLException e) {

e.printStackTrace();

throw new RuntimeException(e);

}

}

/\*\*

\* 创建释放资源的方法

\*/

public static void close(Connection conn, Statement stmt, ResultSet rs) {

if (rs!=null) {

try {

rs.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (stmt!=null) {

try {

stmt.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (conn!=null) {

try {

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

public static void close(Connection conn, Statement stmt) {

close(conn, stmt, null);

}

}

## 数据源工具类的使用

* 需求：向学生表中插入1条学生记录
* 步骤：

1. 通过数据源的工具类得到一个连接对象
2. 创建PreparedStatement语句，使用insert插入1个学生记录。
3. 执行executeUpdate()插入数据到数据库中
4. 通过工具类释放资源

* 代码：

package com.itheima;

import com.itheima.utils.DataSourceUtils;

import java.sql.Connection;

import java.sql.Date;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class Demo03TestDataSource {

public static void main(String[] args) throws SQLException {

//1) 通过数据源的工具类得到一个连接对象

Connection conn = DataSourceUtils.getConnection();

//2) 创建PreparedStatement语句，使用insert插入1个学生记录。

PreparedStatement ps = conn.prepareStatement("INSERT INTO student VALUES (null,?,?,?)");

ps.setString(1,"牛魔王");

ps.setBoolean(2,true);

ps.setDate(3, Date.valueOf("1987-11-23"));

//3) 执行executeUpdate()插入数据到数据库中

int row = ps.executeUpdate();

//4) 通过工具类释放资源

DataSourceUtils.close(conn,ps);

System.out.println(row + "条记录被添加");

}

}

## 连接池小结

DRUID连接池根据Properties对象中的数据作为连接池参数去创建连接池，我们自己定义properties类型的配置文件，名称自己取，也可以放到其他路径，建议放到src目录下方便加载。

不管是C3P0连接池，还是DRUID连接池，配置大致都可以分为2种：

1. 创建数据库连接的参数：用户名，密码，URL，驱动
2. 创建连接池的参数：初始连接数，最大连接数，最长等待时间，最长空闲时间

# JdbcTemplate模板

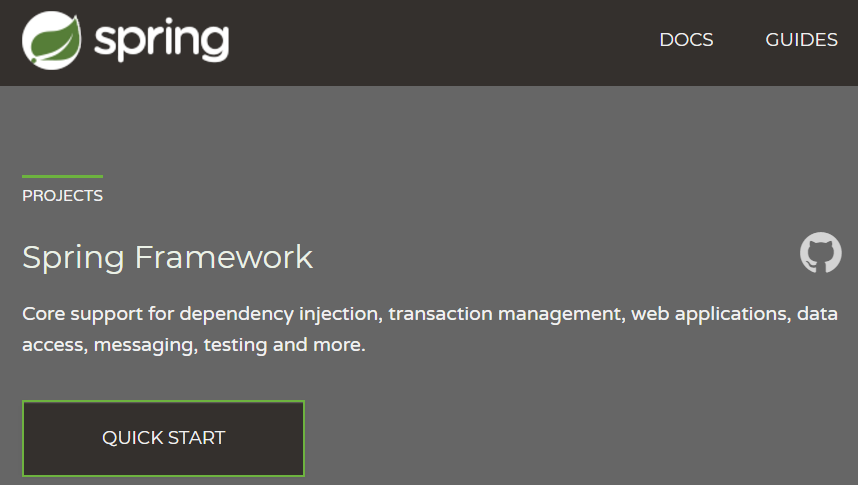
## JdbcTemplate概念

JDBC已经能够满足大部分用户最基本的需求，但是在使用JDBC时，必须自己来管理数据库资源。

如：获取PreparedStatement，设置SQL语句参数，关闭连接等步骤。

JdbcTemplate就是Spring对JDBC的封装，目的是使JDBC更加易于使用。JdbcTemplate是Spring的一部分。JdbcTemplate处理了资源的建立和释放。它运行核心的JDBC工作流，如PreparedStatement的建立和执行，而我们只需要提供SQL语句和提取数据，拿到结果集。

Spring下载地址：http://projects.spring.io/spring-framework/



* 在JdbcTemplate中执行SQL语句的方法大致分为3类：

|  |  |
| --- | --- |
| **JdbcTemplate中的方法** | **功能说明** |
| **execute()** | 用于执行DDL语句，如：建表 |
| **update()** | 用于执行增删改语句，DML |
| **queryXxx()** | 用于执行查询，DQL |

## 执行DDL语句

### API介绍

|  |  |
| --- | --- |
| **JdbcTemplate类的方法** | **说明** |
| **public JdbcTemplate(DataSource dataSource)** | 传入一个数据源对象做为类的构造方法的参数 |
| **public void execute(final String sql)** | 参数：SQL语句  没有返回值 |

### 使用步骤

1. 准备连接池
2. 导入依赖的jar包



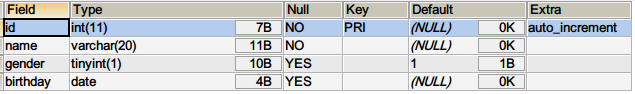
1. 创建JdbcTemplate对象，传入C3P0连接池
2. 调用execute()、update()、queryXxx()等方法

### 案例代码

* 需求：创建一个学生表id,name,gender,birthday

1. id是主键，整数类型，自增长
2. name是varchar(20)，非空
3. 性别是boolean类型
4. 生日是date类型

* 案例效果



* 开发步骤：

1. 创建JdbcTemplate对象
2. 编写建表的SQL语句
3. 使用JdbcTemplate对象的execute()方法执行DDL语句

* 代码：

//运行DDL语句，创建一张表

@Test

public void testDDL() {

//1) 创建JdbcTemplate对象

JdbcTemplate jdbcTemplate = new JdbcTemplate(DataSourceUtils.getDataSource());

//2) 编写建表的SQL语句

String sql = "create table student(id int primary key auto\_increment, " +

"name varchar(20) not null, gender boolean, birthday date)";

//3) 使用JdbcTemplate对象的execute()方法执行DDL语句

jdbcTemplate.execute(sql);

System.out.println("创建成功");

}

## 执行DML语句

### API介绍

|  |  |
| --- | --- |
| **JdbcTemplate类的方法** | **说明** |
| **public int update(final String sql, Object…args)** | 作用：用于执行DML增删改操作  返回值：受影响的行数  参数：  1) 要执行的SQL语句，有可能有占位符?  2) 替换占位符的真实值 |

### 使用步骤

1. 创建JdbcTemplate对象，传入数据源对象。
2. 编写SQL语句
3. 使用JdbcTemplate对象的update方法进行增删改
   1. 每条update向表中添加1条记录，添加6条记录
   2. 修改编号为2的学生名字和生日
   3. 删除编号为6的记录

### 案例代码

public void testAdd() {

//2) 编写SQL语句

//a) 每条update向表中添加1条记录，添加6条记录

jdbcTemplate.update("INSERT INTO student VALUES (NULL,?,?,?)", "孙悟空",true,"1999-11-11");

jdbcTemplate.update("INSERT INTO student VALUES (NULL,?,?,?)", "孙悟天",false,"1999-11-11");

jdbcTemplate.update("INSERT INTO student VALUES (NULL,?,?,?)", "孙悟钣",true,"1995-11-11");

}

@Test

public void testUpdate() {

int row = jdbcTemplate.update("UPDATE student set name=?, birthday=? where id=?", "白骨精","1997-10-24",2);

System.out.println("更新了" + row + "行");

}

@Test

public void testDelete() {

//删除id为3的学生

int row = jdbcTemplate.update("DELETE from student where id=?", 3);

System.out.println("删除" + row + "行记录");

}

### DML的小结

JdbcTemplate的update方法用于执行DML语句。同时还可以在SQL语句中使用?占位符，在update方法的Object... args可变参数中传入对应的参数。

## 执行DQL语句

### API介绍

* + 所有的查询如果没有返回结果，会抛出异常

|  |  |
| --- | --- |
| **方法名** | **功能说明** |
| **query()** | 通用的查询方法，有多个同名方法的重载，可以自定义查询结果集封装成什么样的对象。 |
| **queryForMap()** | 返回Map<String,Object>的查询结果，其中键是列名，值是表中对应的记录。用于查询结果只有1条记录的情况。如果结果集返回多条记录会出现异常。 |
| **queryForObject()** | 返回查询只有单一对象的结果，这个单一结果应该是简单的数据类型，如：Integer.class、Long.class、String.class，不能直接封装成JavaBean对象。  常用于聚合函数的查询。 |
| **queryForList()** | 返回多条记录的查询结果，封装成一个List集合  默认List集合中的每个元素是Map对象，即List<Map<String,Object>>  如果要封装成List<JavaBean>对象，使用query()方法。 |

* 数据准备：实体类，属性名必须与列名一样

import java.sql.Date;  
  
public class Student {  
 private int id; //编号  
 private String name; //姓名  
 private boolean gender; //性别  
 private Date birthday; //生日  
}

### 查询一条记录封装成Map

* 需求：

返回的结果集中如果只有一条记录，可以使用queryForMap()方法，将这条记录被封装成了Map对象

* 开发步骤：

1. 创建JdbcTemplate对象，传入数据源
2. 编写SQL语句
3. 使用JdbcTemplate对象的queryForMap()方法查询结果
4. 返回一个Map对象，并且输出结果

* 代码：

private JdbcTemplate jdbcTemplate = new JdbcTemplate(DataSourceUtils.getDataSource());

//返回的结果集中如果只有一条记录，可以使用queryForMap()方法，将这条记录被封装成了Map对象

@Test

public void testQueryForMap() {

//一条记录

Map<String, Object> map = jdbcTemplate.queryForMap("select \* from student where id=?", 1);

System.out.println(map);

}

* 结果：



* + 1. 查询一条记录封装成自定义的类

## 需求：

查询id为1的学生

* 分析：

使用queryForObject()查询1个学生，但queryForObject()必须要指定查询的结果集与JavaBean属性之间的对应关系，所以这个方法需要传递一个接口做为参数：RowMapper<T>。

|  |  |
| --- | --- |
| **JdbcTemplate中的方法** | **说明** |
| **<T> T queryForObject(String sql, RowMapper<T> rowMapper, Object... args)** | 作用：返回一条记录，并且封装成一个T对象。  参数：   1. SQL语句 2. 接口，我们需要实现这个接口，使用匿名内部类或Lambda 3. 替换占位符真实的值   返回：T对象 |

|  |  |
| --- | --- |
| **RowMapper接口中的方法** | **说明** |
| **T mapRow(ResultSet rs, int rowNum) throws SQLException** | 作用：将结果集中数据取出，封装成一个对象T  返回值：封装好的T对象  参数：  1) 结果集  2) 当前的行号 |

我们需要重写这个方法，指定属性与列之间的映射关系，代码中可以使用匿名内部类。

* 开发步骤：

1. 创建JdbcTemplate对象，传入数据源对象
2. 编写SQL语句
3. 使用JdbcTemplate对象的queryForObject ()方法查询结果，方法中传入一个接口的匿名内部类
4. 也可以使用Lambda表达式封装成JavaBean对象

* 代码：

//3) 使用JdbcTemplate对象的queryForObject ()方法查询结果，方法中传入一个接口的匿名内部类

@Test

public void testQueryForObject() {

Student student = jdbcTemplate.queryForObject("select \* from student where id=?", new RowMapper<Student>() {

@Override

public Student mapRow(ResultSet rs, int i) throws SQLException {

System.out.println(i);

return new Student(rs.getInt("id"), rs.getString("name"),

rs.getBoolean("gender"), rs.getDate("birthday"));

}

}, 2);

System.out.println(student);

}

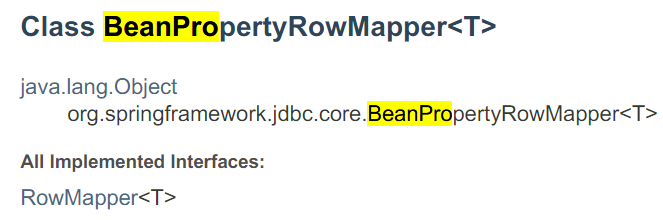
* 结果：



### 查询一条记录封装成实体类

如果每个JavaBean都需要自己封装每个属性，那开发效率将大打折扣，所以Spring JDBC提供了这个接口的实现类BeanPropertyRowMapper，使用起来更加方便。只需要在构造方法中传入Student.class类对象即可，它会自动封装所有同名的属性。请看另一种写法：

使用BeanPropertyRowMapper实现类



表的字段名与实体类属性名映射的规则：

1. 表的字段名与类中的属性名相同，表的字段名大小写不区分。
2. 表的字段名如果有多个单词使用下划线隔开，与Java中驼峰命名的属性相对应。

|  |  |
| --- | --- |
| **表字段名** | **类的属性名** |
| **name或NAME** | name |
| **dept\_name** | deptName |

* 开发步骤：

1. 创建JdbcTemplate对象，传入数据源对象。
2. 编写SQL语句
3. 使用JdbcTemplate对象的queryForObject ()方法查询结果，方法中传入BeanPropertyRowMapper的实现类

* 结果：



@Test

public void testQueryForObject2() {

Student student = jdbcTemplate.queryForObject("select \* from student where id=?", new BeanPropertyRowMapper<>(Student.class), 4);

//如果查询结果为空，会抛出异常

System.out.println(student);

}

* + 1. 查询多条记录封装成List<Map<String,Object>>
* 需求：

查询1号部门中所有的学生，使用queryForList()方法查询元素

注：queryForList这个方法默认返回的List中的每个元素是Map对象

* 代码：

//注：queryForList这个方法默认返回的List中的每个元素是Map对象

@Test

public void testQueryForList() {

List<Map<String, Object>> list = jdbcTemplate.queryForList("select \* from student");

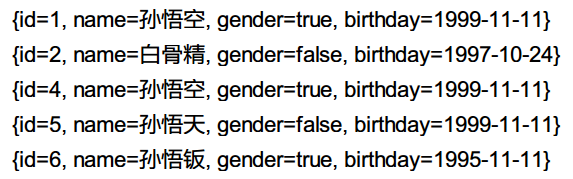
for (Map<String, Object> s : list) {

System.out.println(s);

}

}

* 结果：



### 查询多条记录封装成List<Student>

如果要封装成List<Student>对象怎么办呢？这里需要使用query方法来实现，如果使用queryForList方法则需要自己实现RowMapper接口。

* 代码：

//封装成List<Student>对象

@Test

public void testQuery() {

List<Student> students = jdbcTemplate.query("select \* from student where gender=?", new BeanPropertyRowMapper<>(Student.class), true);

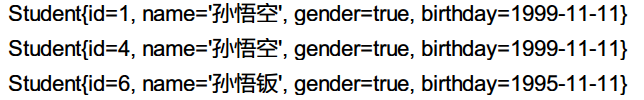
for (Student student : students) {

System.out.println(student);

}

}

* 结果：



* + 注：如果查询没有得到任何的记录，会抛出EmptyResultDataAccessException异常
    1. 查询一列
* 需求：

查询学生名字这一列，可以使用queryForList()方法，并且直接指定List<String>中的每个元素的类型为String.class。

|  |  |
| --- | --- |
| **方法名** | **说明** |
| **public <T> List<T> queryForList(String sql, Class<T> elementType, Object... args)** | 方法返回多行记录，只有一列，这一列的类型是指定的类型  参数：   1. SQL语句 2. 指定数据的类型，如：String.class 3. 替换占位符的值 |

* 代码：

//查询学生名字这一列，可以使用queryForList()方法

@Test

public void testString() {

List<String> names = jdbcTemplate.queryForList("select name from student", String.class);

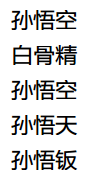
for (String name : names) {

System.out.println(name);

}

}

* 结果：



* + 1. 聚合函数
* 需求：

查询一共有多少个学生，使用queryForObject()方法，指定参数为Integer.class类型

|  |  |
| --- | --- |
| **方法名** | **说明** |
| **public <T> T queryForObject(String sql, Class<T> requiredType, Object... args)** | 查询一条记录  参数：   1. 指定SQL语句 2. 指定返回类型 3. 替换占位符的真实值 |

* 代码：

//查询一共有多少个男生，使用queryForObject()方法

@Test

public void testCount() {

Integer num = jdbcTemplate.queryForObject("select count(\*) from student where gender=?", Integer.class, true);

System.out.println(num);

}

* 结果：



### DQL的小结

JdbcTemplate的query()方法用于执行DQL语句，简化JDBC的代码。同时还可以在SQL语句中使用？占位符通过在query方法的Object... args可变参数中传入对应的参数。

## JdbcTemplate原理

### 模拟JdbcTemplate

我们看到在使用JdbcTemplate的query方法返回对象，需要用到JdbcTemplate类、RowMapper接口。我们使用代码来模拟JdbcTemplate的query方法。

### 步骤分析

1. 定义MyRowMapper<T>接口，模拟RowMapper，添加T mapRow(ResultSet rs) 抛出SQLException异常，方法用于处理一行结果。
2. 定义MyJdbcTemplate类，模拟JdbcTemplate类的功能
3. 在MyJdbcTemplate定义一个构造方法，参数为DataSource
4. 在MyJdbcTemplate定义方法  
   public <T> List<T> query(String sql, MyRowMapper<T> rowMapper, Object...args)  
   用于实现查询返回对象集合。在方法中使用JDBC查询数据表，并且调用MyRowMapper接口中的方法将每一行结果封装成一个T对象，添加到List集合中返回list对象。
5. 定义测试类，编写SQL语句，使用MyJdbcTemplate的query方法来查询数据。

### 案例代码

package com.itheima;

import com.itheima.entity.Student;

import com.itheima.utils.DataSourceUtils;

import javax.sql.DataSource;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

//1) 定义MyRowMapper<T>接口，模拟RowMapper，

interface MyRowMapper<T> {

// 抛出SQLException异常，方法用于处理一行结果。

T mapRow(ResultSet rs) throws SQLException;

}

//2) 定义MyJdbcTemplate类，模拟JdbcTemplate类的功能

class MyJdbcTemplate {

private DataSource ds;

//3) 在MyJdbcTemplate定义一个构造方法，参数为DataSource

public MyJdbcTemplate(DataSource ds) {

this.ds = ds;

}

//4) 在MyJdbcTemplate定义方法

public <T> List<T> query(String sql, MyRowMapper<T> rowMapper, Object...args) {

//用于实现查询返回对象集合。在方法中使用JDBC查询数据表

Connection conn = null;

PreparedStatement ps = null;

ResultSet rs = null;

List<T> list = new ArrayList<>();

try {

conn = ds.getConnection();

//sql有可能有问号

ps = conn.prepareStatement(sql);

//设置了所有的参数

for (int i = 0; i < args.length; i++) {

//第1个参数，数据中[0]

ps.setObject(i+1, args[i]);

}

//查询

rs = ps.executeQuery();

while (rs.next()) {

// 并且调用MyRowMapper接口中的方法将每一行结果封装成一个T对象，添加到List集合中返回list对象

T t = rowMapper.mapRow(rs);

list.add(t);

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

DataSourceUtils.close(conn, ps,rs);

}

return list;

}

}

public class Demo06MyJdbcTemplate {

//使用自己的jdbctemplate

public static void main(String[] args) {

MyJdbcTemplate jdbcTemplate = new MyJdbcTemplate(DataSourceUtils.getDataSource());

List<Student> students = jdbcTemplate.query("select \* from student where gender=?", new MyRowMapper<Student>() {

@Override

public Student mapRow(ResultSet rs) throws SQLException {

return new Student(rs.getInt("id"), rs.getString("name"),

rs.getBoolean("gender"), rs.getDate("birthday"));

}

}, true);

for (Student student : students) {

System.out.println(student);

}

}

}

