

day19-JdbcTemplate

学习目标

1. 能够说出什么是数据库元数据
2. 掌握自定义数据库框架，实现增加、删除、更新方法
3. 掌握JdbcTemplate实现增删改
4. 掌握JdbcTemplate实现增查查询
5. 能够理解分层的作用

一、使用JdbcTemplate完成增删改查的操作

1. 相关知识点

JavaBean就是一个类，在开发中常用于封装数据。具有如下特性

1. 提供私有字段：private 类型 字段名;
2. 提供getter/setter方法：get和set方法一定要是public
3. 提供无参构造
4. 需要实现接口：java.io.Serializable，通常偷懒省略了。

javaBean不是功能, 也不是大公司的一个规定, 全世界的开发人员之间的一个约定俗成. 很多框架就依赖JavaBean属性来设计做功能

javabean属性: get和set方法, 去掉set, 然后把set后面的字段首写字母变小写, 首写字母变成小写的字段就是javabean属性, 但是我们一般的情况下, 字段和javaBean属性一致

2. JdbcTemplate介绍

2.1 概述

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。他帮助我们避免一些常见的错误，比如忘了总要关闭连接。他运行核心的JDBC工作流，如PreparedStatement的建立和执行，而我们只需要提供SQL语句和提取结果。

2.2 JdbcTemplate和Java元数据核心API

方法	作用
<code>public JdbcTemplate(DataSource dataSource)</code>	构造方法，传递数据源做为参数
<code>int update(String sql, Object...args)</code> 执行增,删,改	前面传sql语句，后面用数组给问号赋值，不需要写角标
<code>queryForMap(String sql,?值)</code> 返回Map	键是列名，值是表中对应的记录。一条语句。
<code>queryForObject(sql,new BeanPropertyRowMapper<>(User.class))</code> 返回User对象 <code>queryForObject(sql,Long.class);</code> 返回sql语句中count次数	查询一个对象 <code>User user=jdbcTemplate.queryForObject(sql,new BeanPropertyRowMapper<>(User.class),1);</code> 传入(sql语句, User.class, ?号赋值) <code>Long count=jdbcTemplate.queryForObject(sql,Long.class);</code>
<code>queryForList(sql)</code>	在queryForMap基础上将多个map存放到一个list集合里面,返回list
<code>query(sql, User对象);</code> 返回list	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。 <code>List list=jdbcTemplate.query(sql,new BeanPropertyRowMapper<>(User.class));</code>
<code>ParameterMetaData.getParameterCount()</code>	获得sql语句中问号?的个数, <code>PreparedStatement.setObject(index,value)</code> 方法将sql语句?号填充
<code>ResultSetMetaData.getColumnCount();</code>	获取结果集中列项目的个数,
<code>ResultSetMetaData.getColumnName(int column);</code>	获得数据指定列的列名

3.使用jdbcTemplate完成CRUD

3.1开发步骤

1. 创建项目,导入jar



2. 创建jdbcTemplate对象,传入连接池

3. 调用execute()、update()、queryXxx()等方法

3.2JdbcTemplate实现增删改

3.2.1API介绍

```
public int update(String sql, Object ... params); //固定写法，前面传sql语句，后面用数组给问号赋值，不需要写角标
```

3.2.2代码实现

```
package Utils;

import com.mchange.v2.c3p0.ComboPooledDataSource;
import javax.sql.DataSource;
import java.sql.SQLException;

public class C3P0Utils {
    private static DataSource dataSource = new ComboPooledDataSource();
    public static DataSource getDataSource() throws SQLException {
        return dataSource;
    }
}
```

- 增加

```
import Utils.C3P0Utils;
import org.springframework.jdbc.core.JdbcTemplate;
import java.sql.SQLException;
public class JdbcDemo {
    public static void main(String[] args) throws SQLException {
        JdbcTemplate jdbcTemplate=new JdbcTemplate(C3P0Utils.getDataSource());
        jdbcTemplate.update("insert into cend values (null,?,?)","寻欢",23);
    }
}
```

3.2JdbcTemplate实现查询

3.2.1查询一条记录封装成Map

- 需求: 查询id为1的用户, 封装成map对象
- 开发步骤:

创建JdbcTemplate对象，传入数据源

编写SQL语句

使用JdbcTemplate对象的queryForMap(String sql)方法查询结果

返回是一个Map对象

- 代码实现

```
package JdbcTemplateDemo;

import Utils.C3P0Utils;
import org.springframework.jdbc.core.JdbcTemplate;
import java.sql.SQLException;
import java.util.Map;
import java.util.Set;

public class JdbcDemo {
    public static void main(String[] args) throws SQLException {
        JdbcTemplate jdbcTemplate=new JdbcTemplate(C3P0Utils.getDataSource());
        Map<String,Object> map=jdbcTemplate.queryForMap("select * from cend where id=?",2);
        Set set=map.keySet();
        for (Object o : set) {
            System.out.println(o+": "+map.get(o));
        }
    }
}
```

3.2.2查询一条记录封装成实体对象

- 需求: 查询id为1的用户, 封装成user对象

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User.class类对象即可, 它会自动封装所有同名的属性。使用BeanPropertyRowMapper实现类:

- 开发步骤

创建JdbcTemplate对象, 传入数据源

编写查询的SQL语句

使用JdbcTemplate对象的queryForObject方法, 并传入需要返回的数据的类型

返回是一个实体对象

- 代码实现

```
//执行queryForObject(sql,封装器,参数)的方法来查询;
//要求列名必须和JavaBean属性一致, 如果不一致是不能封装的
package JdbcTemplateDemo;
import Utils.C3P0Utils;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import java.sql.SQLException;
import User.User;
public class JdbcDemo {
    public static void main(String[] args) throws SQLException {
        JdbcTemplate jdbcTemplate=new JdbcTemplate(C3P0Utils.getDataSource());
        String sql="select * from cend where id = ?";
        User user=jdbcTemplate.queryForObject(sql,new BeanPropertyRowMapper<>(User.class),1);

        System.out.println(user);
    }
}
```

```
}  
}
```

3.2.3 查询多条记录封装成 `List<Map<String, Object>>`

- 需求: 查询所有的用户, 封装成 `List<Map<String, Object>> list`
- 开发步骤:
 - 创建JdbcTemplate对象, 传入数据源
 - 编写SQL语句
 - 使用JdbcTemplate对象的query(String sql)方法查询结果
- 代码实现

```
package JdbcTemplateDemo;  
  
import Utils.C3P0Utils;  
import org.springframework.jdbc.core.JdbcTemplate;  
import java.sql.SQLException;  
import java.util.Iterator;  
import java.util.List;  
import java.util.Map;  
import java.util.Set;  
  
public class JdbcDemo {  
    public static void main(String[] args) throws SQLException {  
        JdbcTemplate jdbcTemplate=new JdbcTemplate(C3P0Utils.getDataSource());  
        String sql="select * from cend";  
        List<Map<String, Object>> mapList=jdbcTemplate.queryForList(sql);  
        for (Map<String, Object> map : mapList) {  
            Set<Map.Entry<String, Object>> set=map.entrySet();  
            Iterator it=set.iterator();  
            while(it.hasNext()){  
                Map.Entry entry= (Map.Entry) it.next();  
                System.out.println(entry.getKey()+" "+entry.getValue());  
            }  
        }  
    }  
}
```

3.2.4 查询多条记录封装成 `List<JavaBean>`

- 需求: 查询所有的用户, 封装成 `List<JavaBean> list`
- 开发步骤:
 - 创建JdbcTemplate对象, 传入数据源
 - 编写SQL语句
 - 使用JdbcTemplate对象的query(String sql,new BeanPropertyRowMapper<>)方法查询结果
- 代码实现

```

package JdbcTemplateDemo;

import Utils.C3P0Utils;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import java.sql.SQLException;
import java.util.List;
import User.User;

public class JdbcDemo {
    public static void main(String[] args) throws SQLException {
        JdbcTemplate jdbcTemplate=new JdbcTemplate(C3P0Utils.getDataSource());
        String sql="select * from cend";
        List<User> list=jdbcTemplate.query(sql,new BeanPropertyRowMapper<>(User.class));
        for (User user : list) {
            System.out.println(user);
        }
    }
}

```

3.2.5统计总记录数

- 需求: 统计user的总记录数
- 开发步骤:
 - 创建JdbcTemplate对象，传入数据源
 - 编写SQL语句
 - 使用JdbcTemplate对象的使用queryForObject()方法，指定参数为Integer.class或者Long.class
- 代码实现

```

package JdbcTemplateDemo;

import Utils.C3P0Utils;
import org.springframework.jdbc.core.JdbcTemplate;
import java.sql.SQLException;

public class JdbcDemo {
    public static void main(String[] args) throws SQLException {
        JdbcTemplate jdbcTemplate=new JdbcTemplate(C3P0Utils.getDataSource());
        String sql="select count(*) from cend";
        Long count=jdbcTemplate.queryForObject(sql,Long.class);
        System.out.println(count);
    }
}

```

二,自定义数据库框架(增删改) 框架

1.元数据概述

元数据(MetaData)，即定义数据的数据。打个比方，就好像我们要想搜索一首歌(歌本身是数据)，而我们可以通过歌名，作者，专辑等信息来搜索，那么这些歌名，作者，专辑等等就是这首歌的元数据。因此数据库的元数据就是一些注明数据库信息的数据。

简单来说: 元数据就是数据库、表、列的定义信息。

元数据在建立框架和架构方面是特别重要的知识,我们可以使用数据库的元数据来创建自定义JDBC框架, 模仿jdbcTemplate.

① 由PreparedStatement对象的getParameterMetaData ()方法获取的是ParameterMetaData对象。

② 由ResultSet对象的getMetaData()方法获取的是ResultSetMetaData对象。

2.通过JDBC获得元数据

2.1ParameterMetaData 用于解析sql语句中的问号?

2.1.1概述

ParameterMetaData是由preparedStatement对象通过getParameterMetaData方法获取而来，ParameterMetaData 可用于获取有关 PreparedStatement 对象和其预编译sql语句 中的一些信息. eg:参数个数,获取指定位置占位符的SQL类型

```
select * from user where name=? and password=?
```

ParameterMetaData获得参数的个数，参数类型(mysql不支持)

获得ParameterMetaData:

```
ParameterMetaData parameterMetaData = preparedStatement.getParameterMetaData ()
```

2.1.2ParameterMetaData相关的API

- int getParameterCount(); 获得sql语句中问号?的个数
- int getParameterType(int param) 获取指定参数的SQL类型。(注:MySQL不支持获取参数类型)

2.1.3实例代码

```
package JdbcTemplateDemo;

import Utils.C3P0Utils;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.ParameterMetaData;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class JdbcDemo {

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

```

DataSource ds=C3P0Utils.getDataSource();
Connection c=ds.getConnection();
String sql="select * from cend where name=? and id=?";
PreparedStatement ps=c.prepareStatement(sql);
ParameterMetaData pmd=ps.getParameterMetaData();
    int pc = pmd.getParameterCount();
    System.out.println(pc);
ps.close();
    c.close();
}
}

```

2.2 ResultSetMetaData 用于得到返回表中的字段名及个数,自定义框架暂时用不上

2.2.1概述

ResultSetMetaData是由ResultSet对象通过getMetaData方法获取而来, `ResultSetMetaData` 可用于获取有关 `ResultSet` 对象中列的类型和属性的信息。

ResultSet结果集

id	NAME	age	score
1	张三	25	99.5
2	王五	35	88.5
3	张三	25	99.5
4	王五	35	88.5

ResultSetMetaData获取结果集中的列名和列的类型

获得ResultSetMetaData:

```
ResultSetMetaData resultSetMetaData = resultSet.getMetaData()
```

2.2.2resultSetMetaData 相关的API

- getColumnCount(); 获取结果集中列项目的个数,
- getColumnName(int column); 获得数据指定列的列名

2.2.3实例代码

```

package JdbcTemplateDemo;

import Utils.C3P0Utils;
import javax.sql.DataSource;
import java.sql.*;

public class JdbcDemo {
    public static void main(String[] args) throws SQLException {
        DataSource ds = C3P0Utils.getDataSource();
        Connection c = ds.getConnection();
        String sql = "select * from cend";
        PreparedStatement ps = c.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();
        ResultSetMetaData rsmd=rs.getMetaData();
    }
}

```



```

        int resultcount=rsmd.getColumnCount();
        for(int i=1;i<=resultcount;i++)
            System.out.println(rsmd.getColumnName(i));
        rs.close();
        ps.close();
        c.close();
    }
}

```

3.自定义JDBC框架 ,目前只能做update增删改

利用ParameterMetaData的getParameterCount()方法获取 ? 个数,然后用PreparedStatement的setObject()方法将sql语句填充

```

package Utils;

import com.mchange.v2.c3p0.ComboPooledDataSource;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

public class C3P0Utils {
    private static DataSource dataSource = new ComboPooledDataSource();
    public static DataSource getDataSource() throws SQLException {
        return dataSource;
    }
    public static void release(Connection connection,Statement statement){
        try {
            if(connection!=null){
                connection.close();
            }
            if(statement!=null){
                statement.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

```

package Utils;

import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.ParameterMetaData;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class TemplateUtils {

```

```

private DataSource dataSource;
private TemplateUtils() {
}
public TemplateUtils(DataSource dataSource) {
    this.dataSource = dataSource;
}
public int update(String sql,Object ...params){
    Connection connection =null;
    PreparedStatement ps =null;
    int x=-1;
    try {
        if(dataSource==null){
            throw new RuntimeException("DataSource Exception");
        }
        if(sql==null){
            throw new RuntimeException("Sql Exception");
        }
        connection = dataSource.getConnection();
        ps = connection.prepareStatement(sql);
        ParameterMetaData pmd = ps.getParameterMetaData();
        int count = pmd.getParameterCount();
        for (int i = 1; i < count+1; i++) {
            ps.setObject(i,params[i-1]);
        }
        x=ps.executeUpdate();

    } catch (SQLException e) {
        e.printStackTrace();
    }finally{
        C3P0Utils.release(connection,ps);
    }
    return x;
}
}

```

```

package Server;
import Utils.TemplateUtils;
import Utils.C3P0Utils;
import java.sql.SQLException;

public class Server {
    public static void main(String[] args) {
        TemplateUtils td= null;
        try {
            td = new TemplateUtils(C3P0Utils.getDataSource());
        } catch (SQLException e) {
            e.printStackTrace();
        }
        int res=td.update("insert into cend values(?,?,?)",null,"夏雪宜",31);
        System.out.println(res);
    }
}

```

三,三层架构

1,分层的作用

我们之前的登录案例是将用户输入，数据库的操作，逻辑处理放在了同一个方法中，这样虽然非常直观，但是等项目做大的时候非常不好维护代码，也不好增加功能

- 软件中分层：按照不同功能分为不同层，通常分为三层：表现层，业务层，持久(数据库)层。



- 不同层次包名的命名

分层	包名(公司域名倒写)
表现层(web层) 注: 后续学javaweb	com.itheima.web
业务层(service层)	com.itheima.service
持久层(数据库访问层)	com.itheima.dao
JavaBean	com.itheima.bean
工具类	com.itheima.utils

- 分层的意义:
 1. 解耦：降低层与层之间的耦合性。
 2. 可维护性：提高软件的可维护性，对现有的功能进行修改和更新时不会影响原有的功能。
 3. 可扩展性：提升软件的可扩展性，添加新的功能的时候不会影响到现有的功能。
 4. 可重用性：不同层之间进行功能调用时，相同的功能可以重复使用。

2.使用三层架构改写登录案例

2.1案例需求

在控制台输入用户和密码, 判断用户是否登录成功

2.2 案例思路

2.3代码实现