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

```java
public class User {
    private int id;
    private String username;
    private String password;
    private String nickname;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getNickname() {
        return nickname;
    }

    public void setNickname(String nickname) {
        this.nickname = nickname;
    }
}
```

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。

2.2jdbcTemplate核心API

方法	作用
public JdbcTemplate(DataSource dataSource)	构造方法，传递数据源做为参数
int update(String sql, Object... args)	执行增删改
queryForMap()	返回Map的查询结果，其中键是列名，值是表中对应的记录。
queryForObject()	查询一个对象
queryForList()	返回多条记录的查询结果，封装成一个List集合,但是封装的是Map
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。

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);  //用于执行`INSERT`、`UPDATE`、`DELETE`等语句。
```

3.2.2代码实现

- 增加

```
@Test
//使用JDBC模版进行添加用户
public void fun01(){
    //1. 创建JdbcTemplate对象(需要传入连接池)
    JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());

    //2. 使用update,执行sql
    String sql = "insert into user values (?,?,?,?)";
    Object[] params = {null,"ls","123456","李四"};
    jdbcTemplate.update(sql,params);

}
```

- 更新

```
@Test
//使用JDBC模版进行更新用户; 把id为7的用户的名字改成ww
public void fun02(){
    //1.创建JdbcTemplate对象(需要传入连接池)
    JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());
    //2. 使用update,执行sql语句
    String sql = "update user set username = ? where id = ?";
    Object[] params = {"ww",7};
    jdbcTemplate.update(sql,params);

}
```

- 删除

```
@Test
//使用JDBC模版进行删除用户; 把id为7的用户删除
public void fun03(){
   //1. 创建JdbcTemplate对象(需要传入连接池)
    JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());

    //2,使用update,执行sql语句
    jdbcTemplate.update("DELETE FROM  user WHERE  id = ?",7);

}
```

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

```
 //1. 创建JdbcTemplate对象(需要传入连接池)
 JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());

//2. 执行queryForMap()的方法来查询
 String sql = "select * from user where id = ?";
Map<String, Object> map = jdbcTemplate.queryForMap(sql, 1);
```

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:

  Class BeanPropertyRowMapper<T>

  java.lang.Object
  org.springframework.jdbc.core.BeanPropertyRowMapper<T>

  All Implemented Interfaces:

  RowMapper<T>

- 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

```
 //1. 创建JdbcTemplate对象(需要传入连接池)
JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());

 //2. 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
String sql = "select * from user where id = ?";
User user = jdbcTemplate.queryForObject(sql, new BeanPropertyRowMapper<>(User.class), 1);
```

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

```
//1. 创建JdbcTemplate对象(需要传入连接池)
JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());

//2. 调用queryForList()查询,一条数据就对应一个map, 再把多个map封装成到List集合里面
String sql = "select * from user";
 List<Map<String, Object>> mapList = jdbcTemplate.queryForList(sql);
```

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

```
//1. 创建JdbcTemplate对象(需要传入连接池)
JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());

//2. 调用query(sql, 封装器)查询,一条数据就对应一个User对象, 再把多个对象封装成到List集合里面
String sql = "select * from user";
List<User> list = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(User.class));
```

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

```
 //1. 创建JdbcTemplate对象(需要传入连接池)
JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());

//2. 调用queryForObject(Long.class) 查询
String sql = "select count(*) from user";
Long count = jdbcTemplate.queryForObject(sql, Long.class);
```

4.JDBCTemplate总结

4.1 增删改【重点】

- 创建JDBCTemplate对象, 需要传入DataSource
- 调用 update(sql, Object...params);

4.2 查询

4.2.1 queryForObject 【重点】

- 查询一条记录, 封装成JavaBean对象
- 统计总数量

4.2.2 queryForMap

- 查询一条记录, 封装成Map集合

4.2.3 queryForList

- 查询多条记录, 封装成List list

4.2.4 query【重点】

- 查询多条记录, 封装成List list

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。

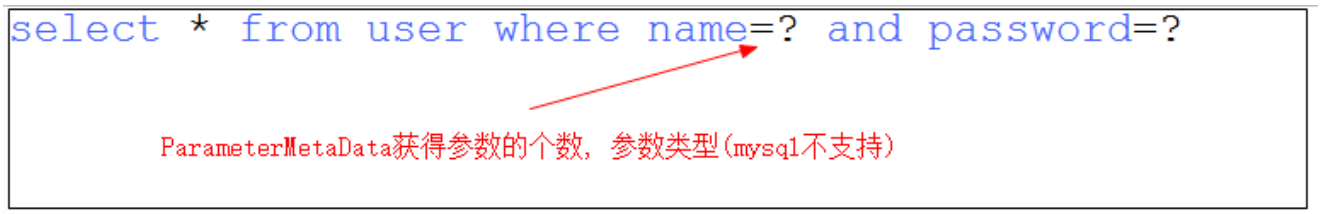
② 由ResultSet对象的getMetaData()方法获取的是ResultSetMetaData对象。

2.通过JDBC获得元数据

2.1ParameterMetaData

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



获得ParameterMetaData:

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

2.1.2ParameterMetaData相关的API

- int getParameterCount(); 获得参数个数
- 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)

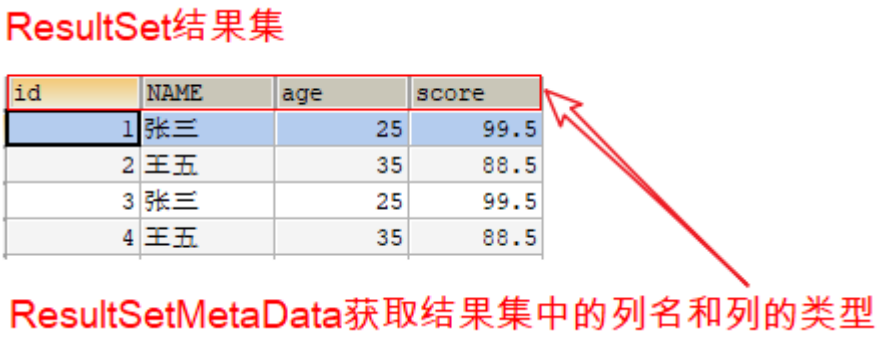
2.1.3实例代码

```
        //3. 获得参数的元数据
        ParameterMetaData parameterMetaData = preparedStatement.getParameterMetaData();
        // 获得sql语句里面参数的个数
        int parameterCount = parameterMetaData.getParameterCount();
        System.out.println("parameterCount="+parameterCount);
```

2.2ResultSetMetaData

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。



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
- getColumnTypeName();获取指定列的SQL类型
- getColumnClassName();获取指定列SQL类型对应于Java的类型

2.2.3实例代码

```
        //4. 获得结果的元数据对象
         ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

         // 获得结果集里面列的数量
         int columnCount = resultSetMetaData.getColumnCount();
         System.out.println("columnCount="+columnCount);//4

         // 获得某一列的列名 eg: 获得第二列
         String columnName = resultSetMetaData.getColumnName(2);
         System.out.println("columnName="+columnName);

         // 获得第三列的sql类型 varchar
         String columnTypeName = resultSetMetaData.getColumnTypeName(3);
         System.out.println(columnTypeName);

         // 获得第三列对应的java类型 String
         String columnClassName = resultSetMetaData.getColumnClassName(3);
         System.out.println("columnClassName="+columnClassName);
```

3.自定义JDBC框架

```java
public class MyJdbcTemplate {

    //1. 需要传入数据源(获得连接, 使用者不需要关注连接的)
    private DataSource dataSource;

    public MyJdbcTemplate(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    /**  封装了JDBC操作数据库的步骤+元数据, 释放资源(使用者不需要关注释放资源了)
     * 进行增,删,修改
     * @param sql  sql语句
     * @param params 参数
     * @return
     * Ctrl+Alt+T
     */
    public int update(String sql, Object ... params){
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        try {
            //0. 非空判断
            if(dataSource == null){
                throw  new RuntimeException("dataSource must not null...");
            }

            if(sql == null){
                throw  new RuntimeException("sql must not null...");
            }

            //1. 从dataSource 获得连接对象
            connection = dataSource.getConnection();
            //2. 创建预编译的sql语句对象 insert into user values (?,?,?,?)
            preparedStatement = connection.prepareStatement(sql);

            //3. 获得参数的元数据对象
            ParameterMetaData parameterMetaData = preparedStatement.getParameterMetaData();
            //4. 获得参数的个数
            int parameterCount = parameterMetaData.getParameterCount();
            //5. 给每一个?赋值
            for(int i = 0; i < parameterCount ;i++){
                preparedStatement.setObject(i+1,params[i]);
            }

            //6. 执行
            int i = preparedStatement.executeUpdate();
            return  i;
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            //释放资源
            C3P0Utils.release(null,preparedStatement,connection);
        }
```

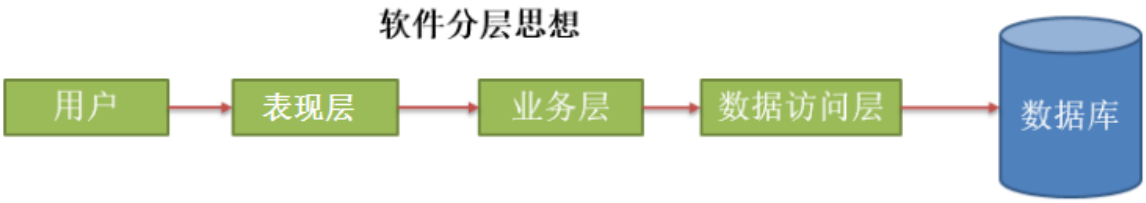
```
            return  -1;
        }
    }
```

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