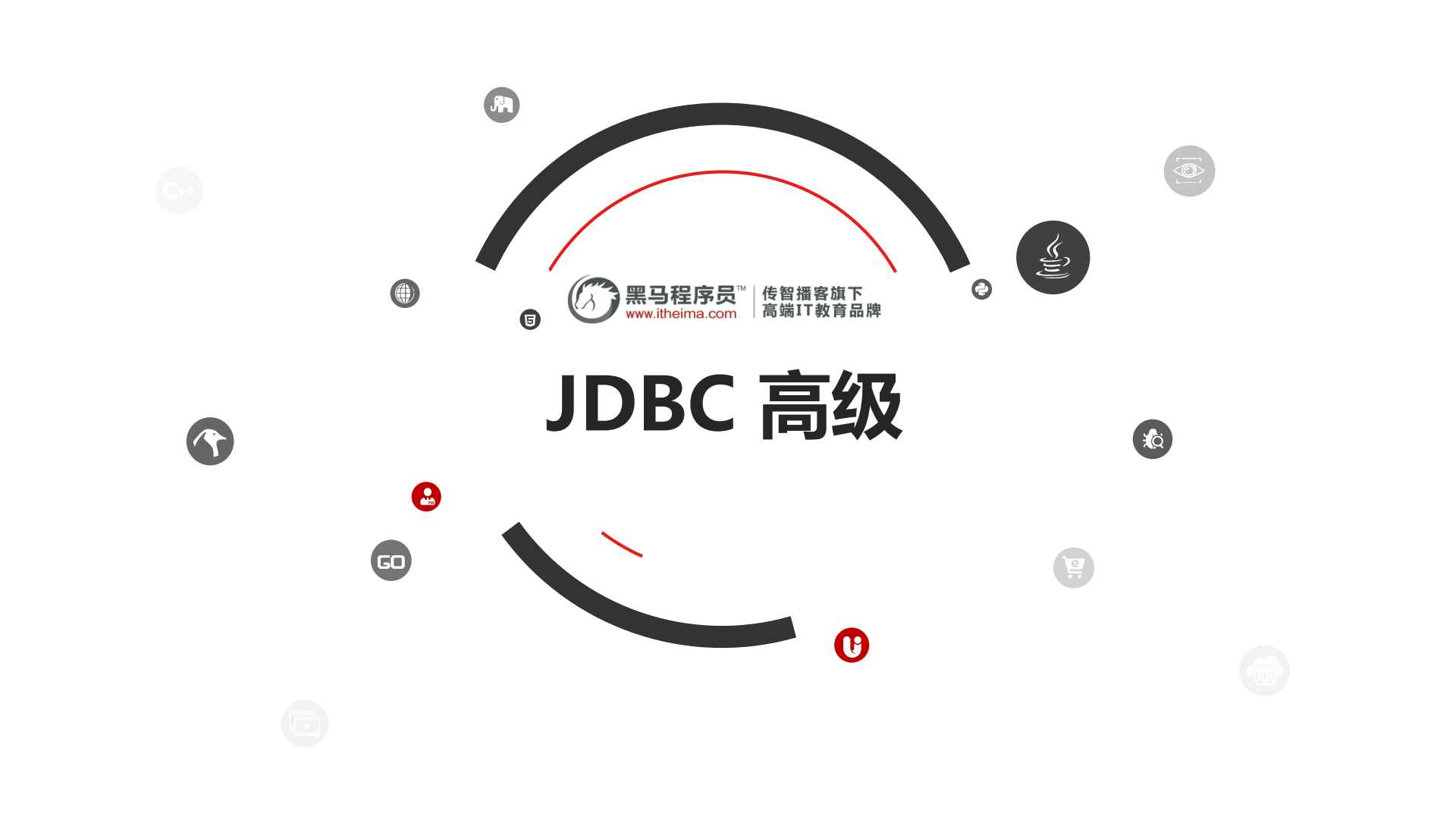


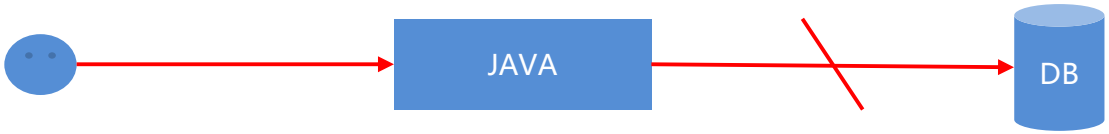
JDBC 高级

目录 Contents

数据库连接池

数据库连接池的概念

1. 数据库连接的背景
   - 数据库连接是一种关键的、有限的、昂贵的资源，这一点在多用户的网页应用程序中体现得尤为突出
   - 对数据库连接的管理能显著影响到整个应用程序的性能指标，数据库连接池正是针对这个问题提出来的

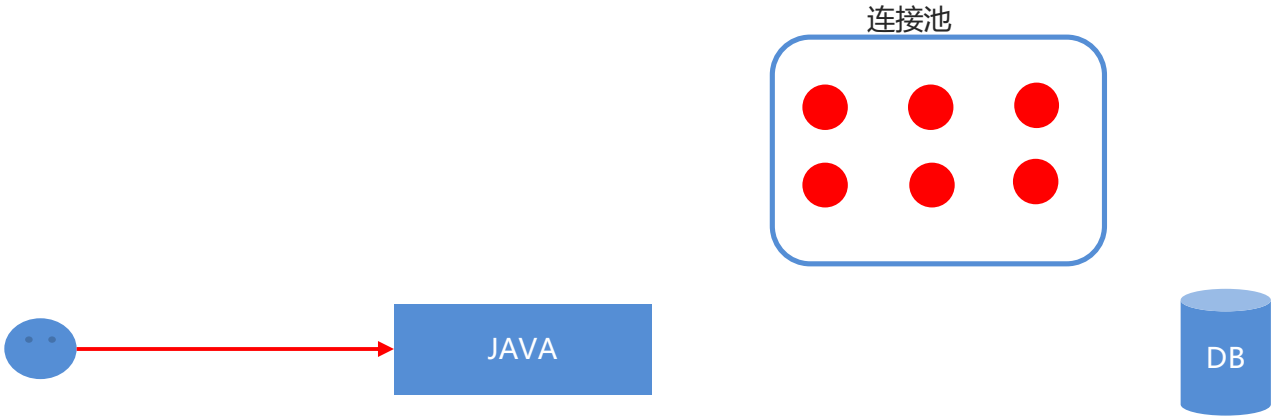


数据库连接池

数据库连接池的概念

2. 数据库连接池

- 数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是再重新建立一个。这项技术能明显提高对数据库操作的性能

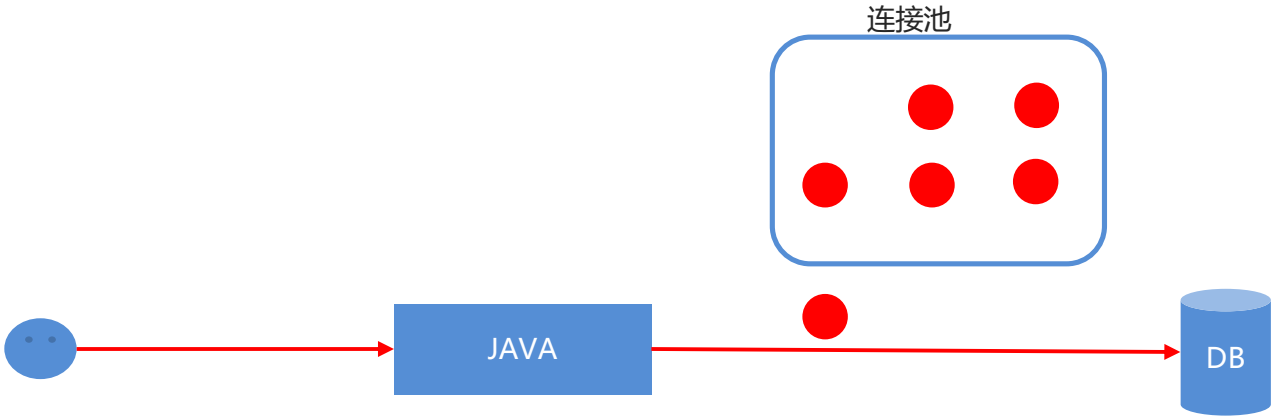


数据库连接池

数据库连接池的概念

2. 数据库连接池

- 数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是再重新建立一个。这项技术能明显提高对数据库操作的性能

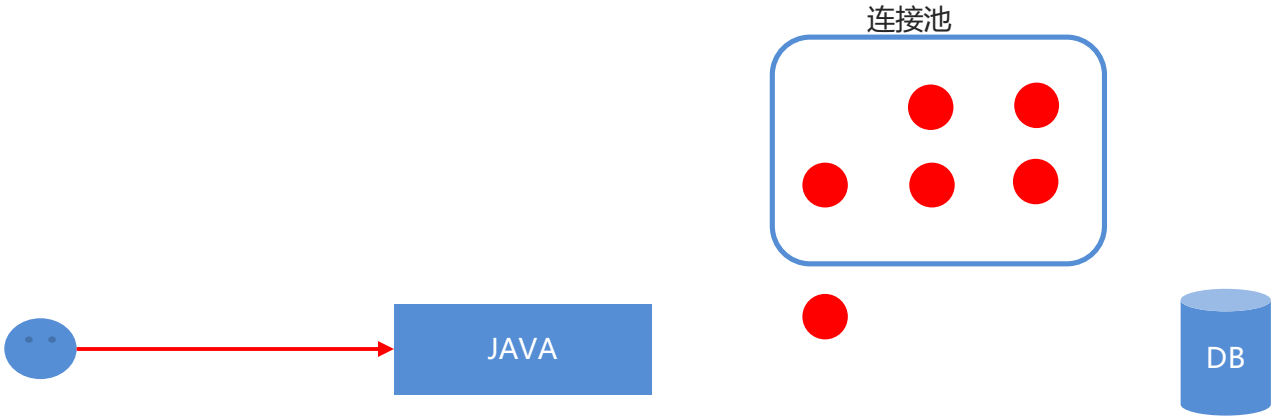


数据库连接池

数据库连接池的概念

2. 数据库连接池

- 数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是再重新建立一个。这项技术能明显提高对数据库操作的性能

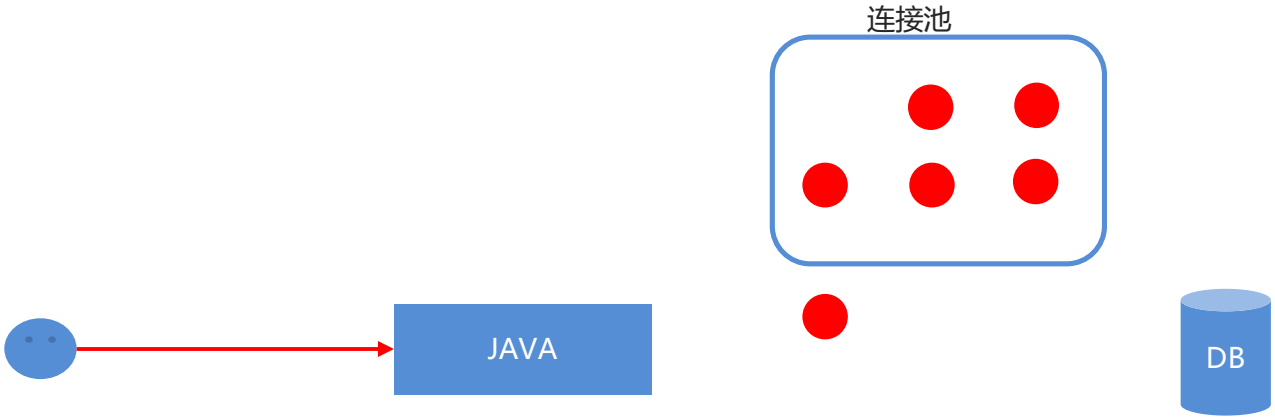


数据库连接池

数据库连接池的概念

2. 数据库连接池

- 数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是再重新建立一个。这项技术能明显提高对数据库操作的性能



目录 Contents

自定义数据库连接池

DataSource

1. DataSource 接口概述
   - javax.sql.DataSource 接口：数据源(数据库连接池)。Java 官方提供的数据库连接池规范(接口)
   - 如果想完成数据库连接池技术，就必须实现 DataSource 接口
   - 核心功能：获取数据库连接对象：Connection getConnection();

自定义数据库连接池

DataSource

2. 自定义数据库连接池

① 定义一个类，实现 DataSource 接口。

② 定义一个容器，用于保存多个 Connection 连接对象。

③ 定义静态代码块，通过 JDBC 工具类获取 10 个连接保存到容器中。

④ 重写 getConnection 方法，从容器中获取一个连接并返回。

⑤ 定义 getSize 方法，用于获取容器的大小并返回。

3. 自定义数据库连接池的测试

- 通过自定义数据库连接池完成查询学生表的全部信息

归还连接

归还数据库连接的方式

- 继承方式
- 装饰设计模式
- 适配器设计模式
- 动态代理方式

自定义数据库连接池

归还连接-继承方式

1. 继承方式归还数据库连接的思想。
   - 通过打印连接对象，发现 DriverManager 获取的连接实现类是 JDBC4Connection
   - 那我们就可以自定义一个类，继承 JDBC4Connection 这个类，重写 close() 方法，完成连接对象的归还

2. 继承方式归还数据库连接的实现步骤。
   ① 定义一个类，继承 JDBC4Connection 。
   ② 定义 Connection 连接对象和连接池容器对象的成员变量。
   ③ 通过有参构造方法完成对成员变量的赋值。
   ④ 重写 close 方法，将连接对象添加到池中。

3. 继承方式归还数据库连接存在的问题。
   - 通过查看 JDBC 工具类获取连接的方法发现：我们虽然自定义了一个子类，完成了归还连接的操作。但是 DriverManager 获取的还是 JDBC4Connection 这个对象，并不是我们的子类对象，而我们又不能整体去修改驱动包中类的功能，所继承这种方式行不通！

自定义数据库连接池

归还连接-装饰设计模式

1. 装饰设计模式归还数据库连接的思想。
   - 我们可以自定义一个类，实现 Connection 接口。这样就具备了和 JDBC4Connection 相同的行为了
   - 重写 close() 方法，完成连接的归还。其余的功能还调用 mysql 驱动包实现类原有的方法即可

2. 装饰设计模式归还数据库连接的实现步骤。
   ① 定义一个类，实现 Connection 接口
   ② 定义 Connection 连接对象和连接池容器对象的成员变量
   ③ 通过有参构造方法完成对成员变量的赋值
   ④ 重写 close() 方法，将连接对象添加到池中
   ⑤ 剩余方法，只需要调用 mysql 驱动包的连接对象完成即可
   ⑥ 在自定义连接池中，将获取的连接对象通过自定义连接对象进行包装

3. 装饰设计模式归还数据库连接存在的问题。
   - 实现 Connection 接口后，有大量的方法需要在自定义类中进行重写

自定义数据库连接池

归还连接-适配器设计模式

1. 适配器设计模式归还数据库连接的思想。
   - 我们可以提供一个适配器类，实现 Connection 接口，将所有方法进行实现(除了close方法)
   - 自定义连接类只需要继承这个适配器类，重写需要改进的 close() 方法即可

2. 适配器设计模式归还数据库连接的实现步骤。
   ① 定义一个适配器类，实现 Connection 接口。
   ② 定义 Connection 连接对象的成员变量。
   ③ 通过有参构造方法完成对成员变量的赋值。
   ④ 重写所有方法(除了 close)，调用mysql驱动包的连接对象完成即可。
   ⑤ 定义一个连接类，继承适配器类。
   ⑥ 定义 Connection 连接对象和连接池容器对象的成员变量，并通过有参构造进行赋值。
   ⑦ 重写 close() 方法，完成归还连接。
   ⑧ 在自定义连接池中，将获取的连接对象通过自定义连接对象进行包装。

3. 适配器设计模式归还数据库连接存在的问题。
   - 自定义连接类虽然很简洁了，但适配器类还是我们自己编写的，也比较的麻烦

自定义数据库连接池

动态代理

- 动态代理：在不改变目标对象方法的情况下对方法进行增强
- 组成

  被代理对象：真实的对象

  代理对象：内存中的一个对象
- 要求

  代理对象必须和被代理对象实现相同的接口
- 实现

  Proxy.newProxyInstance()

归还连接-动态代理方式

1. 动态代理方式归还数据库连接的思想。
   - 我们可以通过 Proxy 来完成对 Connection 实现类对象的代理
   - 代理过程中判断如果执行的是 close 方法，就将连接归还池中。如果是其他方法则调用连接对象原来的功能即可
2. 动态代理方式归还数据库连接的实现步骤。
   ① 定义一个类，实现 DataSource 接口
   ② 定义一个容器，用于保存多个Connection连接对象
   ③ 定义静态代码块，通过 JDBC 工具类获取 10 个连接保存到容器中
   ④ 重写 getConnection 方法，从容器中获取一个连接
   ⑤ 通过 Proxy 代理，如果是 close 方法，就将连接归还池中。如果是其他方法则调用原有功能
   ⑥ 定义 getSize 方法，用于获取容器的大小并返回

3. 动态代理方式归还数据库连接存在的问题。
   - 我们自己写的连接池技术不够完善，功能也不够强大

目录 Contents

开源数据库连接池

开源数据库连接池的使用

1. C3P0 数据库连接池的使用步骤。
   ① 导入 jar 包。
   ② 导入配置文件到 src 目录下。
   ③ 创建 C3P0 连接池对象。
   ④ 获取数据库连接进行使用。

注意：C3P0 的配置文件会自动加载，但是必须叫 c3p0-config.xml 或 c3p0-config.properties 。

开源数据库连接池的使用

2. Druid 数据库连接池的使用步骤。
   ① 导入 jar 包。
   ② 编写配置文件，放在 src 目录下。
   ③ 通过 Properties 集合加载配置文件。
   ④ 通过 Druid 连接池工厂类获取数据库连接池对象。
   ⑤ 获取数据库连接进行使用。

注意：Druid 不会自动加载配置文件，需要我们手动加载，但是文件的名称可以自定义。

目录 Contents

自定义JDBC框架

框架的背景

1. 分析案例中的重复代码。

定义必要的信息、获取数据库的连接、释放资源都是重复的代码，而我们最终的核心功能仅仅只是执行一条 sql 语句，所以我们可以抽取出一个 JDBC 模板类，来封装一些方法(update、query)，专门帮我们执行增删改查的 sql 语句。将之前那些重复的操作，都抽取到模板类中的方法里，就能大大简化我们的使用步骤！

框架的背景

2. 源信息。

DataBaseMetaData：数据库的源信息（了解）

java.sql.DataBaseMetaData 封装了整个数据库的综合信息。

例如：

- String getDatabaseProductName()：获取数据库产品的名称
- int getDatabaseProductVersion()：获取数据库产品的版本号

框架的背景

2. 源信息。

ParameterMetaData：参数的源信息

- java.sql.ParameterMetaData 封装的是预编译执行者对象中每个参数的类型和属性，这个对象可以通过预编译执行者对象中的 getParameterMetaData() 方法来获取
- 核心功能： int getParameterCount() 用于获取 sql 语句中参数的个数

框架的背景

2. 源信息。

ResultSetMetaData：结果集的源信息

- java.sql.ResultSetMetaData：封装的是结果集对象中列的类型和属性，这个对象可以通过结果集对象中的getMetaData()方法来获取
- 核心功能：int getColumnCount() 用于获取列的总数，String getColumnName(int i) 用于获取列名

自定义JDBC框架

框架的编写

1. 用于执行增删改功能的 update() 方法
   ① 定义所需成员变量(数据源、数据库连接、执行者、结果集)。
   ② 定义有参构造，为数据源对象赋值。
   ③ 定义 update() 方法，参数：sql 语句、sql 语句所需参数。
   ④ 定义 int 类型变量，用于接收 sql 语句执行后影响的行数。
   ⑤ 通过数据源获取一个数据库连接。
   ⑥ 通过数据库连接对象获取执行者对象并对 sql 语句预编译。
   ⑦ 通过执行者对象获取 sql 语句中参数的源信息对象。
   ⑧ 通过参数源信息对象获取 sql 语句中参数的个数。
   ⑨ 判断参数个数是否一致。
   ⑩ 为 sql 语句中 ? 占位符赋值。
   ⑪ 执行 sql 语句并接收结果。
   ⑫ 释放资源。
   ⑬ 返回结果。

框架的编写

2. update() 方法的测试。
   ① 定义测试类，模拟 dao 层。
   ② 测试执行 insert 语句。
   ③ 测试执行 update 语句。
   ④ 测试执行 delete 语句。

自定义JDBC框架

框架的编写

3. 用于执行查询功能的方法介绍。

- 查询一条记录并封装对象的方法：queryForObject()
- 查询多条记录并封装集合的方法：queryForList()
- 查询聚合函数并返回单条数据的方法：queryForScalar()

框架的编写

4. 实体类的编写。

   定义一个类，提供一些成员变量。

注意：成员变量的数据类型和名称要和表中的列保持一致。

自定义JDBC框架

框架的编写

5. 处理结果集的接口。

① 定义泛型接口 ResultSetHandler<T> 。

② 定义用于处理结果集的泛型方法 <T> T handler(ResultSet rs) 。

注意：此接口仅用于为不同处理结果集的方式提供规范，具体的实现类还需要自行编写。

框架的编写

6. 处理结果集的接口实现类 1。

① 定义 BeanHandler<T> 类实现 ResultSetHandler<T> 接口。

② 定义 Class 对象类型的变量。

③ 定义有参构造为变量赋值。

④ 重写 handler() 方法，用于将结果集中的一条记录封装到自定义对象中并返回。

⑤ 声明自定义类型对象。

⑥ 创建传递参数的对象，为自定义对象赋值。

⑦ 判断结果集中是否有数据。

⑧ 通过结果集对象获取结果集的源信息对象。

⑨ 通过结果集源信息对象获取列数。

⑩ 通过循环遍历列数。

⑪ 通过结果集源信息获取列名。

⑫ 通过列名获取该列的数据。

⑬ 创建属性描述器对象，将获取到的值通过对象的 set() 方法进行赋值。

⑭ 返回封装好的对象。

自定义JDBC框架

框架的编写

7. 用于查询一条记录并封装对象的方法 queryForObject() 。

① 定义方法 queryForObject()，参数：sql 语句、处理结果集接口、sql 语句中的参数。

② 声明自定义类型对象。

③ 通过数据源获取数据库连接对象。

④ 通过数据库连接对象获取执行者对象并对sql语句进行预编译。

⑤ 通过执行者对象获取参数源信息的对象。

⑥ 通过参数源信息对象获取参数的个数。

⑦ 判断参数数量是否一致。

⑧ 为 sql 语句中 ? 占位符赋值。

⑨ 执行 sql 语句并接收结果集。

⑩ 通过结果集接口对结果进行处理。

⑪ 释放资源。

⑫ 返回结果。

框架的编写

8. queryForObject() 方法的测试。

   测试查询一条记录并封装对象。

框架的编写

9. 处理结果集的接口实现类 2 。

① 定义 BeanListHandler<T> 类实现 ResultSetHandler<T> 接口。

② 定义 Class 对象类型的变量。

③ 定义有参构造为变量赋值。

④ 重写 handler 方法，用于将结果集中的所有记录封装到集合中并返回。

⑤ 创建 List 集合对象。

⑥ 遍历结果集对象。

⑦ 创建传递参数的对象。

⑧ 通过结果集对象获取结果集的源信息对象。

⑨ 通过结果集源信息对象获取列数。

⑩ 通过循环遍历列数。

⑪ 通过结果集源信息获取列名。

⑫ 通过列名获取该列的数据。

⑬ 创建属性描述器对象，将获取到的值通过对象的 set() 方法进行赋值。

⑭ 将封装好的对象添加到集合中。

⑮ 返回集合对象。

自定义JDBC框架

框架的编写

10. 用于查询多条记录并封装集合的方法 queryForList()。
    ① 定义方法 queryForList()，参数：sql 语句、处理结果集接口、sql 语句中的参数。
    ② 创建集合对象。
    ③ 通过数据源获取数据库连接对象。
    ④ 通过数据库连接对象获取执行者对象并对sql语句进行预编译。
    ⑤ 通过执行者对象获取参数源信息的对象。
    ⑥ 通过参数源信息对象获取参数的个数。
    ⑦ 判断参数数量是否一致。
    ⑧ 为 sql 语句中？占位符赋值。
    ⑨ 执行 sql 语句并接收结果集。
    ⑩ 通过结果集接口对结果进行处理。
    ⑪ 释放资源。
    ⑫ 返回结果。

自定义JDBC框架

框架的编写

11. queryForList() 方法的测试。

    测试查询多条记录并封装集合。

自定义JDBC框架

框架的编写

12. 处理结果集的接口实现类 3 。

① 定义 ScalarHandler<T> 类实现 ResultSetHandler<T> 接口。

② 重写 handler() 方法，用于返回一个聚合函数的查询结果。

③ 定义 Long 类型变量。

④ 判断结果集对象是否有数据。

⑤ 通过结果集对象获取结果集源信息的对象。

⑥ 通过结果集源信息对象获取第一列的列名。

⑦ 通过列名获取该列的数据。

⑧ 将结果返回。

自定义JDBC框架

框架的编写

13. 用于执行聚合函数 sql 语句的方法 queryForScalar() 。

① 定义方法 queryForScalar()，参数：sql 语句、处理结果集接口、sql 语句中的参数。
② 创建 Long 类型变量。
③ 通过数据源获取数据库连接对象。
④ 通过数据库连接对象获取执行者对象并对 sql 语句进行预编译。
⑤ 通过执行者对象获取参数源信息的对象。
⑥ 通过参数源信息对象获取参数的个数。
⑦ 判断参数数量是否一致。
⑧ 为 sql 语句中？占位符赋值。
⑨ 执行 sql 语句并接收结果集。
⑩ 通过结果集接口对结果进行处理。
⑪ 释放资源。
⑫ 返回结果。

自定义JDBC框架

框架的编写

14. queryForScalar() 方法的测试。

    测试执行一条聚合函数的 sql 语句。

自定义JDBC框架

框架的分析

分析自定义框架中目前存在的问题，为 MyBatis 的学习做铺垫。

传智播客旗下高端IT教育品牌