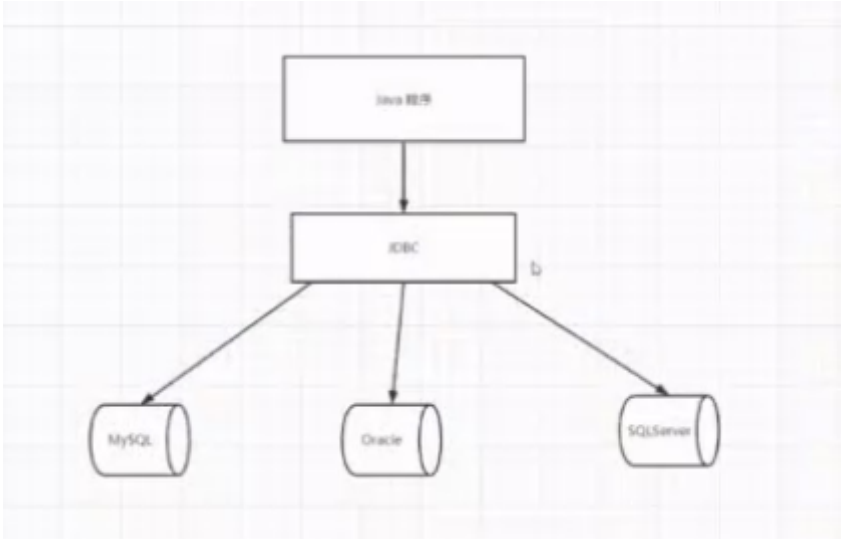
1.JDBC

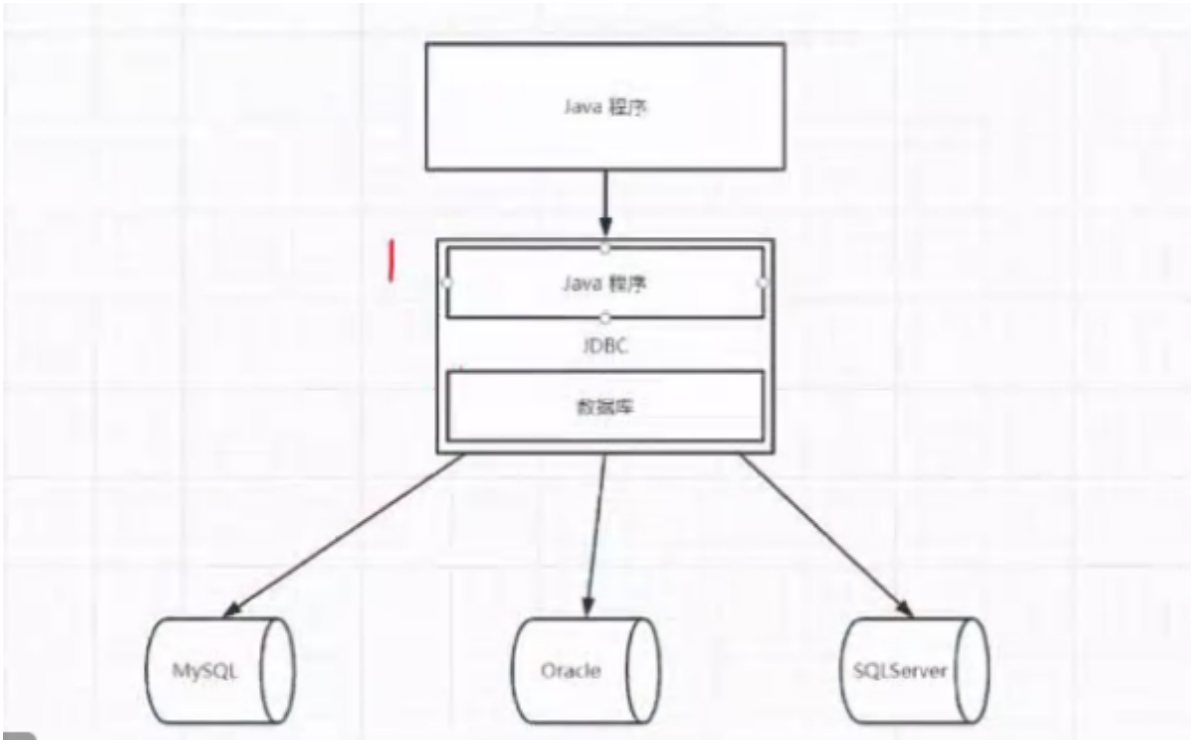
1.1JDBC是什么

java Database Connectivity是一个独立于特定数据库管理系统，通用的SQL数据库存取和操作的公共接口，定义了一组标准，为访问不同的数据库提供了统一的途径

JDBC支持多种数据库，连通java程序和数据库



1.2 JDBC的体系结构



jdbc接口包括两个层面：

1.面向应用的API，供程序员调用

2.面向数据库厂商的API，供数据库厂商开发的驱动程序

jdbc的API：java提供的，供程序员调用的接口，类

在java.sql包和Javax.sql包中

DriverManager类

Connection接口

Statement接口

ResultSet接口

JDBC驱动：数据库厂商提供的，负责连接各种不同的数据库

MYSQL驱动

Oracle驱动

SQLServer驱动

1.3JDBC使用流程

1.加载驱动，Java程序和数据库的桥梁

2.创建Connection一次连接

3.Statement，由Connection产生，执行sql语句

4.ReslutSet保存Statement执行后所产生的结果，将数据库记录映射成java对象

1.4具体操作

entity：

```
public class Account {
    private  String name;
    private Integer money;


    public Account(String name, Integer money) {
        this.name = name;
        this.money = money;
    }

    @Override
    public String toString() {
        return "Account{" +
                "name='" + name + '\'' +
                ", money=" + money +
                '}';
    }
```

JDBCUtil:

```
public class JDBCUtil {
    public static Connection getConnection(){
//        加载驱动
        Connection connection =null;
        Statement statement = null;
        ResultSet resultSet = null;
```

```
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
//            创建连接
            String  url = "jdbc:mysql://localhost:3306/test?serverTimezone=UTC";
            String user = "root";
            String pwd = "123456";
            connection = DriverManager.getConnection(url,user,pwd);


        } catch (Exception e) {
            e.printStackTrace();
        }
        return connection;
    }

//    资源释放
    public static  void release(Connection  connection,Statement
statement,ResultSet resultSet){
        try{
            if (connection!=null) connection.close();
            if (statement!=null) statement.close();
            if(resultSet!=null) resultSet.close();
        }catch (SQLException e){
            e.printStackTrace();

        }

    }
}
```

CRUD

```
public class Test {
    public static void main(String[] args) {
//        query();
//        add("张三",1200);
//        update("张三",5000);
        delete("张三");

    }


//    删除操作
    public static void delete(String name){
        Connection connection = JDBCUtil.getConnection();
        String s = "delete  from account where name = '"+name+"'";
        try {
            Statement statement = connection.createStatement();
            int i = statement.executeUpdate(s);
            System.out.println(i);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

//    修改操作
    public static void update(String name,Integer money){
```

```
        Connection connection = JDBCUtil.getConnection();
        String s = "Update account set money='"+money+"'where name =
'"+name+"'";
        try {
            Statement statement = connection.createStatement();
            int i = statement.executeUpdate(s);
            System.out.println(i);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
//    添加操作
    public static void add(String name,Integer money){
        Connection connection = JDBCUtil.getConnection();
        String s = "Insert into account(name,money)
values('"+name+"','"+money+"')";
        try {
            Statement statement = connection.createStatement();
            int i = statement.executeUpdate(s);
            System.out.println(i);
        } catch (SQLException e) {
            e.printStackTrace();
        }


    }

//    查询操作
    public static void query(){
        Connection connection = JDBCUtil.getConnection();
        String s = "select * from account;";
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(s);
            while(resultSet.next()){
                int anid = resultSet.getInt("money");
                String aname = resultSet.getString("name");
                Account account = new Account(aname, anid);
                System.out.println(account);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }
}
```

1.加载驱动

2.执行sql

ResultSet excuteQuery(String sql) 适用于查询操作，返回的结果是对象集对象

int excuteUpdate(String sql) 适用于增删改操作，返回的是影响的行数

boolean excute（String sql） 适用于增删改查操作，true表示返回的结果是ResultSet，false表示返回的结果不是ResultSet

1.5 PreparedStatement

Statement的子类，闭端：

1.频繁地拼接SQL,出错率较高，不利于程序的开发

2.拼接的SQL的方式存在SQL注入的风险，对系统安全性是一个隐患

SQL注入

利用某些系统没有对用户输入的数据进行充分检查，在用户输入的数据中注入非法的sql语句，从而利用系统的sql引擎完成恶意行为的做法

登录：正常情况下知道正确的用户名和密码才能登陆

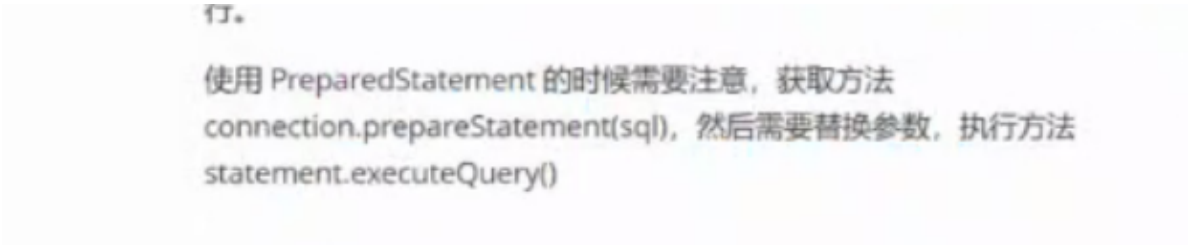
SQL注入：在不知道用户名和密码的情况下仍然能登陆成功

使用PreparedStatement可以解决上述问题，因为PreparedStatement提供了SQL占位符的功能，可以避免SQL注入风险

不需要手动拼接sql

```
public static void login(String name,String pwd){
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    try {
        connection = JDBCUtil.getConnection();
        //定义SQL语句
        String sql = "select * from user where name = ? and pwd = ?";
        //执行SQL
        statement = connection.prepareStatement(sql);
        statement.setString(1, name);
        statement.setString(2, pwd);
        resultSet = statement.executeQuery();
        if (resultSet.next()) System.out.println("登录成功");
        else System.out.println("登录失败");
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        JDBCUtil.release(connection, statement, resultSet);
    }
}
```

既可以避免反复凭借SQL语句的问题，还可以避免sql注入的风险，所以实际开发中选择使用PreparedStatement来完成SQL执行



1.6JDBC如何操作事务

1.关闭Connection的自动提交

2.捕获异常，在异常处理中让数据进行回滚

3.如果没有异常，再进行事务提交

```
public static void transactionTest(){
    Connection connection = null;
    PreparedStatement statement1 = null;
    PreparedStatement statement2 = null;
    ResultSet resultSet = null;
    try {
        connection = JDBCUtil.getConnection();
        //定义SQL语句
        String sql = "update user set money = ? where id = ?";
        //关闭Connection自动提交
        connection.setAutoCommit(false);
        statement1 = connection.prepareStatement(sql);
        statement2 = connection.prepareStatement(sql);
        statement1.setInt(1, 500);
        statement1.setInt(2, 1);
        statement2.setInt(1, 1500);
        statement2.setInt(2, 2);
        statement1.executeUpdate();
        int num = 10/0;
        statement2.executeUpdate();
        //提交事务
        connection.commit();
    } catch (Exception e) {
        e.printStackTrace();
        try {
            //回滚
            connection.rollback();
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    } finally {
        JDBCUtil.release(connection, statement1, resultSet);
        JDBCUtil.release(connection, statement2, resultSet);
    }
}
```

1.7数据库连接池

JDBC开发流程：

1.建立数据库连接

2.执行sql

3.断开数据库连接，销毁资源

JDBC数据库连接使用DriverManager来获取，每次都需要向数据库申请获取连接，每次都要验证用户名和密码，执行完sql之后，断开连接并销毁资源，这样的方式会消耗大量的资源和时间，数据库连接资源并没有得到很好的重复利用，造成了资源的极大浪费，为了解决这一问题，可以使用数据库连接池

数据库连接池的基本思想就是为数据库建立一个缓冲池，预先向缓冲池中存入一定数量的连接对象，当需要获取数据库连接的时候，只需要从缓冲池中取出一个使用即可，用完之后再放回到缓冲池中，供下一次请求使用，这样就可以做到资源的重复利用，允许应用程序重复使用一个现有的数据库对象，而不是每次都重新创建一个连接对象

数据库连接池在初始化的时候会创建一定数量的连接对象，当数据库连接池中没有空闲的连接时，请求会进入等待队列，等待其他线程释放连接对象

JDBC的数据库连接池使用javax.sql.DataSource来完成，DataSource是一个接口，java官方提供的数据库连接池接口。

C3p0是一个第三方的DataSource实现

使用步骤：

1.导入jar包

2.创建c3p0对象

```
setInitialPoolSize：初始化缓冲池大小
setMaxPoolSize：最大连接数
setMinPoolSize：最小连接数
setAcquireIncrement：单次增加的连接数
```

com.mysql.cj.jdbc.ConnectionImpl@2a70a3d8

com.mchange.v2.c3p0.impl.NewProxyConnection@ed17bee

```
package com.southwind.util;

import com.mchange.v2.c3p0.ComboPooledDataSource;

import java.beans.PropertyVetoException;
import java.sql.*;

public class JDBCUtil {

    private static ComboPooledDataSource dataSource = null;

    static {
        try {
            dataSource = new ComboPooledDataSource();
            dataSource.setDriverClass("com.mysql.cj.jdbc.Driver");
            dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test1");
            dataSource.setUser("root");
            dataSource.setPassword("123456");
            dataSource.setInitialPoolSize(5);
            dataSource.setMaxPoolSize(10);
            dataSource.setMinPoolSize(3);
            dataSource.setAcquireIncrement(5);
        } catch (PropertyVetoException e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection(){
        Connection connection = null;
        try {
```

```
            connection = dataSource.getConnection();
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return connection;
    }

    public static void release(Connection connection,Statement
statement,ResultSet resultSet){
        try {
            if(connection != null) connection.close();
            if(statement != null) statement.close();
            if(resultSet != null) resultSet.close();
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }

}
```