# 面向对象与多线程综合实验

## 结合华为云平台的基于JAVASE数据挖掘系统设计与实现

# 实验内容

## 1. 做什么？

◆ 控制台数据记录功能

```
**************************
* 1、数据采集        2、数据匹配 *
* 3、数据记录        4、数据显示 *
* 5、数据发送        0、退出应用 *
**************************
请输入菜单项（0~5）：
```

分别将日志和物流记录保存到文件和MySQL数据库中并从文件和数据库中显示到控制台

## 2. 怎么做？

◆ 1. 搭建数据库基础环境

◆ 2. JDBC及系统框架

◆ 3. 系统功能实现

3.1匹配日志数据保存到数据库

3.2从数据库中读取匹配的日志数据

3.3 测试数据库存取功能

3.4 嵌入到控制台主界面

## 3. 对应项

◆ MySQL与SQLyog安装配置
◆ MySQL数据库操作

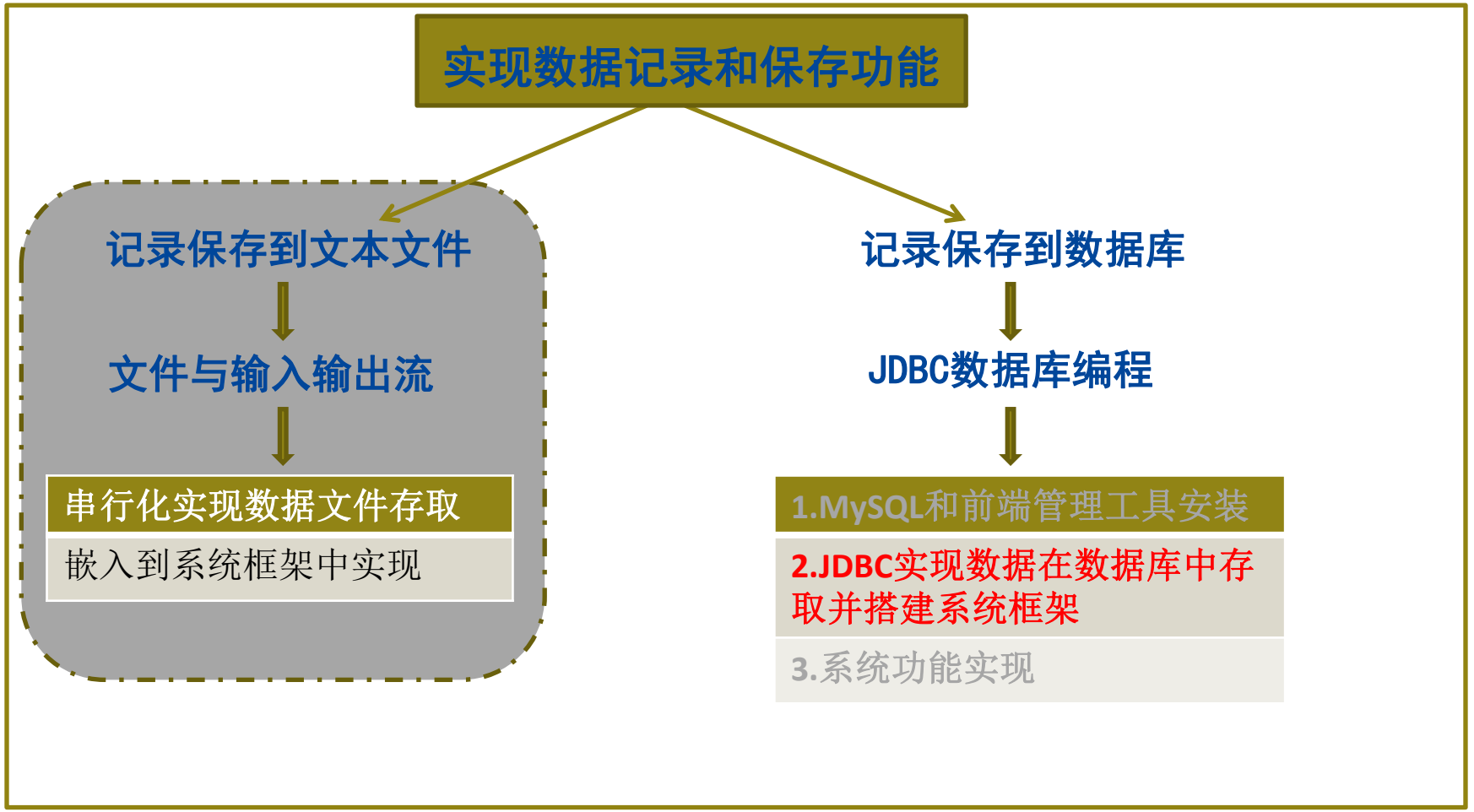
- db-》DBUtil.java
- util-》Config.java
- config/mysql.properties
- libs/mysql-connector

- saveMatchLogToDB.java
- readMatchedLogFromDB.java
- DBDemo.java
- MenuDriver.java

# 实验目标

1. 掌握JDBC的基本步骤，能够实现Java连接数据库的基本操作
2. 进一步熟悉系统框架

# 实验内容


实现数据记录和保存功能
记录保存到文本文件
文件与输入输出流
串行化实现数据文件存取
嵌入到系统框架中实现
记录保存到数据库
JDBC数据库编程
1.MySQL和前端管理工具安装
2.JDBC实现数据在数据库中存取并搭建系统框架
3.系统功能实现

# 实验要求与交付

要求：

1. JDBC中实现对数据库temp的存取操作

   Eclipse中新建一个项目test,将数据库驱动导入到该项目中，并编写JDBC程序连接数据库，实现在temp1数据表中新增记录，以及读取记录并在控制台中显示

2. 搭建第三次实验数据库系统框架

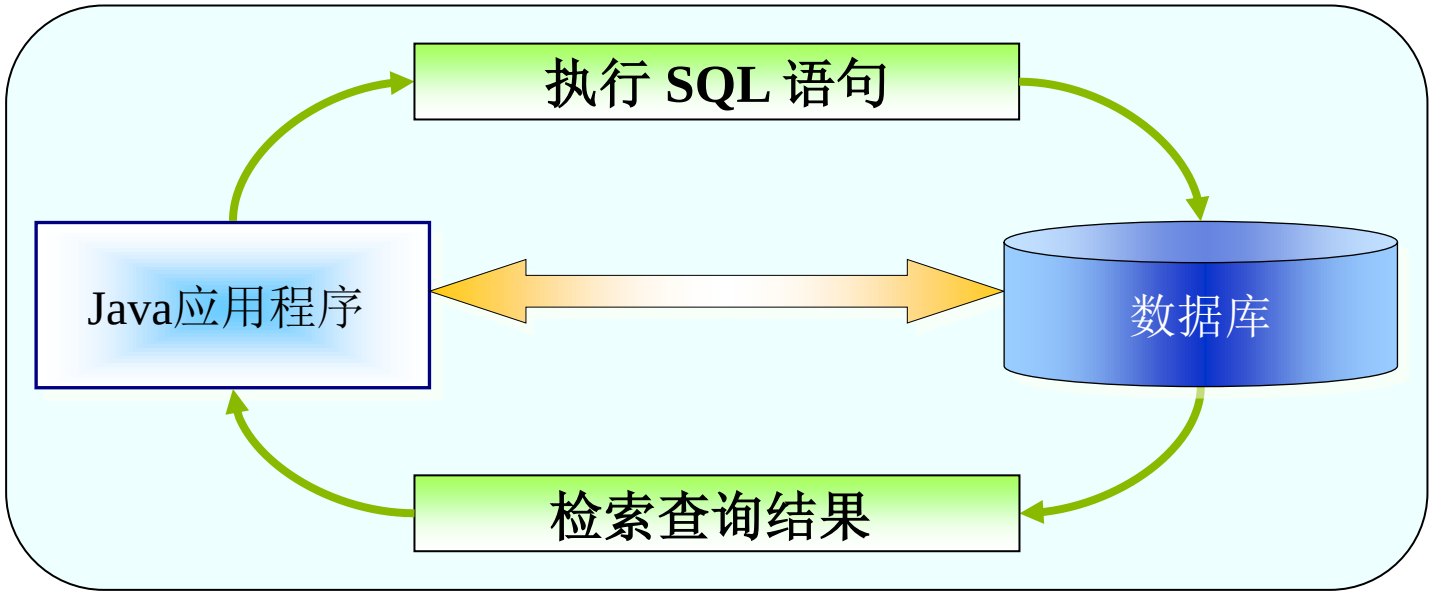
✱交付：课堂验收

# 一、Java实现数据库操作：JDBC

JDBC（Java Database Connectivity，JAVA数据库连接），是一种执行SQL语句的Java API，由一组用JAVA语言编写的类和接口组成。

通过JDBC可以对多种数据库进行统一的访问和操作。

JDBC



# Java实现数据库操作：JDBC

## JDBC实现对数据库进行访问和操作是由是由一系列的类和接口实现（API）

| API | 说明 |
| --- | --- |
| java.sql.Connection | 与特定数据库的连接（会话）。能够通过 getMetaData 方法获得数据库提供的信息、所支持的 SQL 语法、存储过程和此连接的功能等信息。代表了数据库。 |
| java.sql.Driver | 每个驱动程序类必需实现的接口，同时，每个数据库驱动程序都应该提供一个实现 Driver 接口的类。 |
| java.sql.DriverManager (Class) | 管理一组 JDBC 驱动程序的基本服务。作为初始化的一部分，此接口会尝试加载在"jdbc.drivers"系统属性中引用的驱动程序。只是一个辅助类，是工具。 |
| java.sql.Statement | 用于执行静态 SQL 语句并返回其生成结果的对象。 |
| java.sql.PreparedStatement | 继承 Statement 接口，表示预编译的 SQL 语句的对象，SQL 语句被预编译并且存储在 PreparedStatement 对象中。然后可以使用此对象高效地多次执行该语句。 |
| java.sql.CallableStatement | 用来访问数据库中的存储过程。它提供了一些方法来指定语句所使用的输入/输出参数。 |
| java.sql.ResultSet | 指的是查询返回的数据库结果集。 |
| java.sql.ResultSetMetaData | 可用于获取关于 ResultSet 对象中列的类型和属性信息的对象。 |

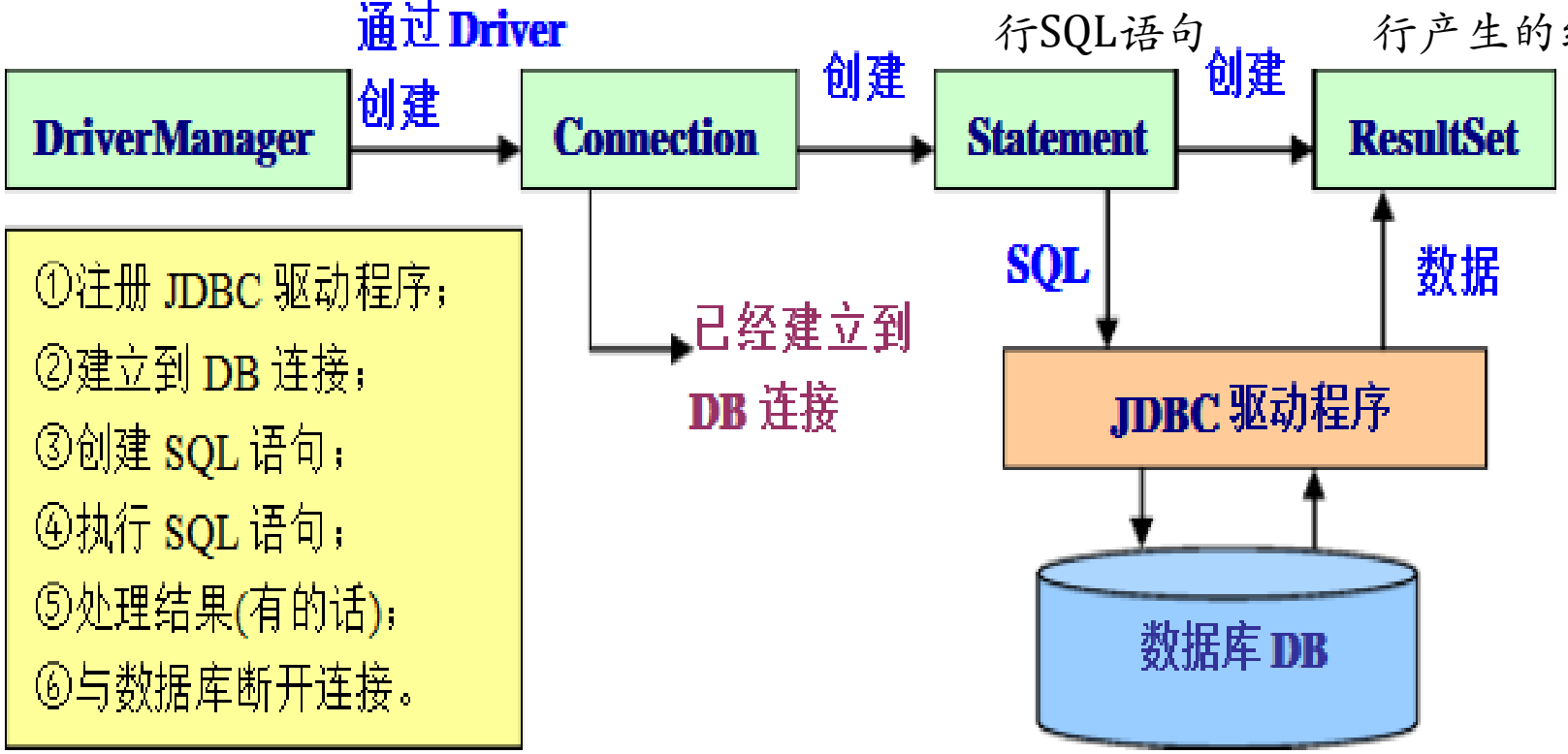
# JDBC程序访问数据库步骤

依据不同的数据库加载JDBC驱动

**连接对象**
连接数据库并传送数据

**语句对象**
由Connection产生，负责执行SQL语句

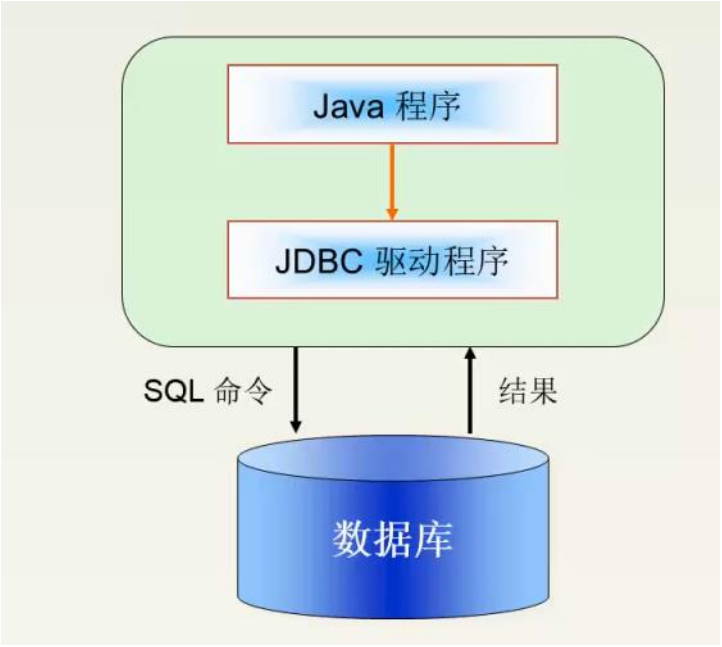
**结果集对象**
负责保存由Statement执行产生的结果

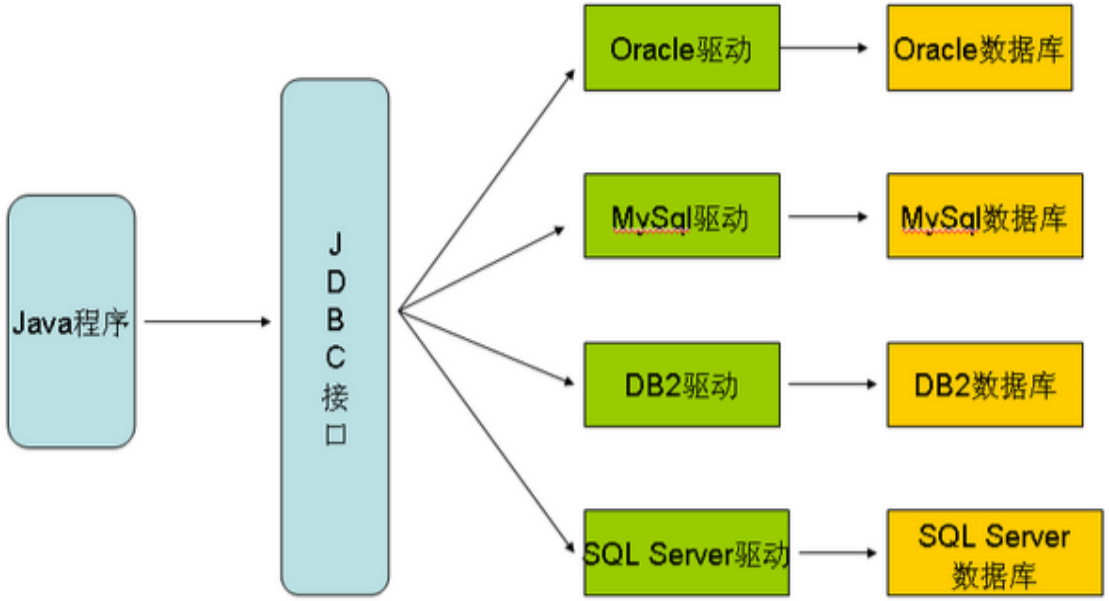



①注册 JDBC 驱动程序；
②建立到 DB 连接；
③创建 SQL 语句；
④执行 SQL 语句；
⑤处理结果(有的话)；
⑥与数据库断开连接。

# 1.1 加载数据库驱动

JAVA程序通过API访问不同的数据库时，需要不同数据库厂商提供相应的驱动程序。



# 1.1 加载数据库驱动

在数据库官网下载java连接数据库的驱动程序

通过Class.forName()方法加载MySQL驱动：

```
driver =com.mysql.jdbc.Driver


// 指定驱动程序
Class.forName(driver);
```

或者直接写成 Class.forName(“com.mysql.jdbc.Driver”)

# 1.2 建立数据库连接

在使用JDBC操作数据库之前，需要先创建一个数据库连接。

调用DriverManager.getConnection()方法，同时获得了connection对象，建立了MySQL数据库连接

```
Connection conn = null;
// 建立数据库连接
conn = DriverManager.getConnection(url, user, pwd);
```

例如：

```
url = jdbc:mysql://localhost:3306/q_dms?useUnicode=true&characterEncoding=UTF-8
user = root
password =
serverIP=127.0.0.1
```

- url：数据库连接字符串：其中localhost是数据库服务器的IP地址，代指本机，还可以用127.0.0.1，3306是端口号，q_dms是数据库实例名
- user：连接数据库的用户名
- password：密码

# 1.3 创建并执行SQL语句

成功连接到数据库，获得Connection对象后，通过Connection对象的createStatement方法来创建语句对象statement；

```
//创建语句对象
Statement st=conn.createStatement();
```

使用语句对象Statement来执行SQL语句，有两种情况：

一种是执行DELETE、UPDATE和INSERT之类的数据库操作语句（DML），这样的语句没有数据结果返回，使用Statement对象的executeUpdate方法

```
//执行数据库操作，处理查询结果
  //1.数据库增删改操作，不需要返回结果集
String sql="insert into temp1 values('002','张三','南京')";
st.executeUpdate(sql);
```

# 1.4 结果集Resultset

使用语句对象Statement来执行SQL语句，有两种情况：

另一种是执行SELECT这样的数据查询语句（DQL），这样的语句将从数据库中获得所需的数据，使用Statement对象的executeQuery方法

获得所需数据在ResultSet结果集对象中

访问结果集中的数据：按行记录指针next()方法,获取当行的数据get()方法

```
  //2.数据库查询操作，需要返回ResultSet结果集
String sql_query="select* from temp1";
ResultSet rs=st.executeQuery(sql_query);
     //操作结果集对象，使用next(),get()方法
     while(rs.next())
     {
          int id=rs.getInt(1);
          String name=rs.getString(2);
          String address=rs.getString(3);
          System.out.println("编号="+id);
          System.out.println("姓名="+name);
          System.out.println("地址="+address);
     }
```

# 1.5 关闭数据库连接

## 调用close()方法关闭所有创建的对象

```
 * 释放资源
 */
public void closeAll() {
    // 如果rs不空，关闭rs
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    // 如果pstmt不空，关闭pstmt
    if (pstmt != null) {
        try {
            pstmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    // 如果conn不空，关闭conn
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

Connection对象、Statement对象和ResultSet对象都有执行关闭的close方法

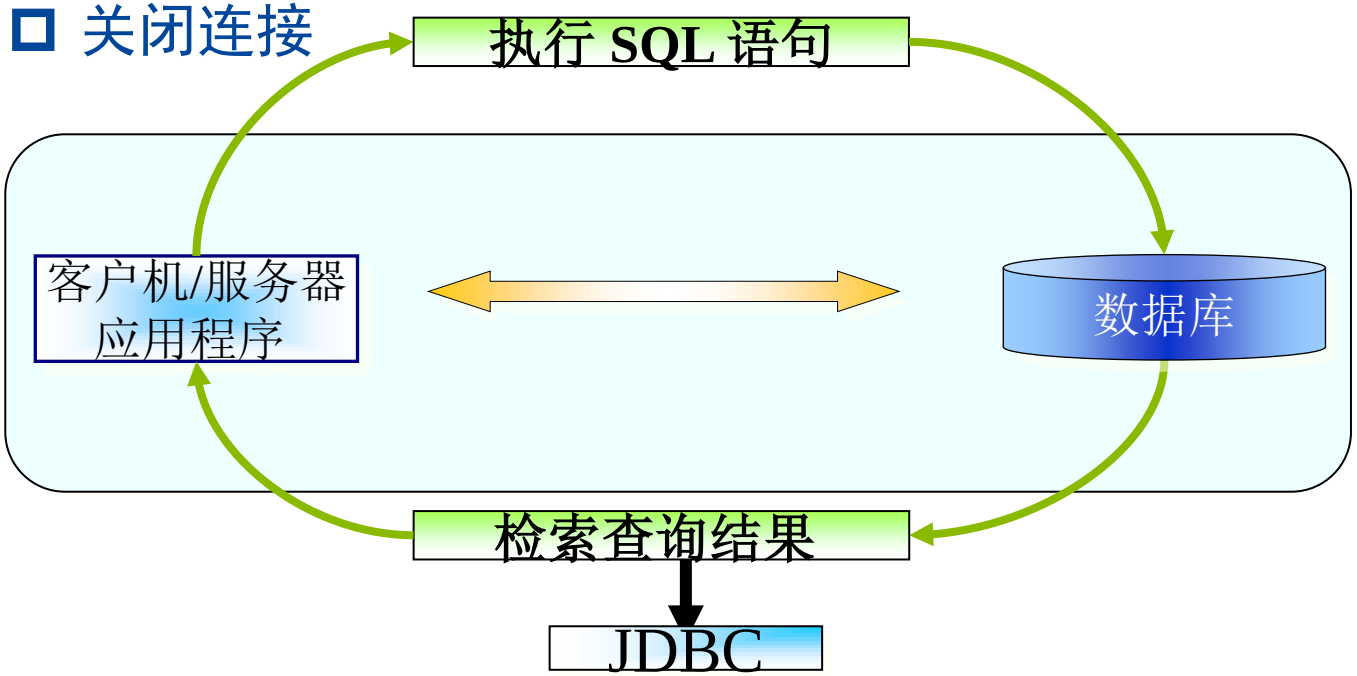
注意先后次序，按次序关闭资源

可能抛出SQLException，要进行异常处理

# Java实现数据库操作：JDBC

**JDBC对数据库进行访问和操作的主要步骤：**

- 加载数据库驱动
- 建立java与MySQL数据库连接
- 执行SQL语句来操作数据库（增删改查），获取执行结果
- 关闭连接



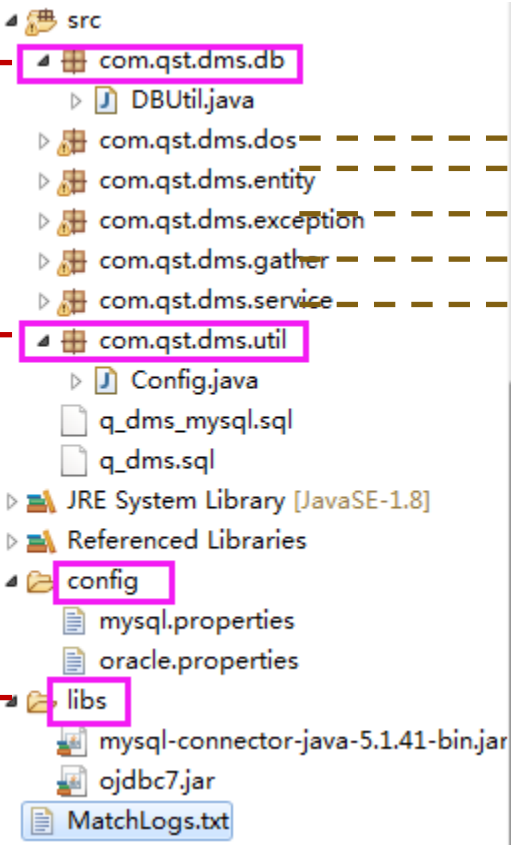
# 二、系统框架

数据库操作

通用工具

参数配置

资源包




控制台程序

数据实体类

自定义异常处理类

用于匹配筛选的数据集合

日志和物流业务逻辑处理

# 2.1 配置文件-Config文件夹

- 将一些易变的配置参数放在XML配置文件或者properties属性文件中
- 将mysql数据库配置资源放在mysql.properties属性文件中

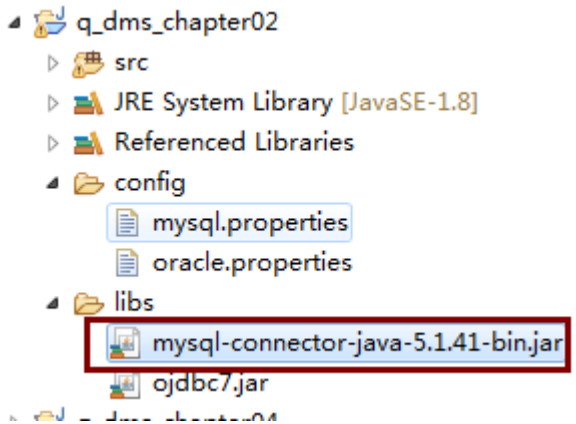

- url：数据库连接字符串

其中，127.0.0.1 是服务器IP地址，也可以使用localhost，3306是端口号，q_dms是数据库实例名

- user：连接数据库的用户名
- password：密码

# 2.2 引用外部Jar资源-libs文件夹

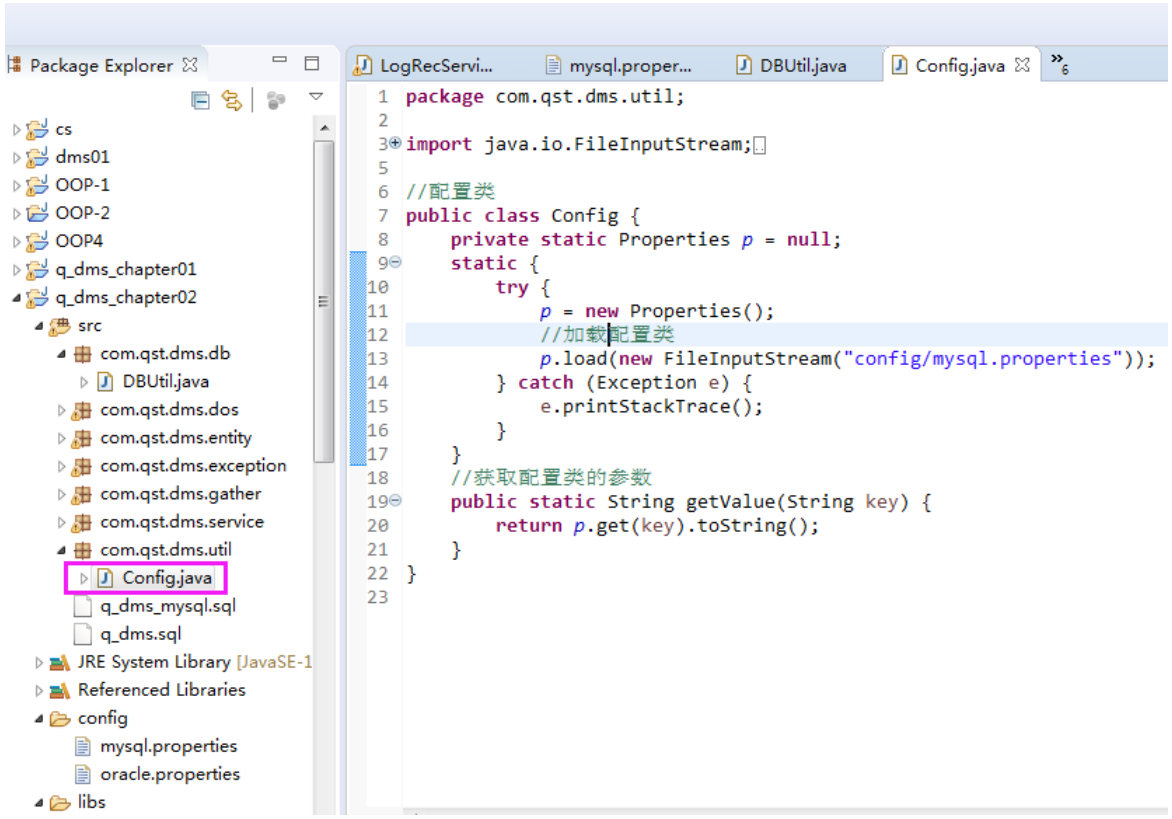
- 项目下新建libs文件夹，将外部的JAR资源放入其中
- 通过配置项目编译路径-引用外部JAR包，将其导入到Eclipse项目中




- Mysql-connector-java-5.1.1.41-bin.jar是MySQL官方JDBC驱动程序
- 将该jar包导入Eclipse项目中具体参见本次实验视频课件。

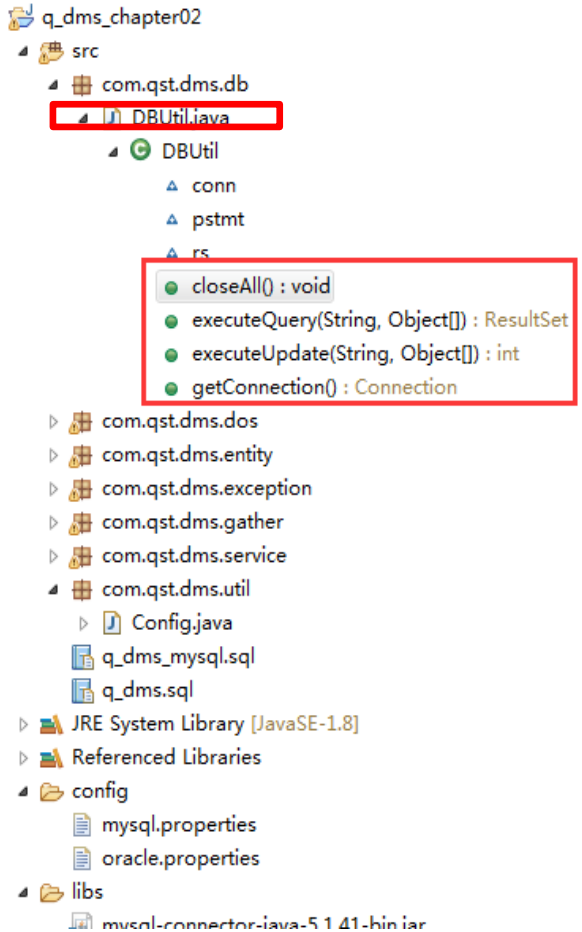
# 2.3 通用工具-Util包

- 新增util包，编写Config类获取数据库配置信息
- 利用Properties的load方法加载已配置的数据库资源



# 2.4 数据库操作-DB包

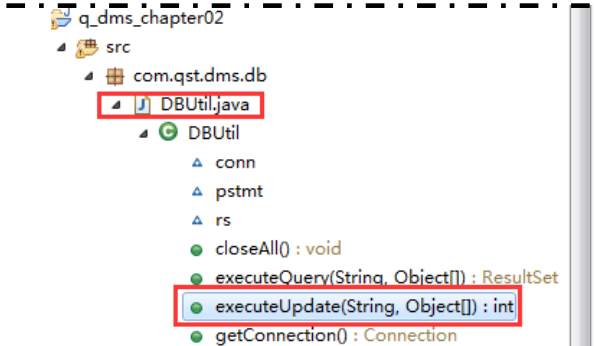
## 新增db包，将数据库操作封装在DBUtil类中




```
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.PreparedStatement;
6  import java.sql.ResultSet;
7  import java.sql.SQLException;
8  import com.qst.dms.util.Config;
9  public class DBUtil {
10     Connection conn = null;
11     PreparedStatement pstmt = null;
12     ResultSet rs = null;
13     /**
14      * 得到数据库连接
15      */
16     public Connection getConnection() throws ClassNotFoundException,
34
35     /**
36      * 释放资源
37      */
38     public void closeAll() {
64
66      * 执行SQL语句，可以进行查询
68     public ResultSet executeQuery(String preparedSql, Object[] param) {
87
89      * 执行SQL语句，可以进行增、删、改的操作，不能执行查询
91     public int executeUpdate(String preparedSql, Object[] param) {
```

# 2.4.1 预编译语句对象执行更新操作

- DBUtil类中的executeUpdate()方法封装数据库更新操作，该方法没有结果集返回
- 创建PreparedStatement对象，执行executeUpadate()方法进行增，删，改记录的操作
- PreparedStatement.setObject(index,Object)方法给带参数的SQL语句的占位符？赋值，index从1开始




```
87
88      /**
89       * 执行SQL语句，可以进行增、删、改的操作，不能执行查询
90       */
91      public int executeUpdate(String preparedSql, Object[] param) {
92          int num = 0;
93          // 处理SQL,执行SQL
94          try {
95              // 得到PreparedStatement对象
96              pstmt = conn.prepareStatement(preparedSql);
97              if (param != null) {
98                  for (int i = 0; i < param.length; i++) {
99                      // 为预编译sql设置参数
100                     pstmt.setObject(i + 1, param[i]);
101                 }
102             }
103             // 执行SQL语句
104             num = pstmt.executeUpdate();
105         } catch (SQLException e) {
106             // 处理SQLException异常
107             e.printStackTrace();
108         }
109         return num;
110     }
```

# 2.4.2 预编译语句对象执行查询操作

- DBUtil类中的executeQuery()方法封装数据库查询操作：
  - 创建PreparedStatement对象，执行executeQuery进行查询
  - 查询结果放在ResultSet结果集对象中

```
/**
 *执行SQL语句，可以进行查询
 */
public ResultSet executeQuery(String preparedSql, Object[] param) {
    // 处理SQL,执行SQL
    try {
        // 得到PreparedStatement对象
        pstmt = conn.prepareStatement(preparedSql);
        if (param != null) {
            for (int i = 0; i < param.length; i++) {
                // 为预编译sql设置参数
                pstmt.setObject(i + 1, param[i]);
            }
        }
        // 执行SQL语句
        rs = pstmt.executeQuery();
    } catch (SQLException e) {
        // 处理SQLException异常
        e.printStackTrace();
    }
    return rs;
}
```

# 2.4.3 处理结果集

通过调用ResultSet结果集对象的next()方法将游标移动到下一条记录，再通过getXXX()方法来获取指定列中的数据

```
// 从数据库读匹配日志信息，返回匹配日志信息集合
public ArrayList<MatchedLogRec> readMatchedLogFromDB() {
    ArrayList<MatchedLogRec> matchedLogRecs = new ArrayList<MatchedLogRec>();
    DBUtil db = new DBUtil();
    try {
        // 获取数据库链接
        db.getConnection();
        // 查询匹配的日志
        String sql = "SELECT i.id,i.time,i.address,i.type,i.username,i.ip,i.logtype,"
                + "o.id,o.time,o.address,o.type,o.username,o.ip,o.logtype "
                + "FROM matched_logrec m,gather_logrec i,gather_logrec o "
                + "WHERE m.loginid=i.id AND m.logoutid=o.id";
        ResultSet rs = db.executeQuery(sql, null);
        while (rs.next()) {
            // 获取登录记录
            LogRec login = new LogRec(rs.getInt(1), rs.getDate(2),
                    rs.getString(3), rs.getInt(4), rs.getString(5),
                    rs.getString(6), rs.getInt(7));
            // 获取登出记录
            LogRec logout = new LogRec(rs.getInt(8), rs.getDate(9),
                    rs.getString(10), rs.getInt(11), rs.getString(12),
                    rs.getString(13), rs.getInt(14));
            // 添加匹配登录信息到匹配集合
            MatchedLogRec matchedLog = new MatchedLogRec(login, logout);
            matchedLogRecs.add(matchedLog);
        }
        // 关闭数据库连接，释放资源
        db.closeAll();
```

# 2.4.4 数据库关闭

- closeAll()方法进行数据库访问中建立的对象资源释放

  Connection对象、Statement对象和ResultSet对象都有执行关闭的close方法；按次序关闭资源

  可能抛出SQLException，要进行异常处理

```
/**
 *释放资源
 */
public void closeAll() {
    // 如果rs不空，关闭rs
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    // 如果pstmt不空，关闭pstmt
    if (pstmt != null) {
        try {
            pstmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    // 如果conn不空，关闭conn
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

# 知识点介绍

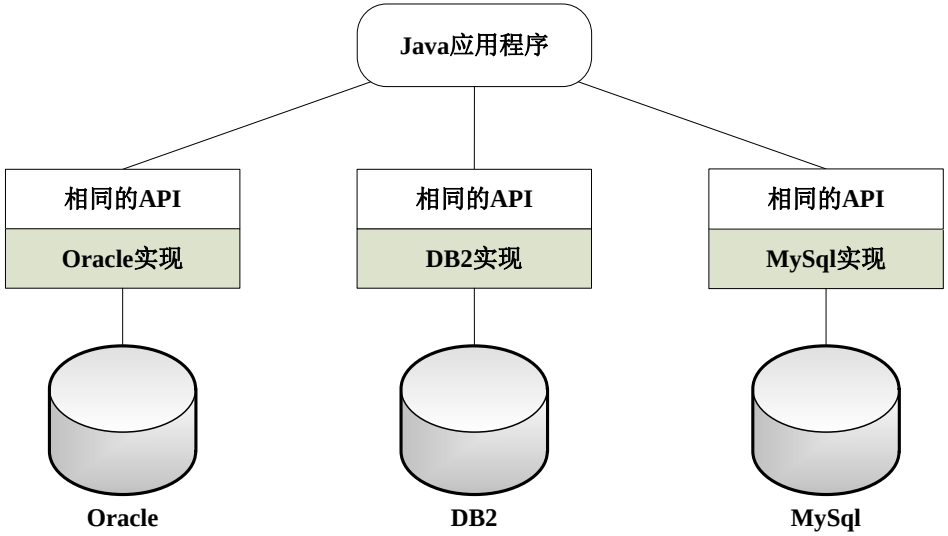
# JDBCJava DataBase Connectivity

- JDBC（Java Database Connectivity）是一种执行SQL语句的Java API
- 可以通过JDBC API连接到关系数据库，并使用SQL结构化查询语言来完成对数据库的增、删、改、查等操作
- 使用JDBC开发的数据库应用程序可以访问不同的数据库，并在不同平台上运行

# JDBC Java DataBase Connectivity

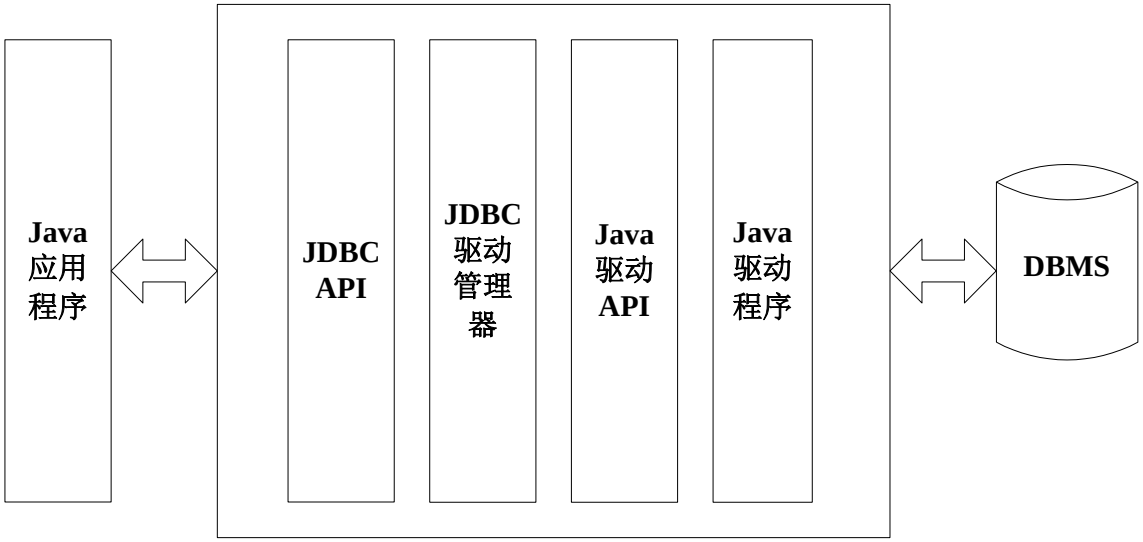
- JDBC程序访问不同的数据库时，需要数据库厂商提供相应的驱动程序。

- JDBC应用程序可以对数据库进行访问和操作，JDBC访问数据库时主要步骤：
  - 建立与数据库的连接
  - 执行SQL语句
  - 获取执行结果
  - 关闭连接



# JDBC驱动

- 数据库驱动程序是JDBC程序和数据库之间的转换层。
- 数据库驱动程序负责将JDBC调用映射成特定的数据库调用。



# JDBC驱动

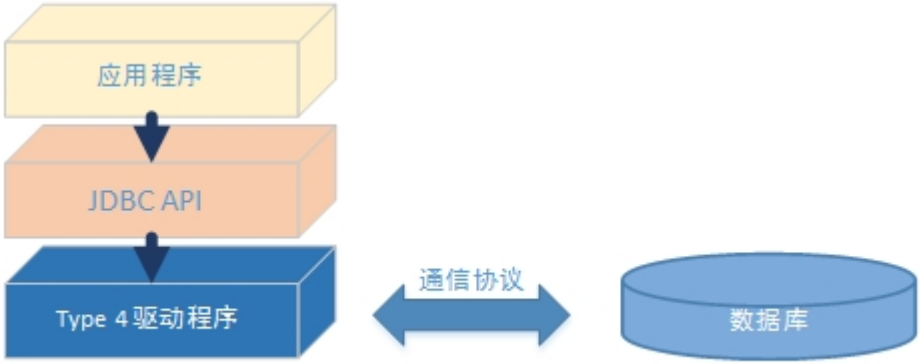
JDBC驱动程序有以下4种类型：

- JDBC-ODBC桥
- 本地API驱动
- 网络协议驱动
- 本地协议驱动

- 使用JDBC-ODBC桥方式连接数据库，其性能完全取决于数据源（ODBC）的性能，并且无法脱离Microsoft的平台，这样将带来不便
- 大部分DBMS产商都为自己的产品开发了纯Java的驱动程序，我们只需要加载相应的驱动，就可以直接连接到数据库，而无需通过ODBC桥连接

# JDBC驱动

本地协议驱动
最常见的驱动程序类型，开发中使用的驱动jar包基本都属于此类，通常由数据库厂商直接提供，例如mysql-connector-java，驱动程序把JDBC调用转换为数据库特定的网络通信协议。因为使用网络通信，驱动程序可以纯Java实现，支持跨平台部署，性能也较好。



# JDBC API

JDBC API提供了一组用于与数据库进行通信的接口和类，这些接口和类都定义在java.sql包中。

```
import java.sql.*;
```

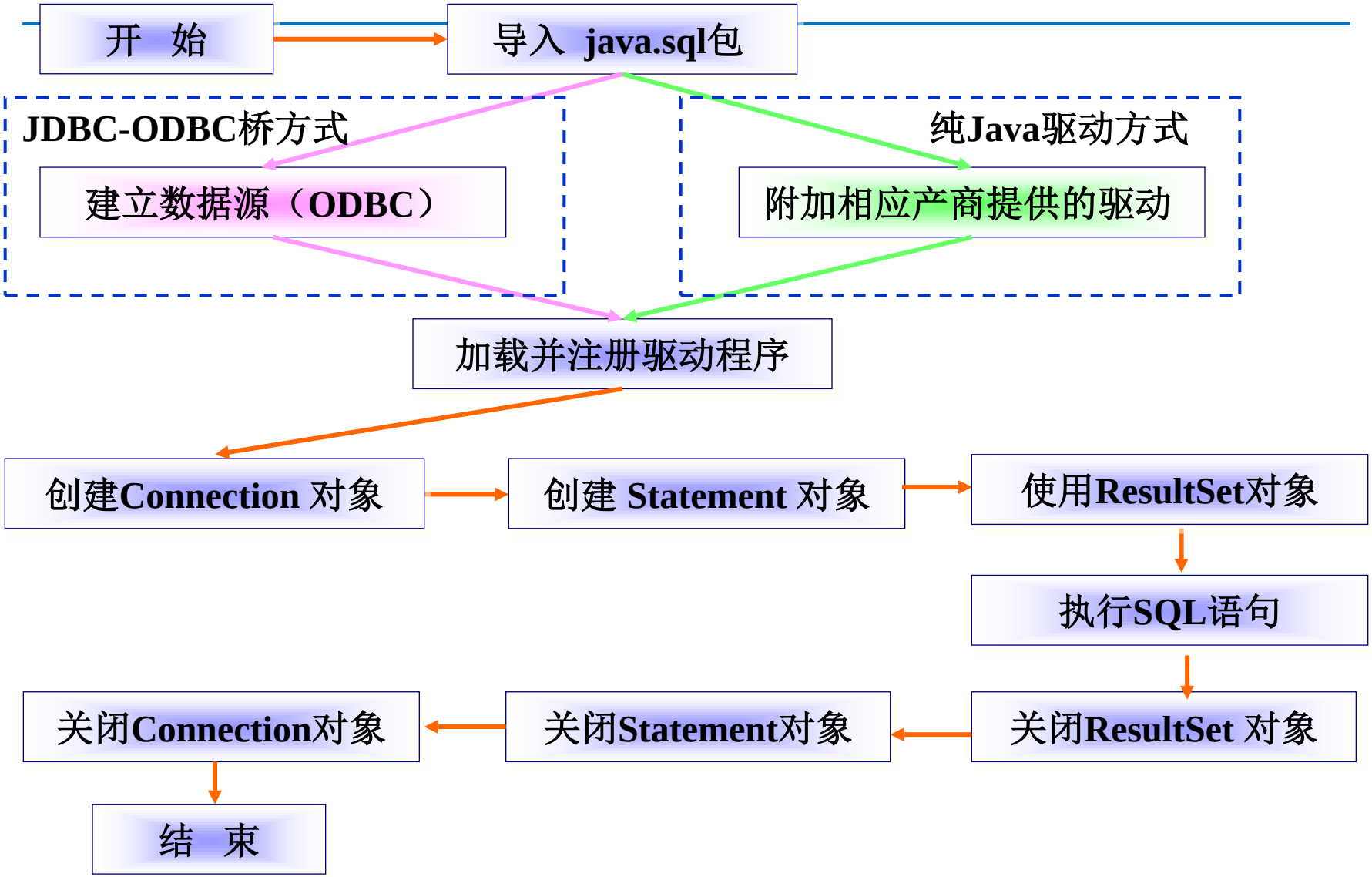
# JDBC API

| 名称 | 说　明 |
| --- | --- |
| Connection | 连接对象，用于与数据库取得连接 |
| DriverManager类 | 驱动程序管理类，用于加载和卸载各种驱动程序，建立与数据库的连接并获取连接对象 |
| Statement | 语句对象，用于执行SQL语句，并将数据检索到结果集（ResultSet）对象中 |
| PreparedStatement | 预编译语句对象，用于执行预编译的SQL语句，执行效率比Statement高 |
| CallableStatement | 存储过程语句对象，用于调用执行存储过程 |
| ResultSet | 结果集对象，包含执行SQL语句后返回的数据的集合 |

# JDBC API

| 类 名 称 | 说　明 |
| --- | --- |
| **SQLException** | 数据库异常类，是其它JDBC异常类的根类，继承于java.lang.Exception，绝大部分对数据库进行操作的方法都有可能抛出该异常 |
| **Date** | 该类中包含有将SQL日期格式转换成Java日期格式的方法 |
| **TimeStamp** | 表示一个时间戳，能精确到纳秒 |

# JDBC程序访问数据库步骤



# JDBC程序访问数据库步骤

- 通过Class.forName()方法加载MySQL驱动；
- 调用DriverManager.getConnection()方法来建立MySQL数据库连接，创建了connection对象；
- 通过连接对象的createStatement()方法来获取Statement对象，调用Statement对象的executeQuery()方法执行SQL语句；
- 调用ResultSet结果集对象的next()方法将游标移动到下一条记录，再通过getXXX()方法来获取指定列中的数据；
- 最后调用close()方法关闭所有创建的对象。

# 1.加载MySQL驱动程序并建立连接

- 使用纯Java驱动连接到MySQL 5.0数据库，加载驱动程序应如下语句：

```
Class.forName("com.mysql.jdbc.Driver");
```

- 在使用JDBC操作数据库之前，需要先创建一个数据库连接。
  建立连接，获得Connection对象

```
DriverManager.getConnection(String url,String user,String pass);
```

- url：数据库连接字符串 "jdbc:mysql://127.0.0.1:3306/test"

  其中，127.0.0.1 是服务器IP地址，也可以使用localhost，3306是端口号，test是数据库实例名

- user：连接数据库的用户名

- pass：密码

# 加载Oracle驱动程序并建立连接

- 使用纯Java驱动连接到Oracle 9i数据库，加载驱动程序应改成如下语句：

  Class.forName("oracle.jdbc.driver.OracleDriver");

- 建立连接，获得Connection对象，连接字符串应如下格式：

  例：

  Connection con = DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:test", "scott", "tiger");

# 2.创建Statement语句对象

一旦成功连接到数据库，获得Connection对象后，通过Connection对象获得Statement语句对象，其主要方法如下：

1. createStatement()：创建基本的Statement对象
2. prepareStatement(String sql)：根据参数化的SQL语句创建一个预编译的PreparedStatement对象

| 方 法 原 型 | 说 明 |
| --- | --- |
| Statement createStatement() throws SQLException | 成功创建返回Statement对象，否则抛出SQLException异常，必须捕捉 |

# 3.执行SQL语句

Statement对象有三种执行SQL语句的方法：

① executeUpdate方法：执行DELETE、UPDATE和INSERT之类的数据库操作语句（DML），这样的语句没有数据结果返回

| 方 法 原 型 | 说 明 |
| --- | --- |
| **int executeUpdate(String sql) throws SQLException** | **参数sql是要执行的SQL语句，执行成功返回受影响的行数，执行失败则抛出SQLException异常，必须捕捉** |

# 执行SQL语句（续）

- executeQuery方法：执行SELECT这样的数据查询语句（DQL），这样的语句将从数据库中获得所需的数据

| 方 法 原 型 | 说 明 |
| --- | --- |
| ResultSet executeQuery(String sql) throws SQLException | 参数sql是要执行的SQL语句，查询成功返回包含有结果数据的ResultSet对象，否则抛出SQLException异常，必须捕捉 |

# 访问结果集

- SQL的查询结果使用ResultSet封装
- 如：使用Statement对象的executeQuery方法成功执行SELECT语句后，将返回一个包含有结果数据的ResultSet对象，使用getXXX方法访问结果集中数据：

| 方 法 原 型 | 说 明 |
| --- | --- |
| **boolean next()<br>throws SQLException** | 将结果集游标往下移动一行，如果已经到达结果集最后，将会返回**false**，有可能抛异常，必须捕捉 |
| **X getX(String columnName)<br>throws SQLException** | 获得某个字段的值，**X**是指具体的数据类型，视数据库表中字段的具体情况而定，该方法有一组，并且每个都有两种重载方法，一种是以字段名称为参数，另一种是以字段索引为参数（字段索引从**1**开始），有可能抛异常，必须捕捉 |
| **X getX(int columnIndex)<br>throws SQLException** | |

# PreparedStatement接口

- 如果要多次执行相似的SQL语句，可以使用PreparedStatemend（预编译语句对象）对象来执行

  【示例】参数化的动态SQL语句

```
String sql = "INSERT INTO gather_logrec(id,time,address,type,username,ip,logtype) VALUES(?,?,?,?,?,?,?)";
```

- 通过Connection对象的prepareStatement方法来创建预编译语句对象

```
Connection conn = null;
PreparedStatement pstmt = null;
// 得到PreparedStatement对象
pstmt = conn.prepareStatement(preparedSql);
```

- PreparedStatement对象会将SQL语句预先编译，这样将会获得比Statement对象更高的执行效率

# PreparedStatement接口

- 在执行带参数的SQL语句前，必须对“？”占位符参数进行赋值。
- PreparedStatement接口中提供了大量的setXXX()方法，通过占位符的索引完成对输入参数的赋值。

| 方法名 | 描述 |
| --- | --- |
| void setArray(int parameterIndex,Array x) | 将指定参数设置为给定的java.sql.Array对象 |
| void setByte(int parameterIndex,byte x) | 将指定参数设置为给定的byte值 |
| void setShort(int parameterIndex,short x) | 将指定参数设置为给定的short值 |
| void setInt(int parameterIndex, int x) | 将指定参数设置为给定的int值 |
| void setLong(int parameterIndex, long x) | 将指定参数设置为给定的long值 |
| void setFloat(int parameterIndex, float x) | 将指定参数设置为给定的float值 |
| void setDouble(int parameterIndex, double x) | 将指定参数设置为给定的double值 |
| void setString(int parameterIndex, String x) | 将指定参数设置为给定的String字符串 |
| … | … |

# PreparedStatement接口

- 设置好参数后，就可以使用PreparedStatement对象的executeUpdate和executeQuery方法来执行SQL语句

| 方 法 原 型 | 说　　明 |
| --- | --- |
| **int executeUpdate()<br>throws SQLException** | **用于执行INSERT、DELETE和UPDATE语句，执行成功返回受影响的行数，否则抛出SQLException异常，必须捕捉** |
| **ResultSet executQuery()<br>throws SQLException** | **用于执行SELECT语句，执行成功返回包含有结果数据的ResultSet对象，否则抛出SQLException异常，必须捕捉** |

# 关闭资源

- 当数据库操作执行完毕或退出应用前，需将数据库访问过程中建立的对象按顺序关闭。Connection对象、Statement对象和ResultSet对象都有执行关闭的close方法；
- 函数原型都是：void close() throws SQLException
  关闭的次序是：
  - 关闭结果集：rs.close()
  - 关闭Statement对象：stmt.close()
  - 关闭连接：conn.close()
- 有可能抛出SQLException异常，必须捕捉；
- 请注意关闭的顺序，最后打开的资源最先关闭，最先打开的资源最后关闭

# 集元数据MetaData

集元数据是有关数据库和表结构的信息，如数据库中的表、列，索引和数据类型等信息

JDBC中的DatabaseMetaData和ResultsSetMetaData接口用来获取这些信息

- DataBaseMetaData接口用于获取数据库相关信息，通过Connection接口的getMetaData()方法进行获取
- ResultSetMetaData接口用来获取结果集的结构，通过ResultSeT的getMetaData()方法获取ResultSetMetaData对象