JDBC

- What?Why?How?

JDBC介绍

- JDBC(Java Database Connectivity)是基于JAVA语言访问数据库的一种技术。
- JDBC（Java Data Base Connectivity,java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。JDBC提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序，同时，JDBC也是个商标名。
- JDBC的设计思想：由SUN公司(JCP)提供访问数据库的接口，由数据库厂商提供对这些接口的实现，程序员编程时都是针对接口进行编程的。
- JDBC包括一套JDBC的API和一套程序员和数据库厂商都必须去遵守的规范。
  - java.sql包：提供访问数据库基本的功能
  - javax.sql包：提供扩展的功能
- 数据库中间件
- JDBC可以做些什么？
  - 连接到数据库
  - 在Java app中执行SQL命令
  - 处理结果。

JDBC

- Jdbc
  - Java反问控制数据库里面数据的一套标准
    - 接口
- Java 面向对象编程
  - 面向接口编程

JDBC

- jdbc: java database connectivity
- 一、职责
- 1、java: 客户端 :接收数据、拼接sql 、发送sql、分析结果、返回结果browser
- 2、db: 服务器 :接收sql, 分析处理，返回结果给java

面向接口编程 java.sql

- 二、面向接口编程 java.sql.*
- 1、java.sql.Driver：-->驱动
- 2、java.sql.Connection -->连接
- 3、java.sql.Statement -->静态处理块
- java.sql.PreparedStatement -->预处理块
- 4、java.sql.ResultSet -->结果集
- 5、java.sql.ResultSetMetaData -->结果集元数据

JDBC访问数据库的过程

驱动管理器　　作用：加载jdbc程序

连接数据库　　作用：建立与数据库的连接

Sql语句　　作用：发送sql语句

结果集　　作用：得到查询结果

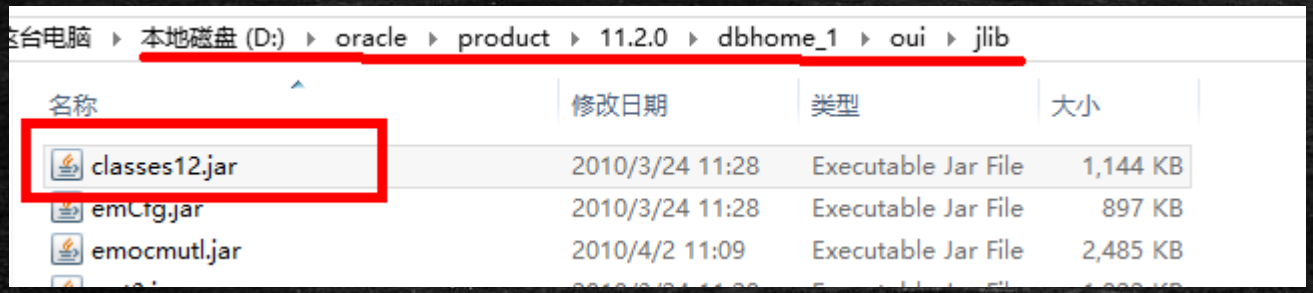
JDBC 的常用接口

- Java.sql.DriverManager用来装载驱动程序，并且为创建新的数据库联接提供支持。
- Java.sql.Connection完成对某一指定数据库的联接
- Java.sql.Statement在一个给定的连接中作为SQL执行声明的容器，他包含了两个重要的子类型。
  - Java.sql.PreparedSatement用于执行预编译的sql声明
  - Java.sql.CallableStatement用于执行数据库中存储过程的调用
- Java.sql.ResultSet对于给定声明取得结果的途径

jdbc

- 1.装载驱动程序

- 链接oracle的jar文件
- **D:\oracle\product\11.2.0\dbhome_1\oui\jlib**



通过JDBC访问数据库

1. 加载驱动程序

使用Class.forName()显式加载驱动程序。

例如:

Class.forName(“oracle.jdbc.driver.OracleDriver”);

加载了oracle提供的jar包内的OracleDriver类

常用数据库链接方式

- **MySQL:**
- String Driver="com.mysql.jdbc.Driver"; //驱动程序
- String URL="jdbc:mysql://localhost:3306/db_name"; //连接的URL,db_name为数据库名
- String Username="username"; //用户名
- String Password="password"; //密码
- Class.forName(Driver);
- Connection con=DriverManager.getConnection(URL,Username,Password);
- **Oracle:**
- String Driver="oracle.jdbc.driver.OracleDriver"; //连接数据库的方法
- String URL="jdbc:oracle:thin:@localhost:1521:orcl"; //orcl为数据库的SID
- String Username="username"; //用户名
- String Password="password"; //密码
- Class.forName(Driver) ; //加载数据库驱动
- Connection con=DriverManager.getConnection(URL,Username,Password);
-

常用数据库链接方式

- **PostgreSQL:**
- String Driver="org.postgresql.Driver"; //连接数据库的方法
- String URL="jdbc:postgresql://localhost/db_name"; //db_name为数据库名
- String Username="username"; //用户名
- String Password="password"; //密码
- Class.forName(Driver) ;
- Connection con=DriverManager.getConnection(URL,Username,Password);
- **DB2:**
- String Driver="com.ibm.dbjdbc.app.DBDriver"; //连接具有DB2客户端的Provider实例
- //String Driver="com.ibm.dbjdbc.net.DBDriver"; //连接不具有DB2客户端的Provider实例
- String URL="jdbc:db2://localhost:5000/db_name"; //db_name为数据库名
- String Username="username"; //用户名
- String Password="password"; //密码
- Class.forName(Driver) ;
- **Microsoft SQL Server :**
- String Driver="com.microsoft.sqlserver.jdbc.SQLServerDriver"; //连接SQL数据库的方法
- String URL="jdbc:sqlserver://localhost:1433;DatabaseName=db_name"; //db_name为数据库名
- String Username="username"; //用户名
- String Password="password"; //密码
- Class.forName(Driver).new Instance(); //加载数据可驱动

与数据库的连接

2. 建立连接

- 指定数据库连接的url,数据源的位置
- 使用DriverManager.getConnection(url);

例如:

```
String url = "jdbc:oracle:thin:@127.0.0.1:1521:orcl";
Connection conn=DriverManager.getConnection(url);
```

或者:

```
Connection conn=DriverManager.getConnection(String
    url , String   user , String  password)
```

获取数据库信息

```
String username="scott";
String pwd="tiger";
String url="jdbc:oracle:thin:@127.0.0.1:1521:orcl";
try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection conn = DriverManager.getConnection(url, username, pwd);
    System.out.println(conn);
    DatabaseMetaData databaseMetaData = conn.getMetaData();
    //获取oracle版本
    System.out.println(databaseMetaData.getDatabaseProductVersion());
    //获取驱动名称
    System.out.println(databaseMetaData.getDriverName());
    //获取驱动版本
    System.out.println(databaseMetaData.getDriverVersion());
    //获取某个用户拥有的表
    ResultSet rs=databaseMetaData.getTables(conn.getCatalog(), "SCOTT",null, new String[]{"TABLE"});
    //迭代器
    while (rs.next()) {
        System.out.println(rs.getString("TABLE_NAME"));

    }
    System.out.println("----------------");
    指定标明就只获取 某个表的字段，null 获取所有表的字段，columnNamePattern null获取列
    (String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern
    ResultSet rs01 = databaseMetaData.getColumns(conn.getCatalog(),"SCOTT",null,null);
    while (rs01.next()) {
        System.out.println(rs01.getString("COLUMN_NAME"));
    }
    conn.close();
```

静态处理块 Statement

- 一、静态处理块 Statement
- 1、特点: 静态sql语句
- 2、创建: 连接.createStatement()
- 3、操作
- ddl --> execute(String sql)
- dml --> executeUpdate(String sql)
- select -->executeQuery(String sql)

查询数据库 (Statement)

1. 创建Statement
   - 用户执行sql语句,
   - Statement stmt=conn.createStatement();
2. 执行查询语句
   - String sql="select * from emp"
   - ResultSet rs=stmt.executeQuery(sql);
   - executeQuery()一般用于执行一个sql语句,返回一个结果集。
3. 关闭Statement
   1. . stmt.close()

检索结果集

前面讲了如何执行sql语句,返回了ResultSet类的对象,这里讲如何对ResultSet对象进行处理

ResultSet的基本处理方法

ResultSet对象包括一个由查询语句返回的一个表,这个表中包含所有的查询结果，按照行和列进行处理.

ResultSet对象维持一个指向当前行的指针.最初,这个指针指向第一行之前.ResultSet类的next()方法使这个指针移向下一行.第一次,使用next()方法,将指针指向结果集的第一行.next方法的返回值是一个boolean值,若为true,则成功移向下一行.若返回false则没有下一行.

getXXX方法可以从某一列中获得结果.其中XXX是jdbc中的java数据类型.如getInt();需要制定检索的列,或名称.

检索结果集

```
Statement stmt=conn.createStatement();
String sql="select a,b,c from mytable";
ResultSet rs=stmt.executeQuery(sql);
While (rs.next())
{
   int i=rs.getInt(1);
   String s=rs.getString("a");
}
```

查询数据库 (PreparedStatement)

Statement对象在每次执行sql语句时都将语句传给数据库，在多次执行同一个语句时,效率比较低.

Statement对象Sql注入引起安全问题

可以使用PreparedStatement,使用数据库的预编译功能,速度可以提高很多.避免sql注入引起的安全问题

PreparedStatement对象的sql语句可以接受参数,每次执行时可以传递不同的参数.

预处理块 PreparedStatement

二、预处理块 PreparedStatement

1、特点: 动态sql语句 凡是Statement 能够处理的 PreparedStatement都能处理 ， 反之不一定

2、创建: 连接.prepareStatement(String sql)

3、操作

1)存在参数，必须填充

setXxx(int parameterIndex, Xxx x)

2)、ddl --> execute()

dml -->executeUpdate()

select -->executeQuery()

ps: 参数 指 值，不是用于关键字 和字段上面

select -->where sal=?

insert --> values(?)

update --> set sal=? where deptno=?

delete --> where sal=?

PreparedStatement

1. 创建
   - String sql="select * from emp where ename=?"
   - PreparedStatement ps=conn.prepareStatement(sql);
2. 执行
   - ps.setInt(1,"SMITH");
   - ResultSet rs=pstmt.executeQuery();
3. 关闭
   - ps.close();

获取结果集的信息

ResultsetMetaData:

- 可以获取结果集中 的列的名称,数据类型等.
- ResultsetMetaData rsmd=rs.getMetaData();
- getColumnCount();
- getColumnName(int column);
- getColumnType(int column); 返回int值
- getColumnTypeName(int column); 返回字符串
- isReadOnly(int column)
- isNullable(int column)

更新数据库

- 包括修改,更新和删除记录,创建和删除表,以及增加和删除列.对应于数据库insert,update ,delete等.
- 对数据库的更新操作也是通过PreparedStatement对象完成的.
- 不使用executeQuery()方法,使用executeUpdate()方法.
- executeUpdate的返回值是它影响的记录的行数.

更新数据库

例如

String sql="update Customer set address ='Peking' where lastname='Li'"

int i=Stmt.executeUpdate(sql)

返回更新的行数.

操作jdbc步骤

四、操作jdbc步骤

1、选择快递公司 -->选择数据库 加载驱动

2、与快递公司建立联系(电话号码...) -->建立连接 (连接信息 user password url)

3、准备包裹 快递员收包裹 -->准备sql语句 选择处理块 (Statement PreparedStatement)

4、打包 投递 -->填充参数 执行(ddl -->execute(sql) dml-->executeUpdate(sql) select-->executeQuery())

5、签字 验收 -->分析结果(ddl-->没有异常 dml-->记录数>0 select -->分析结果集)

6、打发走人 -->释放资源

批处理

多次执行数据更新操作时，可以使用批处理减少连接数据库次数，提高效率.

Statement批处理方式:

```
Statement st = conn.createStatement();
st.addBatch(更新语句1);
st.addBatch(更新语句2);
.....
st.executeBatch();
st.close();
```

建议采用PreparedStatement

PreparedStatement批处理

PreparedStatement批处理方式:

```
PreparedStatement ps = conn.preparedStatement(sql);
    ps.setXXX(索引,参数值);
        … …
        ps.addbatch();
        ps.executeBatch();
        ps.close();
```