

# 数据库编程

---

## 第一节 数据库和SQL

---

- DB (文件集合, 类似.doc,.docx文件)
- DBMS: Database Management System (类似Office/WPS)
  - 操纵和管理数据库的软件, 可建立、使用和维护数据库
- DB种类
  - 文本文件/二进制文件
  - Xls文件
  - Access (包含在office里面, 收费, 只能运行在Windows上。32和64位 office95/97/2000/2003/2007/2010/...)

### DB种类(续)

- Mysql /Postgresql/Berkely DB (免费, 但也有收费版。多平台, 32和64位区分。)
- SQL Server (收费, 只能运行Windows, 32位和64位, 中文文档。SQL Server 2000/2005/2008/2012/...,也有免费版, 但有CPU和内存限制)
- Oracle/DB2 (收费, 全平台, 32和64位, 英文文档, 也有免费版, 但有CPU和内存限制)
- SQLite (免费,手机上使用)

### 内容

- 表: table, 实体
  - 列: 列、属性、字段
  - 行: 记录、元组tuple, 数据
- 数据值域: 数据的取值范围
- 字段类型
  - int :整数 -2147483648~2147483647, 4个字节
  - double: 小数, 8个字节
  - datetime : 时间, 7个字节
  - varchar: 字符串, 可变字节

### SQL基本语句

– 是一种特殊目的的编程语言, 是一种数据库查询和程序设计语言, 用于存取数据以及查询、更新和管理关系数据库系统; 同时也是数据库脚本文件的扩展名。

- create table t1(a int, b varchar(20)); //创建
- insert into t1(a,b) values(1,'abc'); //插入
- select a from t1; //选择
- select a,b from t1 where a > 1; //附加条件的选择
- delete from t1 where a = 10 and b='ab'; //删除
- update t1 set a=2, b = 'cd' where a=1 and b='ab'; //更新覆盖
- drop table t1; //删除整张表

## 第二节 JDBC基础操作

### Java SQL 操作类库

- java.sql, javax.sql; 这2个包只是接口类
- 根据数据库版本和JDBC版本合理选择
- 一般数据库发行包都会提供jar包, 同时也要注意区分32位和64位 (数据库分32/64位, JDK也分32/64位)。
- 连接字符串 (样例)
  - jdbc:oracle:thin:@127.0.0.1:1521:dbname
  - jdbc:mysql://localhost:3306/mydb
  - jdbc:sqlserver://localhost:1433; DatabaseName=dbname

### Java连接数据库操作步骤

- 构建连接 (搭桥)
  - 注册驱动, 寻找材质, class.forName("..."); //确定数据库类型
  - 确定对岸目标, 建桥 Connection // 端口 IP地址
- 执行操作 (派个人过桥, 提着篮子, 去拿数据)
  - Statement (执行者)
  - ResultSet(结果集)
- 释放连接 (拆桥) connection.close();



- Statement 执行者类
  - 使用executeQuery()执行select语句, 返回结果放在ResultSet
  - 使用executeUpdate()执行insert/update/delete, 返回修改的行数
  - 一个Statement对象一次只能执行一个命令
- ResultSet 结果对象
  - next() 判断是否还有下一条记录
  - getInt/getString/getDouble/.....
  - 可以按索引位置, 可以按照列名

## 基本代码实现

\*\* 查询

```
import java.sql.*;

public class SelectTest {
    public static void main(String[] args){

        //构建Java和数据库之间的桥梁介质
        try{
            Class.forName("com.mysql.jdbc.Driver");
            System.out.println("注册驱动成功!");
        }catch(ClassNotFoundException e1){
            System.out.println("注册驱动失败!");
            e1.printStackTrace();
            return;
        }

        String url="jdbc:mysql://localhost:3306/test";
        Connection conn = null;
        try {
            //构建Java和数据库之间的桥梁: URL, 用户名, 密码
            conn = DriverManager.getConnection(url, "root", "123456");

            //构建数据库执行者
            Statement stmt = conn.createStatement();
            System.out.println("创建Statement成功!");

            //执行SQL语句并返回结果到ResultSet
            ResultSet rs = stmt.executeQuery("select bookid, bookname, price
from t_book order by bookid");

            //开始遍历ResultSet数据
            while(rs.next())
            {
                System.out.println(rs.getInt(1) + "," + rs.getString(2) + "," +
rs.getInt("price"));
            }

            rs.close();
            stmt.close();

        } catch (SQLException e){
            e.printStackTrace();
        }
        finally
        {
            try
            {
                if(null != conn)
                {
                    conn.close();
                }
            }
            catch (SQLException e){
                e.printStackTrace();
            }
        }
    }
}
```

```

    }
}

}

}

** 更新

import java.sql.*;

public class UpdateTest {
    public static void main(String[] args){
        executeUpdate("update t_book set price = 300 where bookid = 1");
        executeUpdate("insert into t_book(bookid, bookname, price) values(4, '编译原理', 90)");
        executeUpdate("delete from t_book where id = 4");
    }
    public static void executeUpdate(String sql)
    {
        //构建Java和数据库之间的桥梁介质
        try{
            Class.forName("com.mysql.jdbc.Driver");
            System.out.println("注册驱动成功!");
        }catch(ClassNotFoundException e1){
            System.out.println("注册驱动失败!");
            e1.printStackTrace();
        }

        String url="jdbc:mysql://localhost:3306/test";
        Connection conn = null;
        try {
            //构建Java和数据库之间的桥梁: URL, 用户名, 密码
            conn = DriverManager.getConnection(url, "root", "123456");

            //构建数据库执行者
            Statement stmt = conn.createStatement();
            System.out.println("创建Statement成功!");

            //执行SQL语句
            int result = stmt.executeUpdate(sql);

            stmt.close();

        } catch (SQLException e){
            e.printStackTrace();
        }
        finally
        {
            try
            {
                if(null != conn)
                {
                    conn.close();
                }
            }
            catch (SQLException e){
                e.printStackTrace();
            }
        }
    }
}

```

```
}  
}
```

## 注意事项

- ResultSet不能多个做笛卡尔积连接
- ResultSet最好不要超过百条，否则极其影响性能
- ResultSet也不是一口气加载所有的select结果数据
- Connection 很昂贵，需要及时close
- Connection所用的jar包和数据库要匹配

```
//两个ResultSet做笛卡尔积  
while(rs1.next())  
{  
    while(rs2.next())  
    {  
        //这个循环次数 = rs1的条数*rs2的条数  
    }  
}
```

## 第三节 JDBC高级操作

### 事务

- 数据库事务， Database Transaction。
- 作为单个逻辑工作单元执行的一系列操作，要么完全地执行，要么完全不执行。
- 事务，必须满足所谓的ACID（原子性、一致性、隔离性和持久性）属性。
- 事务是数据库运行中的逻辑工作单位，由DBMS中的事务管理子系统负责事务的处理。

### JDBC事务

- 关闭自动提交，实现多语句同一事务：
- connection.setAutoCommit(false);
- connection.commit(); 提交事务
- connection.rollback(); 回滚事务
- 保存点机制
  - connection.setSavepoint()
  - connection.rollback(Savepoint)

```
//key part  
  
insertBook(conn, "insert into t_book values(101, 'aaaa', 10)");  
insertBook(conn, "insert into t_book values(102, 'bbbb', 10)");  
insertBook(conn, "insert into t_book values(103, 'cccc', 10)");  
Savepoint phase1 = conn.setSavepoint(); //设置一个保存点  
insertBook(conn, "insert into t_book values(104, 'cccc', 10)");  
insertBook(conn, "insert into t_book values(105, 'cccc', 10)");  
conn.rollback(phase1); //回滚到phase1保存点，即上面2行无效
```

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Savepoint;
import java.sql.Statement;

public class TransactionTest {

    public static void main(String[] args) throws Exception {
        // 构建Java和数据库之间的桥梁介质
        try {
            Class.forName("com.mysql.jdbc.Driver");
            System.out.println("注册驱动成功!");
        } catch (ClassNotFoundException e1) {
            System.out.println("注册驱动失败!");
            e1.printStackTrace();
        }

        String url = "jdbc:mysql://localhost:3306/test";
        Connection conn = null;
        try {
            // 构建Java和数据库之间的桥梁: URL, 用户名, 密码
            conn = DriverManager.getConnection(url, "root", "123456");
            conn.setAutoCommit(false);

            insertBook(conn, "insert into t_book values(101, 'aaaa', 10)");
            insertBook(conn, "insert into t_book values(102, 'bbbb', 10)");
            insertBook(conn, "insert into t_book values(103, 'cccc', 10)");
            Savepoint phase1 = conn.setSavepoint(); //设置一个保存点
            insertBook(conn, "insert into t_book values(104, 'cccc', 10)");
            insertBook(conn, "insert into t_book values(105, 'cccc', 10)");
            conn.rollback(phase1); //回滚到phase1保存点, 即上面2行无效
            conn.commit();

            System.out.println("操作成功");

        } catch (SQLException e) {
            e.printStackTrace();
            conn.rollback();
        } finally {
            if (null != conn) {
                conn.close();
            }
        }
    }

    public static void insertBook(Connection conn, String sql) {
        try {
            // 构建数据库执行者
            Statement stmt = conn.createStatement();
            //System.out.print("创建Statement成功!");

            // 执行SQL语句
            int result = stmt.executeUpdate(sql);

            stmt.close();
        }
    }
}

```

```

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

## PreparedStatement

- 使用PreparedStatement的好处：
  - 防止注入攻击
  - 防止繁琐的字符串拼接和错误
  - 直接设置对象而不需要转换为字符串
  - PreparedStatement使用预编译速度相对Statement快很多

```

String sql = "insert into t_book(bookid,bookname,price) values(?,?,?)";

//构建数据库执行者
PreparedStatement pstmt = conn.prepareStatement(sql);

//执行SQL语句
int bookid = 10;
String bookName = "Effective Java";delete from t_book;insert into t_book
values(101, 'faked book');
int price = 50;

//values(1, 'Effective Java', 50)
pstmt.setInt(1, bookid);
pstmt.setString(2, bookName);
pstmt.setInt(3, price);

int result = pstmt.executeUpdate();

```

## addBatch

高效率执行大量的sql语句的方法

```

conn = DriverManager.getConnection(url, "root", "123456");

String sql = "insert into t_book(bookid,bookname,price) values(?,?,?)";

//构建数据库执行者
PreparedStatement pstmt = conn.prepareStatement(sql);

//执行SQL语句

String bookName = "aaaaaaaaaaaaaaaa";
int price = 50;

//values(1, 'Effective Java', 50)

```

```

for(int i=200;i<210;i++)
{
    pstmt.setInt(1, i);
    pstmt.setString(2, bookName);
    pstmt.setInt(3, price);
    pstmt.addBatch();
}

pstmt.executeBatch();

pstmt.close();

System.out.println("操作成功");

```

## ResultSetMetaData

- ResultSet可以用来承载所有的select语句返回的结果集
- ResultSetMetaData来获取ResultSet返回的属性（如，每一行的名字类型等）
  - getColumnCount(), 返回结果的列数
  - getColumnClassName(i), 返回第i列的数据的Java类名
  - getColumnTypeName(i), 返回第i列的数据库类型名称
  - getColumnType(i), 返回第i列的SQL类型
- 使用ResultSetMetaData解析ResultSet

```

//构建数据库执行者
Statement stmt = conn.createStatement();
System.out.println("创建Statement成功！");

//执行SQL语句并返回结果到ResultSet
ResultSet rs = stmt.executeQuery("select bookid, bookname, price from t_book
order by bookid");

//获取结果集的元数据
ResultSetMetaData meta = rs.getMetaData();
int cols = meta.getColumnCount();
for(int i=1;i<=cols;i++)
{
    System.out.println(meta.getColumnName(i) + "," + meta.getColumnTypeName(i));
}

rs.close();
stmt.close();

```

## 第四节 数据库连接池

### 需要连接池的原因：

- Connection是Java和数据库两个平行系统的桥梁
- 桥梁构建不易，成本很高，单次使用成本昂贵



## 实现连接池的方式：

- 运用共享技术来实现数据库连接池(享元模式)
  - 降低系统中数据库连接Connection对象的数量
  - 降低数据库服务器的连接响应消耗
  - 提高Connection获取的响应速度
- 享元模式, Flyweight Pattern
  - 经典23个设计模式的一种, 属于结构型模式。
  - 一个系统中存在大量的相同的对象, 由于这类对象的大量使用, 会造成系统内存的耗费, 可以使用享元模式来减少系统中对象的数量。

## 连接池的基础参数：

初始数、最大数、增量、超时时间

- driverClass 驱动class, 这里为mysql的驱动
- jdbcUrl jdbc链接
- user password数据库用户名密码
- initialPoolSize 初始数量: 一开始创建多少条链接
- maxPoolSize 最大数: 最多有多少条链接
- acquireIncrement 增量: 用完每次增加多少个
- maxIdleTime最大空闲时间: 超出的链接会被抛弃

## 常用的数据库连接池：

- 常用的数据库连接池
  - DBCP (Apache, <http://commons.apache.org/>, 性能较差)
  - C3P0 (<https://www.mchange.com/projects/c3p0/>)
  - Druid (Alibaba, <https://github.com/alibaba/druid>)

## C3P0配置

```
* C3p0Factory1.java

import java.sql.Connection;

import com.mchange.v2.c3p0.ComboPooledDataSource;

public class C3p0Factory1 {

    private static ComboPooledDataSource dataSource = null;

    public static void init() throws Exception {

        dataSource = new ComboPooledDataSource();
        dataSource.setDriverClass( "com.mysql.jdbc.Driver" );
        dataSource.setJdbcUrl( "jdbc:mysql://localhost:3306/test" );
        dataSource.setUser("root");
        dataSource.setPassword("123456");
    }
}
```

```

        // the settings below are optional -- c3p0 can work with defaults
        dataSource.setMinPoolSize(5);
        dataSource.setAcquireIncrement(5);
        dataSource.setMaxPoolSize(20);

        // The DataSource dataSource is now a fully configured and usable pooled
DataSource

    }

    public static Connection getConnection() throws Exception {
        if(null == dataSource)
        {
            init();
        }
        return dataSource.getConnection();
    }
}

```

## Druid配置

```

* DruidFactory1.java

import java.sql.Connection;

import com.alibaba.druid.pool.DruidDataSource;

public class DruidFactory1 {
    private static DruidDataSource dataSource = null;

    public static void init() throws Exception {

        dataSource = new DruidDataSource();
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");
        dataSource.setUsername("root");
        dataSource.setPassword("123456");
        dataSource.setUrl("jdbc:mysql://127.0.0.1:3306/test");
        dataSource.setInitialSize(5);
        dataSource.setMinIdle(1);
        dataSource.setMaxActive(10);
        // 启用监控统计功能 dataSource.setFilters("stat");//
    }

    public static Connection getConnection() throws Exception {
        if(null == dataSource)
        {
            init();
        }
        return dataSource.getConnection();
    }
}

```

## 运行程序

```
* SelectTest.java

import java.sql.*;

public class SelectTest {
    public static void main(String[] args){

        Connection conn = null;
        try {
            //从c3p0获取
            //conn = C3p0Factory1.getConnection();

            //从Druid获取
            //conn = DruidFactory1.getConnection();

            //构建数据库执行者
            Statement stmt = conn.createStatement();
            System.out.println("创建Statement成功!");

            //执行SQL语句并返回结果到ResultSet
            ResultSet rs = stmt.executeQuery("select bookid, bookname, price
from t_book order by bookid");

            //开始遍历ResultSet数据
            while(rs.next())
            {
                System.out.println(rs.getInt(1) + "," + rs.getString(2) + "," +
rs.getInt("price"));
            }

            rs.close();
            stmt.close();

        } catch (Exception e){
            e.printStackTrace();
        } finally {
            try {
                if(null != conn) {
                    conn.close();
                }
            } catch (SQLException e){
                e.printStackTrace();
            }
        }
    }
}
```