[JDBC 1](#_Toc264024951)

[JDBC中的主要接口 1](#_Toc264024952)

[常用JDBC操作数据库 1](#_Toc264024953)

[JDBC的高级应用 1](#_Toc264024954)

[***JDBC代码摘抄*** 2](#_Toc264024955)

[连接池 6](#_Toc264024956)

[连接池的建立 6](#_Toc264024957)

[连接池的管理 7](#_Toc264024958)

[连接池的关闭 7](#_Toc264024959)

[连接池的配置 7](#_Toc264024960)

[自定义连接池示例 7](#_Toc264024961)

[数据源 10](#_Toc264024962)

[Tomcat数据源的配置和使用 10](#_Toc264024963)

# JDBC

## JDBC中的主要接口

DriverManager类

Connection接口

Statement接口

ResultsSet接口

## 常用JDBC操作数据库

使用JDBC-ODBC桥方式 连接 和 操作数据库

使用纯JDBC驱动的方式连接和操作数据库

## JDBC的高级应用

PreparedStatement接口

Transaction(事务处理)

***JDBC代码摘抄***

|  |
| --- |
| /\*\*  \* 使用JDBC  \*/  **package** util;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.SQLException;  **public** **class** JDBCUtil {  /\*\*  \* **@return** 数据源中的链接  \*/  **public** Connection initConnection() {  Connection conn = **null**;// 定义数据库链接变量  String driver = "com.mysql.jdbc.Driver";  String url = "jdbc:mysql://172.16.14.85/phone";  String name = "root";  String password = "root";  **try** {  Class.*forName*(driver);// 1.注册驱动  conn = DriverManager.*getConnection*(url, name, password);// 2.获取数据库的链接  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  } **catch** (SQLException s) {  s.printStackTrace();  }  **return** conn;  }  /\*\*  \* 关闭链接  \* **@param** conn  \*/  **public** **void** closeConnection(Connection conn) {  **try** {  **if** (conn != **null**) {  conn.close();  conn = **null**;  }  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |
| **package** util;  **import** java.sql.Connection;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.sql.Statement;  **public** **class** Test {  /\*\*  \* 测试  \*/  **public** **void** test() {  String sql = "select \* from user where id=1";  DataSourceUtil ds = **new** DataSourceUtil(); //使用数据源  // JDBCUtil ds = new JDBCUtil(); //使用JDBC  Connection conn = ds.initConnection();  **try** {  Statement statement = conn.createStatement();  ResultSet rs = statement.executeQuery(sql);  **while** (rs.next()) {  System.*out*.println(rs.getString("name"));  }  } **catch** (SQLException e) {  e.printStackTrace();  }  ds.closeConnection(conn);  }  } |

# 连接池

|  |
| --- |
| 连接池的原理:  1. 连接复用, 通过建立一个数据库连接池以及一套连接使用, 分配, 管理策略, 使得该连接池中的连接可以得到高效, 安全的复用. 避免了数据库连接频繁建立, 关闭的开销.  2. 由于对JDBC中的原始连接进行了封装, 从而方便了数据库应用对于连接的使用, 提高了开发效率, 也正是因为这个封装层的存在, 隔离了应用本身的处理逻辑和具体数据库访问逻辑, 使应用本身的复用成为可能. |

## 连接池的建立

|  |  |
| --- | --- |
| 应用程序中建立的连接池其实是一个静态的连接池(连接池中的连接在系统初始化时就根据系统配置创建好连接, 并放置在连接池中, 且不能任意关闭连接, 以便需要使用时能从连接池中获取.)  java中提供了很多容器类可以方便的构建连接池, 如: Vector, Stack, Servlet, Bean等, 通过读取连接属性文件Connections.properties与数据库实例建立连接.   |  | | --- | | **public** **class** Conn {  **private** **static** HashMap *connectionPoolManager* = **new** HashMap();  **private** **static** String *dbDriver* = "com.microsoft.sqlserver.jdbc.SQLServerDriver";  **private** **static** String *dbUrl* = "jdbc:sqlserver://localhost:1433; DatabaseName = DBName";  **private** **static** String *userName* = "sa";  **private** **static** String *userPassword* = "123456";  **private** **static** **int** *MAX\_POOL* = 100;//默认100个连接  Connection conn = **null**;    **public** **void** createConn(){  **for**(**int** i=0; i<*MAX\_POOL*; i++){  **try**{  Class.*forName*(*dbDriver*);  conn = java.sql.DriverManager.*getConnection*(*dbUrl*, *userName*, *userPassword*);  *connectionPoolManager*.put("连接" + i, conn);  }**catch**(Exception e){  e.printStackTrace();  }  }  }    } | |

## 连接池的管理

|  |
| --- |
| 当客户请求数据库连接时, 首先查看连接池中是否有空闲的连接(指当前没有分配出去的连接).  如果存在空闲连接, 则把连接分配给客户并作相应处理(即标记该连接为正在使用, 引用计数器加1).  假设没有空闲连接, 则查看当前所开的连接数是不是已经达到maxConn(最大连接数), 假设没有达到就创建一个连接给请求的用户; 假设达到就按设定的maxWaitTime(最大等待时间)进行等待, 假如等待maxWaitTime后仍没有空闲连接, 就抛出无空闲连接异常给用户. |
| 当客户释放数据库连接时, 先判定该连接的引用次数是否超过了规定值, 假如超过就删除该连接, 并判定当前连接池内总的连接数是否小于minConn(最小连接数), 若小于就将连接池布满; 假如没超过就将该连接标记为开放状态, 可供再次复用. |

## 连接池的关闭

|  |
| --- |
| conn.close(); |

## 连接池的配置

|  |
| --- |
| 最小连接数(minConn): 是连接池一直保持的数据库连接 |
| 最大连接数(maxConn): |
| 1. 如果最小连接数与最大连接数相差太大, 那么最先的连接请求将会获利, 之后超过最小连接数量的连接请求等价于重新建立一个新的数据库连接.  不过这些大于最小连接数的数据库连接在使用完, 不会马上被释放, 它将被放到连接池中等待重复使用或是空闲超时后被释放. |

## 自定义连接池示例

|  |
| --- |
| **package** conn;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.util.HashMap;  **import** java.util.Map;  **import** java.util.Vector;  **public** **class** ConnectionPool {  //private static String dbDriver = "com.microsoft.sqlserver.jdbc.SQLServerDriver"; //SQLServer驱动  **private** **static** String *dbDriver* = "com.mysql.jdbc.Driver"; //Mysql驱动  **private** **static** String *dbUrl* = "jdbc:mysql://172.16.14.85:3306; DatabaseName=testdb";  **private** **static** String *userName* = "root";  **private** **static** String *userPassword* = "root";    **private** **static** **int** *MAX\_POOL* = 100; //最大100个连接  **private** **static** Vector *noUseConnectionPool* = **new** Vector();//没有使用的连接, 用vector实现同步  **private** **static** HashMap *nowUseConnectionPool* = **new** HashMap(); //正在使用的连接    **public** ConnectionPool(String driver, String url, String name, String password, **int** max)**throws** ClassNotFoundException{  Class.*forName*(driver);  *dbUrl* = url;  *userName* = name;  *userPassword* = password;  *MAX\_POOL* = max;  }    //通过连接池获得没有使用的连接  **public** Connection getConnection() **throws** java.sql.SQLException{  Connection conn = **null**;  **synchronized**(*noUseConnectionPool*){  **if**(*noUseConnectionPool*.size()>0){  conn = (Connection)*noUseConnectionPool*.firstElement();  *noUseConnectionPool*.remove(conn);  **return** conn;  }  }    //如果数据库连接池没有连接了, 再创建一个连接  **if**(conn != **null** && conn.isClosed()){  *noUseConnectionPool*.remove(conn);  }    //如果连接没有达到最大数, 就创建连接, 否则不创建  **if**((*noUseConnectionPool*.size() + *noUseConnectionPool*.size())<*MAX\_POOL*){  conn = *createConnection*(*dbDriver*, *dbUrl*, *userName*, *userPassword*);  conn.setAutoCommit(**false**);  ((Map) *noUseConnectionPool*).put(conn, conn);  }  **return** conn;  }    //创建一个新连接  **private** **static** Connection createConnection(String driver, String url, String user, String password)**throws** java.sql.SQLException{  Connection conn = DriverManager.*getConnection*(url, user, password);  **return** conn;  }    **public** **void** releaseConnection(Connection conn, **boolean** isCommit)**throws** java.sql.SQLException{  **if**(isCommit){  conn.commit();  }**else**{  conn.rollback();  }  *nowUseConnectionPool*.remove(conn);  **if**((*noUseConnectionPool*.size() + *noUseConnectionPool*.size())<*MAX\_POOL*){  **synchronized**(*noUseConnectionPool*){  *noUseConnectionPool*.add(conn);  }  }**else**{  conn.close();  }  }  } |
| **测试连接池**  **package** conn;  **import** java.io.IOException;  **import** java.io.PrintWriter;  **import** java.sql.Connection;  **import** java.sql.SQLException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **public** **class** PoolTest **extends** HttpServlet {  **private** **static** **int** *MAX\_POOL* = 100;  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {    response.setContentType("text/html");  response.setCharacterEncoding("utf-8");  PrintWriter out = response.getWriter();    String dbDriver = "com.mysql.jdbc.Driver"; //Mysql驱动  String dbUrl = "jdbc:mysql://172.16.14.85:3306; DatabaseName=testdb";  String userName = "root";  String userPassword = "root";    **try**{  ConnectionPool pool = **new** ConnectionPool(dbDriver, dbUrl, userName, userPassword, *MAX\_POOL*);  **for**(**int** i=0; i<10; i++){  Connection conn = pool.getConnection();  out.println("用户连接"+i+"<br/>");  }  }**catch**(SQLException ex1){  System.*err*.println("ex1");  out.print(ex1.getMessage());  ex1.printStackTrace();  }**catch**(ClassNotFoundException ex2){  System.*err*.println("ex2");  out.print(ex2.getMessage());  ex2.printStackTrace();  }  out.flush();  out.close();  }  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  doGet(request, response);  }  } |

# 数据源

## Tomcat数据源的配置和使用

|  |  |
| --- | --- |
| 第1步(这一步必须要有): 修改Tomcat服务器的conf目录下的Context.xml文件中的<Context>元素中加入<Resource>元素来定义JNDI数据源.   |  | | --- | | <Resource  name=*"phone\_pool"*  auth=*"Container"*  type=*"javax.sql.DataSource"*  driverClassName=*"com.mysql.jdbc.Driver"*  url=*"jdbc:mysql://172.16.14.85/phone"* 说明: phone是数据库名  username=*"root"*  password=*"root"*  maxActive=*"100"*  maxIdle=*"30"*  maxWait=*"1000"*  >  </Resource> | |
| 第2步(这一步必须要有): 把驱动jar包拷贝到tomcat目录下的common/lib目录中 |
| **package** util;  **import** java.lang.reflect.Method;  **import** java.sql.Connection;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.util.ArrayList;  **import** javax.naming.Context;  **import** javax.naming.InitialContext;  **import** javax.naming.NamingException;  **import** javax.sql.DataSource;  **public** **class** DBUtil\_1 {    /\*\*  \* 从数据库连接池中获取连接  \* **@return**  \*/  **public** **static** Connection initConnection() {  Connection conn = **null**;  Context ctx;  DataSource ds;  **try** {  ctx = **new** InitialContext(); //1. 获得数据源的引用  ds = (DataSource) ctx.lookup("java:comp/env/picture\_pool"); //这个picture\_pool,与Tomcat配置的数据源的name属性值相同  conn = ds.getConnection(); //2. 获得数据库连接对象.  } **catch** (NamingException e) {  e.printStackTrace();  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** conn;  }  /\*\*  \* 释放连接  \* **@param** conn  \*/  **public** **void** closeConnection(Connection conn) {  **try** {  conn.close(); //3. 返回 数据库连接 到连接池 . 注意: 在连接池中使用close()和在非连接池中使用close()方法的区别是: 前者仅仅是把数据库连接对象返回到数据库连接池中, 是连接对象又恢复到空闲状态, 而非关闭数据库连接, 而后者是直接关闭和数据库的连接.  } **catch** (SQLException e) {  e.printStackTrace();  }  }    /\*\*  \* 增、删、改  \*  \* **@param** sql: 增、删、改SQL语句  \* **@return**:  \*/  **public** **static** **int** ExecuteUD(String sql){  Connection conn = DBUtil\_1.*initConnection*();  PreparedStatement pstmt = **null**;  **try** {  pstmt = conn.prepareStatement(sql);  **return** pstmt.executeUpdate();  } **catch** (Exception e) {  e.printStackTrace();  }  **finally**{  DBUtil\_2.*CloseConn*(**null**, pstmt, conn);  }  **return** 0;  }      /\*\*  \* 查  \*  \* **@param** sql: 查询SQL语句  \* **@param** obj: 要查询的实体  \* **@return**: 实体集合  \*/  @SuppressWarnings("unchecked")  **public** **static** ArrayList<Object> ExecuteQuery(String sql,Object obj){  Connection conn = DBUtil\_1.*initConnection*();  PreparedStatement pstmt = **null**;  ResultSet rs = **null**;  ArrayList list = **new** ArrayList();  **try** {  pstmt = conn.prepareStatement(sql);//创建执行SQL语句的pstmt对象  rs = pstmt.executeQuery();//执行SQL语句,并且返回结果集  **while**(rs.next()){ //遍历结果集  //使用Java反射机制，获取对象信息，根据对象的方法，动态调用类中的所有set方法，完成对象的赋值  Class cla = Class.*forName*(obj.getClass().getName());//动态加载类  Method [] ms = cla.getMethods();//获取类的方法  Object currentobj = cla.newInstance();//创建该类的实例  **for**(**int** i=0;i<ms.length;i++){//遍历所有的方法  **if**(ms[i].getName().startsWith("set")){//获取以set开头的方法  Class[] cc = ms[i].getParameterTypes();//获取当前方法的参数  **if**(cc.length>0){//判断参数的个数，是否大于0  **if**("int".equals(cc[0].getName().toLowerCase())){//获取参数的类型，int类型  ms[i].invoke(currentobj,rs.getInt(ms[i].getName().substring(3)));  }**else** **if**("float".equals(cc[0].getName().toLowerCase())){//获取参数的类型，float类型  ms[i].invoke(currentobj,rs.getFloat(ms[i].getName().substring(3)));  }**else** **if**("double".equals(cc[0].getName().toLowerCase())){//获取参数的类型，double类型  ms[i].invoke(currentobj,rs.getDouble(ms[i].getName().substring(3)));  }**else** **if**("char".equals(cc[0].getName().toLowerCase())){//获取参数的类型，char类型  ms[i].invoke(currentobj,**new** String(rs.getString(ms[i].getName().substring(3))).charAt(0));  }**else** **if**(cc[0].getName().indexOf("String")!=0){//获取参数的类型，string类型  ms[i].invoke(currentobj,rs.getString(ms[i].getName().substring(3)));  }**else**{  System.*out*.println("常用数据类型不匹配！（只能支持：int、float、double、char、String数据类型）");  }  }**else**{ //set方法没有传递参数  System.*out*.println("set方法必须有且只有一个参数！");  }  }  }  list.add(currentobj);//将新添加的实例，存放在List中  }  **return** list;  } **catch** (Exception e) {  e.printStackTrace();  }**finally**{  DBUtil\_2.*CloseConn*(rs, pstmt, conn);//关闭连接  }  **return** **null**;  }  } |

## 严道永使用的数据库连接池

|  |
| --- |
| **package** util;  **import** java.lang.reflect.Method;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** java.util.ArrayList;  **public** **class** DBUtil\_2 {  //连接字符串  **private** **final** **static** String *DRIVER* = "com.mysql.jdbc.Driver";  **private** **final** **static** String *URL* = "jdbc:mysql://localhost/pic"; //pic是数据库名  **private** **final** **static** String *USER* = "root";  **private** **final** **static** String *PWD* = "root";  /\*\*  \* 从数据库连接池中获取连接  \* **@return**  \*/  **public** **static** Connection getConn() {  **try** {  Class.*forName*(*DRIVER*);  **return** DriverManager.*getConnection*(*URL*, *USER*, *PWD*);  } **catch** (Exception ex) {  ex.printStackTrace();  }  **return** **null**;  }    /\*\*  \* 释放连接  \* **@param** conn  \*/  **public** **static** **void** CloseConn(ResultSet rs,PreparedStatement pstmt,Connection conn) {  **try** {  **if**(rs!=**null**){  rs.close();  }  **if**(pstmt!=**null**){  pstmt.close();  }  **if** (conn != **null** && conn.isClosed() == **false**) {  conn.close();  }  } **catch** (Exception ex) {  ex.printStackTrace();  }  }    /\*\*  \* 增、删、改  \*  \* **@param** sql: 增、删、改SQL语句  \* **@return**:  \*/  **public** **static** **int** ExecuteUD(String sql){  Connection conn = DBUtil\_2.*getConn*();  PreparedStatement pstmt = **null**;  **try** {  pstmt = conn.prepareStatement(sql);  **return** pstmt.executeUpdate();  } **catch** (Exception e) {  e.printStackTrace();  }  **finally**{  DBUtil\_2.*CloseConn*(**null**, pstmt, conn);  }  **return** 0;  }      /\*\*  \* 查  \*  \* **@param** sql: 查询SQL语句  \* **@param** obj: 要查询的实体  \* **@return**: 实体集合  \*/  @SuppressWarnings("unchecked")  **public** **static** ArrayList<Object> ExecuteQuery(String sql,Object obj){  Connection conn = DBUtil\_2.*getConn*();//获取数据库连接  PreparedStatement pstmt = **null**;  ResultSet rs = **null**;  ArrayList list = **new** ArrayList();  **try** {  pstmt = conn.prepareStatement(sql);//创建执行SQL语句的pstmt对象  rs = pstmt.executeQuery();//执行SQL语句,并且返回结果集  **while**(rs.next()){ //遍历结果集  //使用Java反射机制，获取对象信息，根据对象的方法，动态调用类中的所有set方法，完成对象的赋值  Class cla = Class.*forName*(obj.getClass().getName());//动态加载类  Method [] ms = cla.getMethods();//获取类的方法  Object currentobj = cla.newInstance();//创建该类的实例  **for**(**int** i=0;i<ms.length;i++){//遍历所有的方法  **if**(ms[i].getName().startsWith("set")){//获取以set开头的方法  Class[] cc = ms[i].getParameterTypes();//获取当前方法的参数  **if**(cc.length>0){//判断参数的个数，是否大于0  **if**("int".equals(cc[0].getName().toLowerCase())){//获取参数的类型，int类型  ms[i].invoke(currentobj,rs.getInt(ms[i].getName().substring(3)));  }**else** **if**("float".equals(cc[0].getName().toLowerCase())){//获取参数的类型，float类型  ms[i].invoke(currentobj,rs.getFloat(ms[i].getName().substring(3)));  }**else** **if**("double".equals(cc[0].getName().toLowerCase())){//获取参数的类型，double类型  ms[i].invoke(currentobj,rs.getDouble(ms[i].getName().substring(3)));  }**else** **if**("char".equals(cc[0].getName().toLowerCase())){//获取参数的类型，char类型  ms[i].invoke(currentobj,**new** String(rs.getString(ms[i].getName().substring(3))).charAt(0));  }**else** **if**(cc[0].getName().indexOf("String")!=0){//获取参数的类型，string类型  ms[i].invoke(currentobj,rs.getString(ms[i].getName().substring(3)));  }**else**{  System.*out*.println("常用数据类型不匹配！（只能支持：int、float、double、char、String数据类型）");  }  }**else**{ //set方法没有传递参数  System.*out*.println("set方法必须有且只有一个参数！");  }  }  }  list.add(currentobj);//将新添加的实例，存放在List中  }  **return** list;  } **catch** (Exception e) {  e.printStackTrace();  }**finally**{  DBUtil\_2.*CloseConn*(rs, pstmt, conn);//关闭连接  }  **return** **null**;  }  } |