# 今日内容

# 测试昨天工具类完成商品的增删改查

【自己封装的工具类】

1获取连接资源的方法， 2释放连接资源的方法

|  |
| --- |
| public class MyDbUtils {    //读取配置文件 注册驱动  static String *driverClass* = null;  static String *url* = null;  static String *username* = null;  static String *password* = null;  static {//在静态代码块中读取配置文件  // 读取配置文件  InputStream in = MyDbUtils.class.getClassLoader().getResourceAsStream("db.properties");  // 创建Properties工具类  Properties p = new Properties();  try {  p.load(in);  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  *driverClass* = p.getProperty("driverClass");  *url* = p.getProperty("url");  *username* = p.getProperty("username");  *password* = p.getProperty("password");  try {  Class.*forName*(*driverClass*);  } catch (ClassNotFoundException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  /\*\*  \* 获取连接  \* @throws SQLException  \*/  public static Connection getConnection() throws SQLException{  return DriverManager.*getConnection*(*url*, *username*, *password*);  }  /\*\*  \* 关闭资源  \*/  public static void release(Connection conn,Statement st,ResultSet set){  try {  if (null != set) {  set.close();  set = null;// 经验  }  } catch (SQLException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  try {  if (null != st) {  st.close();  st = null;// 经验  }  } catch (SQLException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  try {  if (null != conn) {  conn.close();  conn = null;// 经验  }  } catch (SQLException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  } |

【测试完成CRUD】

|  |
| --- |
| public class TestUtil {  Connection conn = null;  Statement st = null;  @Before  /\*\*  \* 在单元测试的每个方法执行前执行此方法  \*  \* @throws SQLException  \*/  public void init() throws SQLException {  // 获取连接  conn = MyDbUtils.*getConnection*();  // 获取执行sql语句的对象  st = conn.createStatement();  }  @Test  /\*\*  \* 测试完成添加商品  \* @throws SQLException  \*/  public void add() throws SQLException {  // sql语句  String sql = "insert into products values(null,'六角函数',12.00,'数学',100,'不难')";  // 执行插入操作  int i = st.executeUpdate(sql);  System.*out*.println(i);  MyDbUtils.*release*(conn, st, null);  }  @Test  /\*\*  \* 测试完成删除商品  \* @throws SQLException  \*/  public void delete() throws SQLException {  // sql语句  String sql = "delete from products where id = 11";  // 执行插入操作  int i = st.executeUpdate(sql);  System.*out*.println(i);  MyDbUtils.*release*(conn, st, null);  }  @Test  /\*\*  \* 测试完成更新商品  \* @throws SQLException  \*/  public void update() throws SQLException {  // sql语句  String sql = "update products set name = '七角函数' where id = 12";  // 执行插入操作  int i = st.executeUpdate(sql);  System.*out*.println(i);  MyDbUtils.*release*(conn, st, null);  }  @Test  /\*\*  \* 测试完成查询商品  \* @throws SQLException  \*/  public void select() throws SQLException {  // sql语句  String sql = "select \* from products";  // 执行插入操作  ResultSet set = st.executeQuery(sql);  while (set.next()) {  String id = set.getString("id");//指定列名查询  String name = set.getString(2);//指定列号查询  System.*out*.println(id + "=" + name);  }  MyDbUtils.*release*(conn, st, set);  }  } |

# jdbc中PreparedStatement对象的使用

在查询操作的时候，很多参数是从页面或者用户直接输入的数据，在数据中存在着sql注入的风险 （XX or 1=1） ,所以在执行sql语句的时候先对其进行预编译 。

|  |
| --- |
| @Test  public void run() throws SQLException{  //String name="zhangsan or 1=1" ;  String sql = "select \* from user1 where username = ? and password = ?" ;  //获取执行sql语句的对象  PreparedStatement ps = conn.prepareStatement(sql) ;  //select \* from user1 where username =zhangsan or 1=1 and password=abc ;  ps.setString(1, "zhangsan");  ps.setString(2, "123");  //在执行sql语句之前  ResultSet resultSet = ps.executeQuery() ;  while(resultSet.next()){  String name = resultSet.getString("username");  System.*out*.println(name);  }  } |

【方法】

获取对象

|  |
| --- |
| PreparedStatement ps = conn.prepareStatement(sql) ;  ps.setString(index, "熊二"); |

执行sql语句的方法

|  |
| --- |
| ResultSet resultSet = ps.executeQuery() ;//查询的方法  int i = ps.executeUpdate();//更新操作的方法 增删改 |

执行更新的代码

|  |
| --- |
| public void run1() throws SQLException{  //String name="zhangsan or 1=1" ;  String sql = "insert into user1 values(null,?,?,?,?,?,?,?,null)" ;  //获取执行sql语句的对象  PreparedStatement ps = conn.prepareStatement(sql) ;  //select \* from user1 where username =zhangsan or 1=1 and password=abc ;  ps.setString(1, "熊二");  ps.setString(2, "111");  ps.setString(3, "nan");  ps.setString(4, "123@qq.com");  ps.setString(5, "12345678912");  ps.setString(6, "1");  ps.setString(7, "主角");  //在执行sql语句之前  int i = ps.executeUpdate();  System.*out*.println(i);  MyDbUtils.*release*(conn, ps,null);  } |

# Jdbc中的常用对象介绍

|  |  |  |  |
| --- | --- | --- | --- |
| **DriverManger** **作用**   1. 注册驱动b、 2. 获取与数据库的链接   **改进注册驱动：**  DriverManager.registerDriver(new com.mysql.jdbc.Driver());  缺点：严重依赖具体的驱动类；会导致驱动被注册2次。  替代方案：Class.forName("com.mysql.jdbc.Driver");  **获取与数据库的链接**  DriverManager.getConnection("jdbc:mysql://localhost:3306/ssm", "root", "hang"); //ssm为数据库名   |  | | --- | | Class.forName("com.mysql.jdbc.Driver");  // 获取数据库连接  Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/ssm", "root", "hang"); |  Connection 所有的与数据库的交互都是基于链接的基础之上的，想要对数据库进行操作，首先要获取此对象，从连接对象中获取执行数据库的statement对象  Statement stmt = conn.createStatement()：创建向数据库发送sql的statement对象 Statement **作用：**  向数据库发送并执行具体的SQL语句  **常用方法：**  常用方法：   * ResultSet executeQuery(String sql):只适合查询，返回的是查询的结果集 * int executeUpdate(String sql):适合DML，或者没有返回结果集的DDL,返回的是影响的记录行数 * boolean execute(String sql):执行任何的SQL语句，返回的不是成功与否。有结果集的返回true，没有返回false。  |  | | --- | | //有结果集的返回true，没有返回false  **boolean** b = st.execute(sql);  //返回的是影响的记录行数  **int** update = st.executeUpdate(sql);  //返回结果集  ResultSet set = st.executeQuery(sql); |  PreparedStatement **PreparedStatement实现Statement**   * PreparedStatement需要预编译以及需要参数 * 由于PreparedStatement有缓存区，所以效率更高 * 由于PreparedStatement有缓存区，所以更安全，防止了注入（1=1） * 语句中的参数可以使用占位符(?)   IMG_256   |  | | --- | | // 2：准备sql预编译语句  // ?占用一个参数位  String sql = "INSERT INTO user (NAME,sex,age) VALUES (?,?,?);";  // 3：执行sql预编译语句（检查语法）  stsm=conn.prepareStatement(sql);  // 4：设置传递的参数  stsm.setString(1,"张三");  stsm.setString(2,"男");  stsm.setInt(3,20);  // 5：发送参数，执行sql  // 注意：这里的方法后面没有参数  **int** result = stsm.executeUpdate(); |   **能用PreparedStatement就不要使用Statement** ResultSet **作用：**  代表者查询语句的查询结果集 |

# 数据库连接池的基本概念（知道）

|  |
| --- |
| 对数据库频繁的操作就需要频繁的建立连接等对象和销毁这些对象，这是非常耗费资源的，所以有了数据库连接池来帮助我们创建和管理这些对象 。 |

# 模拟一个数据库连接池（测试+了解）

创建池子（LinkedList）-- 在你创建的时候应该有一些连接（构造器创建连接对象）；

写一个getConnection（） Connection对象

back（Connection conn）方法

|  |
| --- |
| public class MyDataPool {  //创建一个池子 里面是connection对象  List<Connection> list = new LinkedList<Connection> ();  //当创建对象的时候，池子中要有一定数量的连接  public MyDataPool() throws Exception{  for (int i = 0; i < 5; i++) {  Connection conn = MyDbUtils.*getConnection*();  list.add(conn);  }  }  /\*\*  \* 从池子中获取连接  \* @throws SQLException  \*/  public Connection getConnection() throws SQLException{  if(list.size()==0){  for (int i = 0; i < 5; i++) {  Connection conn = MyDbUtils.*getConnection*();  list.add(conn);  }  }  Connection connection = list.remove(0);  return connection;  }  /\*\*  \* 将连接归还到池子中  \* @throws SQLException  \*/  public void addBack(Connection conn) throws SQLException{  if(list.size()>8){  conn.close();  }  list.add(conn) ;  }  } |

# Dbcp连接池的手动配置使用(了解)



开发步骤

1 导入 两个jar包

2 创建DBCP数据库连接池

3 手动设置参数

4 获取连接 ........

|  |
| --- |
| public class Demo1 {    public static void main(String[] args) throws SQLException {  //创建数据库连接池  BasicDataSource dataSource = new BasicDataSource();  //设置四大参数 读取配置文件  dataSource.setDriverClassName("com.mysql.jdbc.Driver");  dataSource.setUrl("jdbc:mysql:///mydb2?characterEncoding=UTF-8");  dataSource.setUsername("root");  dataSource.setPassword("123456");  //获取连接  Connection conn = dataSource.getConnection();  //获取执行sql语句的对象  String sql = "update user set name=? where uid=?";  PreparedStatement ps = conn.prepareStatement(sql);  ps.setString(1, "大朗");  ps.setString(2, "4");  ps.executeUpdate();  ps.close();  conn.close();  }  } |

# DBCP读取配置文件的基本使用 (知道)

配置文件 dbcp.properties key值不能变

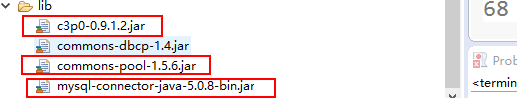
|  |
| --- |
| driverClassName=com.mysql.jdbc.Driver  url=jdbc:mysql://localhost:3306/mydb2?characterEncoding=UTF-8  username=root  password=123456 |

代码

|  |
| --- |
| public class Demo2 {  public static void main(String[] args) throws Exception {  // 加载配置文件  Properties p = new Properties();  p.load(new FileInputStream("src/dncp.properties"));  // 创建数据库连接数据元  DataSource dataSource = BasicDataSourceFactory.*createDataSource*(p);  // 获取连接  Connection connection = dataSource.getConnection();  // 获取执行sql语句的对象  String sql = "update user set name=? where uid=?";  PreparedStatement ps = connection.prepareStatement(sql);  //预编译sql语句  ps.setString(1, "大朗2");  ps.setString(2, "4");  //执行更新  ps.executeUpdate();  //释放连接资源  ps.close();  connection.close();  }  } |

# C3p0数据库连接池的基本使用

Jar



【配置文件】

指定名称指定在src目录下

C3p0.properties

|  |
| --- |
| c3p0.driverClass=com.mysql.jdbc.Driver  c3p0.jdbcUrl=jdbc:mysql:///mydb2?characterEncoding=UTF-8  c3p0.user=root  c3p0.password=123456 |

c3p0-config.xml 可以配置多个数据库连接并且可以指定名字去加载

ComboPooledDataSource dataSource = new ComboPooledDataSource("my");

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <c3p0-config>  <!-- 默认的配置，默认读取该配置 -->  <default-config>  <property name=*"driverClass"*>com.mysql.jdbc.Driver</property>  <property name=*"jdbcUrl"*>jdbc:mysql:///ssm</property>  <property name=*"user"*>root</property>  <property name=*"password"*>hang</property>  </default-config>    <!-- 指定名称的配置 -->  <named-config name=*"my"*>  <property name=*"driverClass"*>com.mysql.jdbc.Driver</property>  <property name=*"jdbcUrl"*>jdbc:mysql:///ssm</property>  <property name=*"user"*>root</property>  <property name=*"password"*>hang</property>  </named-config>  </c3p0-config> |

测试代码+数据封装

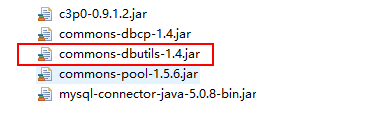
|  |
| --- |
| public class Demo1 {  public static void main(String[] args) throws SQLException {  //获取c3p0的数据库元数据  ComboPooledDataSource dataSource = new ComboPooledDataSource();  //获取数据库连接  Connection conn = dataSource.getConnection();  String sql = "select \* from products" ;  //获取执行SQL语句的对象  PreparedStatement ps = conn.prepareStatement(sql);  ResultSet resultSet = ps.executeQuery();  List<Product> list = new ArrayList<>();  while(resultSet.next()){  Product product = new Product();  //封装结果集  String id = resultSet.getString(1);  String name = resultSet.getString(2);  String price = resultSet.getString(3);  String category = resultSet.getString(4);  String pnum = resultSet.getString(5);  String description = resultSet.getString(6);  product.setId(Integer.*parseInt*(id));  product.setName(name);  //product.setPrice(Double.parseDouble(price));  product.setCategory(category);  product.setPnum(Integer.*parseInt*(pnum));  product.setDescription(description);  list.add(product);  System.*out*.println(price);  }  System.*out*.println(list.size());  //释放连接资源  MyDbUtils.*release*(conn, ps, resultSet);  }  } |

Product 实体类

|  |
| --- |
| public class Product {  /\*  \* CREATE TABLE `products` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `name` varchar(40) DEFAULT NULL,  `price` double DEFAULT NULL,  `category` varchar(40) DEFAULT NULL,  `pnum` int(11) DEFAULT NULL,  `description` varchar(255) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=17 DEFAULT CHARSET=utf8;  \*/  private int id ;  private String name ;  private double price ;  private String category ;  private int pnum ;  private String description ;    public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public double getPrice() {  return price;  }  public void setPrice(double price) {  this.price = price;  }  public String getCategory() {  return category;  }  public void setCategory(String category) {  this.category = category;  }  public int getPnum() {  return pnum;  }  public void setPnum(int pnum) {  this.pnum = pnum;  }  public String getDescription() {  return description;  }  public void setDescription(String description) {  this.description = description;  }  @Override  public String toString() {  return "Product [id=" + id + ", name=" + name + ", price=" + price + ", category=" + category + ", pnum=" + pnum  + ", description=" + description + "]";  }  } |

# C3p0数据库连接池和阿帕奇的工具类集合完成商品的CRUD

1. 基本使用导入阿帕奇工具类的jar包



|  |
| --- |
| public class Demo2 {  public static void main(String[] args) throws SQLException {  //创建数据库连接池的元数据  ComboPooledDataSource dataSource = new ComboPooledDataSource();  //使用阿帕奇的工具类  QueryRunner runner = new QueryRunner(dataSource);  //操作sql语句  String sql = "select \* from products where id = ?";  //查询 返回一条解果  Product product = runner.query(sql, new BeanHandler<Product>(Product.class), "8") ;  System.*out*.println(product);  //不需要任何操作  }  } |

【测试完商品的增删改查】

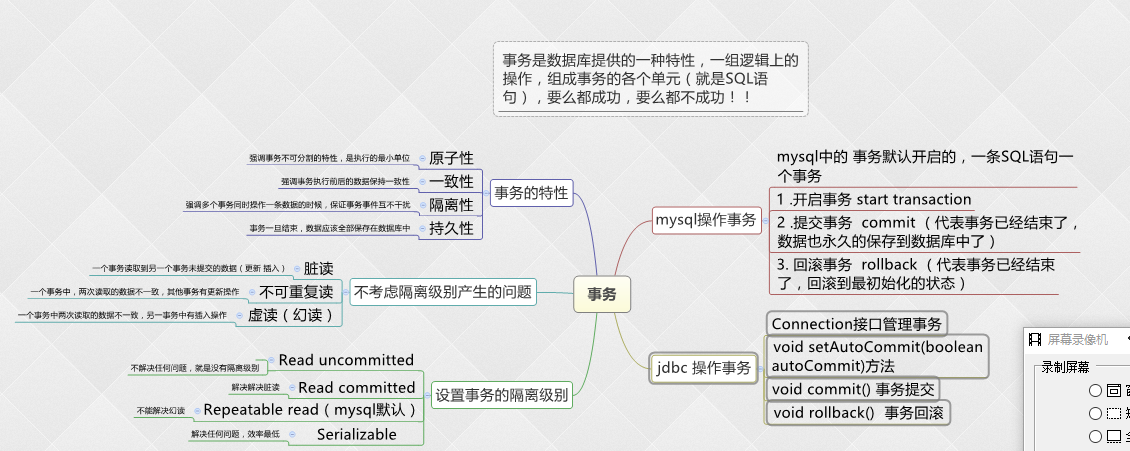
|  |
| --- |
| public class DemoCRUD {  QueryRunner runner = null ;  @Before  public void init() {  // 创建数据库连接池的元数据  ComboPooledDataSource dataSource = new ComboPooledDataSource();  // 使用阿帕奇的工具类  runner = new QueryRunner(dataSource);  }  @Test  public void add() throws SQLException {  String sql = "insert into products values(null,?,?,?,?,?)" ;  runner.update(sql, "八角函数",12.00,"高等数学",1000,"你学不会");  }  @Test  public void delete() throws SQLException {  String sql = "delete from products where id = ?" ;  runner.update(sql, "15");  }  @Test  public void update() throws SQLException {  String sql = "update products set name = ? where id = ?" ;  runner.update(sql,"我和青春有个约会","16");  }  @Test  public void select() throws SQLException {  String sql = "select \* from products" ;  List<Product> list = runner.query(sql, new BeanListHandler<Product>(Product.class));  }  } |

【查询的结果两种】

1 单条数据 new BeanHandler<>(type);

2 多条数据 new BeanListHandler<>(type);

# 事务简介，事务的特性，隔离级别

1. 

# Mysql中事务演示

|  |
| --- |
| package shiwu;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.Statement;  public class TestShiWu {  public static void main(String[] args) throws Exception {  // DriverManager.registerDriver(new Driver());  Class.*forName*("com.mysql.jdbc.Driver");  // 获取连接  Connection conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/mydb2?characterEncoding=UTF-8",  "root", "123456");  // 获取执行SQL语句的对象  Statement statement = conn.createStatement();  conn.setAutoCommit(false);  String sql1 = "update user set money= money-100 where uid = 3";  //西门庆花钱  int i = statement.executeUpdate(sql1);  int h = 1/0 ;  //大朗收钱  String sql2 = "update user set money = money+100 where uid = 4";  int j = statement.executeUpdate(sql2);  conn.commit();  // 释放资源  statement.close();  conn.close();  }  } |