**每日作业卷答案**

**jdbc连接池**

传智播客.黑马程序员

# 关卡1

## 训练案例1

### 训练描述

使用C3P0获得10个连接对象

要求：不使用配置文件

### 操作步骤答案

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception {  // 获得连接池对象  ComboPooledDataSource ds = **new** ComboPooledDataSource();  // 设置数据库连接参数  // 设置用户名  ds.setUser("root");  // 设置密码  ds.setPassword("123456");  // 设置数据库连接URL  ds.setJdbcUrl("jdbc:mysql://localhost:3306/day08");  // 设置数据库驱动你  ds.setDriverClass("com.mysql.jdbc.Driver");    // 设置连接池参数  // 设置初始连接数  ds.setInitialPoolSize(5);  // 设置最大连接数  ds.setMaxPoolSize(10);  // 设置最大空闲时间:毫秒  ds.setMaxIdleTime(2000);  // 设置最大等待时间  ds.setCheckoutTimeout(1000);    // 获得连接对象  **for** (**int** i = 0; i < 10; i++) {  Connection conn = ds.getConnection();  System.***out***.println("第" + (i + 1) + "个:" + conn);  }  } |

## 训练案例2

### 训练描述

使用C3P0连接池获得10个连接对象。

要求：分别使用默认配置和命名配置创建连接池对象。再通过连接池对象获得连接对象。

### 操作步骤答案

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception {  // test01();  *test02*();  }    // 使用默认配置(default-config)  **public** **static** **void** test01() **throws** Exception{  // 获得连接池对象  ComboPooledDataSource ds = **new** ComboPooledDataSource();  // 获得连接对象  **for** (**int** i = 0; i < 10; i++) {  Connection conn = ds.getConnection();  System.***out***.println("第" + (i + 1) + "个:" + conn);  **if**(i == 3){  // 把连接对象放回连接池  conn.close();  }  }  }    // 使用命名配置(named-config)  **public** **static** **void** test02() **throws** Exception{  // 获得连接池对象  ComboPooledDataSource ds = **new** ComboPooledDataSource("pkxing-config");  // 获得连接对象  **for** (**int** i = 0; i < 10; i++) {  Connection conn = ds.getConnection();  System.***out***.println("第" + (i + 1) + "个:" + conn);  }  } |

## 训练案例3

### 训练描述

自定义C3P0工具类，要求如下：

1.创建私有静态数据源(连接池对象)成员变量

2.创建公有的得到数据源(连接池对象)的方法

3.创建共有的得到连接对象的方法

### 操作步骤答案

|  |
| --- |
| **public** **class** C3P0Utils {  //连接池  **private** **static** ComboPooledDataSource *dataSource* = **new** ComboPooledDataSource();    /\*\*  \* 获得数据源(连接池)  \* **@return**  \*/  **public** **static** DataSource getDataSource(){  **return** *dataSource*;  }  /\*\*  \* 获得连接  \* **@return**  \* **@throws** SQLException  \*/  **public** **static** Connection getConnection(){  //将从连接池中获得连接  **try** {  **return** *dataSource*.getConnection();  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** **null**;  }  } |

## 训练案例4

### 训练描述

使用DBCP连接池获得10个连接对象，要求如下：

1.纯代码编写配置信息(不使用配置文件)

2.设置初始为5个，最大10个，最大等待时间3秒，最大空闲个数3个

### 操作步骤答案

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** SQLException {  // 创建连接池对象  BasicDataSource ds = **new** BasicDataSource();  // 设置数据库连接参数  ds.setUsername("root");  ds.setPassword("123456");  ds.setUrl("jdbc:mysql:///day08");  ds.setDriverClassName("com.mysql.jdbc.Driver");    // 设置连接参数  ds.setInitialSize(5);  ds.setMaxActive(10);  // 最大空闲时间  ds.setMaxWait(2000);  ds.setMaxIdle(5);    // 获得连接对象  **for** (**int** i = 0; i < 10 ; i++) {  Connection conn = ds.getConnection();  System.***out***.println("第" + (i + 1) + "个:" + conn.hashCode());  }  } |

## 训练案例5

### 训练描述

使用DBCP连接池获得10个连接对象，要求如下：

使用配置文件配置数据库参数和连接池参数

### 操作步骤答案

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception {  Properties prop = **new** Properties();  prop.load(Test01\_05.**class**.getResourceAsStream("/dbcp.properties"));  // 1.创建连接池  BasicDataSource ds = (BasicDataSource) BasicDataSourceFactory.*createDataSource*(prop);  // 2.获取连接  **for** (**int** i = 1; i <= 10; i++) {  Connection conn = ds.getConnection();  System.***out***.println(conn.hashCode());  }  }  配置文件信息(dbcp.properties)  # database connection configuration  url=jdbc:mysql://localhost:3306/day08  username = root  password = 123456  driverClassName = com.mysql.jdbc.Driver  # connection pool configuration  initialSize = 5  maxActive = 10  maxWait = 3000  maxIdle = 3000 |

## 训练案例6

### 训练描述

自定义DBCP工具类，要求如下：

1. 声明静态连接对池成员变量
2. 编写静态语句块：加载类路径下的属性文件，并且使用工厂类创建数据源(连接池对象)
3. 编写两个公共静态方法，返回数据源(连接池对象)和返回连接对象。
4. 编写测试类测试该工具类的相关方法。

### 操作步骤答案

|  |
| --- |
| **public** **class** DBCPUtils {  //1 创建连接池  **private** **static** DataSource *dataSource*;  // 编写固定代码  **static**{  **try** {  //1) 加载properties文件，获得Properties对象  InputStream is =  DBCPUtils.**class**.getClassLoader().getResourceAsStream("dbcp.properties");  Properties prop = **new** Properties();  prop.load(is);  //2) 通过工厂，创建连接池  *dataSource* = BasicDataSourceFactory.*createDataSource*(prop);  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  }  }    /\*\*  \* 获得数据库(连接池)  \* **@return**  \*/  **public** **static** DataSource getDataSource(){  **return** *dataSource*;  }  /\*\*  \* 获得连接  \* **@return**  \* **@throws** SQLException  \*/  **public** **static** Connection getConnection(){  **try** {  **return** *dataSource*.getConnection();  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** **null**;  }  } |

## 训练案例7

### 训练描述

1.创建一个用户表(包含字段：用户编号，用户名，性别，地址)

2.向用户表添加多条用户记录。

### 操作步骤答案

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception {  *createTable*();  *insert*();  }    /\*  \* 创建用户表  \*/  **public** **static** **void** createTable() **throws** Exception{  // 创建用户表Sql  String sql01 = "create table user(id int primary key auto\_increment,"  + "username varchar(20) unique not null,"  + "gender varchar(2),"  + "address varchar(30));";  // 创建QueryRunner对象  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  // 执行创表SQL语句  qr.update(sql01);  }  /\*  \* 向用户表添加多条用户记录。  \*/  **public** **static** **void** insert() **throws** SQLException{  // 得到数据源对象  DataSource ds = C3P0Utils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  // 要执行的SQL语句  String sql = "insert into user(username,gender,address) values('jack','男','广州'),('rose','女','北京'),('lily','女','天津')";  // 执行插入操作  **int** row = qr.update(sql);    System.***out***.println(row);  } |

## 训练案例8

### 训练描述

更新用户表中id为2的记录，修改性别和居住地。

### 操作步骤答案

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception {  // 得到数据源对象  DataSource ds = DBCPUtils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  // 执行更新操作  **int** row = qr.update("update user set gender=?,address=? where id=?","男","深圳",2);  System.***out***.println(row);  } |

## 训练案例9

### 训练描述

删除用户表中id为3的记录

### 操作步骤答案

|  |
| --- |
| **public** **class** Test01\_09 {  **public** **static** **void** main(String[] args) **throws** SQLException {  // 得到数据源对象  DataSource ds = DBCPUtils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  // 执行删除操作  **int** row = qr.update("delete from user where id = ?",3);  System.***out***.println(row);  }  } |

## 训练案例10

### 训练描述

查询用户表的所有用户数据，要求如下：

1.只查询用户名和性别两个字段信息。

2.查询结果是一个集合，集合中存放所有的用户对象。

### 操作步骤答案

|  |
| --- |
| **public** **class** Test01\_10 {  **public** **static** **void** main(String[] args) **throws** Exception{  // 数据源对象  DataSource ds = C3P0Utils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  // 查询用户数据  List<User> list = qr.query("select username,gender from user",  **new** ResultSetHandler<List<User>>() {  @Override  **public** List<User> handle(ResultSet rs) **throws** SQLException {  // 创建集合存放用户对象  List<User> users = **new** ArrayList<User>();  **while**(rs.next()){  // 获得用户名  String username = rs.getString("username");  // 获得性别  String gender = rs.getString("gender");  System.***out***.println(username + gender);  // 创建用户对象  User u = **new** User(0,username,gender,**null**);  users.add(u);  }  **return** users;  }  });  }  } |

# 关卡2

## 训练案例1

### 训练描述

查询用户表中的第一条数据。并将数据封装成对象数组。

### 操作步骤答案

|  |
| --- |
| **public** **class** Test02\_01 {    **public** **static** **void** main(String[] args) **throws** Exception {  // 获得数据源对象  DataSource ds = C3P0Utils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  // 查询所有用户数据  Object[] users = qr.query("select \* from user", **new** ArrayHandler());  System.***out***.println(Arrays.*toString*(users));  }  } |

## 训练案例2

### 训练描述

查询用户表中的所有用户记录，将每一条记录都封装成一个Object[]数组中，返回集合对象。

### 操作步骤答案

|  |
| --- |
| **public** **class** Test02\_02 {  **public** **static** **void** main(String[] args) **throws** SQLException {  // 获得数据源对象  DataSource ds = C3P0Utils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  // 查询所有用户数据  List<Object[]> users = qr.query("select \* from user", **new** ArrayListHandler());  // 遍历集合  **for** (Object[] objArr : users) {  System.***out***.println(Arrays.*toString*(objArr));  }  }  } |

## 训练案例3

### 训练描述

查询用户表中的第一条用户记录并将该记录封装成一个JavaBean对象。

注意事项：JavaBean属性名和用户表的列名要相同。

### 操作步骤答案

|  |
| --- |
| **public** **class** Test02\_03 {  **public** **static** **void** main(String[] args) **throws** SQLException {  // 获得数据源对象  DataSource ds = C3P0Utils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  // 查询所有用户  User user = qr.query("select \* from user", **new** BeanHandler<User>(User.**class**));  System.***out***.println(user);  }  } |

## 训练案例4

### 训练描述

1.定义一个方法：查询用户表的所有用户记录并将每一条记录封装成JavaBean对象存放到集合中。

2.定义一个方法：查询用户表中用户记录的数量。

### 操作步骤答案

|  |
| --- |
| **public** **class** Test02\_04 {  **public** **static** **void** main(String[] args)  **throws** SQLException {  *test01*();  *test02*();  }    // 2.定义一个方法：查询用户表中用户记录的数量。  **public** **static** **void** test02() **throws** SQLException{  // 获得数据源对象  DataSource ds = C3P0Utils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  Long count = qr.query("select count(\*) from user", **new** ScalarHandler<Long>());  System.***out***.println("一共有用户：" + count);  }    // 1.定义一个方法：查询用户表的所有用户记录并将每一条记录封装成JavaBean对象存放到集合中。  **public** **static** **void** test01() **throws** SQLException{  // 获得数据源对象  DataSource ds = C3P0Utils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  // 查询所有用户  List<User> users = qr.query("select \* from user", **new** BeanListHandler<>(User.**class**));  **for** (User user : users) {  System.***out***.println(user);  }  }  } |

## 训练案例5

### 训练描述

1.定义一个方法，查询用户表，获得所有用户的名字存放到集合中。

2.定义一个方法，查询用户表，获得所有用户的所有信息，返回一个Map集合(Map集合的

key是用户id，value是每一个用户的信息，也是一个Map集合。)

### 操作步骤答案

|  |
| --- |
| **public** **class** Test02\_05 {  **public** **static** **void** main(String[] args) **throws** SQLException {  *test01*();  *test02*();  }  /\* 2.定义一个方法，查询用户表，获得所有用户的所有信息，返回一个Map集合(Map集合的  key是用户id，value是每一个用户的信息，也是一个Map集合。)  \*/  **public** **static** **void** test02() **throws** SQLException{  // 获得数据源对象  DataSource ds = C3P0Utils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  //将多条记录封装成Map<某列类型,Map<字段名,字段值>>，KeyedHandler<某列的类型>  Map<Integer,Map<String,Object>> users = qr.query("select  \* from user",  **new** KeyedHandler<Integer>("id"));  // 遍历map集合  **for**(Entry<Integer,Map<String,Object>> entry:users.entrySet()) {  System.***out***.println(entry.getKey());  System.***out***.println(entry.getValue());  }  }  // 1.定义一个方法，查询用户表，获得所有用户的名字存放到集合中。  **public** **static** **void** test01() **throws** SQLException{  // 获得数据源对象  DataSource ds = C3P0Utils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  // 查询某一列，封装成List<字段类型>  List<String> usernames = qr.query("select id,username,gender from user",  **new** ColumnListHandler<String>("username"));  **for** (String username : usernames) {  System.***out***.println(username);  }  }  } |

## 训练案例6

### 训练描述

1.定义一个方法：查询用户表获取第一条用户记录并封装成Map集合(key是字段名称，value是字段值)。

2.定义一个方法：查询用户表获取所有用户记录并返回一个集合，集合中存放的都是Map对象，一个Map对象封装对应一个用户记录。

### 操作步骤答案

|  |
| --- |
| **public** **static** **void** main(String[] args)**throws** SQLException{  *test01*();  *test02*();  }  // 1.定义一个方法：查询用户表获取第一条用户记录并封装成Map集合(key是字段名称，value是字段值)。  **public** **static** **void** test01() **throws** SQLException{  // 获得数据源对象  DataSource ds = C3P0Utils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  Map<String,Object> map = qr.query("select \* from user", **new** MapHandler());  System.***out***.println(map);  }  // 2.定义一个方法：查询用户表获取所有用户记录并返回一个集合，集合中存放的都是Map对象，一个Map对象封装对应一个用户记录。  **public** **static** **void** test02() **throws** SQLException{  // 获得数据源对象  DataSource ds = C3P0Utils.*getDataSource*();  // 创建SQL运行对象  QueryRunner qr = **new** QueryRunner(ds);  List<Map<String,Object>> list = qr.query("select \* from user", **new** MapListHandler());  // 遍历集合  **for** (Map<String, Object> map : list) {  System.***out***.println(map);  }  }  } |

# 关卡3

## 训练案例1

### 训练描述

1.在数据库创建部门表和员工表，包含字段如下：

部门表包含字段有：部门编号，部门名称(唯一且不能为空)

员工表包含字段有：员工编号，员工姓名(唯一且不能为空)，员工性别，员工职位，员工工资，入职日期。部门编号(外键)

2.先添加多个部门数据，再添加多条员工数据

3.编写方法接收一个员工编号和工资两个参数，方法内将指定编号的员工工资修改为新的工资。

4.编写方法查询指定职位所有员工的信息，返回List<Employee>集合。

5.编写方法查询指定姓名的员工信息，返回Employee对象。

6.编写方法根据员工姓名删除指定的员工信息。

7.编写方法查询所有姓张员工的工资并输出在控制台，输出格式如下：

张三=10000

张飞=20000

…………………

8.编写方法接收一个工资参数，方法内查询工资大于等于传入的工资的员工，返回符合条件所有员工信息List<Employee>集合。

9.编写方法查询指定部门的所有员工信息，返回List<Employee>集合

要求：以上所有的增删改查操作都必须使用DbUtils工具类型相关API完成。

### 操作步骤答案

|  |
| --- |
| **public** **static** **void** main(String[] args)  **throws** Exception{  // 创建表  // createTable();    // 1.添加部门数据  // addDept();    // 2.添加员工数据  // addEmployees();    // 3.修改id=2的员工工资为16000  // updateSalary(2, 16000);    // 4.获得所有员工信息  /\*List<Employee> employees = findAllEmployee();  for (Employee employee : employees) {  System.out.println(employee);  } \*/    // 5. 查询指定姓名的员工信息  /\*Employee e = findEmployeeByName("林青霞");  System.out.println(e);\*/      // 6.删除name为林青霞的员工  // System.out.println(deleteByName("林青霞"));    // 7.编写方法查询所有姓张员工的工资并输出在控制台  // querySalarys();    // 8.查询工资大于10000的员工  /\*List<Employee> employees = findEmployeeBySalary(10000);  for (Employee employee : employees) {  System.out.println(employee);  }\*/    // 9.编写方法查询指定部门的所有员工信息，返回List<Employee>集合  List<Employee> employees = *findEmployeeByDeptName*("研发部");  **for** (Employee employee : employees) {  System.***out***.println(employee);  }  }    // 9.编写方法查询指定部门的所有员工信息，返回List<Employee>集合  **public** **static** List<Employee> findEmployeeByDeptName(String deptName) **throws** SQLException {  // 要执行的SQL语句  String sql = "select \* from employee where deptid in("  + "select id from dept where dept\_name = ?"  + ");";  // 创建QueryRunner对象  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  // 执行sql语句并返回结果  **return** qr.query(sql, **new** BeanListHandler<>(Employee.**class**),deptName);  }    // 8.编写方法接收一个工资参数，方法内查询工资大于等于传入的工资的员工，  // 返回符合条件所有员工信息List<Employee>集合。  **public** **static** List<Employee> findEmployeeBySalary(**double** salary) **throws** SQLException {  // 要执行的SQL语句  String sql = "select \* from employee where salary >= ?";  // 创建QueryRunner对象  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  // 执行sql语句并返回结果  **return** qr.query(sql, **new** BeanListHandler<>(Employee.**class**),salary);  }    // 7.编写方法查询所有姓张员工的工资并输出在控制台，输出格式如下：  **public** **static** **void**  querySalarys() **throws** SQLException {  //要执行的SQL语句  String sql = "select salary,name from employee where name like'张%' ";  // 创建QueryRunner对象  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  // 执行SQL语句并获得返回结果  List<Employee> list = qr.query(sql, **new** BeanListHandler<>(Employee.**class**));  // 遍历集合  **for** (Employee employee : list) {  System.***out***.println(employee.getName() + "=" + employee.getSalary());  }  }    // 6.编写方法根据员工姓名删除指定的员工信息。  **public** **static** **boolean** deleteByName(String name) **throws** SQLException{  // 要执行的SQL语句  String sql = "delete from employee where name = ?";  // 创建QueryRunner对象  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  // 执行SQL语句并获得返回结果  **int** row = qr.update(sql, name);  **return** row == 1;  }    // 5.编写方法查询指定姓名的员工信息，返回Employee对象。  **public** **static** Employee findEmployeeByName(String name) **throws** SQLException{  // 要执行的SQL语句  String sql = "select \* from employee where name = ? " ;  // 创建QueryRunner对象  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  // 执行SQL语句并获得返回结果  **return** qr.query(sql,**new** BeanHandler<>(Employee.**class**),name);  }    // 4.编写方法查询指定职位所有员工的信息，返回List<Employee>集合。  **public** **static** List<Employee> findAllEmployee() **throws** SQLException {  // 要执行的SQL语句  String sql = "select \* from employee";  // 创建QueryRunner对象  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  // 执行sql语句并返回结果  **return** qr.query(sql, **new** BeanListHandler<>(Employee.**class**));  }    // 3.编写方法接收一个员工编号和工资两个参数，方法内将指定编号的员工工资修改为新的工资。  **public** **static** **void** updateSalary(**int** id,**double** salary)  **throws** Exception {  // 创建QueryRunner对象  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  // Sql语句  String sql = "update employee set salary = ? where id = ?;";  // 执行SQL语句  **int** row = qr.update(sql,salary,id);  System.***out***.println("更新了"+row+"行记录");  }    // 添加多条员工数据  **public** **static** **void** addEmployees() **throws** Exception{  // 创建QueryRunner对象  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  // SQL语句  String sql = "insert into employee(name,gender,post,salary,edate,deptid) "  + "values('张三','男','Java研发工程师',9000,'2016-10-10',1),"  + "('林青霞','女','销售经理',15000,'2015-06-15',2),"  + "('李四','男','产品经理',10000,'2014-12-11',3),"  + "('张三丰','男','Android研发工程师',8000,'2017-05-26',1);";  // 执行SQL语句  **int** row = qr.update(sql);  System.***out***.println("插入了"+row+"行记录");  }      // 添加多个部门数据  **public** **static** **void** addDept() **throws** Exception{  // 创建QueryRunner对象  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  // 执行的SQL语句  String sql = "insert into dept(dept\_name) values(?),(?),(?);";  // 执行语句  **int** row = qr.update(sql, **new** Object[]{"研发部","销售部","产品部"});  System.***out***.println("插入了"+row+"行记录");  }    /\*  \*  1.在数据库创建部门表和员工表，包含字段如下：  部门表包含字段有：部门编号，部门名称(唯一且不能为空)  员工表包含字段有：员工编号，员工姓名(唯一且不能为空)，员工性别，员工职位，员工工资，入职日期。部门编号(外键)  \*/  **public** **static** **void** createTable() **throws** Exception{  // 创建部门表Sql  String sql01 = "create table dept(id int primary key auto\_increment,"  + "dept\_name varchar(20) unique not null);";  // 创建员工表Sql  String sql02 = "create table employee(id int primary key auto\_increment,"  + "name varchar(20) unique not null,"  + "gender varchar(2),"  + "post varchar(20),"  + "salary double,"  + "edate date,"  + "deptid int,"  + "constraint foreign key (deptid) references dept(id))";    // 创建QueryRunner对象  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  // 执行创表SQL语句  qr.update(sql01);  qr.update(sql02);  } |