JDBC常见面试题

JDBC操作数据库的步骤？

> JDBC操作数据库的步骤？

1. 注册数据库驱动。
2. 建立数据库连接。
3. 创建一个Statement。
4. 执行SQL语句。
5. 处理结果集。
6. 关闭数据库连接

代码如下：

```
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        try {

            /*
            * 加载驱动有两种方式
            *
            * 1：会导致驱动会注册两次，过度依赖于mysql的api，脱离的mysql的开发包，程序则无法编译
            * 2：驱动只会加载一次，不需要依赖具体的驱动，灵活性高
            *
            * 我们一般都是使用第二种方式
            * */

            //1.
            //DriverManager.registerDriver(new com.mysql.jdbc.Driver());

            //2.
            Class.forName("com.mysql.jdbc.Driver");

            //获取与数据库连接的对象-Connetcion
            connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/zhongfucheng",
"root", "root");

            //获取执行sql语句的statement对象
            statement = connection.createStatement();

            //执行sql语句,拿到结果集
            resultSet = statement.executeQuery("SELECT * FROM users");

            //遍历结果集，得到数据
            while (resultSet.next()) {
```

```
                System.out.println(resultSet.getString(1));

                System.out.println(resultSet.getString(2));
            }

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

            /*
            * 关闭资源，后调用的先关闭
            *
            * 关闭之前，要判断对象是否存在
            * */

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

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

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

            }

        }
```

JDBC中的Statement 和PreparedStatement，CallableStatement的区别?

> JDBC中的Statement 和PreparedStatement的区别?

区别：

- PreparedStatement是预编译的SQL语句，效率高于Statement。

- PreparedStatement支持?操作符，相对于Statement更加灵活。
- PreparedStatement可以防止SQL注入，安全性高于Statement。
- CallableStatement适用于执行存储过程。

JDBC中大数据量的分页解决方法?

> JDBC中大数据量的分页解决方法?

最好的办法是利用sql语句进行分页，这样每次查询出的结果集中就只包含某页的数据内容。

mysql语法：

```
SELECT *
FROM 表名
LIMIT [START], length;
```

oracle语法：

```
SELECT *FROM (
    SELECT 列名,列名,ROWNUM rn
    FROM 表名
    WHERE ROWNUM<=(currentPage*lineSize)) temp

WHERE temp.rn>(currentPage-1)*lineSize;
```

说说数据库连接池工作原理和实现方案?

> 说说数据库连接池工作原理和实现方案?

工作原理：

- JAVA EE服务器启动时会建立一定数量的池连接，并一直维持不少于此数目的池连接。客户端程序需要连接时，池驱动程序会返回一个未使用的池连接并将其表记为忙。如果当前没有空闲连接，池驱动程序就新建一定数量的连接，新建连接的数量有配置参数决定。当使用的池连接调用完成后，池驱动程序将此连接表记为空闲，其他调用就可以使用这个连接。

实现方案：**连接池使用集合来进行装载，返回的Connection是原始Connection的代理，代理Connection的close方法，当调用close方法时，不是真正关连接，而是把它代理的Connection对象放回到连接池中，等待下一次重复利用。**

具体代码：

```
@Override
   public Connection getConnection() throws SQLException {

       if (list.size() > 0) {
           final Connection connection = list.removeFirst();

           //看看池的大小
           System.out.println(list.size());
```

```
            //返回一个动态代理对象
            return (Connection) Proxy.newProxyInstance(Demo1.class.getClassLoader(),
connection.getClass().getInterfaces(), new InvocationHandler() {

                @Override
                public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {

                    //如果不是调用close方法，就按照正常的来调用
                    if (!method.getName().equals("close")) {
                        method.invoke(connection, args);
                    } else {

                        //进到这里来，说明调用的是close方法
                        list.add(connection);

                        //再看看池的大小
                        System.out.println(list.size());

                    }
                    return null;
                }

            });
        }
        return null;
    }
```

Java中如何进行事务的处理?

> Java中如何进行事务的处理?

1. **事务是作为单个逻辑工作单元执行的一系列操作。**
2. 一个逻辑工作单元必须有四个属性，称为**原子性、一致性、隔离性和持久性 (ACID) 属性，只有这样才能成为一个事务**

Connection类中提供了4个事务处理方法:

- setAutoCommit(Boolean autoCommit):设置是否自动提交事务,默认为自动提交,即为true,通过设置false禁止自动提交事务;
- commit():提交事务;
- rollback():回滚事务.
- savepoint:保存点
- - 注意：savepoint不会结束当前事务，普通提交和回滚都会结束当前事务的

修改JDBC代码质量

> 下述程序是一段简单的基于JDBC的数据库访问代码,实现了以下功能:从数据库中查询product表中的所有记录,然后打印输出到控制台.该代码质量较低,如没有正确处理异常,连接字符串以"魔数"的形式直接存在于代码中等,请用你的思路重新编写程序,**完成相同的功能,提高代码质量.**

原来的代码:

```
public void printProducts(){
    Connection c = null;
    Statements s = null;
    ResultSet r = null;
    try{

c=DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:sid","username","password");
        s=c.createStatement();
        r=s.executeQuery("select id, name, price from product");
        System.out.println("Id\tName\tPrice");
        while(r.next()){
            int x = r.getInt("id");
            String y = r.getString("name");
            float z = r.getFloat("price");
            System.out.println(x + "\t" + y + "\t" + z);
        }
    } catch(Exception e){

    }
}
```

修改后的代码:

```
class Constant{
    public static final String URL="jdbc:oracle:thin:@127.0.0.1:1521:sid";
    public static final String USERNAME="username";
    public static final String PASSWORD="password";
}

class DAOException extends Exception{
    public DAOException(){
        super();
    }
    public DAOException(String msg){
        super(msg);
    }
}

public class Test{

    public void printProducts() throws DAOException{
        Connection c = null;
        Statement s = null;
        ResultSet r = null;
        try{
            c = DriverManager.getConnection(Constant.URL,Constant.USERNAME,Constant.PASSWORD);
```

```
            s = c.createStatement();
            r = s.executeQuery("select id,name,price from product");
            System.out.println("Id\tName\tPrice");
            while(r.next()){
                int x = r.getInt("id");
                String y = r.getString("name");
                float z = r.getFloat("price");
                System.out.println(x + "\t" + y + "\t" + z);
            }
        } catch (SQLException e){
            throw new DAOException("数据库异常");
        } finally {
            try{
                r.close();
                s.close();
                c.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

修改点:

- url、password等信息不应该直接使用字符串“写死”，可以使用常量代替
- catch中应该回滚事务，抛出RuntimeException也是回滚事务的一种方法
- 关闭资源

写出一段JDBC连接本机MySQL数据库的代码

> 写出一段JDBC连接本机MySQL数据库的代码

```
Class.forName("com.mysql.jdbc.Driver");
String url="jdbc:mysql://localhost/test";
Stirng user='root';
String password='root';
Connection conn = DriverManager.getConnection(url,user,password);
```

JDBC是如何实现Java程序和JDBC驱动的松耦合的?

> JDBC是如何实现Java程序和JDBC驱动的松耦合的?

通过制定接口，数据库厂商来实现。我们只要通过接口调用即可。随便看一个简单的JDBC示例，**你会发现所有操作都是通过JDBC接口完成的，而驱动只有在通过Class.forName反射机制来加载的时候才会出现。**

execute，executeQuery，executeUpdate的区别是什么?

> execute，executeQuery，executeUpdate的区别是什么?

- Statement的execute(String query)方法用来**执行任意的SQL查询**，如果查询的结果是一个ResultSet，这个方法就返回true。如果结果不是ResultSet，比如insert或者update查询，它就会返回false。我们可以通过它的getResultSet方法来获取ResultSet，或者通过getUpdateCount()方法来获取更新的记录条数。
- Statement的executeQuery(String query)接口用来执行select查询，并且返回ResultSet。即使查询不到记录返回的ResultSet也不会为null。我们通常使用executeQuery**来执行查询语句，这样的话如果传进来的是insert或者update语句的话，它会抛出错误信息为 “executeQuery method can not be used for update”的java.util.SQLException。
- Statement的executeUpdate(String query)方法**用来执行insert或者update/delete（DML）语句，或者 什么也不返回DDL语句**。返回值是int类型，如果是DML语句的话，它就是更新的条数，如果是DDL的话，就返回0。
- 只有当你不确定是什么语句的时候才应该使用execute()方法，否则应该使用executeQuery或者executeUpdate方法。

PreparedStatement的缺点是什么，怎么解决这个问题?

> PreparedStatement的缺点是什么，怎么解决这个问题?

PreparedStatement的一个缺点是，**我们不能直接用它来执行in条件语句**；需要执行IN条件语句的话，下面有一些解决方案：

- 分别进行单条查询——这样做性能很差，不推荐。
- 使用存储过程——这取决于数据库的实现，不是所有数据库都支持。
- 动态生成PreparedStatement——这是个好办法，但是不能享受PreparedStatement的缓存带来的好处了。
- 在PreparedStatement查询中使用NULL值——如果你知道输入变量的最大个数的话，这是个不错的办法，扩展一下还可以支持无限参数。

JDBC的脏读是什么？哪种数据库隔离级别能防止脏读?

> JDBC的脏读是什么？哪种数据库隔离级别能防止脏读?

脏读：**一个事务读取到另外一个事务未提交的数据**

例子：A向B转账，A执行了转账语句，但A还没有提交事务，B读取数据，发现自己账户钱变多了！B跟A说，我已经收到钱了。A回滚事务【rollback】，等B再查看账户的钱时，发现钱并没有多。

下面的三种个隔离级别都可以防止：

- Serializable【TRANSACTION_SERIALIZABLE】
- Repeatable read【TRANSACTION_REPEATABLE_READ】
- Read committed【TRANSACTION_READ_COMMITTED】

什么是幻读，哪种隔离级别可以防止幻读?

> 什么是幻读，哪种隔离级别可以防止幻读?

是指在一个事务内读取到了别的事务插入的数据，导致前后读取不一致。

只有TRANSACTION_SERIALIZABLE隔离级别才能防止产生幻读。

JDBC的DriverManager是用来做什么的?

> JDBC的DriverManager是用来做什么的?

- JDBC的DriverManager是一个**工厂类**，我们**通过它来创建数据库连接。**
- 当JDBC的Driver类被加载进来时，它会自己注册到DriverManager类里面
- 然后我们会把数据库配置信息传成DriverManager.getConnection()方法，**DriverManager会使用注册到它里面的驱动来获取数据库连接，并返回给调用的程序。**

JDBC的ResultSet是什么?

> JDBC的ResultSet是什么?

- **在查询数据库后会返回一个ResultSet，它就像是查询结果集的一张数据表。**
- ResultSet对象维护了一个游标，指向当前的数据行。开始的时候这个游标指向的是第一行。如果调用了ResultSet的next()方法游标会下移一行，如果没有更多的数据了，next()方法会返回false。可以在for循环中用它来遍历数据集。
- 默认的ResultSet是不能更新的，游标也只能往下移。也就是说你只能从第一行到最后一行遍历一遍。不过也可以创建可以回滚或者可更新的ResultSet
- **当生成ResultSet的Statement对象要关闭或者重新执行或是获取下一个ResultSet的时候，ResultSet对象也会自动关闭。**
- 可以通过ResultSet的getter方法，传入列名或者从1开始的序号来获取列数据。

有哪些不同的ResultSet?

> 有哪些不同的ResultSet?

根据创建Statement时输入参数的不同，会对应不同类型的ResultSet。如果你看下Connection的方法，你会发现createStatement和prepareStatement方法重载了，以支持不同的ResultSet和并发类型。

一共有三种ResultSet对象。

- **ResultSet.TYPE_FORWARD_ONLY：这是默认的类型，它的游标只能往下移。**
- **ResultSet.TYPE_SCROLL_INSENSITIVE：游标可以上下移动，一旦它创建后，数据库里的数据再发生修改，对它来说是透明的。**
- **ResultSet.TYPE_SCROLL_SENSITIVE：游标可以上下移动，如果生成后数据库还发生了修改操作，它是能够感知到的。**

ResultSet有两种并发类型。

- ResultSet.CONCUR_READ_ONLY:**ResultSet是只读的，这是默认类型。**
- ResultSet.CONCUR_UPDATABLE:**我们可以使用ResultSet的更新方法来更新里面的数据。**

JDBC的DataSource是什么，有什么好处

> JDBC的DataSource是什么，有什么好处

DataSource即数据源，它是定义在javax.sql中的一个接口，跟DriverManager相比，它的功能要更强大。**我们可以用它来创建数据库连接，当然驱动的实现类会实际去完成这个工作。除了能创建连接外，它还提供了如下的特性：**

- **缓存PreparedStatement以便更快的执行**
- **可以设置连接超时时间**
- **提供日志记录的功能**
- **ResultSet大小的最大阈值设置**
- **通过JNDI的支持，可以为servlet容器提供连接池的功能**

如何通过JDBC的DataSource和Apache Tomcat的JNDI来创建连接池?

> 如何通过JDBC的DataSource和Apache Tomcat的JNDI来创建连接池?

Tomcat服务器也给我们提供了连接池，内部其实就是DBCP

步骤:

1. 在META-INF目录下配置context.xml文件【文件内容可以在tomcat默认页面的 JNDI Resources下Configure Tomcat's Resource Factory找到】
2. 导入Mysql或oracle开发包到tomcat的lib目录下
3. 初始化JNDI->获取JNDI容器->检索以XXX为名字在JNDI容器存放的连接池

context.xml文件的配置:

```
<Context>

  <Resource name="jdbc/EmployeeDB"
            auth="Container"
            type="javax.sql.DataSource"

            username="root"
            password="root"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/zhongfucheng"
            maxActive="8"
            maxIdle="4"/>
</Context>
```

```
        try {

            //初始化JNDI容器
            Context initCtx = new InitialContext();

            //获取到JNDI容器
            Context envCtx = (Context) initCtx.lookup("java:comp/env");

            //扫描以jdbc/EmployeeDB名字绑定在JNDI容器下的连接池
            DataSource ds = (DataSource)
                    envCtx.lookup("jdbc/EmployeeDB");

            Connection conn = ds.getConnection();
            System.out.println(conn);

        }
```

Apache的DBCP是什么?

> Apache的DBCP是什么

如果用DataSource来获取连接的话，**通常获取连接的代码和驱动特定的DataSource是紧耦合的。**另外，除了选择DataSource的实现类，剩下的代码基本都是一样的。

Apache的DBCP就是用来解决这些问题的，它提供的DataSource实现成为了应用程序和不同JDBC驱动间的一个抽象层。**Apache的DBCP库依赖commons-pool库，所以要确保它们都在部署路径下。**

使用DBCP数据源的步骤：

1. 导入两个jar包【Commons-dbcp.jar和Commons-pool.jar】
2. 读取配置文件
3. 获取BasicDataSourceFactory对象
4. 创建DataSource对象

```
    private static DataSource dataSource = null;

    static {
        try {
            //读取配置文件
            InputStream inputStream =
Demo3.class.getClassLoader().getResourceAsStream("dbcpconfig.properties");
            Properties properties = new Properties();
            properties.load(inputStream);

            //获取工厂对象
            BasicDataSourceFactory basicDataSourceFactory = new BasicDataSourceFactory();
            dataSource = basicDataSourceFactory.createDataSource(properties);

        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection() throws SQLException {
        return dataSource.getConnection();

    }

    //这里释放资源不是把数据库的物理连接释放了，是把连接归还给连接池【连接池的Connection内部自己做好了】
    public static void release(Connection conn, Statement st, ResultSet rs) {

        if (rs != null) {
            try {
                rs.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            rs = null;
        }
        if (st != null) {
            try {

                st.close();
```

```
            } catch (Exception e) {
                e.printStackTrace();
            }

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

        }


    }
```

常见的JDBC异常有哪些?

> 常见的JDBC异常有哪些?

有以下这些:

- java.sql.SQLException——这是JDBC异常的基类。
- java.sql.BatchUpdateException——当批处理操作执行失败的时候可能会抛出这个异常。这取决于具体的JDBC驱动的实现，它也可能直接抛出基类异常java.sql.SQLException。
- java.sql.SQLWarning——SQL操作出现的警告信息。
- java.sql.DataTruncation——字段值由于某些非正常原因被截断了（不是因为超过对应字段类型的长度限制）。

JDBC中存在哪些不同类型的锁?

> JDBC中存在哪些不同类型的锁?

从广义上讲，有两种锁机制来防止多个用户同时操作引起的数据损坏。

- **乐观锁——只有当更新数据的时候才会锁定记录。**
- **悲观锁——从查询到更新和提交整个过程都会对数据记录进行加锁。**

java.util.Date和java.sql.Date有什么区别?

> java.util.Date和java.sql.Date有什么区别?

java.util.Date包含日期和时间，而java.sql.Date只包含日期信息，而没有具体的时间信息。**如果你想把时间信息存储在数据库里，可以考虑使用Timestamp或者DateTime字段**

SQLWarning是什么，在程序中如何获取SQLWarning?

> SQLWarning是什么，在程序中如何获取SQLWarning?

SQLWarning是SQLException的子类，**通过Connection, Statement, Result的getWarnings方法都可以获取到它。** SQLWarning不会中断查询语句的执行，只是用来提示用户存在相关的警告信息。

如果java.sql.SQLException: No suitable driver found该怎么办?

> 如果java.sql.SQLException: No suitable driver found该怎么办?

如果你的SQL URL串格式不正确的话，就会抛出这样的异常。不管是使用DriverManager还是JNDI数据源来创建连接都有可能抛出这种异常。它的异常栈看起来会像下面这样。

```
org.apache.tomcat.dbcp.dbcp.SQLNestedException: Cannot create JDBC driver of class
'com.mysql.jdbc.Driver' for connect URL ''jdbc:mysql://localhost:3306/UserDB'
    at
org.apache.tomcat.dbcp.dbcp.BasicDataSource.createConnectionFactory(BasicDataSource.java:1452)
    at org.apache.tomcat.dbcp.dbcp.BasicDataSource.createDataSource(BasicDataSource.java:1371)
    at org.apache.tomcat.dbcp.dbcp.BasicDataSource.getConnection(BasicDataSource.java:1044)
java.sql.SQLException: No suitable driver found for 'jdbc:mysql://localhost:3306/UserDB
    at java.sql.DriverManager.getConnection(DriverManager.java:604)
    at java.sql.DriverManager.getConnection(DriverManager.java:221)
    at com.journaldev.jdbc.DBConnection.getConnection(DBConnection.java:24)
    at com.journaldev.jdbc.DBConnectionTest.main(DBConnectionTest.java:15)
Exception in thread "main" java.lang.NullPointerException
    at com.journaldev.jdbc.DBConnectionTest.main(DBConnectionTest.java:16)
```

解决这类问题的方法就是，检查下日志文件，像上面的这个日志中，URL串是'jdbc:mysql://localhost:3306/UserDB，只要把它改成jdbc:mysql://localhost:3306/UserDB就好了。

JDBC的RowSet是什么，有哪些不同的RowSet?

> JDBC的RowSet是什么，有哪些不同的RowSet?

RowSet用于存储查询的数据结果，和ResultSet相比，**它更具灵活性。RowSet继承自ResultSet，因此ResultSet能干的，它们也能，而ResultSet做不到的，它们还是可以。**RowSet接口定义在javax.sql包里。

RowSet提供的额外的特性有：

- 提供了Java Bean的功能，可以通过settter和getter方法来设置和获取属性。RowSet使用了JavaBean的事件驱动模型，它可以给注册的组件发送事件通知，比如游标的移动，行的增删改，以及RowSet内容的修改等。
- RowSet对象默认是**可滚动，可更新的，因此如果数据库系统不支持ResultSet实现类似的功能，可以使用RowSet来实现。**

RowSet分为两大类：

- A. **连接型RowSet——这类对象与数据库进行连接**，和ResultSet很类似。JDBC接口只提供了一种连接型RowSet，javax.sql.rowset.JdbcRowSet，它的标准实现是com.sun.rowset.JdbcRowSetImpl。
- B. **离线型RowSet——这类对象不需要和数据库进行连接，因此它们更轻量级，更容易序列化。它们适用于在网络间传递数据。**
- - CachedRowSet——可以通过他们获取连接，执行查询并读取ResultSet的数据到RowSet里。我们可以在离线时对数据进行维护和更新，然后重新连接到数据库里，并回写改动的数据。
  - WebRowSet继承自CachedRowSet——他可以读写XML文档。
  - JoinRowSet继承自WebRowSet——它不用连接数据库就可以执行SQL的join操作。
  - FilteredRowSet继承自WebRowSet——我们可以用它来设置过滤规则，这样只有选中的数据才可见。

- 有四种不同的离线型RowSet的实现。

什么是JDBC的最佳实践?

> 什么是JDBC的最佳实践?

- 数据库资源是非常昂贵的，**用完了应该尽快关闭它。**Connection, Statement, ResultSet等JDBC对象都有close方法，调用它就好了。
- **养成在代码中显式关闭掉ResultSet，Statement，Connection的习惯**，如果你用的是连接池的话，连接用完后会放回池里，但是没有关闭的ResultSet和Statement就会造成资源泄漏了。
- 在finally块中关闭资源，保证即便出了异常也能正常关闭。
- **大量类似的查询应当使用批处理完成。**
- **尽量使用PreparedStatement而不是Statement，以避免SQL注入，同时还能通过预编译和缓存机制提升执行的效率。**
- 如果你要将大量数据读入到ResultSet中，**应该合理的设置fetchSize以便提升性能。**
- 你用的数据库可能没有支持所有的隔离级别，用之前先仔细确认下。
- 数据库隔离级别越高性能越差，确保你的数据库连接设置的隔离级别是最优的。
- 如果在WEB程序中创建数据库连接，最好通过JNDI使用JDBC的数据源，这样可以对连接进行重用。
- **如果你需要长时间对ResultSet进行操作的话，尽量使用离线的RowSet。**