当获得了与数据库的连接后，就可以与数据库进行交互了。 JDBC Statement，CallableStatement和PreparedStatement接口定义了可用于发送SQL或*PL/SQL*命令，并从数据库接收数据的方法和属性。
它们还定义了有助于在Java和SQL数据类型的数据类型差异转换的方法。
下表提供了每个接口定义，以及使用这些接口的目的的总结。

接口	推荐使用
Statement	用于对数据库进行通用访问，在运行时使用静态SQL语句时很有用。 Statement接口不能接受参数。
PreparedStatement	当计划要多次使用SQL语句时使用。PreparedStatement接口在运行时接受输入参数。
CallableStatement	当想要访问数据库存储过程时使用。CallableStatement接口也可以接受运行时输入参数。

1. Statement对象

1.1. 创建Statement对象

在使用Statement对象执行SQL语句之前，需要使用Connection对象的createStatement()方法创建一个Statement对象，如以下示例所示：

```
Statement stmt = null;
try {
   stmt = conn.createStatement( );
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

在创建Statement对象后，可以使用它来执行一个SQL语句，它有三个执行方法可以执行。它们分别是 -

- boolean execute (String SQL)： 如果可以检索到ResultSet对象，则返回一个布尔值true; 否则返回false。使用此方法执行SQLDDL语句或需要使用真正的动态SQL，可使用于执行创建数据库，创建表的SQL语句等等。
- int executeUpdate (String SQL): 返回受SQL语句执行影响的行数。使用此方法执行预期会影响多行的SQL语句，例如:INSERT，UPDATE或DELETE语句。
- ResultSet executeQuery(String SQL)：返回一个ResultSet对象。 当您希望获得结果集时，请使用此方法，就像使用SELECT语句一样。

1.2. 关闭Statement对象

就像关闭一个Connection对象一样，以保存数据库资源一样，由于同样的原因，还应该关闭Statement对象。

一个简单的调用close()方法将执行该作业(工作)。 如果先关闭Connection对象，它也会关闭Statement对象。 但是，应该始终显式关闭Statement对象，以确保正确的清理顺序。

```
Statement stmt = null;
try {
   stmt = conn.createStatement( );
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   stmt.close();
}
```

2. PreparedStatement对象

PreparedStatement接口扩展了Statement接口，它添加了比Statement对象更好一些优点的功能。

此语句可以动态地提供/接受参数。

2.1 创建PreparedStatement对象

```
PreparedStatement pstmt = null;
try {
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

setXXX()方法将值绑定到参数，其中XXX表示要绑定到输入参数的值的Java数据类型。 如果忘记提供绑定值，则将会抛出一个SQLException。

每个参数标记是它其顺序位置引用。第一个标记表示位置1，下一个位置2等等。 该方法与Java数组索引不同(它不从0开始)。

所有Statement对象与数据库交互的方法(a)execute()，(b)executeQuery()和(c)executeUpdate()也可以用于PreparedStatement对象。 但是，这些方法被修改为可以使用输入参数的SQL语句。

2.2. 关闭PreparedStatement对象

就像关闭Statement对象一样，由于同样的原因(节省数据库系统资源)，也应该关闭PreparedStatement对象。

简单的调用close()方法将执行关闭。 如果先关闭Connection对象，它也会关闭PreparedStatement对象。 但是，应该始终显式关闭PreparedStatement对象，以确保以正确顺序清理资源。

```
PreparedStatement pstmt = null;
try {
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   pstmt.close();
}
```

3. CallableStatement对象

类似Connection对象创建Statement和PreparedStatement对象一样，它还可以使用同样的方式创建CallableStatement对象，该对象将用于执行对数据库存储过程的调用。

3.1. 创建CallableStatement对象

假设需要执行以下Oracle存储过程 -

```
CREATE OR REPLACE PROCEDURE getEmpName
   (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END;
SQL
```

注意：上面的存储过程是针对Oracle编写的，但是如果您使用MySQL数据库，可使用以下方式来编写MySQL相同的存储过程，如下在EMP数据库中创建它 -

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
   (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END $$
```

DELIMITER ;
SQL
存在三种类型的参数：IN，OUT和INOUT。 PreparedStatement对象只使用IN参数。
CallableStatement对象可以使用上面三个参数类型。
以下是上面三种类型参数的定义 -

参数	描述
IN	创建SQL语句时其参数值是未知的。 使用setXXX()方法将值绑定到IN参数。
OUT	由SQL语句返回的参数值。可以使用getXXX()方法从OUT参数中检索值。
INOUT	提供输入和输出值的参数。使用setXXX()方法绑定变量并使用getXXX()方法检索值。

以下代码片段显示了如何使用Connection.prepareCall()方法根据上述存储过程来实例化一个CallableStatement对象 -

```
CallableStatement cstmt = null;
try {
   String strSQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

String变量strSQL表示存储过程，带有两个参数占位符。
使用CallableStatement对象就像使用PreparedStatement对象一样。 在执行语句之前，必须将值绑定到所有参数，否则将抛出一个SQLException异常。
如果有IN参数，只需遵循适用于PreparedStatement对象的相同规则和技术; 使用与绑定的Java数据类型相对应的setXXX()方法。
使用OUT和INOUT参数时，必须使用一个额外的CallableStatement对象方法registerOutParameter()。 registerOutParameter()方法将JDBC数据类型绑定到存储过程并返回预期数据类型。
当调用存储过程，可以使用适当的getXXX()方法从OUT参数中检索该值。 此方法将检索到的SQL类型的值转换为对应的Java数据类型。

关闭CallableStatement对象

就像关闭其他Statement对象一样，由于同样的原因(节省数据库系统资源)，还应该关闭CallableStatement对象。

简单的调用close()方法将执行关闭CallableStatement对象。 如果先关闭Connection对象，它也会关闭CallableStatement对象。 但是，应该始终显式关闭CallableStatement对象，以确保按正确顺序的清理资源。

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   cstmt.close();
}
```