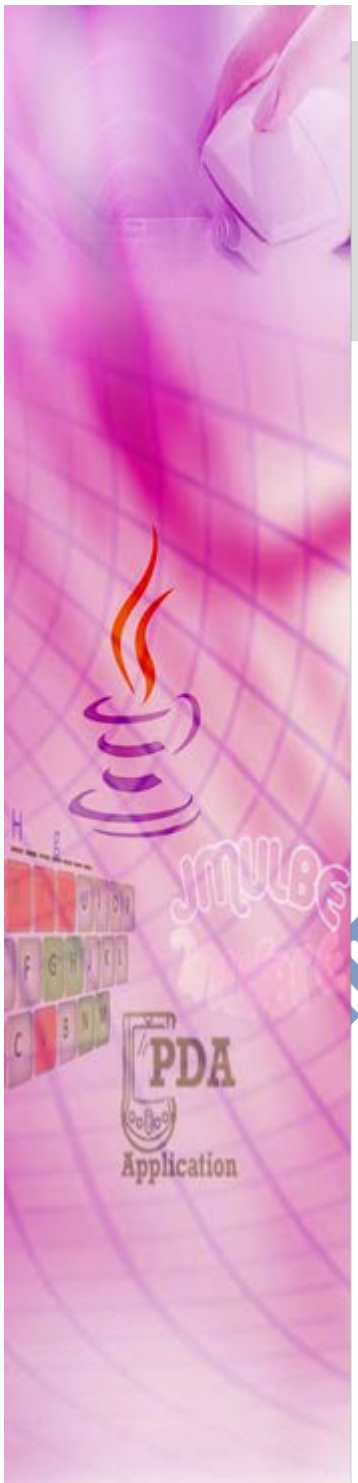



第 9 章

# 介绍 JDBC

Java 数据库连接 (JDBC) 是执行 SQL 语句的 API。它由一组使用 Java 编程语言编写的类和接口组成。接口通过使用 JDBC 能很容易地连接到任何数据库。Java 和 JDBC 结合使用使分发信息的过程变得简单经济。

本章介绍 JDBC 架构。它也概述了 JDBC 支持的各类驱动程序。还详细描述了使用类型 4 JDBC 驱动程序建立与 SQL Server 的连接的方法。

## 目标

在本章中，您将学习：

- 了解 JDBC 架构中的层
- 了解 JDBC 驱动程序的类型
- 使用 JDBC API
- 访问结果集


NIIT
training
china
.com

# 了解 JDBC 架构中的层

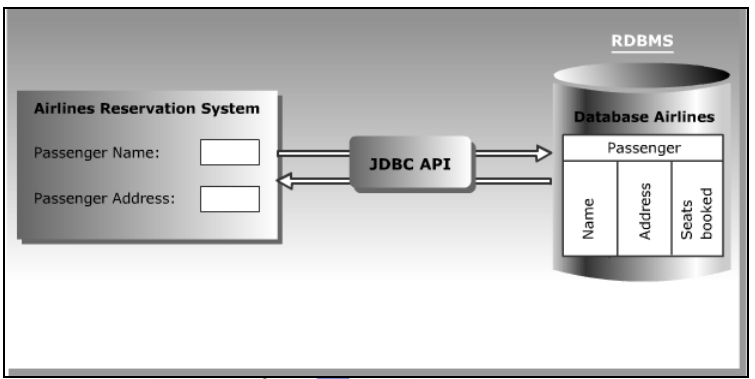
思考场景：您需要为航空公司开发一个应用程序以维护日常事务记录。安装 SQL Server 作为 RDBMS，设计航空数据库，并要求航空人事使用该数据库。航空人事是否可单独使用数据库执行任何任务？

答案是否定的。单单使用 SQL 语句在 SQL Server 数据库中更新数据的任务是个繁复的过程。需要开发用户友好型应用程序以向客户提供通过按键检索、添加和修改数据的选项。

因此，您需要开发与数据库通信从而执行以下任务的应用程序：

- 在数据库中存储和更新数据。
- 检索数据库中存储的数据并将其以适当格式呈现给用户。

下图显示了用 Java 开发的航空预订系统，它使用 JDBC API 与航空 (Airlines) 数据库交互。




*使用 JDBC API 的数据库连接*

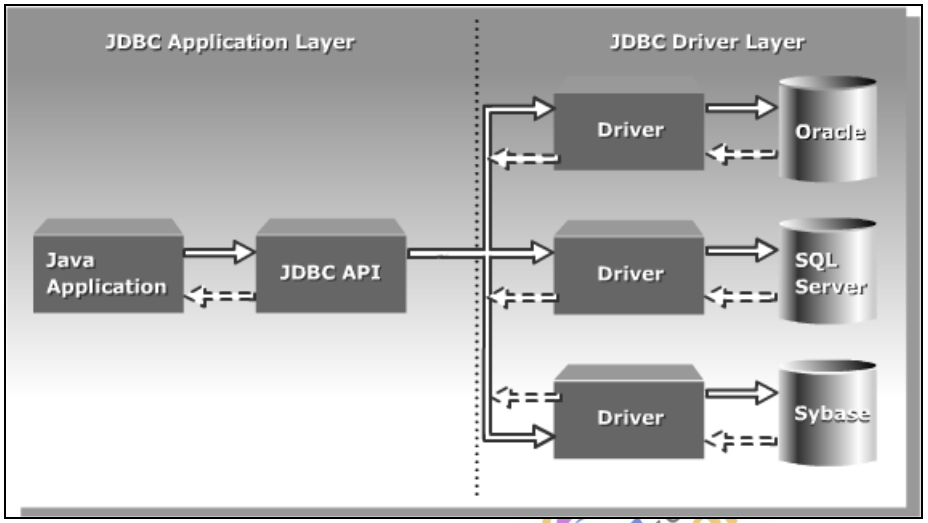
## JDBC 架构

Java 应用程序不能直接与数据库通信来提交数据和检索查询结果。这是因为数据库只能解释 SQL 语句，不能解释 Java 语言语句。因此，您需要一个将 Java 语句翻译成 SQL 语句的机制。JDBC 架构为此类翻译提供机制。

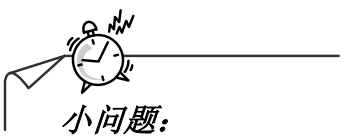
JDBC 架构具有以下两个层：

- **JDBC 应用程序层**：表示使用 JDBC API 与 JDBC 驱动程序交互的 Java 应用程序。JDBC 驱动程序是 Java 应用程序用来访问数据库的软件。
- **JDBC 驱动程序层**：充当 Java 应用程序和数据库之间的接口。这个层包含一个驱动程序，比如 SQL Server 驱动程序或 Oracle 驱动程序，允许连接到数据库。驱动程序向数据库发送 Java 应用程序的请求。处理该请求后，数据库将响应发送回驱动程序。然后驱动程序翻译响应并发送到 JDBC API。JDBC API 最后将该响应转发到 Java 应用程序。

下图显示了 JDBC 架构。




*JDBC 架构*



***小问题：***

*说出 JDBC 架构中的两个层*

***答案：***

*JDBC 架构的两个层是：*

1. *JDBC 应用程序层*
2. *JDBC 驱动程序层*

# 了解 JDBC 驱动程序的类型

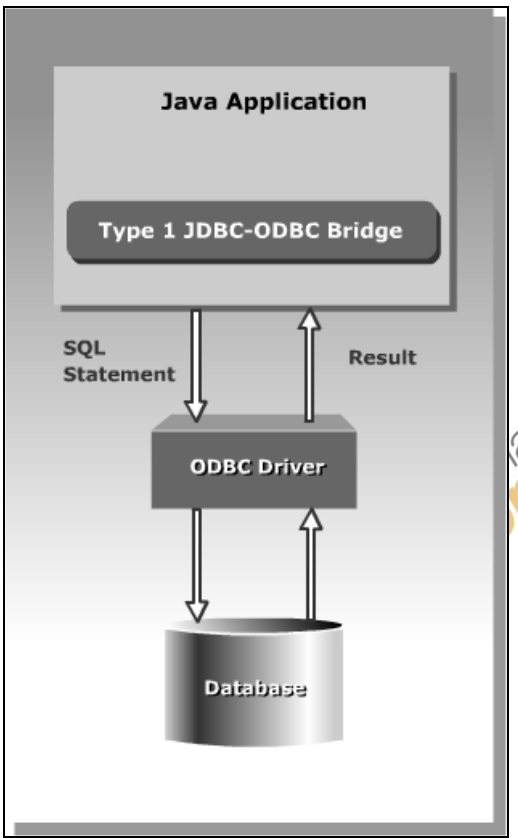
开发 JDBC 应用程序时，需要使用 JDBC 驱动程序将查询转换为特殊数据库可以解释的一种形式。JDBC 驱动程序也检索 SQL 语句的结果并将结果转换成 Java 应用程序可以使用的等效的 JDBC API 类对象。由于 JDBC 驱动程序只管与数据库的交互，数据库的任何更改都不会影响应用程序。JDBC 支持以下类型的驱动程序：

- JDBC ODBC 桥驱动程序
- 本机 API 驱动程序
- 网络协议驱动程序
- 本机协议驱动程序

## JDBC-ODBC 桥驱动程序

JDBC-ODBC 桥驱动程序称为类型 1 驱动程序。JDBC-ODBC 桥驱动程序将 JDBC 方法调用转换为*开放数据库连接 (ODBC)* 函数调用。ODBC 是与数据库通信的开放标准 API。JDBC-ODBC 桥驱动程序使 Java 应用程序能够使用支持 ODBC 驱动程序的任何数据库。Java 应用程序不能直接与 ODBC 驱动程序交互。因此，应用程序使用 JDBC-ODBC 桥驱动程序充当应用程序和 ODBC 驱动程序之间的接口。要使用 JDBC-ODBC 桥驱动程序，需要在客户机上安装 ODBC 驱动程序。JDBC-ODBC 桥驱动程序通常使用在单独的应用程序中。

下图显示了 JDBC-ODBC 桥驱动程序如何工作。

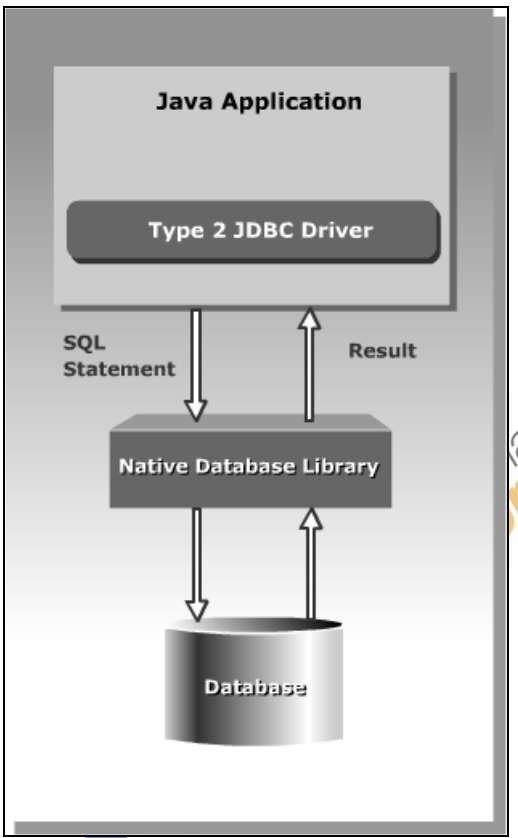



*JDBC-ODBC 桥驱动程序*

## 本机 API 驱动程序

本机 API 驱动程序称为类型 2 驱动程序。它使用数据库供应商提供的本机库来访问数据库。JDBC 驱动程序将 JDBC 调用映射到本机方法调用，它们传递到本机 *调用级接口 (CLI)*。此接口由使用 C 语言编写的函数组成，访问数据库。要使用类型 2 驱动程序，需要在客户机上加载 CLI。与 JDBC-ODBC 桥驱动程序不同的是，本机 API 驱动程序没有 ODBC 中间层。因此，该驱动程序性能要优于 JDBC-ODBC 桥驱动程序，通常用于基于网络的应用程序。

下图显示了本机 API 驱动程序如何工作。

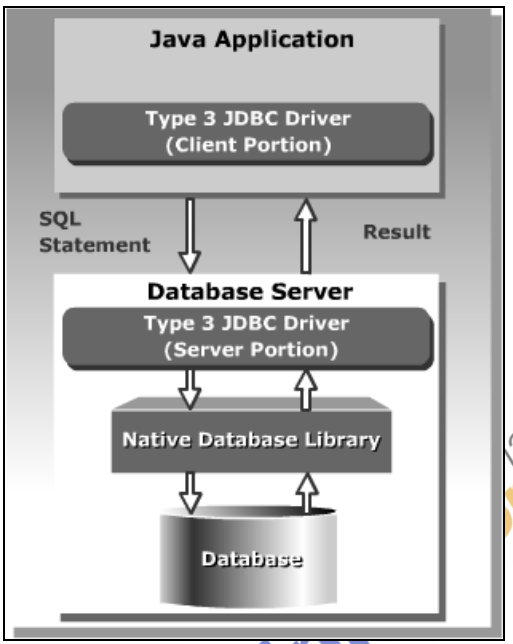



*本机 API 驱动程序*

## 网络协议驱动程序

网络协议驱动程序称为类型 3 驱动程序。网络协议驱动程序由客户机和服务器部分组成。客户机部分包含纯 Java 函数，服务器部分包含 Java 和本机方法。Java 应用程序将 JDBC 调用发送到网络协议驱动程序客户机部分，后者将 JDBC 调用转换为数据库调用。数据库调用被发送到网络协议驱动程序的服务器部分，这部分将请求转发给数据库。使用网络协议驱动程序时，在服务器上加载 CLI 本机库。

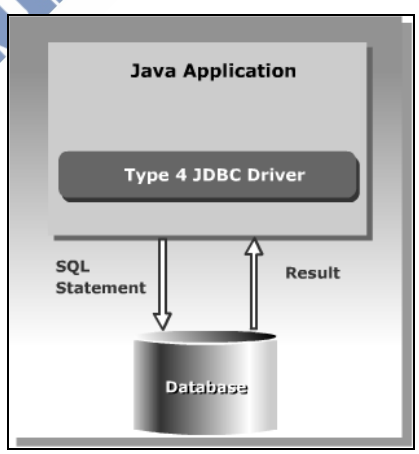
下图显示了网络协议驱动程序如何工作。




*网络协议驱动程序*

## 本机协议驱动程序

本机协议驱动程序称为类型 4 驱动程序。它是一种使用特定于供应商的网络协议直接与数据库交互的 Java 驱动程序。与其他 JDBC 驱动程序不同的是，您无需安装任何供应商特定库就能使用类型 4 驱动程序。每个数据库都具有特定的类型 4 驱动程序。下图显示了本机协议驱动程序如何工作。




*本机协议驱动程序*

# 使用 JDBC API

您需要在开发 Java 应用程序时使用数据库驱动程序和 JDBC API 在数据库中检索或存储数据。JDBC API 类和接口在 `java.sql` 和 `javax.sql` 包中提供。类和接口执行一系列任务，例如建立和关闭与数据库的连接，向数据库发送请求，从数据库检索数据，更新数据库中的数据等。JDBC API 中常用的类和接口有：

- `DriverManager` 类：为数据库加载驱动程序。
- `Driver` 接口：表示数据库驱动程序。所有 JDBC 驱动程序类都必须实现 `Driver` 接口。
- `Connection` 接口：使您能够在 Java 应用程序和数据库之间建立连接。
- `Statement` 接口：能让您执行 SQL 语句。
- `ResultSet` 接口：表示从数据库检索到的信息。
- `SQLException` 类：提供有关在与数据库交互时发生的异常的信息。

要查询数据库并使用 Java 应用程序显示结果，您需要：

- 加载驱动程序。
- 连接到数据库。
- 创建和执行 JDBC 语句。
- 处理 SQL 异常。

## 加载驱动程序

开发 JDBC 应用程序的第一步是使用驱动程序管理器加载和注册所需的驱动程序。可以通过以下方法加载和注册驱动程序：

- 使用 `java.lang.Class` 类的 `forName()` 方法。
- 使用 `DriverManager` 类的 `registerDriver()` 静态方法。

### 使用 `forName()` 方法

`forName()` 方法加载 JDBC 驱动程序并注册该驱动程序。以下签名用于加载 JDBC 驱动程序以访问数据库：

```
Class.forName("package.sub-package.sub-package.DriverClassName");
```

您可以使用以下代码段为 SQL Server 加载 JDBC 类型 4 驱动程序：

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

### 使用 `registerDriver()` 方法

可以创建 JDBC 驱动程序实例并将引用传递到 `registerDriver()` 方法。以下语法用于创建 JDBC 驱动程序的实例：

```
Driver d=new <driver name>;
```

您可以使用以下代码段创建 JDBC 驱动程序的实例：

```
Driver d = new com.microsoft.sqlserver.jdbc.SQLServerDriver();
```

一旦创建了 Driver 对象，调用 registerDriver() 方法注册驱动程序，如以下代码段所示：

```
DriverManager.registerDriver(d);
```

## 连接到数据库

您需要创建 Connection 接口的对象来建立 Java 应用程序与数据库之间的连接。可以在 Java 应用程序中创建多个 Connection 对象来访问和检索多个数据库中的数据。DriverManager 类提供 getConnection() 方法来创建 Connection 对象。getConnection() 方法是具有以下三种形式的重载方法：

- `public static Connection getConnection(String url)`
- `public static Connection getConnection(String url, Properties info)`
- `public static Connection getConnection(String url, String user, String password)`

getConnection (String url) 方法接受数据库的 JDBC URL，您需要作为参数对其进行访问。您可以使用以下代码段，通过具有单个参数的 getConnection() 方法连接数据库：

```
String url =
"jdbc:sqlserver://localhost;user=MyUserName;password=password@123";
Connection con = DriverManager.getConnection(url);
```

以下签名用于作为参数传递到 getConnection() 方法的 JDBC 统一资源定位符 (URL)：

```
<protocol>:<subprotocol>:<subname>
```

JDBC URL 具有以下三个组件：

- protocol：表示用于访问数据库的协议的名称。在 JDBC 中，访问协议的名称始终是 jdbc。
- subprotocol：指示关系数据库管理系统 (RDBMS) 的供应商。例如，如果您使用 SQL Server 的 JDBC 类型 4 驱动程序访问其数据库，那么 subprotocol 的名称将是 sqlserver。
- subname：指示*数据源信息*，其中包含用于访问数据库服务器的数据库信息，例如数据库服务器的位置、数据库的名称、用户名和密码。

## 创建和执行 JDBC 语句

创建连接后，您需要编写要执行的 JDBC 语句。需要创建 Statement 对象将请求发送到数据库并从中检索结果。Connection 对象提供 createStatement() 方法来创建 Statement 对象。您可以使用以下代码段创建 Statement 对象：

```
Connection
con=DriverManager.getConnection("jdbc:sqlserver://sqlserver01;databaseName=Li
brary;user=user1;password=password#1234");
Statement stmt = con.createStatement();
```

可以使用静态 SQL 语句向数据库发送请求。不包含运行时参数的 SQL 语句称为静态 SQL 语句。可以使用 Statement 对象向数据库发送 SQL 语句。Statement 接口包含向数据库发送静态 SQL 语句的以下方法：

- ResultSet executeQuery(String str)：执行 SQL 语句并返回类型 ResultSet 的一个对象。此对象向您提供从结果集访问数据的方法。executeQuery() 方法应该在您需要使用 SELECT 语句从数据库表检索数据时使用。使用 executeQuery() 方法的语法是：

  ```
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery(<SQL statement>);
  ```

  在上述语法中，stmt 是对 Statement 接口对象的引用。executeQuery() 方法执行 SQL 语句，从数据库检索的结果存储在 ResultSet 对象 rs 中。

- int executeUpdate(String str)：执行 SQL 语句并返回在处理 SQL 语句后受影响的行数。当您需要使用数据操作语言 (DML) 语句 INSERT、DELETE 和 UPDATE 修改数据库表中的数据时，可以使用 executeUpdate() 方法。使用 executeUpdate() 方法的语法是：

  ```
  Statement stmt = con.createStatement();
  int count = stmt.executeUpdate(<SQL statement>);
  ```

  在上述语法中，executeUpdate() 方法执行 SQL 语句，数据库中受影响的行数存储在 int 类型变量的 count 中。

- boolean execute(String str)：执行一条 SQL 语句，并返回布尔值。动态执行未知 SQL 字符串时可以使用该方法。如果 SQL 语句的结果是 ResultSet 的对象，execute() 方法返回 true，否则返回 false。使用 execute() 方法的语法是：

  ```
  Statement stmt = con.createStatement();
  stmt.execute(<SQL statement>);
  ```

  可以使用 DML 语句 INSERT、 UPDATE 和 DELETE 在 Java 应用程序中修改存储在数据库表中的数据。还可以使用数据定义语言 (DDL)语句 CREATE、ALTER 和 DROP 在 Java 应用程序中定义或更改数据库对象的结构。

## 查询表

使用 SELECT 语句，您可以从表中检索数据。SELECT 语句是使用 executeQuery() 方法执行的，并且以 ResultSet 对象形式返回输出。您可以使用以下代码段从 Authors 表检索数据：

```
String str = "SELECT * FROM Authors";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(str);
```

在上述代码段中，str 包含从 Authors 表检索数据的 SELECT 语句。结果存储在 ResultSet 对象 rs 中。

需要从表检索选定行时，检索行的条件在 SELECT 语句的 WHERE 子句中指定。您可以使用以下代码段从 Authors 表检索选定行：

```
String str = "SELECT * FROM Authors WHERE city='London'";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(str);
```

在上述代码段中，executeQuery() 方法从 Authors 表检索特殊城市的详细信息。

### 在表中插入行

可以使用 INSERT 语句在现有表中添加行。executeUpdate() 方法使您能在表中添加行。您可以使用以下代码段在 Authors 表中插入行：

```
String str = " INSERT INTO Authors (au_id, au_name, phone, address, city,
state, zip) VALUES ('A004', 'Ringer Albert', '8018260752', ' 67 Seventh Av.',
'Salt Lake City', 'UT', '100000078')";
Statement stmt = con.createStatement();
int count = stmt.executeUpdate(str);
```

在上述代码段中，str 包含需要发送给数据库的 INSERT 语句。Statement 接口的对象 stmt 使用 executeUpdate() 方法执行 INSERT 语句，并将在表中插入的行数返回给 count 变量。

### 在表中更新行

可以使用 UPDATE 语句修改表中的现有信息。您可以使用以下代码段在 Authors 表中修改行：

```
String str = "UPDATE Authors SET address='10932 Second Av.' where au_id=
'A001'";
Statement stmt = con.createStatement();
int count = stmt.executeUpdate(str);
```

在上述代码段中，str 包含需要发送给数据库的 UPDATE 语句。Statement 对象使用 executeUpdate() 方法执行此语句，并将在表中修改的行数返回给 count 变量。

### 从表删除行

可以使用 DELETE 语句从表删除现有信息。您可以使用以下代码段删除 Authors 表中的行：

```
String str = "DELETE FROM Authors WHERE au_id='A005'";
Statement stmt = con.createStatement();
int count = stmt.executeUpdate(str);
```

在上述代码段中，str 包含需要发送给数据库的 DELETE 语句。Statement 对象使用 executeUpdate() 方法执行此语句，并将从表中删除的行数返回给 count 变量。

### 创建表

您可以使用 CREATE TABLE 语句在数据库中创建和定义表结构。您可以使用以下代码段在 Java 应用程序中创建表：

```
String str="CREATE TABLE Publishers"
+"(pub_id VARCHAR(5),"
+"pub_name VARCHAR(50),"
+"phone INTEGER,"
+"address VARCHAR(50), "
+"city VARCHAR(50), "
+"ZIP VARCHAR(20))";
Statement stmt=con.createStatement();
stmt.execute(str);
```

在上述代码段中，str 包含 CREATE TABLE 语句以用于创建 Publishers 表。execute() 方法用于处理 CREATE TABLE 语句。

### 改变或丢弃表

您可以使用 ALTER 语句修改数据库对象的定义。使用 ALTER TABLE 语句修改表结构。例如，使用此语句在表中添加新列、更改现有列的数据类型和宽度，并向列添加约束。可以使用以下代码段通过 ALTER 语句在 Java 应用程序中向 Books 表添加列：

```
String str="ALTER TABLE Books ADD price INTEGER";
Statement stmt=con.createStatement();
stmt.execute(str);
```

您可以使用 DROP 语句从数据库中删除对象。使用 DROP TABLE 语句从数据库丢弃表。您可以使用以下代码段删除使用 Java 应用程序的 MyProduct 表：

```
String str="DROP TABLE MyProduct";
Statement stmt=con.createStatement();
stmt.execute(str);
```

创建 JDBC 应用程序时，需要处理异常。execute() 方法在执行 SQL 语句时会抛出 SQLException。

## 处理 SQL 异常

java.sql 包提供 SQLException 类，它派生自 java.lang.Exception 类。SQLException 由 JDBC API 中的各种方法抛出，能让您确定在将 Java 应用程序连接到数据库时发生的错误的原因。可以使用 try 和 catch 异常处理块在 Java 应用程序中捕捉 SQLException。SQLException 类提供以下错误信息：

- **错误消息**：是描述错误的字符串。
- **错误代码**：是与错误相关的整数值。错误代码是供应商特定的，取决于使用的数据库。
- **SQL 状态**：是标识错误的 *XOPEN* 错误代码。各个供应商提供不同的错误消息来定义相同错误。因此，错误可能有不同的错误消息。XOPEN 错误代码是与错误相关的标准消息，可以在多个数据库中标识错误。

SQLException 类包含各种方法，提供错误信息。SQLException 类中的一些方法有：

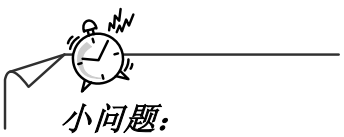
- int getErrorCode()：返回与发生错误相关的错误代码。
- String getSQLState()：返回 SQLException 对象的 SQL 状态。
- SQLException getNextException()：返回异常链中的下一个异常。

例如，您可以使用以下代码段捕捉 SQLException：

```
try
{
      String str = "DELETE FROM Authors WHERE au_id='A002'";
      Statement stmt = con.createStatement();
      int count = stmt.executeUpdate(str);
```

```
}
catch(SQLException sqlExceptionObject)
{
      System.out.println("Display Error Code");
           System.out.println("SQL Exception "+
           sqlExceptionObject.getErrorCode());
}
```

在上述代码段中，如果 DELETE 语句抛出 SQLException 异常，那么它将由 catch 块处理。sqlExceptionObject 是 SQLException 类的对象，用于调用 getErrorCode() 方法。



***小问题：***

*说出 java.sql 包的哪个接口必须被所有 JDBC 驱动程序类实现。*

***答案：***

*Driver*

## 活动 9.1：创建 JDBC 应用程序以查询数据库

# 访问结果集

当您执行查询使用 Java 应用程序从表检索数据时，查询的输出以表格格式存储在 ResultSet 对象中。ResultSet 对象保留一个*光标*，使您能够在 ResultSet 对象中存储的行之间移动。ResultSet 对象默认保留仅朝前移动的光标。因此，它从 ResultSet 中的第一行移动到最后一行。 此外，默认 ResultSet 对象不可更新，意味着不能更新默认对象中的行。ResultSet 对象中的光标起初指在第一行之前。

## 结果集类型

可以在执行 SQL 语句后创建各种类型的 ResultSet 对象来存储数据库返回的输出。ResultSet 对象的各种类型有：

- **只读**：允许您只读 ResultSet 对象中的行。
- **只进**：允许您只以向前的方向从第一行向最后一行移动结果集光标。
- **可滚动**：允许您在结果集中向前或向后移动结果集光标。
- **可更新**：允许更新从数据库表检索的结果集行。

可以在使用 Connection 接口的 createStatement() 方法时指定 ResultSet 对象的类型。createStatement() 方法接受 ResultSet 字段作为参数来创建不同类型的 ResultSet 对象。

下表列出了 ResultSet 接口的各种字段。

| ***ResultSet 字段*** | ***描述*** |
|---|---|
| *TYPE_SCROLL_SENSITIVE* | *指定 ResultSet 对象的光标可滚动，它反映其他用户对数据所作的更改。* |
| *TYPE_SCROLL_INSENSITIVE* | *指定 ResultSet 对象的光标可滚动，但不反映其他用户对数据所作的更改。* |
| *TYPE_FORWARD_ONLY* | *指定 ResultSet 对象的光标只从第一行向最后一行向前移动。* |
| *CONCUR_READ_ONLY* | *指定不允许您更新 ResultSet 对象的并发模式。* |
| *CONCUR_UPDATABLE* | *指定允许您更新 ResultSet 对象的并发模式。* |
| *CLOSE_CURSOR_AT_COMMIT* | *指定在提交当前事务时关闭开放 ResultSet 对象的可保持性方式。* |

| *ResultSet 字段* | *描述* |
|---|---|
| *HOLD_CURSOR_OVER_COMMIT* | *指定在提交当前事务时保持 ResultSet 对象处于打开状态的可保持性方式。* |

*ResultSet 接口的字段*

createStatement() 方法具有以下重载形式：

- Statement createStatement()：不接受任何参数。该方法创建 Statement 对象以向数据库发送 SQL 语句。它创建只允许向前滚动的默认 ResultSet 对象。
- Statement createStatement(int resultSetType, int resultSetConcurrency):接受两个参数。第一个参数指示 ResultSet 类型，它决定结果集光标是可滚动还是仅向前。第二个参数指出结果集的并发模式，决定结果集中的数据是可更新还是只读。此方法创建具有给定类型和并发的 ResultSet 对象。
- Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability):接受三个参数。除了 ResultSet 类型和并发模式，此方法还接受第三个参数。该参数指示开放结果集对象的可保持性方式。此方法创建具有给定类型、并发和可保持性方式的 ResultSet 对象。

## ResultSet 接口的方法

ResultSet 接口包含使您能够在结果集中移动光标的各种方法。下表列出 ResultSet 接口的一些常用方法。

| *方法* | *描述* |
|---|---|
| *boolean first()* | *将结果集光标的控制转移到结果集的第一行。* |
| *boolean isFirst()* | *确定结果集光标是否指向结果集的第一行。* |
| *void beforeFirst()* | *将结果集光标的控制转移到结果集的第一行之前。* |
| *boolean isBeforeFirst()* | *确定结果集光标是否指向结果集的第一行之前。* |
| *boolean last()* | *将结果集光标的控制转移到结果集的最后一行。* |
| *boolean isLast()* | *确定结果集光标是否指向结果集的最后一行。* |
| *void afterLast()* | *将结果集光标的控制转移到结果集的最后一行之后。* |

| *方法* | *描述* |
|---|---|
| *boolean isAfterLast()* | *确定结果集光标是否指向结果集的最后一行之后。* |
| *boolean previous()* | *将结果集光标的控制转移到结果集的上一行。* |
| *boolean absolute(int i)* | *将结果集光标控制转移到指定为参数的行号。* |
| *boolean relative(int i)* | *将结果集光标控制相对于指定为参数的行号向前或向后转移。此方法接受正值或负值作为参数。* |

*ResultSet 接口的方法*

可以创建可滚动的结果集从而在结果集的行中向前或向后滚动。您可以使用以下代码段创建只读可滚动结果集：

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet rs=stmt.executeQuery ("SELECT * FROM Authors");
```

可以使用 ResultSet 接口中的方法确定结果集光标的位置。可以使用以下代码段确定结果集光标是否在结果集中的第一行之前：

```
if(rs.isBeforeFirst()==true)
System.out.println("Result set cursor is before the first row in the result
set");
```

在上述代码段中，rs 是调用 isBeforeFirst() 方法的 ResultSet 对象。

可以使用 ResultSet 接口中的这些方法在结果集中移动到特殊行，比如第一行或最后一行。可以使用以下代码段将结果集光标移动到结果集中的第一行：

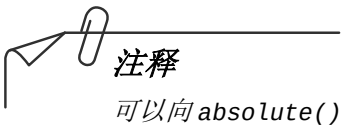
```
if(rs.first()==true)
System.out.println(rs.getString(1) + ", " + rs.getString(2)+ ", " +
rs.getString(3));
```

在上述代码段中，rs 是调用 first() 方法的 ResultSet 对象。

同样，可以使用 last() 方法在结果集中将结果集光标移动到最后一行。

要移动结果集的任何特殊行，可以使用 absolute() 方法。例如，如果结果集光标位于第一行而您想滚动到第四行，应该在调用 absolute() 方法时输入 4 作为参数，如以下代码段所示：

```
System.out.println("Using absolute() method");
rs.absolute(4);
int rowcount = rs.getRow();
System.out.println("row number is " + rowcount);
```



*注释*

*可以向 absolute() 方法传递负值以将光标设置到与结果集最后一行有关的行。例如，要将光标设置到最后一行的前一行，指定 rs.absolute(-2)。*

JDBC 允许您创建使您能够在结果集中修改行的可更新结果集。下表列出了一些可与可更新结果集一起使用的方法。

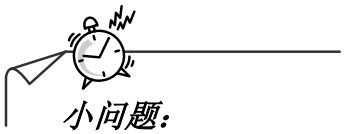
| *方法* | *描述* |
|---|---|
| *void updateRow()* | *更新 ResultSet 对象和基础数据库表的当前行。* |
| *void insertRow()* | *在当前 ResultSet 对象和基础数据库表中插入一行。* |
| *void deleteRow()* | *从 ResultSet 对象和基础数据库表中删除当前行。* |
| *void updateString(int columnIndex, String x)* | *使用给定字符串值更新指定列。它接受需要更改其值的列索引。* |
| *void updateString(String columnLabel, String x)* | *使用给定字符串值更新指定列。它接受需要更改其值的列名。* |
| *void updateInt(int columnIndex, int x)* | *使用给定 int 值更新指定列。它接受需要更改其值的列索引。* |
| *void updateInt(String columnLabel, int x)* | *使用给定 int 值更新指定列。它接受需要更改其值的列名。* |

*用于可更新结果集的方法*

您可以使用以下代码段来修改使用可更新结果集的作者信息：

```
Statement stmt = con.createStatement();
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT au_id, city, state FROM Authors
WHERE au_id='A004'");
rs.next();
rs.updateString("state", "NY");
rs.updateString("city", "Columbia");
rs.updateRow();
```

在上述代码段中，从作者 id 是 A004 的 Authors 表中检索行。在检索的行中，state 列中的值被更改为 NY， city 列中的值被更改为 Columbia。



***小问题：***

*指出 ResultSet 接口中允许光标可滚动并在数据中反映更改的字段。*

1. *TYPE_SCROLL_SENSITIVE*
2. *TYPE_SCROLL_INSENSITIVE*
3. *CONCUR_READ_ONLY*
4. *CONCUR_UPDATAB*

***答案：***

1. *TYPE_SCROLL_SENSITIVE*

除了 ResultSet，Java 还提供了 RowSet。RowSet 接口提供已连接和未连接环境。在已连接的环境中，RowSet 对象使用 JDBC 驱动程序建立与基础数据库的连接并在连接生命周期内保持连接。而另一方面，在未连接环境中，RowSet 对象建立与基础数据库的连接只为读取 ResultSet 对象数据或写回数据库。读取或写入数据后，RowSet 对象断开连接。

为建立已连接或未连接的环境，JDBC API 提供了以下接口：

- JdbcRowSet：提供与 ResultSet 对象最相似的已连接环境。它使 ResultSet 对象可滚动并可更新。
- CachedRowSet：提供未连接环境。它具有 JdbcRowSet 对象的所有功能。此外，它还具有以下附加功能：
  - 它获取与数据源的连接并执行查询。
  - 并从 ResultSet 对象读取数据，将数据填充到 CachedRowSet 对象中。
  - 它在断开连接时更改数据。
  - 它重新连接到数据源以将更改写回数据源。
  - 并检查和解决与数据源的冲突。
- WebRowSet：提供未连接环境。它具有 CachedRowSet 对象的所有功能。此外，它还将其对象转换为可扩展标记语言 (XML) 文档。并且它还使用 XML 文档填充其对象。
- JoinRowSet：提供未连接环境。它具有 WebRowSet 对象的所有功能。此外，它允许您在 RowSet 对象之间创建 SQL JOIN。
- FilteredRowSet：提供未连接环境。它具有 JoinRowSet 对象的所有功能。此外，它应用过滤器以使选定数据可用。

上述接口可用于特定需要，例如已连接或未连接。然而，还有其他方法提供 RowSet 实现。为此，RowSetProvider 类提供 API 以获取可用于实例化适当 RowSet 实现的 RowSetFactory 实现。它提供以下两种方法：

- static RowSetFactory newFactory()：用于创建 RowSetFactory 实现的实例。
- static RowSetFactory newFactory(String factoryClassName, ClassLoader c1):用于从指定的工厂类名创建 RowSetFactory 实例。

RowSetFactory 接口定义用于获取不同类型 RowSet 实现的工厂类实现。它提供以下五种方法：

- CachedRowSet createCachedRowSet()：用于创建 CachedRowSet 实例。
- FilteredRowSet createFilteredRowSet()：用于创建 FilteredRowSet 实例。
- JdbcRowSet createJdbcRowSet()：用于创建 JdbcRowSet 实例。
- JoinRowSet createJoinRowSet()：用于创建 JoinRowSet 实例。
- WebRowSet createWebRowSet()：用于创建 WebRowSet 实例。

# 练习问题

1. 以下哪种方法用于将光标移到结果集中的特定行？
   a. `absolute()`
   b. `first()`
   c. `isFirst()`
   d. `relative()`

2. JDBC URL 的以下哪个组件指示 RDBMS 的供应商？
   a. Subname
   b. Supername
   c. Subprotocol name
   d. Superprotocol name

3. 指出 `ResultSet` 接口中允许光标可滚动但不在数据中反映更改的字段。
   a. `TYPE_SCROLL_INSENSITIVE`
   b. `TYPE_FORWARD_ONLY`
   c. `TYPE_SCROLL_SENSITIVE`
   d. `TYPE_READ_ONLY`

4. `Connection` 对象提供 _____ 方法来创建 `Statement` 对象。

5. 下面哪个 Java 驱动程序使用供应商特定的网络协议直接与数据库交互？
   a. JDBC ODBC 桥驱动程序
   b. 本机 API 驱动程序
   c. 网络协议驱动程序
   d. 本机协议驱动程序

# 小结

在本章中，您学习了：

- JDBC 架构具有以下两个层：
  - JDBC 应用程序层
  - JDBC 驱动程序层
- JDBC 支持以下四种驱动程序：
  - JDBC ODBC 桥驱动程序
  - 本机 API 驱动程序
  - 网络协议驱动程序
  - 本机协议驱动程序
- JDBC API 的类和接口在 `java.sql` 和 `javax.sql` 包中定义。
- 您可以加载驱动程序并使用 `forName()` 方法或 `registerDriver()` 方法将其注册到驱动程序管理器。
- `Connection` 对象在 Java 应用程序和数据库之间建立连接。
- `Statement` 对象发送请求并将结果检索到数据库或从数据库检索结果。
- 可以在 Java 应用程序中使用 DML 语句在表中插入、更新和删除数据。
- 可以在 Java 应用程序中使用 DDL 语句在数据库中创建、更改和删除表。
- 执行 SQL 语句后，`ResultSet` 对象将存储从数据库中检索的结果。
- 您可以创建各种类型的 `ResultSet` 对象，例如：只读、可更新和只进。