# 第 2 章

# Java Web 应用实现数据库访问

## 技能目标

- 会使用 JDBC 读取数据
- 会使用接口优化业务逻辑
- 掌握连接池与数据源
- 掌握 JavaBean 的使用
- 掌握 JSP 标签

## 本章任务

学习本章，需要完成以下 4 个工作任务。记录学习过程中遇到的问题，可以通过自己的努力或访问 kgc.cn 解决。

任务 1：在 Java 中实现新闻信息的查询

使用 JDBC 访问新闻系统数据库，查询新闻信息并在控制台显示。

任务 2：使用 JDBC 编辑新闻信息

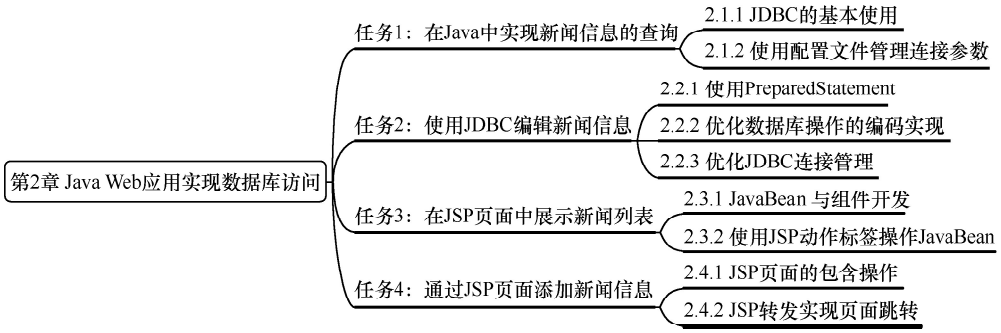
升级任务 1，实现对新闻信息的编辑（新增）操作，并将新增后的新闻信息在控制台显示。

任务 3：在 JSP 页面中展示新闻列表

使用 JDBC 访问数据库，从数据库中读取新闻信息，在 JSP 中以列表方式显示新闻信息。

任务 4：通过 JSP 页面添加新闻信息

在新闻系统中，编写代码实现新闻信息添加功能。




# 任务 1　在 Java 中实现新闻信息的查询

关键步骤如下。

- 使用 JDBC 访问数据库。
- 使用 JDBC 操作数据库。
- 将查询到的数据在控制台输出显示。

## 2.1.1　JDBC 的基本使用

在之前的学习中，新闻标题等数据的存储和显示都是在 JSP 中直接通过变量来实现的，但是当页面中需要显示大量的数据信息时，就不能再使用变量来实现了。使用数据库来存储数据，通过访问数据库实现数据的读取和编辑，是进行项目开发必须要掌握的一门技术。

1．JDBC 技术

（1）JDBC 的概念

JDBC（Java DataBase Connectivity）是一种 Java 数据库连接技术，能实现 Java 程序对各种数据库的访问。由一组使用 Java 语言编写的类和接口组成，这些类和接口称为 JDBC API，它们位于 java.sql 以及 javax.sql 包中。

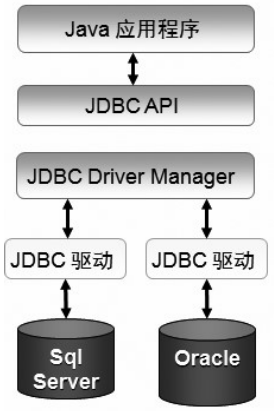



图 2.1　JDBC 工作原理

（2）JDBC 的作用

在项目开发中，使用 JDBC 可以实现应用程序与数据库之间的数据通信，简单来说，JDBC 的作用有以下 3 点。

1）建立与数据库之间的访问连接。

2）将编写好的 SQL 语句发送到数据库执行。

3）对数据库返回的结果进行处理。

（3）JDBC 的工作原理

JDBC 在执行时有一套固定的流程，图 2.1 所示为 JDBC

的工作原理。

从图 2.1 中可以看到一个 JDBC 程序有几个重要的组成要素。顶层是自己编写的 Java 应用程序，Java 应用程序可以使用 java.sql 和 javax.sql 包中的 JDBC API 来连接和操作数据库。了解 JDBC 工作原理请扫描二维码。

JDBC 原理

2. 使用 JDBC 访问数据库

（1）JDBC API

使用 JDBC 访问数据库，就必须要使用到 JDBC API。JDBC API 可以完成 3 件事情：与数据库建立连接、发送 SQL 语句和处理数据库返回的结果，如图 2.2 所示。

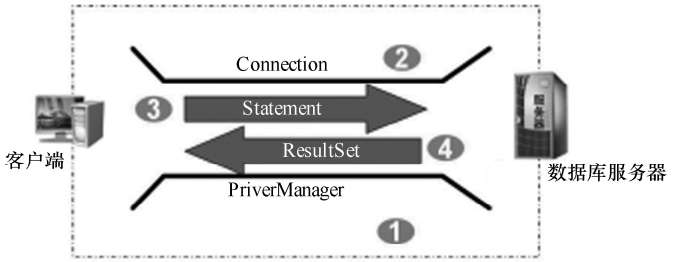



图 2.2　JDBC API

图 2.2 展示了在使用 JDBC API 时，JDBC API 工作的 4 个重要环节涉及的相关类和接口。

1）DriverManager 类：负责依据数据库的不同，管理 JDBC 驱动。

2）Connection（连接）接口：负责连接数据库并担任传送数据的任务。

3）Statement 接口：由 Connection 产生，负责执行 SQL 语句。

4）ResultSet 接口：负责保存 Statement 执行后所产生的执行结果。

（2）JDBC 访问数据库的步骤

实现数据库的访问，需要执行以下几个步骤。

1）使用 Class.forName() 方法加载 JDBC 驱动类。如果系统中不存在给定的类，则会引发异常，异常类型为“ClassNotFoundException”。加载驱动的语法如下：

```
Class.forName("JDBC 驱动类的名称");
```

2）使用 DriverManager 类获取数据库的连接。

DriverManager 类跟踪已注册的驱动程序，当调用 getConnection() 方法时，它会搜索整个驱动程序列表，直到找到一个能够连接至数据库连接字符串中指定的数据库的驱动程序。加载此驱动程序之后，将使用 DriverManager 类的 getConnection() 方法建立与数据库的连接。此方法接收 3 个参数，分别表示数据库 URL、数据库用户名和密码。其中，数据库用户名和密码是可选的。获取数据库连接的语法如下：

```
Connection connection = DriverManager.getConnection( 数据库 URL, 数据库用户名 , 密码 );
```

**示例 1**

使用 JDBC 访问新闻系统数据库，加载数据库驱动，获取数据库连接。

关键代码：

```
public void getNewsList(){
   try {
        //(1) 使用 Class.forName() 加载驱动
        Class.forName("com.mysql.jdbc.Driver");
        //(2) 获得数据库连接
        Connection connection=DriverManager.getConnection("jdbc:mysql://localhost:3306/news",
        "root","root");
   } catch (ClassNotFoundException e) {
        e.printStackTrace();
   }
}
```

使用 JDBC 访问数据库的第一步就是要加载驱动，然后是获取数据库连接，这个过程可能会产生异常，所以在代码中使用了 try-catch 语句对异常进行捕捉处理。

3）发送 SQL 语句，并得到结果集。一旦建立连接，就可以使用该连接创建 Statement 接口的实例，并将 SQL 语句传递给它所连接的数据库，若执行的是查询语句，会返回类型为 ResultSet 的对象，它包含执行 SQL 查询语句的结果。

创建 Statement 接口实例的语句如下。

```
Statement stmt = connection.createStatement();
```

获取结果集对象的语句如下。

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM table1");
```

**示例 2**

在示例 1 的基础上，执行 SQL 语句并获得结果集。

关键代码：

```
……
   //(1) 使用 Class.forName() 加载驱动
   Class.forName("com.mysql.jdbc.Driver");
   //(2) 获得数据库连接
   Connection connection=DriverManager.getConnection("jdbc:mysql://localhost:3306/news",
   "root","root");
   //(3) 获得 Statement 对象， 执行 SQL 语句
   String sql="select * from news_detail";
   Statement stmt=connection.createStatement();
   ResultSet rs=stmt.executeQuery(sql);
……
```

4）处理结果。

执行 SQL 查询语句后，会返回一个结果集 ResultSet 对象。对结果集进行处理的步骤概括如下。

- 使用 ResultSet 对象的 next() 方法判断结果集是否包含数据。
- 在 next() 方法返回 true 的情况下调用 ResultSet 对象的 getXxx() 方法，得到记录中字段对应的值。

**示例 3**

在示例 2 的基础上，对结果集进行处理。

关键代码：

```
……
  //(4) 处理执行结果 (ResultSet)
  while(rs.next()){
      int id=rs.getInt("id");
      String title=rs.getString("title");
      String summary=rs.getString("summary");
      String content=rs.getString("content");
      String author=rs.getString("author");
      Timestamp time=rs.getTimestamp("createdate");
      System.out.println(id + "\t" + title + "\t" + summary + "\t"+
      content + "\t" + author + "\t" + time);
  }
……
```

5）释放资源。在结束数据库访问后，应及时地释放资源。释放资源时需要注意如下两个问题。

- 释放资源应按照创建的顺序逐一进行释放，先创建的后释放，后创建的先释放。
- 由于资源释放不考虑程序本身运行是否正常，所以将释放资源置于 finally 语句块中，确保程序最终会执行资源释放的语句。

**示例 4**

结束数据库访问后，释放资源。

关键代码：

```
……
finally{
  //(5) 释放资源
  try {
      if(rs!=null){
          rs.close();                  // 关闭结果集对象
      }
      if(stmt!=null){
          stmt.close();                // 关闭 Statement 对象
```

```
            }
            if(connection!=null){
                connection.close();                    // 关闭连接对象
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    ……
```

**注意**

如果对一个已经关闭的或者没有实例化的 Connection 对象进行关闭，系统会抛出异常。所以在关闭时，一个良好的习惯是，首先对关闭对象进行判断，判断其是否为 NULL，然后再确定是否关闭。

### 2.1.2 使用配置文件管理连接参数

使用 JDBC 访问数据库时，除了可以把数据库参数写在代码中，还可以使用配置文件的形式保存数据库连接参数。使用配置文件方式访问数据库的优势在于，可以一次编写，随时调用，并且一旦数据库发生变化，只需要修改配置文件即可，无需修改源代码。

1. 配置文件的创建与设置

（1）配置文件的创建

在项目中创建配置文件的方式很简单，具体的操作步骤在这里不再赘述。需要强调的是，配置文件的扩展名是“.properties”。

（2）配置文件的设置

创建好配置文件后，就可以在配置文件中进行数据库参数的相关配置。在配置文件中，采用 key-value 对（键—值对）的方式进行内容的组织。

**示例 5**

修改新闻系统数据库访问方式，通过配置文件来存储访问信息。

关键代码：

```
jdbc.driver_class=com.mysql.jdbc.Driver
jdbc.connection.url=jdbc:mysql://localhost:3306/news
jdbc.connection.username=root
jdbc.connection.password=root
```

**提示**

以 key-value 对方式进行配置文件的编写，等号左边表示键（key），等号右边表示值（value）。

### 2. 读取配置文件

由于将数据库参数保存在配置文件中，所以在进行数据库连接时就需要对配置文件进行读取。在本书中，使用 Properties 对象的 load() 方法来实现配置文件的读取，这就涉及使用流来实现文件的读操作。

通常在进行文件读取时，都会将方法置于一个工具类中，并在构造这个工具类的同时来进行配置文件的读取。

**示例 6**

构建数据库访问的工具类，用于读取配置文件。

关键代码：

```
// 读取配置文件 （属性文件） 的工具类
public class ConfigManager {
    private static ConfigManager configManager;
    private static Properties properties;
    // 在构造工具类时， 进行配置文件的读取
    private ConfigManager(){
      String configFile="database.properties";
      properties=new Properties();
      InputStream in=ConfigManager.class.getClassLoader().getResource AsStream (configFile);
      try {
            // 读取配置文件
            properties.load(in);
            in.close();
       } catch (IOException e) {
            e.printStackTrace();
       }
    }
    // 通过单例模式设置实例化的个数
    public static ConfigManager getInstance(){
       if(configManager==null){
            configManager=new ConfigManager();
       }
       return configManager;
    }
    // 通过 key 获取对应的 value
    public String getString(String key){
       return properties.getProperty(key);
    }
}
```

在工具类编写完成后，就可以在程序中进行调用了，先通过读取配置文件，获取连接数据库相关的参数信息，然后建立数据库连接，获取到相关访问数据后，再实现对数

据库的操作。

**示例 7**

使用配置文件方式实现数据库访问。

关键代码：

```
public void getNewsList(){
    Connection connection=null;
    Statement stmt=null;
    ResultSet rs=null;
    // 通过工具类读取配置文件相关信息
    String driver=ConfigManager.getInstance().getString("jdbc.driver_class");
    String url=ConfigManager.getInstance().getString("jdbc.connection.url");
    String username=ConfigManager.getInstance().getString("jdbc.connection.username");
      String password=ConfigManager.getInstance().getString("jdbc.connection.password");
      try {
          //(1) 使用 Class.forName() 方法加载驱动
          Class.forName(driver);
          //(2) 获得数据库连接
          connection=DriverManager.getConnection(url,username,password);

          //(3) 获得 Statement 对象， 执行 SQL 语句
          ……
}
```

在示例 7 中，将读取新闻信息时的数据库访问方式修改成通过配置文件读取方式实现，修改完毕后，运行程序依然可以实现数据的读取显示。

## 任务 2 使用 JDBC 编辑新闻信息

关键步骤如下。

- 使用 JDBC 访问数据库。
- 使用 PreparedStatement 对象实现信息编辑。
- 对执行结果进行处理。

### 2.2.1 使用 PreparedStatement

1. PreparedStatement 对象

PreparedStatement 接口继承自 Statement 接口，PreparedStatement 对象比普通的 Statement 对象使用起来更加灵活、更有效率。

PreparedStatement 实例包含已编译的 SQL 语句，SQL 语句可具有一个或多个输入

参数。这些输入参数的值在 SQL 语句创建时未被指定，而是为每个输入参数保留一个问号“?”作为占位符。

在执行 PreparedStatement 对象之前，必须设置每个输入参数的值。可通过调用 setXxx() 方法来完成，其中 Xxx 是与该参数对应的类型。例如，如果参数是 Java 类型 int，则使用的方法就是 setInt()。

setXxx() 方法的第一个参数是要设置的参数的序数位置，第二个参数是设置给该参数的值。

**注意**

① 如果数据类型为日期格式，可采用如下语句。

```
setTimestamp( 参数位置 , new java.sql.Timestamp(createdate.getTime()));
```

createdate 为一个日期对象的实例。

② 如果数据类型为 CLOB 类型，则可以将其视为 String 类型进行设置。

**经验**

PreparedStatement 对象对 SQL 语句进行了预编译，所以其执行速度要快于 Statement 对象。因此，多次执行的 SQL 语句应使用 PreparedStatement 对象处理，以提高效率。

### 2. 使用 PreparedStatement 实现数据的编辑

对于数据库数据的操作，归纳起来就是数据的增、删、改、查 4 种操作类型。在本章任务 1 中，已经实现了新闻信息的查询，所以在这里只以增加新闻的功能为例进行介绍，而新闻的删除、修改在实现逻辑上的思路是一样的，仅仅是在 SQL 语句的编写上存在一些区别，就不再做过多的描述了。

实现新闻信息增加的功能，需要执行以下几个步骤。

1）编写增加新闻的 SQL 语句。

2）创建 PreparedStatement 对象的实例，并为占位符赋值。

3）执行 SQL 语句。由于 PreparedStatement 对象对 SQL 语句实现了预编译，所以在执行时，直接调用 executeUpdate() 方法即可。

4）根据执行结果进行处理。对数据库记录的增、删、改操作，都会有一个 int 类型的返回结果，表示操作所影响的记录数，如果这个记录数的数值大于 0，表示 SQL 语句成功影响若干行记录，否则表示 SQL 语句未影响任何记录。

**提示**

数据库的增、删、改操作，除了 SQL 语句不同，其他的操作步骤完全相同，在学习时只需要掌握增加操作的实现过程，然后举一反三，即可掌握另两种数据库操作方式。

## 2.2.2 优化数据库操作的编码实现

1. BaseDao 类

（1）BaseDao 类的作用

到目前为止，我们已经学习了如何使用 JDBC 查询数据库以及如何使用 JDBC 实现对新闻信息的编辑。仔细观察，不难发现，在进行数据库操作时，很多代码是重复编写的，如获取数据库连接、释放资源。而不同的则是 SQL 语句、参数数量及对 SQL 执行结果的处理。因此，数据库操作代码是可以进行优化的，将需要重复编写的代码进行提取，单独存放到一个类中，在实际应用开发中，通常将这个类定义为 BaseDao 类。

（2）编写 BaseDao 类

编写 BaseDao 类，需要实现以下几个功能。

- 获取数据库的连接。
- 执行数据库的增、删、改、查操作。
- 执行每次访问结束后的资源释放工作。

按照 BaseDao 类的作用，编写方法逐一实现获取数据库连接，数据的增、删、改、查以及释放资源。

**示例 8**

编写一个 BaseDao 类，能够获取数据库连接，具备数据库增、删、改、查的通用方法，以及释放资源的通用方法。

关键代码：

```
// 基类 ： 数据库操作通用类
public class BaseDao {
    protected Connection conn;
    protected PreparedStatement ps;
    protected Statement stmt;
    protected ResultSet rs;
    // 获取数据库连接
    public boolean getConnection() {
        return true;
    }
    // 增、 删、 改
    public int executeUpdate(String sql, Object[] params) {
```

```
            int updateRows = 0;
            return updateRows;
        }
        // 查询
        public ResultSet executeSQL(String sql,Object[] params) {
            return rs;
        }
        // 关闭资源
        public boolean closeResource() {
            return true;
        }
    }
```


代码优化

在示例 8 中列举了在 BaseDao 类中需要完成的方法，具体的实现方法这里不再详细描述。了解具体实现请扫描二维码。

**注意**

在本节中，将实现数据库信息增、删、改、查的通用方法编写在 BaseDao 类中，这种方式不是必须的。在实际开发中 BaseDao 类可以只包含数据库访问相关的方法，而对于数据库记录操作的方法可以单独编写在一个专门的类中。

### 2. 使用接口优化新闻编辑

在之前完成新闻信息查询和新闻信息编辑时，将实现的方法均写在类中，现在只需要将类转换成接口，然后通过实现这个接口中的方法就能够实现对新闻信息的读取和编辑操作。

**示例 9**

编写接口，实现对新闻信息的增、删、改、查。

关键代码：

```
public interface NewsDao {
    // 查询新闻信息
    public void getNewsList();
    // 增加新闻信息
    public void add(int id, int categoryId, String title, String summary,
            String content, Date createdate);
    // 删除新闻信息
    public void delete(int id);
    // 修改新闻标题信息
    public void update(int id, String title);
}
```

接口编写完毕后，就需要编写接口的实现类来实现接口中定义的方法。

**示例 10**

编写新闻信息接口的具体实现类，并实现相应的增、删、改、查方法。

关键代码：

```
public class NewsDaoImpl extends BaseDao implements NewsDao {
  // 查询新闻信息
  public void getNewsList(){
      ……
  }
  // 增加新闻信息
  public void add(int id, int categoryId, String title, String summary,
        String content, Date createdate) {
      ……
  }
  // 删除新闻信息
  public void delete(int id) {
      ……
  }
  // 修改新闻标题信息
  public void update(int id, String title) {
      ……
  }
}
```

在示例 10 中编写了一个 NewsDaoImpl 接口实现类，继承了 BaseDao 的同时又实现了示例 9 中的 NewsDao 接口，重写了其方法，具体实现方法这里不再赘述。

**提示**

在编写接口实现类时，在实现接口的同时，一般还继承 BaseDao 类，就是因为在实现类中可以直接调用 BaseDao 类中定义好的方法，而不用再去导入相应的类。

## 2.2.3 优化 JDBC 连接管理

### 1. 数据源与连接池技术

数据源是在 JDBC 2.0 中引入的一个概念。在 JDBC 扩展包中定义了 javax.sql.DataSource 接口，它负责建立与数据库的连接，在应用程序访问数据库时不必编写连接数据库的代码，可以直接从数据源获得数据库连接。

DataSource 的全称为“javax.sql.DataSource”，它有一组特性可以用于确定和描述它所表示的现实存在的数据源，我们配置好的数据库连接池也是以数据源的形式存在的。

在 DataSource 中事先建立了多个数据库连接，这些数据库连接保存在连接池（Connection Pool）中。Java 程序访问数据库时，只需从连接池中取出处于空闲状态的数据库连接，当程序结束数据库访问时，再将数据库连接返回给连接池，这样做可以提高访问数据库的效率。

简单来说，数据源（DataSource）的作用是获取数据库连接，而连接池则是对已经创建好的连接对象进行管理，二者的作用不同。连接池的工作原理如图 2.3 所示。

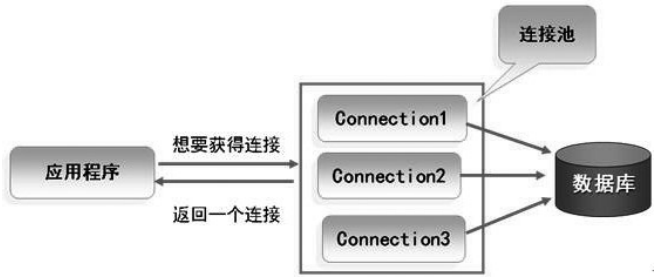



图 2.3　数据源与连接池工作原理

2．*数据源的配置*

数据源的配置有固定模式，例如配置 Tomcat 服务器的配置文件，只需在 Tomcat 服务器的 conf/context.xml 文件中添加如下配置信息：

```
<Resource name="jdbc/news"
        auth="Container"  type="javax.sql.DataSource"  maxActive="100"
        maxIdle="30" maxWait="10000" username="root"  password="root"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://localhost:3306/news"/>
```

Resource 元素中各个属性的含义如表 2-1 所示。

**表2-1　Resource元素属性说明**

| 属性 | 说　明 |
|---|---|
| name | 指定 Resource 的 JNDI 名称 |
| auth | 指定管理 Resource 的 Manager（Container 由容器创建和管理，Application 由 Web 应用创建和管理） |
| type | 指定 Resource 所属的 Java 类 |
| maxActive | 指定连接池中处于活动状态的数据库连接的最大数量 |
| maxIdle | 指定连接池中处于空闲状态的数据库连接的最大数量 |
| maxWait | 指定连接池中连接处于空闲的最长时间，超过这个时间会提示异常，取值为 -1，表示可以无限期等待，单位为毫秒（ms） |

至此，数据源的配置已经完成，下面介绍如何从程序中来访问数据源。

3．*使用 JNDI 读取数据源*

JNDI（Java Naming and Directory Interface，Java 命名与目录接口）是一个为应用

程序设计的 API，为开发人员提供了查找和访问各种命名和目录服务的通用、统一的接口。

我们可以把 JNDI 简单地理解为一种将对象和名字绑定的技术，即指定一个资源名称，将该名称与某一资源或服务相关联。由于数据源是由 Tomcat 容器创建的，因此需要使用 JNDI 来获取数据源。

获取数据源时，javax.naming.Context 提供了查找 JNDI Resource 的接口，通过该对象的 lookup() 方法，就可以找到之前创建好的数据源。lookup() 方法的语法如下。

```
lookup("java:comp/env/ 数据源名称 ")
```

"java:comp/env/" 这个前缀是 Java 的语法要求，必须要写上，其后才是在 context.xml 文件中 <Resource> 元素的 name 属性的值，也就是数据源的名称。

**示例 11**

配置数据源，编写程序获取数据源。

关键代码：

```
// 获取数据库连接
public Connection getConnection2() {
        try {
            // 初始化上下文
            Context cxt=new InitialContext();
            // 获取与逻辑名相关联的数据源对象
            DataSource ds=(DataSource)cxt.lookup("java:comp/env/jdbc/news");
            conn=ds.getConnection();
        } catch (NamingException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return conn;
 }
```

在示例 11 中，Context 对象的实例调用 lookup() 方法来获取数据源，数据源是 DataSource 类型，所以需要进行类型转换。

**注意**

读取数据源获取数据库连接时，首先要确保 Tomcat 服务器已经启动，其次确保读取数据源的代码运行在 Tomcat 中。

#### 4. 调用数据源得到连接

在应用程序中调用数据源获取连接的代码很简单，只需要实例化获取数据源方法的所在类，然后调用获取数据源的方法就可以得到一个 Connection 对象。

**示例 12**

在 JSP 中编写代码，实现数据源调用，获得访问连接。

关键代码：

```
……
<%
   BaseDao baseDao=new BaseDao();
   Connection conn=baseDao.getConnection2();
%>
<%=conn %>
……
```

运行效果如图 2.4 所示。


图 2.4　使用 JNDI 访问数据源

## 任务 3　在 JSP 页面中展示新闻列表

关键步骤如下。

- 使用 JavaBean 封装数据。
- 使用 JavaBean 封装业务。
- 使用 JSP 显示数据列表。
- 使用 JSP 标签实现 JavaBean 属性的读取设置。

### 2.3.1　JavaBean 与组件开发

1. JavaBean 概述

JavaBean 是用 Java 开发的可以跨平台的可重用组件，在 Web 程序中常用来封装业务逻辑和进行数据库操作。在程序开发中，程序员们所要处理的无非是业务逻辑和数据，而这两种操作都要用到 JavaBean，因此 JavaBean 很重要。

JavaBean 实际上就是一个 Java 类，这个类可以重用。JavaBean 从功能上可以分为以下两类。

- 封装数据。
- 封装业务。

JavaBean 一般情况下应满足以下要求。

- 是一个公有类，并提供无参的公有的构造方法。
- 属性私有。
- 具有公有的 getter 和 setter 方法。

符合上述条件的类，我们都可以把它看成 JavaBean 组件。

2. JavaBean 的应用

（1）用 JavaBean 封装数据

使用 JavaBean 封装数据，实际上就是将数据库中某一张表的字段进行封装，因此用 JavaBean 封装数据时，每一个属性都要与数据表中的字段一一对应。为了方便对 JavaBean 中的属性进行操作，分别设置了 setXxx() 方法和 getXxx() 方法来实现对属性的赋值与读取。

**示例 13**

使用 JavaBean 封装新闻信息。

```
// 新闻信息的 JavaBean
public class News {
    // 新闻属性
    private int id;
    private int categoryId;
    ……
    //setter 以及 getter 方法
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public int getCategoryId() {
        return categoryId;
    }
    public void setCategoryId(int categoryId) {
        this.categoryId = categoryId;
    }
    ……
}
```

（2）用 JavaBean 封装业务

相对于一个封装数据的 JavaBean，一般都会有一个封装该类的业务逻辑和操作的 JavaBean 与之对应。实际上在之前的代码中已经实现了使用 JavaBean 封装业务逻辑，如 BaseDao 类、NewsDao 接口及 NewsDaoImpl 接口实现类。

**示例 14**

使用 JavaBean 封装业务操作。

```
public interface NewsDao {
    // 查询新闻信息
    public List<News> getNewsList();
    // 增加新闻信息
    public boolean add(News news) ;
    // 删除新闻信息
    public boolean delete(int id) ;
    // 修改新闻
    public boolean update(News news) ;
}

public class NewsDaoImpl extends BaseDao implements NewsDao {
    // 查询新闻信息
    public List<News> getNewsList(){
        List<News> newList=new ArrayList<News>();
        try {
            // 执行 SQL 语句
            String sql="select * from news_detail";
            Object[] params={};
            ResultSet rs=this.executeSQL(sql, params);
            // 处理执行结果
            while(rs.next()){
                int id=rs.getInt("id");
                // 读取结果集数据
                // 封装成新闻信息对象
                News news=new News();
                news.setId(id);
                news.setTitle(title);
                news.setSummary(summary);
                news.setContent(content);
                news.setAuthor(author);
                news.setCreateDate(time);
                // 将新闻对象放进集合中
                newList.add(news);
            }
        }
      // 异常处理和释放资源
      return newList;
  }
```

在实际开发中通常还会创建一个 Service 层，用于存放与业务逻辑相关的操作。Service 层中的接口和类对 Dao 类的方法实现了封装和调用。

**示例 15**

编写 NewsService 接口及实现类。

```
public interface NewsService {
    // 更新选择的新闻
    public boolean updateNews(News news);
    // 添加新闻
    public boolean addNews(News news);
    // 删除新闻
    public boolean deleteNews(int id);
    // 查询新闻信息
    public List<News> getNewsList();
}

public class NewsServiceImpl implements NewsService {
    private NewsDao newsDao;
    public NewsDao getNewsDao() {
        return newsDao;
    }
    public void setNewsDao(NewsDao newsDao) {
        this.newsDao = newsDao;
    }
    public boolean updateNews(News news) {
        return newsDao.update(news);
    }
    public boolean addNews(News news) {
        return newsDao.add(news);
    }
    public boolean deleteNews(int id) {
         return newsDao.delete(id);
    }
    public List<News> getNewsList() {
        return newsDao.getNewsList();
    }
}
```

在示例 15 中，NewsServiceImpl 类实现了 NewsService 接口，在实现方法中，不难发现对于新闻的增、删、改、查操作仅仅是调用 NewsDao 接口中的方法，而具体的增、删、改、查是如何实现的，在 Service 中并不重要。这也符合程序代码间低耦合的设计要求。

编写 Service 最大的作用就是将业务逻辑和数据操作分离，就是说不管数据增、删、改、查做了怎样的改动，在 Service 中控制程序执行时都不会受到影响，这也是 Service 存在的意义。

3. 使用 JSP 脚本显示新闻列表

到目前为止，我们已经完成了对于新闻信息的增、删、改、查的代码编写，下面需

要做的就是将新闻信息数据显示在 JSP 中。实现方式很简单，就是在 JSP 中使用脚本方式调用已经写好的后台代码。

**示例 16**

使用 JSP 脚本输出显示新闻列表。

分析如下。

实现新闻列表显示，首先要清楚列表显示的实质就是使用表格显示数据，表格的行数对应数据库中新闻信息的记录数，而表格的列则与一条记录中的字段相对应，显示该字段的内容。其次要理解新闻信息的数据是从数据库中查询得到的，表格的行数应该动态循环添加，与查询结果的总数相同。

关键代码：

```
<tbody>
<%
  NewsServiceImpl newsService=new NewsServiceImpl();
  NewsDao newsDao=new NewsDaoImpl();
  newsService.setNewsDao(newsDao);
  List<News> newsList=newsService.getNewsList();
  // 新闻行数
  int i=0;
  for(News news:newsList){
      i++;
%>
      // 判断行数是否为偶数，实现隔行变色显示
      <tr <% if(i%2==0){%>class="admin-list-td-h2"<%} %>>
          <td><a href='adminNewsView.jsp?id=3'><%=news.getTitle() %></a>
</td>
          <td><%=news.getAuthor() %></td>
          <td><%=news.getCreateDate() %></td>
          <td><a href='adminNewsCreate.jsp?id=3'> 修改 </a>
              <a href="javascript:if(confirm(' 确认是否删除此新闻？ '))
                  location='adminNewsDel.jsp?id=3'"> 删除 </a>
          </td>
      </tr>
<%  } %>
</tbody>
```

至此，已经基本实现了任务 3 要求的对新闻信息的显示。但是仔细观察代码不难发现，在页面中使用 JSP 脚本与 HTML 标签的混合方式，使得代码构成很乱，不易读，更不易维护。

其实在 JSP 中还提供了一种方式，就是使用 JSP 标签来优化页面显示，下面就来学习使用 JSP 标签。

### 2.3.2 使用 JSP 动作标签操作 JavaBean

JSP 动作标签是在 JSP 中已经定义好的动作指令，这些指令实现了最常用的基本功能。通过动作标签，开发人员可以在 JSP 中把页面的显示功能部分封装起来，使整个代码更简洁、更易于维护。

1. 创建 JavaBean 标签 <jsp:useBean>

<jsp:useBean> 标签的作用就是在 JSP 中创建一个 JavaBean 的实例，并指定它的名称和作用范围。<jsp:useBean> 标签的语法如下：

```
<jsp:useBean id="name" class="package.class" scope="scope"/>
```

- id：表示创建的 JavaBean 的名称，这个名称可以不与 Java 类名相同。
- class：表示创建的 JavaBean 名称所引用或者指向的 JavaBean 类的完整限定名。
- scope：表示这个 JavaBean 的作用范围以及 id 名称的有效范围，总共有 4 个范围，分别是 page（默认值）、request、session 和 application。

在 JSP 中编写代码：

```
<jsp:useBean id="newsService" class="com.kgc.news.service.impl.NewsServiceImpl"scope="page"/>
<jsp:useBean id="newsDao" class="com.kgc.news.dao.impl.NewsDaoImpl" scope= "page"/>
```

等同于如下代码。

```
NewsServiceImpl newsService=new NewsServiceImpl();
NewsDao newsDao=new NewsDaoImpl();
pageContext.setAttribute("newsService",newsService);
pageContext.setAttribute("newsDao", newsDao);
```

2. 设置 JavaBean 属性 <jsp:setProperty>

在 JSP 中使用 <jsp:useBean> 标签创建 JavaBean 后，就可以对 JavaBean 中的属性进行设置。设置 JavaBean 属性的标签就是 <jsp:setProperty>。<jsp:setProperty> 标签的语法如下。

```
<jsp:setProperty name="BeanName" property="name" value="value"/>
```

- name：表示被赋值的对象（JavaBean）名称。
- property：表示被赋值对象中，需要进行赋值操作的属性名称。
- value：表示需要给被赋值属性所赋的值。

在 JSP 中编写代码。

```
<jsp:useBean id="newsService" class="com.kgc.news.service.impl.NewsServiceImpl" scope="page"/>
<jsp:useBean id="newsDao" class="com.kgc.news.dao.impl.NewsDaoImpl" scope="page"/>
<jsp:setProperty property="newsDao" name="newsService" value="<%= newsDao %>"/>
```

等同于如下代码。

```
<%
NewsServiceImpl newsService=new NewsServiceImpl();
```

```
NewsDao newsDao=new NewsDaoImpl();
newsService.setNewsDao(newsDao);
%>
```

至此，我们已经使用了 JSP 标签来代替 JSP 脚本实现新闻信息的列表显示，运行页面的效果与图 2.1 所示的效果是相同的。

**扩充知识**

JSP 标签既然可以实现对 JavaBean 属性的设置，当然也可以实现对 JavaBean 属性的获取，可以将 <jsp:getProperty> 标签与 <jsp:setProperty> 标签结合使用。这部分内容属于自学内容。

### 3. 获取 JavaBean 的属性 <jsp:getProperty>

<jsp:getProperty> 的作用很简单，就是获取 JavaBean 的属性值，用于在页面中显示。<jsp:getProperty> 标签的语法如下：

```
<jsp:getProperty name="BeanName" property="PropertyName"/>
```

- name：useBean 中使用的 JavaBean 的 id。
- property：指定要获取 JavaBean 的属性名称。

**示例 17**

编写一个 JSP，使用 JSP 标签对 News 类的 title 属性进行赋值，并读取显示。

关键代码：

```
<jsp:useBean id="news" class="com.kgc.news.entity.News" scope="page"/>
<jsp:setProperty name=" news" property="title" value=" 中国首艘航母交付使用 "/>
<jsp:getProperty name= "news" property="title"/>
```

运行效果如图 2.5 所示。


图 2.5　获取 JavaBean 的属性

本任务运行效果如图 2.6 所示。


图 2.6　显示新闻信息列表

## 任务 4　通过 JSP 页面添加新闻信息

关键步骤如下。

- 在 JSP 中实现页面包含。
- 获取 JSP 提交的信息。
- 访问数据库并实现信息的增加。
- 根据数据库执行结果实现页面跳转。

### 2.4.1　JSP 页面的包含操作

在 JSP 中实现页面包含的方式有两种，一种是使用 <%@include%> 指令，另一种是使用 <jsp:include/> 标签。虽然都是实现页面包含，但是二者在使用时存在一些区别。

1. 使用 include 指令实现静态包含

使用 <%@include%> 指令属于静态包含，静态包含是指将被包含的文件插入 JSP 中，简单地说就是将另一个文件中的代码复制到一个 JSP 中。被包含的文件代码将会在 JSP 中被执行。<%@include%> 指令的语法如下。

```
<%@include file="URL" %>
```

file：表示需要包含的页面路径。

例如：

```
<%@include file="common/common.jsp" %>
```

将 common 目录下的 common.jsp 文件包含到当前页面中。

2. 使用 JSP 标签实现动态包含

<jsp:include/> 标签实现的是动态包含页面，允许包含一个静态或者动态的文件。<jsp:include/> 在实现页面包含时，采用的是先执行被包含页面的代码，然后将结果包含

到当前页面中的包含方式。<jsp:include/> 动态包含的特点如下。

- 当包含文件为静态文件时，效果等同于 <%@include%> 指令。
- 当包含文件为动态文件时，被包含文件也会被 JSP 编译器执行。

<jsp:include/> 标签的语法如下。

```
<jsp:include page="URL" />
```

page：表示需要包含的页面路径。

**示例 18**

使用 <jsp:include/> 标签制作 admin.jsp 页面。运行效果如图 2.7 所示。

分析如下。

从图 2.7 中可以看出，admin.jsp 页面分为四个部分，只需要在相应的位置将对应的 JSP 文件包含进来就可以了。

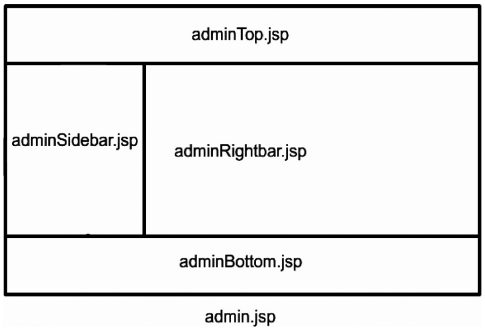



图 2.7　使用 <jsp:include> 标签实现页面包含

关键代码：

```
<!-- 页面顶部 -->
<jsp:include page="adminTop.jsp" />
<!-- 页面中部 -->
<div id="content" class="main-content clearfix">
   <jsp:include page="adminSidebar.jsp"/>
   <jsp:include page="adminRightbar.jsp"/>
</div>
<!-- 页面底部 -->
<jsp:include page="adminBottom.jsp"/>
```

3. ***动态包含与静态包含的区别***

动态包含 <jsp:include> 与静态包含 <%@include%> 都可以实现在当前 JSP 页面中插入另一个文件，当然这个文件不仅限于 JSP 文件，还可以是 HTML 文件或者文本文件。动态包含与静态包含的比较说明如表 2-2 所示。

**表2-2　静态包含与动态包含的比较**

| 静态包含 | 动态包含 |
|---|---|
| <%@include file="url"%> | <jsp:include page="url" /> |
| 先将页面包含，后执行页面代码，即将一个页面的代码复制到另一个页面中 | 先执行页面代码，后将页面包含，即将一个页面的运行结果包含到另一个页面中 |
| 被包含的页面内容发生变化时，包含页面将会被重新编译 | 被包含页面内容发生变化时，包含页面不会重新编译 |

## 2.4.2　JSP 转发实现页面跳转

<jsp:forward/> 标签的实质与 request.getRequestDispatcher(URL).forward(request, response) 语句相同，用于实现页面的跳转。

<jsp:forward/> 标签的语法如下：

```
<jsp:forward page="URL" />
```

page：需要跳转的页面路径。

本任务的运行效果如图 2.8 所示。

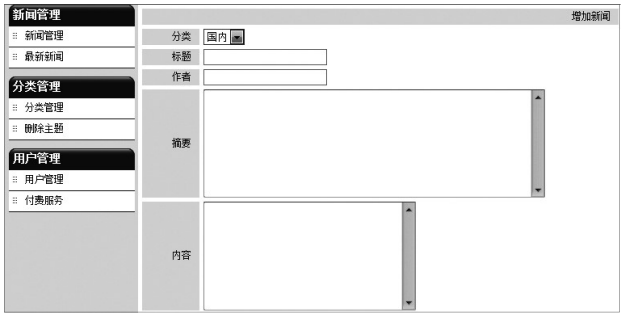



图 2.8　新闻信息添加

## 本章总结

本章学习了以下知识点。

- JDBC 的概念。
- 使用 JDBC 访问数据库。
- 数据源与连接池。
- JavaBean 的概念。
- JavaBean 的应用。
- JSP 标签
  - <jsp:useBean/>
  - <jsp:setProperty/>
  - <jsp:getProperty/>
  - <jsp:include/>
  - <jsp:forward/>
- 静态包含与动态包含。

## 本章练习

1．使用数据库，创建用户登录信息表，表中包含用户名和密码两个字段，输入若干条测试数据，然后编写代码实现数据库访问，在控制台输出所有记录信息。

2．在作业 1 的基础上进行修改，使用配置文件实现数据库访问。

3．在作业 2 的基础上进行修改，配置数据源和连接池，使用 JNDI 实现数据库访问。

4．在作业 3 的基础上，使用 JavaBean 封装数据，编写 Service 来控制逻辑，实现在 JSP 中显示查询数据。