第 13 章　数据库连接池技术

在第十二章的实例中，使用的是传统数据库连接方法，即每一次数据库操作都需要进行一次数据库连接请求，使用完之后再进行资源的释放，以防止资源大量占有而造成排队等待。有关传统数据库连接方式，经过上一章的学习，读者应该有所了解。概括起来，传统数据库连接操作过程如下：

（1）针对数据库类型装载数据库驱动器，可以选择隐式或者显式方式。显式是使用 Class.forName()方法装载。

（2）在 JSP 或者 JavaBean 程序中（下一章将重点讲解 JavaBean 的使用）建立数据库连接，然后使用 DriverManager.getConnection()方法获得一个 Connection 实例对象。

（3）使用 Connection 对象中的 createStatement()方法生成 Statement 或者 preparedSatement()方法生成 preparedStatement 对象，并进行相应 SQL 操作，例如查询、删除和添加等。

（4）获取到返回结果 ResultSet，进行数据操作。

（5）最后断开数据库连接，其中包括 Connection、Statement（或者 preparedStatement）和 ResultSet。

传统的数据库连接模式存在很多的不足，由于每次操作都需要重新建立数据库连接，这就限制数据库操作的性能（特别数据库访问量特别大的站点）。本章将介绍一种新的数据库连接技术，即数据库连接池技术，它提高了数据访问速度和性能。

本章要点包括以下内容：

- 数据库连接池技术的介绍
- 传统数据库连接与连接池技术的比较
- 连接池技术原理
- 连接池的配置

13.1　连接池介绍

顾名思义，连接池技术就是预先建立一些数据库连接，并放置在内存“池”对象中以备用户进行数据库操作时直接使用。连接池技术使得程序不再需要一次操作都必须进行一次数据库连接操作，只须从内存“池”中取出一个连接即可。当程序操作完数据库之后，只需把连接重新放回内存“池”。数据库连接的建立、断开等都由连接池统一来管理。

连接池技术大大提高了 Web 应用系统的性能，系统管理人员还可以根据系统访问量来动态设置连接池的连接数（内存“池”预先创建的连接数量）、每个连接的最大使用次数等等参数。

13.2　两种数据库连接技术的比较

下面将传统数据库连接和数据库连接池技术作进一步的比较，在比较中了解它们之间的区别、各自原理和优缺点。

13.2.1　传统数据库连接模式

总结起来，传统模式的特点和不足如下：

（1）每一次 Web 请求（例如查询数据库表中的数据）都要建立一次数据库连接，系统开销相当大。

对于小型系统或者访问量比较小的应用系统来讲，或许觉察不到系统的这种开销。但是，对于访问量比较大的 Web 应用来讲，即使在某一较短的时间段内，其操作请求数也远远不是一两次，而是数十上百次。事实上，在一个基于数据库的 Web 应用中，建立数据库连接将是系统中代价最大的操作之一。所以这样的数据库连接模式往往是网站速度的瓶颈。

（2）传统的数据库连接必须确保使用完后被正确关闭。

在传统的数据库连接模式下，如果出现程序异常而使得某些连接未能正确地关闭，将导致数据库系统中的内存泄露，最终将不得不重启数据库。这种错误对安全性和稳定性要求非常高的系统来说是致命的。

正是由于传统数据库连接模式存在以上的不足，这就制约了它在大型系统中的使用，迫切需要有一种新的技术来弥补这样的不足。

13.2.2　连接池技术

和传统连接方法相对比，连接池技术克服了其很多不足之处，重点也包括如下两点：

（1）提升了数据库操作性能。

由于每次进行数据库操作再也不需要重新建立连接，大大节省了很多重复的系统资源开支。只需要连接池管理器分配一个数据库连接即可，使用完毕之后再将连接放回到池中。

（2）降低了由于大量资源占有而造成排队等待现象的风险。

传统连接模式，在操作完数据库之后，一定要将建立的连接资源释放，不然容易造成大量资源占有，而是系统性能降低。而数据库连接池技术，所有的连接和释放工作是统一由连接池管理器进行协调分配和管理。非特殊情况，在程序当中不需要涉及资源的释放。

在实际系统，特别是大型系统中都应该使用数据库连接池技术。这项技术也已经非常成熟，有关连接池的配置方式有很多，在本书的后面章节中会逐一介绍比较常用的方式。

13.3　连接池原理

有关连接池原理在上面文字中已有所介绍，下面通过一张图更加清晰地进行描述，如图 13.1 所示。

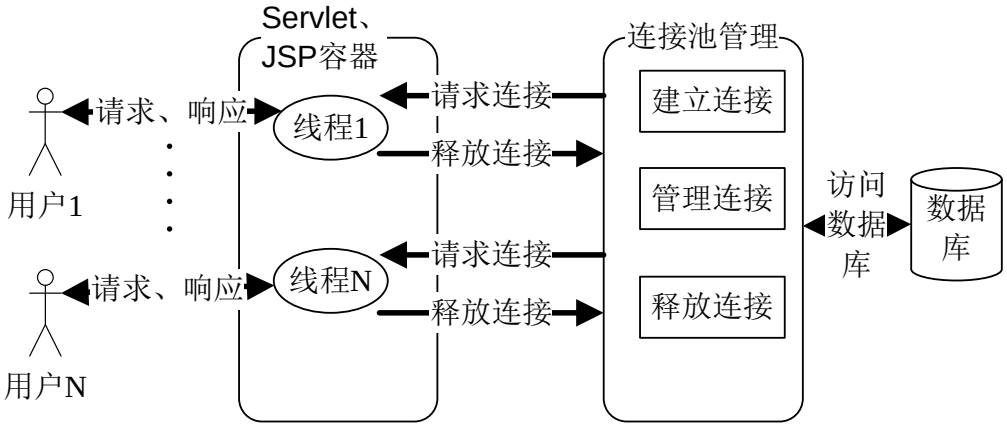

图 13.1　连接池示意图

从图中可以看出，JSP 或者 Servlet、JavaBean 程序需要操作数据库时，可以直接向连接池管理器发出一个请求，然后再由管理器统一分配一个连接操作，最后再由连接池管理器收回资源。

13.4　连接池配置

接下来介绍如何进行连接池的配置以及从连接池中获取到一个连接资源。至于连接的释放，程序是不需要实现的，是由连接池管理器统一回收。有关连接池的配置方法有很多种，下面介绍如何在 Tomcat 服务器上配置一个连接池资源。

13.4.1　Tomcat连接池配置

在使用连接池进行数据库访问之前还需要解决一个问题。

JDBC 连接包的配置：经过实践可知，之前 MySQL 的 JDBC 连接包 mysql-connector-java-3.1.12-bin.jar 放置在 Register\WEB-INF\lib 目录（Register 是上一章创建的 Web 模块名）下时，使用传统的数据库连接方法不会出现错误。但是当使用连接池连接数据库的时候就会有问题，这时需要把 JDBC 连接包 mysql-connector-java-3.1.12-bin.jar 放置在 Tomcat 5.0\common\lib 目录下，这个目录是 Register 模块发布时会自动加载的一个包目录。

配置 Tomcat 连接池有如下两种方式：

（1）针对上一章创建的 Register 模块，下面手动编辑 Tomcat 5.0\conf\Catalina\localhost\register.xml 文件（适合 Tomcat5.x.x 版本，低版本需要在 server.xml 进行修改，如果 register.xml 文件不存在，需要创建），修改的文件内容如下：

```
<?xml version='1.0' encoding='utf-8'?>
<Context docBase="D:/eclipse/EclipseSource/MyProject/eshopping" path="/eshopping" reloadable="true" workDir="D:\eclipse\EclipseSource\MyProject\bin">
    <Logger className="org.apache.catalina.logger.FileLogger" prefix="localhost_DBTest_log." suffix=".txt" timestamp="true"/>
    <Resource name="jdbc/mysql" type="javax.sql.DataSource"/>
    <ResourceParams name="jdbc/mysql">
        <parameter>
            <name>factory</name>
```

```
            <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </parameter>
            <!—配置 MySQL 数据库驱动-->
        <parameter>
            <name>driverClassName</name>
            <value>com.mysql.jdbc.Driver</value>
        </parameter>
            <!—配置数据库连接字串，mydb1 为需要连接的数据库-->
        <parameter>
            <name>url</name>
            <value>jdbc:mysql://localhost/mydb1</value>
        </parameter>
            <!—数据库登陆的用户名-->
        <parameter>
            <name>username</name>
            <value>root</value>
        </parameter>
            <!—用户名对应的密码-->
        <parameter>
            <name>password</name>
            <value>12345</value>
         </parameter>
             <!—最大连接数，开发者可根据系统的需求进行配置-->
          <parameter>
              <name>maxActive</name>
              <value>10</value>
          </parameter>
              <!—最大空闲连接数 -->
          <parameter>
               <name>maxIdle</name>
               <value>6</value>
          </parameter>
                <!—最大等待连接限制-->
          <parameter>
                <name>maxWait</name>
                <value>10000</value>
          </parameter>
        </ResourceParams>
</Context>
```

然后在 Tomcat 5.0\conf\server.xml 文件的<GlobalNamingResources> </GlobalNamingResources>之间添加如下一段代码：

```
<ResourceLink name="jdbc/mysql" type="javax.sql.DataSource" global="jdbc/mysql"/>
```

这样就完成了 Tomcat 连接池的配置工作，接下来需要修改 DBConnect 类中的 Init()方法。

（2）在 IE 浏览器中输入地址：http://localhost:8080/admin（或者单击 Tomcat 服务器主页面的 Tomcat Administration 链接，如第 2 章图 2.16 所示），打开 Tomcat 服务器的配置页面。

单击左边的树结点“Data Source”选项，然后选择右上角的下拉菜单“Create New Data Source”选项，在表格中填写和第一种配置方法相对应的 JNDI 名称（连接池名称，Java 程序就是通过这个名字进行识别的）、数据库连接地址（Data Source URL）、JDBC 驱动、数据库登陆名、登陆密码以及最大连

接数、最大空闲连接数和最大等待连接限制。

然后在 eshopping.xml 文件的<Context>　</Context>之间（如果你是在 server.xml 文件中配置的模块目录，就在 server.xml 文件的相应位置进行修改）添加如下一段代码：

```
<ResourceLink name="jdbc/mysql" type="javax.sql.DataSource" global="jdbc/mysql"/>
```

可以看出第二种的配置方法比较直观和方便，但是推荐使用第一种方法，可以让读者更加了解 Tomcat 服务器的运行机制。

在使用连接池技术时，还使用到另外两个概念：DataSource 对象（全名为 javax.sql.DataSource），它有一组特性，可以用于确定和描述它表示的现实世界的数据源，之上配置的一个数据库连接池就是以数据源存在的；另外使用 Java 命名和目录接口（JNDI）命名服务来使组件定位到其他组件和资源（例如数据库这样的资源）。在使用连接池进行数据库连接操作时，会使用到 DataSource 和 JNDI 对象中方法。下面对这两个概念作详细介绍。

13.4.2　JNDI命名

特别是在分布式应用系统中，一些组件需要访问其他组件和资源（例如数据库），这时就需要使用 Java 命名和目录接口（JNDI）命名服务来组件定位到其他组件和资源。例如，要定位 JDBC 资源（在上一小节，已经通过 Tomcat 配置了一个数据库连接池资源，并取名了 java:comp/env/jdbc/mysql，前面 java:comp/env 是必须的，它的类型是 javax.sql.DataSource），这时候就可以编写代码使用 JNDI 的 lookup()方法来定位到这个资源。

JNDI 命名服务有一组将名称与对象联系在一起的绑定。JNDI 中的 lookup 方法传递一个 JNDI 参数（例如 java:comp/env/jdbc/mysql），返回相应的对象（返回的对象类型是 DataSource，如果连接数据库，则调用 DataSource 中的 getConnection()方法获取数据库连接）。

有关使用 JNDI 来获取数据库连接资源的例子将在下面小节讲解。

注意：**为了避免** JNDI **命名空间中的资源名称起冲突，并且避免可移植性问题**，J2EE **应用程序中的所有名称应该以字符串** java:comp/env **开始**

13.4.3　DataSource对象和连接池

在 JDBC API 中，是可以通过 DataSource 对象来访问数据库的。DataSource 有一组特性，用于确定和描述它表示的现实世界的数据源（使用 JNDI 定位的 jdbc/mysql 资源，返回的类型就是 DataSource）。这些属性可以包括像数据源服务器的位置、数据库名称、与数据库服务器通信的网络协议等信息。在 Application Server 中，数据源被称为 JDBC 资源。

应用程序利用连接访问数据源。可以把 DataSource 对象看作是到 DataSource 实例表示的具体数据源的连接工厂。在基本 DataSource 实现中，对 getConnection 方法的调用返回一个连接对象，即与该数据源的物理连接。

如果用 JNDI 命名服务注册 DataSource 对象，应用程序就可以使用 JNDI API 访问 DataSource 对象，然后使用这个 DataSource 对象连接它表示的数据源。

实现连接池的 DataSource 对象也生成到 DataSource 类表示的具体数据源的连接。GetConnection 方法返回的连接对象是一个 PooledConnection 对象的句柄，而不是一个物理连接。应用程序使用连接对象的方法与它使用连接的方法相同。连接池对应用程序的代码没用影响，但像所有连接一样，汇集入池的连接应该始终被明确关闭。当应用程序关闭汇集入池的连接时，将把连接返回到一个可重用连接池中。

下一次调用 getConnection 方法时，如果有一个汇集入池的连接可用，则 getConnection 方法返回这些汇集入池连接中的一个连接的句柄。因为连接池避免了每次需要连接时建立新的物理连接，因此可以极大地提高应用程序的运行效率。

DataSource 对象以及 JNDI 的使用来连接数据库将一起在下一小节介绍。

13.4.4　连接池数据库连接

使用连接池连接数据库，其中使用到 DataSource 对象来访问数据库，另外 JNDI 定位到已经定义耗的数据库连接资源，执行过程分如下三步骤：

（1）指定数据库连接的逻辑名。在 Tomcat 中配置数据库连接池时，已经指定了逻辑名，起名为 java:comp/env/jdbc/mysql（前面 java:comp/env 是必须加上的）。详情见 13.4.1 小节的配置文件。

（2）获取与逻辑名相关联的 DataSource 对象。该段代码如下：

```
InitialContext ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/mysql");
```

如果给定资源的逻辑名，lookup 方法返回该目录下与此 JNDI 名称绑定的 DataSource 对象。

（3）由 DataSource 对象获取 Connection 对象。

```
Connection con = ds.getConnection();
```

综合以上所有步骤，即可以使用数据库连接池来修改以前所有的数据库连接，详细的数据库连接代码如下：

```
Connection con;
private void init(){
    try{
        InitialContext ctx = new InitialContext();
         DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/mysql");
         conn = ds.getConnection();
    }
    catch(Exception e) {
         e.printStackTrace();
    }
}
```

该段代码使用连接池技术，返回一个 Connection 对象，至于其他数据库操作方法和传统数据库连接方式一样，这里就不具体介绍。

当今连接池技术已经广泛使用，提供连接池的产品也很多，Tomcat 连接池只是其中一个，例如用户也可以使用第三方提供的 PoolMan 连接池以及后面将要讲的 Hibernate 持久层技术。对于其他连接池配置方法后面将会有专门章节进行归纳总结。

注意：使用 Tomcat 连接池，也有其不足的地方，当使用 JavaBean 来编写数据库连接部分时，不能编写 Java 语句来进行测试，只能在 JSP 文件中编写测试代码。这样带来了 Java 程序员的一些麻烦。

13.5　本章小结

本章主要重点介绍了数据库连接池的技术，并将其与传统数据库连接进行了比较，在比较中发现它的优点。从实际应用出来，本章也重点教会了读者如何在 Tomcat 服务器中配置数据库连接池，并使用 JNDI 以及 DataSource 对象来返回一个数据库连接。