第5章 chapter 5

服务器端设计与开发

本章任务

- 设计和实现一个服务器程序以实现用户认证、令牌管理及从数据库提取数据并传送到客户端的功能；
- 设计和实现一个框体以监测在线人数、登录用户信息、令牌持续时间、令牌剩余时间及服务器占用内存的情况。

5.1 相关知识

5.1.1 线程类

1. 创建线程

线程是程序中的执行程序。Java 虚拟机允许应用程序并发地运行多个执行程序。每个线程都有一个优先级，高优先级线程的执行优先于低优先级线程。每个线程可以标记为一个守护程序，其实现方法有两种。

(1) 继承 Thread

当某个线程中运行的代码创建一个新 Thread 对象时，该新线程的初始优先级被设定为创建线程的优先级，并且仅当创建的线程是守护线程时，新线程才是守护程序。当 Java 虚拟机启动时，通常都会有单个非守护线程(它通常会调用某个指定类的 main 方法)，Java 虚拟机会继续执行线程，直到下列任一情况出现时为止：

- 调用了 Runtime 类的 exit()方法，并且安全管理器允许退出操作发生。
- 非守护线程的所有线程都已停止运行，无论是从对 run()方法的调用中返回，还是抛出一个传播到 run()方法之外的异常。

(2) 实现 Runnable

Runnable 接口应该由那些准备通过某一线程执行其实例的类来实现。类必须定义一个名为 run()的无参数方法。

2. 线程的运行控制

(1) 线程的启动 start()、join()与停止 stop()

线程的 start()和 join()方法用来启动一个线程,这个线程会一直运行下去,可以通过 stop()方法来停止一个线程。

(2) 线程的休眠 sleep()与挂起 yield()

线程可以使用 sleep()方法让当前线程暂停一定的事件之后继续执行,也可让同优先级和高优先级的线程有执行机会。可以使用两个休眠方法:

```
static void sleep(long millis);                 //休眠 millis 毫秒
ststic void sleep(long millis,int nanos);       //休眠 millis 毫秒,外加 nanos 纳秒
```

线程 yield()方法与 sleep()方法相似,但它不能由用户指定线程暂停多长时间,且只能使同优先级的线程有执行机会。

(3) 线程的同步 synchronized

许多线程在执行中必须考虑与其他线程之间共享数据或协调执行状态,这就是同步机制。在 Java 中每个对象都有一把锁与其对应。Java 不提供单独的 lock 和 unlock 操作,它由高层的结构隐式实现,以保证操作的对应。有两种方式使用 synchronized 修饰符:

```
public synchronized void funname(){
//保护函数
}
synchronized(this){
//保护代码块
}
```

(4) 线程的同步锁机制:wait()、notify()和 notiftyAll()

在 synchronized 代码被执行期间,线程可以调用对象的 wait()方法释放对象锁标志,进入等待状态;并且可以调用 notify()或者 notifyAll()方法来通知正在等待的其他线程。notify()通知的是等待队列中的第一个线程,notifyAll()是通知等待队列中的所有线程。

代码如下:

```
package chapter5;

import java.util.Date;

public class Test5_1_10 extends Thread{
    public int time;                            //休眠时间
    public String user;                         //调用用户
    //构造函数
    public Test5_1_10(int time,String user){
```

```
        this.time=time;
        this.user=user;
    }

    public void run(){
        while(true){
            try {
                System.out.println(user+"休息"+time+"ms -"+new Date());

                Thread.sleep(time);
            } catch(InterruptedException e) {
                //TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

public static void main(String[] arg){
        Test5_1_10 thread1=new Test5_1_10(1000,"jack");
        Test5_1_10 thread2=new Test5_1_10(1000,"Mike");
        thread1.setPriority(1);
        thread2.setPriority(Thread.MAX_PRIORITY);
        thread1.start();
        thread2.start();
    }
}
```

5.1.2 数据库连接 JDBC

JDBC 提供了一系列的类供 Java 应用程序来操作数据库。最重要的如下：

(1) java.sql.DriverManager：加载不同的 JDBC 驱动程序并且为创建新的数据库连接提供支持。

(2) java.sql.Connection：完成对某一指定数据库的连接。

(3) java.sql.Statement：在一个已经创建的连接中作为执行 SQL 语句的容器。它包含两个重要的子类：

- java.sql.PreparedStatement：执行预编译的 SQL 语句。
- java.sql.CallableStatement：执行数据库中已经创建好的存储过程。

(4) java.sql.Result：代表特定 SQL 语句执行后的数据库结果集。

5.1.3 客户端套接字 Socket

1. Socket

Socket 是网络上运行的两个程序间双向通信的一端。它既可以接收请求，也可以发

送请求。利用它可以较为方便地实现网络上数据的传递。

2. Socket 的工作过程

连接到远程机器→绑定到端口→监听从远程机器到来的绑定端口上的连接→监听到达的数据→发送数据→接收数据→关闭连接。

5.1.4 服务端套接字 ServerSocket

1. ServerSocket

ServerSocket 用来建立客户端的连接,发起请求;而被请求端需要建立监听,此时需要使用 ServerSocket 来建立服务器端的监听程序。每个服务器套接字运行在服务器上特定的端口,监听在这个端口的 TCP 连接。当远程客户端的 Socket 试图与服务器指定端口建立连接时,服务器被激活,判定客户程序的连接,并打开与主机之间固有的连接。一旦客户端与服务器建立了连接,两者之间就可以传送数据,而数据是通过这个固有的套接字传送的。

2. ServerSocket 的工作过程

(1) 用 ServerSocket()方法在指定端口创建一个新的 ServerSocket 对象。

(2) ServerSocket 对象调用 accept()方法在指定的端口监听到来的连接。accept()一直处于阻塞状态,直到有客户端试图建立连接,这时 accept()方法返回连接客户端与服务器的 Socket 对象。

(3) 调用 getInputStream()方法或者 getOutputStream()方法,或者两者全部调用,建立与服务端交互的输入流和输出流。

(4) 服务器与客户端根据一定的协议交互,直到关闭连接。

(5) 服务器、客户机或者两者都关闭连接。

(6) 服务器回到第(2)步,继续监听下一次连接。

5.1.5 序列化与反序列化

1. 序列化

序列化是一种用来处理对象流的机制。对象流就是将对象的内容进行流化。可以对流化后的对象进行读写操作,也可将流化后的对象传输于网络之间。序列化可解决在对对象流进行读写操作时所引发的问题。

2. 序列化的实现

首先对需要被序列化的类实现 Serializable 接口,然后使用输出流(如 FileOutputStream)来构造 ObjectOutputStream(对象流)对象,再使用 ObjectOutputStream 对象的 writeObject

(Object obj)方法就可以将参数为 obj 的对象写出(即保存其状态),要恢复的话则用输入流实现。

3. 反序列化的实现

与序列化相反,反序列化对需要被反序列化的类实现 Serializable 接口,然后使用输入流(如 FileInputStream)来构造 ObjectInputStream(对象流)对象。最后,使用 ObjectInputStream 对象的 readObject(Object obj)方法就可以对参数为 obj 的对象实现读取(即保存其状态)。

5.1.6 远程过程调用

1. RMI 系统原理

RMI 应用程序通常包括两个独立的程序:服务器程序和客户机程序。典型的服务器应用程序将创建多个远程对象,使这些远程对象能够被引用,然后等待客户机调用这些远程对象的方法。在 RMI 分布式应用系统中,服务器与客户机之间传递的 Java 对象必须是可以序列化的对象,不可序列化的对象不能在对象流中传递。

2. RMI 程序的实现步骤

(1) 远程接口必须声明为 public。

(2) 远程对象扩展 java.rmi.Remote 接口。

(3) 除了所有应用程序特定的例外之外,每个方法还必须抛出 java.rmi.RemoteException 例外。

(4) 任何作为参数或者返回值传送的远程对象的数据类型必须声明为远程接口类型,而不是实现类。

5.2 界面设计

服务器监控界面设计如图 5.1 所示。服务器的监控界面显示当前在线人数、服务器占用内存以及在线人员列表,其中人员列表包含用户名、令牌、令牌持续时间、令牌剩余时间。

图 5.1 服务器监控界面设计

5.3 认证流程

服务器和客户端认证流程如图 5.2 所示。用户在客户端输入用户名和密码信息后，客户端将用户名和密码发送到服务器端。服务器将客户端发送过来的用户名和密码与存储在数据库中的用户信息进行比对：如果比对成功，则生成一个令牌，并发送到客户端，用户可以使用该令牌进行后续操作，登录成功；如果比对失败，则向客户端返回错误信息。

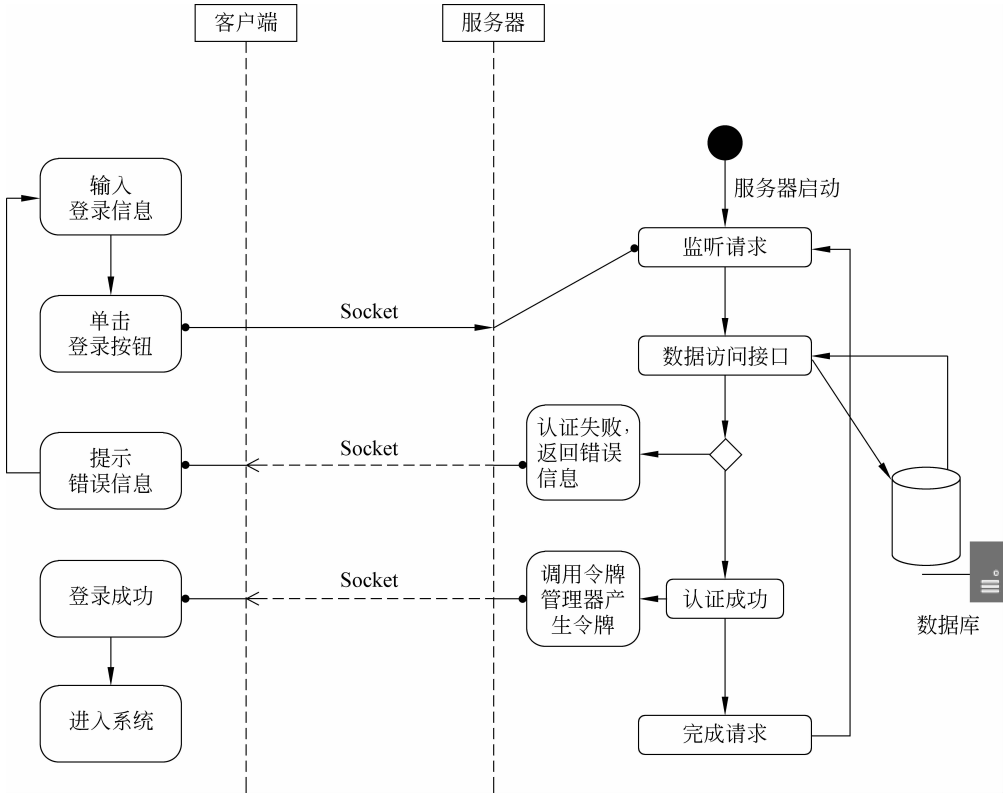

图 5.2 认证流程

5.4 服务器包结构

服务器包结构如图 5.3 所示，从图中可看出与服务器相关的包和类的层次结构。

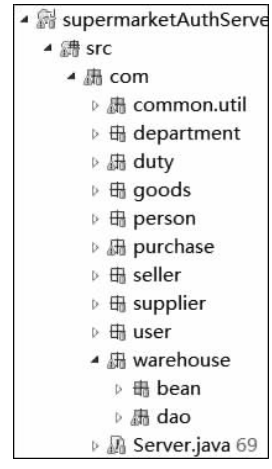

图 5.3 服务器包结构

5.5 代码实现

5.5.1 创建数据库连接工具

1. 导入数据库驱动包

在工程下新建目录 lib，并将 SQL Server 数据库驱动包 sqljdbc.jar 放置到 lib 文件夹下，如图 5.4 所示。

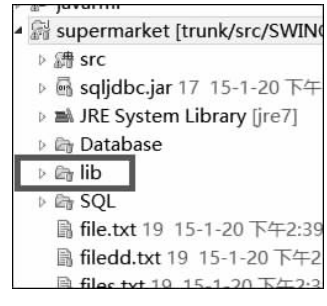

图 5.4 将驱动包放在新建的 lib 文件夹下

导入 sqljdbc.jar，右键单击项目，在弹出的菜单中选择 Build Path 后，会弹出 Configure Build Path。单击进入后可进行编译路径配置，如图 5.5 所示。

选择 Add JARs，在弹出框中选择 sqljdbc.jar，再单击 OK 按钮，如图 5.6 所示。

2. 创建服务器项目

在 Eclipse 中新建项目 supermarketAuthServer 作为服务器端程序，并导入 sqljdbc.jar。

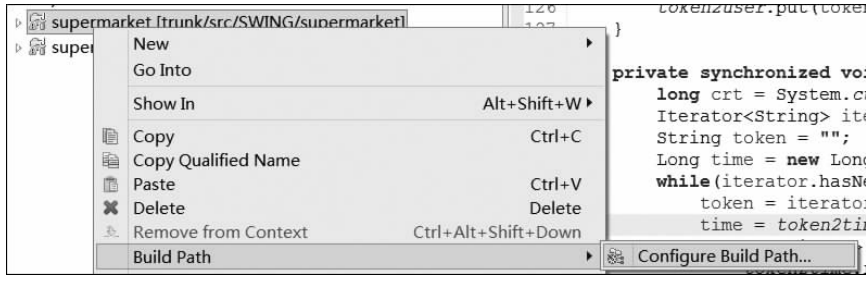

图 5.5 配置编译路径

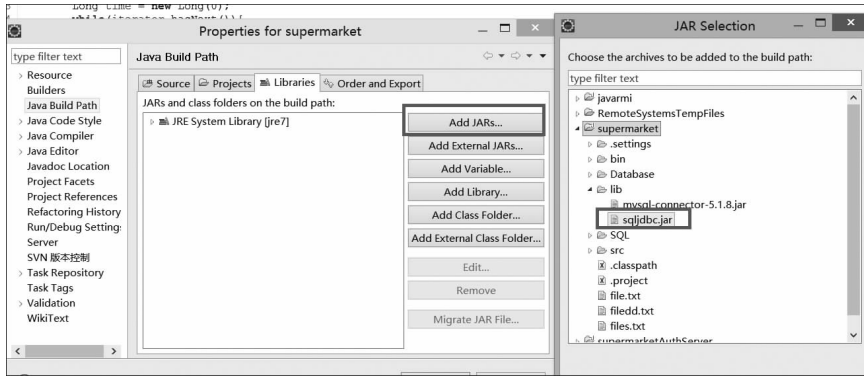

图 5.6 选择驱动包

3. 定义数据库连接类

DataConnection 中定义了一个构造函数，并使用 Class. forName()来加载一个驱动类。不同的数据库有不同的驱动类，本项目采用 SQLServer，其驱动类名为 com. microsoft. sqlserver. jdbc. SQLServerDriver，在第 2 章导入的 sqljdbc. jar 包里。

代码如下：

```
public class DataConnection {
private Connection con;                          //定义数据库连接类对象
private PreparedStatement pstm;
private String user="sa";                        //连接数据库用户名
private String password="123456";                //连接数据库密码
private String className="com.microsoft.sqlserver.jdbc.SQLServerDriver";
                                                 //数据库驱动
private String url="jdbc:sqlserver://localhost:1433;DatabaseName=db_
//supermarket";                                  //连接数据库的 URL
public DataConnection(){
    try{
        Class.forName(className);                //加载驱动
    }catch(ClassNotFoundException e){
```

```
        System.out.println("加载数据库驱动失败!");
        e.printStackTrace();
    }
}
```

4. 将数据库地址、账户和密码传递给 DriverManager. getConnection()方法已获取数据库连接类 Connection 的实例 con

代码如下：

```
public Connection getCon(){
    try {
        con=DriverManager.getConnection(url,user,password); //获取数据库连接
    } catch(SQLException e) {
        System.out.println("创建数据库连接失败!");
        con=null;
        e.printStackTrace();
    }
    return con;          //返回数据库连接对象
}
```

5. 执行 SQL 查询的方法 doPstm()(其中参数 params 为传递的参数值数组)

代码如下：

```
public void doPstm(String sql,Object[] params){
    if(sql!=null&&!sql.equals("")){
        if(params==null)
            params=new Object[0];            //创建参数数组
        getCon();
        if(con!=null){
            try{
                System.out.println(sql);
    pstm=con.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
                for(int i=0;i<params.length;i++){   //for 循环,用于设置查询参数
                    pstm.setObject(i+1,params[i]);
                }
                pstm.execute();
            }catch(SQLException e){
                System.out.println("doPstm()方法出错!");
                e.printStackTrace();
            }
        }
    }
}
```

6. 获取查询结果的方法 getRs()

返回一个查询结果数据集 ResultSet 类的实例,代码如下:

```
public ResultSet getRs() throws SQLException{
    return pstm.getResultSet();
}
```

7. 获取受影响数据行数及关闭数据库连接的方法

代码如下:

```
public int getCount() throws SQLException{
        return pstm.getUpdateCount();
    }
    public void closed(){
        try{
            if(pstm!=null)
                pstm.close();
        }catch(SQLException e){
            System.out.println("关闭 pstm 对象失败!");
            e.printStackTrace();
        }
        try{
            if(con!=null){
                con.close();
            }
        }catch(SQLException e){
            System.out.println("关闭 con 对象失败!");
            e.printStackTrace();
        }
    }
}
```

5.5.2 创建和实现用户数据访问接口

1. 定义用户数据访问接口

DAO 层主要完成数据持久层的工作,负责与数据库进行联络的一些任务都封装在此。

UserDao 接口封装了用户类与数据库之间的操作,声明了一个获取用户的方法 getUser():

```
public interface UserDao {
```