Programmieren II

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2010

Sockets

Über einen *Socket* kann eine Anwendung die Implementierung des Netzwerkprotokolls des darunter liegenden Betriebssystems ansprechen.

Hierbei unterscheidet man

Datagram Sockets zum verbindungslosen Austausch einzelner Nachrichten (Telegramme), i. A. über UDP

Stream Socket zum Austausch von Zeichen bzw. Bytes in einem bidirektionalen Datenstrom.

Diese Vorlesung beschränkt sich auf die Stream Sockets.

Grundsätzlicher Ablauf

- Auf einem *Host* läuft ein Serverprozess, der an einem bestimmten *Port* lauscht.
- Ein Client fordert nun eine Verbindung zu diesem Port auf dem Host an, hierfür wird auf dem Client-Host ein Port für die Verbindung geöffnet.
- Der Server akzeptiert die Verbindungsanfrage und legt einen neuen Socket an, der mit dem Port des Clients verbunden ist.

Beispiel: EchoServer

Nun folgt ein Beispiel für einen Server, der alle gelesenen Daten mit einem Echo beantwortet.

```
import java.io.*;
import java.net.*;

public class EchoServer {
    public static void main(String[] args) {
        ServerSocket echoServer;
        Socket clientSocket;
        String line;
        try {
            echoServer = new ServerSocket(9999);
```


Beispiel: EchoServer (Forts.)

```
            while (true) {
                clientSocket = echoServer.accept();
                BufferedReader in =
                    new BufferedReader(
                            new InputStreamReader(
                                clientSocket.getInputStream()));
                PrintWriter os =
                    new PrintWriter(clientSocket.getOutputStream());
                while ( (line=in.readLine()) != null) {
                    os.println(line);
                    os.flush();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Analyse

- Mit `new ServerSocket(9999)` wird ein Socket geöffnet, der auf Port 9999 lauscht.
- Die Zeile `echoServer.accept()` wartet darauf, dass sich ein Client verbindet und liefert dann einen Socket zurück.
- Die Kommunikation mit dem Client-Socket erfolgt dann über Streams:
  - `clientSocket.getInputStream()`
  - `clientSocket.getOutputStream()`

Beispiel: EchoClient

Nun folgt das entsprechende Gegenstück, der Client:

```
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) {
        try {
            Socket client = new Socket("localhost", 9999);
            System.out.println("Verbindung");
            BufferedReader in = new BufferedReader(
                    new InputStreamReader(
                        client.getInputStream()));
            PrintWriter out =
                    new PrintWriter(client.getOutputStream());
```


Beispiel: EchoClient (Forts.)

```
            out.println("Hallo");
            out.flush();
            System.out.println(in.readLine());
            out.println("Echo");
            out.flush();
            System.out.println(in.readLine());
            client.close();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Beispiel: SMTP-Client

Im Folgenden soll ein (einfacher!) SMTP-Client implementiert werden. Dabei besteht ein SMTP-Dialog zum Verschicken einer Mail über einen SMTP-Server aus einer Reihe von im Protokoll festgelegten Befehlen.

Diese Befehlsreihenfolge wird im Beispiel über eine Zustandsmaschine (*State Machine*) gelöst, unter Verwendung des Entwurfsmuster *State Pattern*: Jeder Schritt der Kommunikation wird über eine einzelne Klasse gelöst, wobei im Fall erfolgreicher Kommunikation zum nächsten Schritt, ansonsten zum Fehlerfall weitergeleitet wird.

Implementierung

```
public class Smtp {
    private String host;
    private int port;
    private String user;
    private String pwd;
    private String from;
    private String to;
    private String subject;
    private String body;
    private Socket sock;
    private int status;
    private String response;
    private SmtpState state;

    public Smtp(String host, int port, String user, String pwd) {
        this.host = host;
        this.port = port;
        this.user = user;
        this.pwd = pwd;
    }
```


Implementierung (Forts.)

```
    public void send(String from, String to, String subject, String body)
        this.from = from; this.to = to;
        this.subject = subject; this.body = body;
        setState(new ConnectState());
        while (getState()!= null) {
            if (getState() instanceof ErrorState) {
                System.err.println("Error: "+response);
                break;
            }
            getState().operate();
            System.out.format("Status: %d; Text: %s\n\n",
                status, response);
        }
    }

    public SmtpState getState() {
        return state;
    }

    public void setState(SmtpState state) {
        this.state = state;
    }
```

Implementierung (Forts.)

```
abstract class SmtpState {
   public void send(String s) {
      try {
         sock.getOutputStream().write(s.getBytes());
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

   public String read() {
      StringBuffer buf = new StringBuffer();
      try {
         BufferedReader in = new BufferedReader(
            new InputStreamReader(sock.getInputStream()));
         String line;
         while ( (line=in.readLine()) != null ) {
            status = Integer.valueOf(line.substring(0, 3));
            buf.append(line.substring(4));
            if (line.charAt(3)!='-') break;
            buf.append('\n');
         }
         response = buf.toString();
```


Implementierung (Forts.)

```
      } catch (IOException e) {
         e.printStackTrace();
      }
      return buf.toString();
   }

   public abstract void operate();
}

class ErrorState extends SmtpState {
   @Override
   public void operate() {
      setState(null);
   }
}
```


Implementierung (Forts.)

```
class ConnectState extends SmtpState {
   @Override
   public void operate() {
      try {
         sock = new Socket(host,port);
         read();
         setState(status<300 ? new EhloState() : new ErrorState());
      } catch (UnknownHostException e) {
         setState(new ErrorState());
      } catch (IOException e) {
         setState(new ErrorState());
      }
   }
}

class EhloState extends SmtpState {
   @Override
   public void operate() {
      send("EHLO .\r\n");
      read();
      setState(status<300 ? new LoginState() : new ErrorState());
   }
}
```

Implementierung (Forts.)

```
class LoginState extends SmtpState {
   @Override
   public void operate() {
      send("AUTH LOGIN\r\n");
      response=Base64Coder.decodeString(read());
      setState(status<400 ? new UserState() : new ErrorState());
   }
}

class UserState extends SmtpState {
   @Override
   public void operate() {
      send(Base64Coder.encodeString(user)+"\r\n");
      response=Base64Coder.decodeString(read());
      setState(status<400 ? new PwdState() : new ErrorState());
   }
}
```


Implementierung (Forts.)

```
class PwdState extends SmtpState {
   @Override
   public void operate() {
      send(Base64Coder.encodeString(pwd)+"\r\n");
      read();
      setState(status<300 ? new FromState() : new ErrorState());
   }
}

class FromState extends SmtpState {
   @Override
   public void operate() {
      send("MAIL FROM:<klaus.hoeppner@gmx.de>\r\n");
      read();
      setState(status<300 ? new RcptToState() : new ErrorState());
   }
}
```


Implementierung (Forts.)

```
class RcptToState extends SmtpState {
   @Override
   public void operate() {
      send("RCPT TO:<klaus.hoeppner@gmail.com>\r\n");
      read();
      setState(status<300 ? new DataState() : new ErrorState());
   }
}

class DataState extends SmtpState {
   @Override
   public void operate() {
      send("DATA\r\n");
      read();
      setState(status<400 ? new SendPayloadState() : new ErrorState());
   }
}
```

Implementierung (Forts.)

```
class SendPayloadState extends SmtpState {
   @Override
   public void operate() {
      send(String.format("From: %s\r\n"+
            "To: %s\r\nSubject: %s\r\n"+
            "\r\n"+
            "%s\r\n.\r\n",from,to,subject,body));
      read();
      setState(status<300 ? new CloseState() : new ErrorState());
   }
}

class CloseState extends SmtpState {
   @Override
   public void operate() {
      try {
         sock.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
      setState(null);
   }
}
```