Bölüm 24

Java Ağ Uygulamaları

24.1 Java Soket Programlama

Soket, bir sunucu programı ve bir veya birden çok istemci programı arasında çift yönlü iletişim kuran bir yazılım uç noktasıdır.

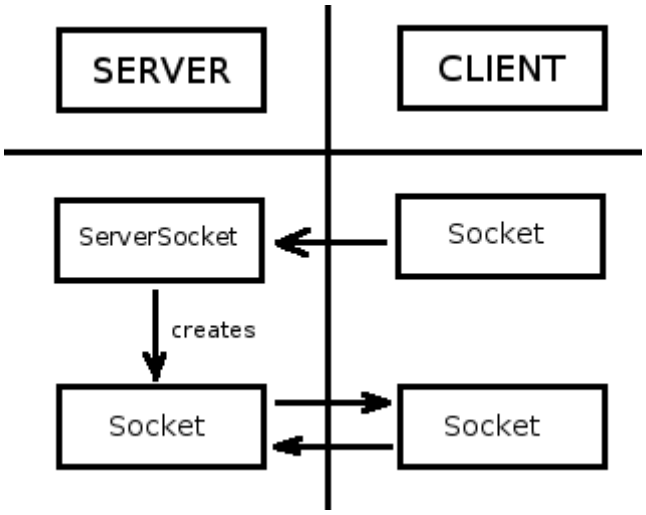
Sunucu programı genellikle istemci programları ağına kaynak sağlar. İstemci programları, sunucu programına istekler gönderir ve server programı da bu isteklere yanıt verir.

Thread(iş parçacığı) kullanarak, multi-threaded(çoklu iş parçacığı) sunucu programı istemciden genel bağlantıyı kabul edebilir, bu bağlantı için thread başlatılır ve diğer istemcilerin istekleri dinlenilmeye devam edilir.

java.net paketi 2 yaygın ağ protokolü için destek sağlar:

- **TCP:** TCP, Transmission Control Protocol anlamına gelir ve bu protokol iki uygulama arasında güvenilir iletişim sağlar. TCP genellikle, TCP/IP olarak anılan Internet Protocol üzerinden kullanır.
- **UDP:** UDP, User Datagram Protocol anlamına gelir, ve bu bağlantısız protokol uygulamalar arasında aktarılacak veri paketleri sağlar

Bağlantı kurulduğunda, sunucu iletişimin sonunda bir soket objesi yaratır. İstemci ve sunucu bundan soketten yazıp okuyarak iletişim kurabilirler.



Aşağıdaki adımlar, soketleri kullanan iki bilgisayar arasında TCP bağlantısı kurulurken meydana gelir.

1. Sunucu, port numarası iletişimi oluştuğunu gösteren bir ServerSocket nesnesini somutlaştırır.

2. Sunucu, ServerSocket sınıfındaki **accept()** metodunu çalıştırır. Bu metot, istemcinin verilen port numarası üstündeki sunucuya bağlamasına kadar devam eder.

3. İstemci, bağlantı kurulacak sunucu adı ve port numarsını belirterek Socket nesnesini somutlaştırır.

4. Socket sınıfının constructor'ı, belirtilen sunucu ve port numarasından istemciye bağlanma girişiminde bulunur. Eğer iletişim kurulursa, istemci sunucu ile iletişim kurma yeteneğine sahip Socket nesnesini edinir.

Bağlantılar kurulduktan sonra, iletişim **I/O streams** kullanılarak meydana gelebilir. Her soket hem bir OutputStream hem de bir InputStream'e sahiptir. İstemcinin OutputStream'i, sunucunun InputStream'ine bağlanır ve istemcinin InputStream'I sunucunun OutputStream'ine bağlanır. TCP iki yönlü iletişim protokolüdür, bu yüzden veri, eş zamanlı her iki stream'in karşısına gönderilebilir.

SimpleServer.Java

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

public class SimpleServer extends Thread
{
   private ServerSocket serverSocket;

   public SimpleServer(int port) throws IOException
   {
      serverSocket = new ServerSocket(port);
      serverSocket.setSoTimeout(100000);
   }

   public void run()
   {
      while(true)
      {
         try
         {
          System.out.println("Waiting for client on port "+ serverSocket.getLocalPort());
            Socket server = serverSocket.accept();

            System.out.println("Just connected to " + server.getRemoteSocketAddress());
            DataInputStream in = new DataInputStream(server.getInputStream());
            System.out.println(in.readUTF());

            DataOutputStream out = new DataOutputStream(server.getOutputStream());
            out.writeUTF("Thank you for connecting to " + server.getLocalSocketAddress()
+ "\nGoodbye!");

            server.close();
         }catch(SocketTimeoutException s)
         {
            System.out.println("Socket timed out!");
            break;
         }catch(IOException e)
         {
            e.printStackTrace();
            break;
         }
      }
   }
   public static void main(String [] args)
   {
      try
      {
         // use TCP 6666 port
          Thread t = new SimpleServer(6666);
         t.start();

      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

SimpleClient.Java

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

public class SimpleClient
{
   public static void main(String [] args)
   {
      String serverName = "localhost";
      int port = 6666;
      try
      {
         System.out.println("Connecting to " + serverName + " on port " + port);
         Socket client = new Socket(serverName, port);
        System.out.println("Just connected to " + client.getRemoteSocketAddress());

         OutputStream outToServer = client.getOutputStream();
         DataOutputStream out = new DataOutputStream(outToServer);
         out.writeUTF("Hello from "  + client.getLocalSocketAddress());

         InputStream inFromServer = client.getInputStream();
         DataInputStream in = new DataInputStream(inFromServer);
         System.out.println("Server says " + in.readUTF());

         client.close();
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

24.2 E-mail Gönderme

Java Uygulamanızı kullanarak email göndermek oldukça kolaydır ama başlamak için öncellikle **JavaMail API'**si makinanıza kurulu olmalıdır.

- **JavaMail'**in son versiyonunu, Java'nın standart web sitesinden indirebilirsiniz.

  http://www.oracle.com/technetwork/java/javamail/index-138643.html

Dosyaları indirin ve açın, her iki uygulama için bir dizi jar dosyası bulacaksınız. Öncelikle **mail.jar** dosyalarını CLASSPATH konumuna eklemeniz gerekmektedir(Eclipse içindeki src dizinine sürükleyip bırakın, mail.jar dosyasına sağ tıklayıp **Build Path**-> **Add to Build Path** kısmını seçin)

Burada, **GOOGLE GMAIL SMTP SERVER** aracılığıyla basit bir email gönderme örneğini görüyoruz. Burada, **localhost**'unuzun internete bağlı olduğu ve email göndermek için yeterli olduğu varsayılıyor.

```java
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;

public class SendEmail {
        public static void main(String[] args) {

                // Sender's email ID needs to be mentioned
                String from = "sender@gmail.com";
                String pass = "123456";

                // Recipient's email ID needs to be mentioned.
                String to = "someone@hotmail.com";


                String host = "smtp.gmail.com";
                // Get system properties
                Properties properties = System.getProperties();
                // Setup mail server
                properties.put("mail.smtp.starttls.enable", "true");
                properties.put("mail.smtp.host", host);
                properties.put("mail.smtp.user", from);
                properties.put("mail.smtp.password", pass);
                properties.put("mail.smtp.port", "587");
                properties.put("mail.smtp.auth", "true");

                // Get the default Session object.
                Session session = Session.getDefaultInstance(properties);

        try {
                // Create a default MimeMessage object.
                MimeMessage message = new MimeMessage(session);

                // Set From: header field of the header.
                message.setFrom(new InternetAddress(from));

                // Set To: header field of the header.
                message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));

                // Set Subject: header field
                message.setSubject("This is the Subject Line!");

                // Now set the actual message
                message.setText("This is actual message");

                // Send message
                Transport transport = session.getTransport("smtp");
                transport.connect(host, from, pass);
                transport.sendMessage(message, message.getAllRecipients());
                transport.close();
                System.out.println("Sent message successfully....");

        } catch (MessagingException mex) {
                        mex.printStackTrace();
                }
        }
}
```