Soketové připojení

Při socketovém připojení spolu komunikují dvě nezávislé JVM pomocí streamů odpovídajících typů. Po síti můžete posílat textové i binární informace, je možno posílat i objekty, které byly do streamů převedeny pomocí serializace. Pro zvýšení bezpečnosti komunikace je možno použít další javovské knihovny a komunikovat např. přes SSL. Pomocí bezpečnostních politik je možno zajistit i další bezpečnostní omezení, např. klienti mohou být pouze ze zadané domény atd.

Třídy potřebné pro připojení pomocí socketu najdete v balíčku java.net.

Klient

Pro vytvoření spojení potřebujete instanci třídy **InetAddress**, která obsahuje informaci o internetové adrese serveru. Tato třída nemá veřejný konstruktor a její instance se vytvářejí pomocí statické metody **getByName** nebo **getByAddress()**. Instanci této třídy lze vytvořit buď pomocí IP adresy nebo logického jména, např.:

InetAddress adresa = InetAddress.getByName(“146.102.64.82”);

Tato metoda může vyvolat výjimku **UnknowHostException** a případně i výjimku **SecurityException**.

Na straně klienta je třeba vytvořit instanci třídy **Socket**, která má v konstruktoru dva parametry. První parametr je typu InetAddress s adresou serveru a druhý typu int udává číslo portu, na který se klient u serveru připojí. Číslo portu by mělo být vyšší než 1025.

Po připojení je nutné vytvořit komunikační streamy pomocí metod instance Socketu **getInputStream()** a **getOutputStream()**. Na konci komunikace je třeba zavřít socket metodou close() a to i v případě, že při spojení došlo k chybě.

Všechny tyto činnosti mohou vyvolat výjimku **IOException**.

Server

Na straně serveru je třeba nejdříve vytvořit instanci třídy **ServerSocket**, která má v konstruktoru číslo portu. Toto číslo se musí shodovat s číslem portu uvedeným v socketu klienta. Pak spustíme metodu instance ServerSocketu **accept()**, která vrací instanci třídy **Socket** po připojení nějakého klienta.

Na straně serveru pak opět vytvoříme streamy pro čtení a zápis. Na konci komunikace je třeba uzavřít socket i server socket metodami close().

Všechny tyto činnosti mohou vyvolat výjimku **IOException**.

Takto vytvořený server je pouze jednouživatelský a při každém spuštění použitelný pouze jednou. Aby měl server smysl, musíte vytvořit podporu pro víceuživatelský přístup. Díky podpoře vláken v Javě to není takový problém. Server bude ve smyčce čekat na připojení klienta, obslouží ho v samostatném vláknu. Toto vlákno vytvoří odpovídající streamy a po skončení komunikace s klientem je uzavře, uzavře socket a ukončí se. Pokud aplikace potřebuje, aby o sobě jednotliví klienti věděli, je třeba vlákna zařazovat do skupin. Pokud jednotliví klienti využívají na serveru omezený zdroj, např. soubor, je nutná synchronizace jim příslušných vláken.

Jednoduchý příklad

Jak vytvořit jednoduchého klienta a server, si ukážeme na následujícím příkladě.

Server poběží na localhostu na portu 1234, bude čekat na připojení klienta. Po připojení bude zprávy posílané klientem zapisovat do souboru a to tak dlouho dokud zpráva od klienta nebude řetězec konec. Poté server zavře soubor i socketové připojení.

Klient se připojí k serveru a bude mu posílat zprávy načítané z konzole do té doby dokud uživatel nevloží řetězec konec. Pak ukončí spojení se serverem a skončí.

Kód serveru:

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

public class Server1 {

    ServerSocket serverSocket;
    Socket socket;
    BufferedReader vstup;
    PrintWriter vystup, soubor;

    public Server1(int port) throws IOException {
        serverSocket = new ServerSocket(port);
        System.out.println("Server spusten");
        soubor = new PrintWriter(new FileWriter("text.txt"));
    }

    public void navazSpojeni() throws IOException {
        socket = serverSocket.accept();
        try{
            vstup = new BufferedReader(new
                    InputStreamReader(socket.getInputStream()));
            vystup = new PrintWriter(new
                OutputStreamWriter(socket.getOutputStream()),true);
        }
        catch (IOException e){
            System.out.println("Problem pri vytvareni spojeni");
            socket.close();
            throw e;
        }
    }

    public void zapisNaServeru() throws IOException {
        try{
            String radek;
            while (!(radek = vstup.readLine()).equals("konec")) {
                System.out.println(radek);
                vystup.println("Zapsal jsem: "+ radek);
                soubor.println(radek);
            }
        }
        catch (IOException e){
            System.out.println("Problem v prubehu spojeni");
        }
        finally {
            socket.close();
        }
    }
```

```
    public static void main (String [] args) throws IOException {
        Server1 server = new Server1(1234);
        try{
            server.navazSpojeni();
            server.zapisNaServeru();
        }
        catch (IOException e){
            System.out.println("Problem pri vytvareni socketu");
        }
        finally {
            server.serverSocket.close();
            server.soubor.close();
        }
    }
}
```

Kód klienta:

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

public class Klient {
    InetAddress adresaServeru;
    Socket socket;
    BufferedReader vstup, konzola;
    PrintWriter vystup;

    public Klient (int port) throws IOException {
        adresaServeru= InetAddress.getByName("localhost");
        socket = new Socket(adresaServeru,port);
    }

    public void navazSpojeni() throws IOException {
        try{
            vstup = new BufferedReader(new
                InputStreamReader(socket.getInputStream()));
            vystup = new PrintWriter(new
                OutputStreamWriter(socket.getOutputStream()),true);
        }
        catch (IOException e){
            System.out.println("Problem pri vytvareni spojeni,
                      zaviram socket v metode vytvor spojeni");
            socket.close();
            throw e;
        }
    }
```

```
    public void zapisNaServer() throws IOException {
        try{
            konzola = new BufferedReader(new
                InputStreamReader(System.in));
            String radek;
            while (true) {
                System.out.print("Napis text: ");
                radek = konzola.readLine();
                vystup.println(radek);
                System.out.println(vstup.readLine());
                if (radek.equals("konec")){
                    break;
                }
            }
        }
        catch (IOException e){
            System.out.println("Problem v prubehu spojeni");
        }
        finally {
            socket.close();
            System.out.println("Zaviram socket v metode zapis");
        }
    }

    public static void main (String [] args) throws IOException {
        try{
            Klient klient = new Klient(1234);
            try{
                klient.navazSpojeni();
                klient.zapisNaServer();
            }
            catch (IOException e) {
                System.out.println("Problem pri vytvareni
                                spojeni");
            }
        }
        catch (ConnectException e){
            System.out.println("Problem na serveru");
        }
    }
}
```

Pro bližší pochopení tématu si vyzkoušejte upravit server tak, aby byl schopen komunikovat s více klienty. Nezapomeňte na synchronizaci souboru.