Gniazda komunikacji sieciowej w środowisku Java

Dariusz Wawrzyniak

Dariusz.Wawrzyniak@cs.put.poznan.pl

1. Obsługa gniazd
   - Reprezentacja adresów
   - Tworzenie gniazd
2. Obsługa komunikacji peer-to-peer
   - Obsługa komunikacji strumieniowej
   - Obsługa komunikacji pakietowej

Pakiet `java.net`

Pakiet `java.net` zawiera klasy do obsługi komunikacji sieciowej.

- java.lang.Object
  - java.net.InetAddress
  - java.net.SocketAddress
  - java.net.Socket
  - java.net.ServerSocket
  - java.net.DatagramSocket

Gniazda w środowisku Java

Gniazdo

Gniazdo jest punktem końcowym dwukierunkowej komunikacji pomiędzy odległymi procesami.

Pakiet `java.net` definiuje kilka klas do obsługi gniazd:

- klasa `Socket` dla gnizada komumnikacji strumieniowej peer-to-peer,
- klasa `ServerSocket` dla gniazda nasłuchu po stronie serwera,
- klasa `java.net.DatagramSocket` dla gniazda komunikacji pakietowej,
- klasa `java.net.MulticastSocket` dla gniazda rozgłoszeniowego.

Adres gniazda

Adres gniazda składa się z:

- adresu komputera (hosta)
- numeru portu komunikacji sieciowej

Adres komputera może być w postaci:

- nazwy tekstowej zgodnej z wymogami odpwiedniego serwisu nazewniczego,
- adresu IP w postaci tekstowej lub numerycznej.

Klasy do reprezentacji adresów

- java.lang.Object
  - java.net.InetAddress
    - java.net.Inet4Address
    - java.net.Inet6Address
  - java.net.SocketAddress
    - java.net.InetSocketAddress

Klasa `InetAddress`

- Klasa reprezentuje adres IP.
- Klasa nie definiuje publicznych konstruktorów, a obiekty tworzone są jako wynik wywołania metod statycznych (metod klasy)
  - `static InetAddress getByAddress(byte[] addr)` — tworzy obiekt na podstawie adresu IP w postaci binarnej,
  - `static InetAddress getByAddress(String host, byte[] addr)` — tworzy obiekt na podstawie nazwy hosta i binarnego adresu IP (bez odwoływania się do serwisu nazw),
  - `static InetAddress getByName(String host)` — tworzy obiekt na podstawie nazwy hosta,
  - `static InetAddress getLocalHost()` — tworzy obiekt reprezentujący adres lokalny hosta.

Klasy `Inet4Address` i `Inet6Address`

- Klasy reprezentują odpowiednio adres IP w wersji 4 (IPv4) i adres IP w wersji 6 (IPv6).
- Klasy wywiedzione są z `InetAddress`, skąd dzidziczą większość własności. Dzidziczenie ma charakter ostateczny (final).
- Część metod klasy `InetAddress` jest redefiniowana, między innymi:
  - `byte[] getAddress()` — zwraca adres IP w postaci binarnej,
  - `String getHostAddress()` — zwraca napis z reprezentację tekstową adresu IP.

Klasa `SocketAddress` i `InetSocketAddress`

- `SocketAddress` jest klasą abstrakcyjną, reprezentującą ogólny adres gniazda (nie związany z żadnym protokołem).
- Klasa `InetSocketAddress` jest pochodną klasy `SocketAddress` i jest typem adresu gniazda w dziedzinie Internetu. W skład obiektu klasy `InetSocketAddress` wchodzi adres IP hosta oraz numer portu.
- Adres do obiektu klasy `InetSocketAddress` przypisywany jest tylko w jednym z konstruktorów:
  - `InetSocketAddress(InetAddress addr, int port)`
  - `InetSocketAddress(int port)`
  - `InetSocketAddress(String hostname, int port)`

Klasy do reprezentacji gniazd

- java.lang.Object
  - java.net.Socket
    - javax.net.ssl.SSLSocket
  - java.net.ServerSocket
    - javax.net.ssl.SSLServerSocket
  - java.net.DatagramSocket
    - java.net.MulticastSocket

Klasa `Socket`

Klasa `Socket` udostępnia mechanizmy:

- komunikacji strumieniowej peer-to-peer zarówno dla serwera, jak i dla klienta,
- nawiązywania połączenia z serwerem po stronie klienta.

Obiekt klasy `Socket` jest tworzony:

- jawnie przez klienta w celu nawiązania połączenia z serwerem,
- jako wynik metody `accept` gniazda klasy `ServerSocket` po stronie serwera w konsekwencji nawiązania połączenia przez klienta.

Nawiązywanie połączenia przez klienta

Podjęcie próby nawiązania połączenia z serwerem po stronie klienta następuję:

- w ramach tworznie obiektu klasy `Socket`, co wymaga użycia odpowiedniego konstruktora,
- w wyniku wywołania metody `connect` obiektu klasy `Socket`.

Konstruktory obiektu klasy `Socket`

`Socket()` — utworzenie gniazda,
`Socket(InetAddress address, int port)` — utworzenie gniazada oraz próba nawiązania połączenia z serwerem,
`Socket(InetAddress address, int port, InetAddress localAddr, int localPort)` — utworzenie gniazada z dowiązaniem lokalnym oraz próba nawiązania połączenia z serwerem,
`Socket(String host, int port)` — utworzenie gniazada oraz próba nawiązania połączenia z serwerem na maszynie o podanej nazwie,
`Socket(String host, int port, InetAddress localAddr, int localPort)` — utworzenie gniazada z dowiązaniem lokalnym oraz próba nawiązania połączenia z serwerem na maszynie o podanej nazwie.

Nawiązywanie połączenia w konstruktorze

```
Socket socket;
try {
   socket = new Socket("localhost", 6000);
   System.out.println("Polaczenie nawiazane");
}
   catch (Exception e) {
      System.err.println( e.getMessage() );
      e.printStackTrace();
   }
```

Podstawowa obsługa gniazda komunikacji strumieniowej

`void bind(SocketAddress bindpoint)` — dowiązanie do lokalnego adresu (nadanie nazwy),
`void close()` — zamknięcie gniazda,
`void connect(SocketAddress endpoint)` — próba nawiązania połączenia z serwerem,
`void connect(SocketAddress endpoint, int timeout)` — próba nawiązania połączenia z serwerem w określonym czasie.

Nawiązywanie połączenia w sposób jawny

```
Socket socket;
InetSocketAddress address;
try {
   socket = new Socket();
   address = new InetSocketAddress("localhost", 6000);
   socket.connect(address);
   System.out.println("Polaczenie nawiazane");
}
   catch (Exception e) {
      System.err.println( e.getMessage() );
      e.printStackTrace();
   }
```

Klasa `ServerSocket`

- Obiekt klasy `ServerSocket` reprezentuje gniazdo, na którym serwer oczekuje zgłaszających się klientów.
- Numer portu przypisany do gniazda klasy `ServerSocket` jest używany przez klienta w celu nawiązania połączenia.
- Adres (nazwa ganizda) przypisywany jest w czasie tworzenia obiektu (co wymaga użycia odpowiedniego konstruktora) lub po jego utworzeniu poprzez jawne wywołanie metody `bind` obiektu klasy `ServerSocket`.

Konstruktory obiektu klasy `ServerSocket`

`ServerSocket()` — utworzenie gniazda bez nadanej nazwy (bez dowiązania).
`ServerSocket(int port)` — utworzenie gniazda z przypisanym domyślnym adresem IP oraz podanym numerem portu.
`ServerSocket(int port, int backlog)` — utworzenie gniazda na podanym numerze portu z określeniem maksymalnej długości kolejki oczekujących żądań.
`ServerSocket(int port, int backlog, InetAddress bindAddr)` — utworzenie gniazda podobnie jak wyżej z dodaktowym podaniem adresu IP, jaki ma być przpisany do gniazda.

Obsługa połączenia przez serwer

```
ServerSocket server_socket;
Socket socket;
try {

   server_socket = new ServerSocket(6000);
   socket = server_socket.accept();
   System.out.println("Zglosil sie klient");
}
   catch (Exception e) {
      System.err.println( e.getMessage() );
      e.printStackTrace();
   }
```

Podstawowa obsługa gniazda nasłuchu

`Socket accept()` — oczekiwanie na zgłoszenie klientów lub przyjęcie wcześniej zarejestrowanego zgłoszenia.
`void bind(SocketAddress endpoint)` — jawne przypisanie adresu (nadanie nazwy).
`void bind(SocketAddress endpoint, int backlog)` — jawne przypisanie nazwy i określenie maksymalnej długości kolejki oczekujących żądań.
`void close()` — zamknięcie zgniazda.

Jawne dowiązanie gniazda

```
ServerSocket server_socket;
InetSocketAddress address;
Socket socket;
try {
   server_socket = new ServerSocket();
   address = new InetSocketAddress(6000);
   server_socket.bind(address);
   socket = server_socket.accept();
   System.out.println("Zglosil sie klient");
}
   catch (Exception e) {
      System.err.println( e.getMessage() );
      e.printStackTrace();
   }
```

Zamykanie gniazda komunikacji strumieniowej

Następujące metody obiektu klasy `Socket` służą do zamykania gniazda:

- `void close()` — zamknięcie gniazda do komunikacji w obu kierunkach,
- `void shutdownInput()` — zamknięcie kanału wejściowego, co uniemożliwia dalszy dobiór danych,
- `void shutdownOutput()` — zamknięcie kanału wyjściowego, co uniemożliwia dalsze wysyłanie danych.

Współbieżna obsługa wielu połączeń

```
ServerSocket server_socket;
Socket socket;
int number = 0;
try {
   server_socket = new ServerSocket(6000);
   while (true) {
      socket = server_socket.accept();
      System.out.println("Zglosil sie klient");
      number++;
      new ServiceThread( socket, number ).start();
   }
}
   catch (Exception e) {
      System.err.println( e.getMessage() );
      e.printStackTrace();
   }
```

Przykład wątku serwera współbieżnego

```
public class ServiceThread
                    extends java.lang.Thread {
   private Socket socket;
   private int number;
   public ServiceThread(Socket s, int n){
      socket = s;
      number = n;
   }
   public void run() {

...

   }
}
```

Klasa `DatagramSocket`

Klasa `DatagramSocket` udostępnia mechanizmy:

- wysyłania i odbioru datagramów,
- rozgłaszania datagramów w ramach segmentu sieci lokalnej.

Obiekt klasy `DatagramSocket` jest tworzony:

- przez klienta w celu skomunikowania się z serwerem,
- przez serwer w celu oczekiwania na komunikaty od klientów.

Konstruktory obiektu klasy `DatagramSocket`

`DatagramSocket()` — utworzenie gniazda i dowiązanie go do jakiegoś wolnego portu na lokalnej maszynie.
`DatagramSocket(int port)` — utworzenie gniazda i dowiązanie go do podanego portu na lokalnej maszynie.
`DatagramSocket(int port, InetAddress laddr)` — utworzenie gniazda i dowiązanie go do podanego portu i podanego adresu IP.
`DatagramSocket(SocketAddress bindaddr)` — utworzenie gniazda i dowiązanie go do podanego adresu.

Podstawowa obsługa gniazda komunikacji pakietowej

`void bind(SocketAddress addr)` — dowiązanie gniazda do podanego addressu.
`void close()` — zamknięcie gniazda.
`void connect(InetAddress address, int port)`, `void connect(SocketAddress addr)` — związanie gniazda z podanym adresem zdalnym oraz portem (ograniczenie możliwości komunikacji wyłącznie do tego adresu).
`void disconnect()` — rozłączenie gniazda.

Przekazywanie danych przez gniazdo strumieniowe

W trybie połączeniowym gniazdo udostępnia strumień wejściowy oraz strumień wyjściowy, za pośrednictwem których można odpowiednio odbierać i wysyłać dane. Referencje do strumieni zwracają następujące metody obiektu klasy `Socket`:

- `OutputStream getOutputStream()` — referencja do strumienia wyjściowego, przez który **wysyłane** są dane,
- `IntputStream getIntputStream()` — referencja do strumienia wejściowego, przez który **odbierana** są dane.

`OutputStream` oraz `OutputStream` są abstrakcyjnymi klasami, zdefiniowanymi wraz z kilkoma konkretnymi pochodnymi w pakiecie `java.io`.

Pakiet `java.io`

Pakiet udostępnia klasy do obsługi wejścia-wyjścia poprzez strumienie danych.

- java.lang.Object
  - java.io.InputStream
    - ByteArrayInputStream
    - FileInputStream
    - PipedInputStream
    - FilterInputStream
      - BufferedInputStream
      - DataInputStream

- Java.lang.Object
  - java.io.OutputStream
    - ByteArrayOutputStream
    - FileOutputStream
    - PipedOutputStream
    - FilterOutputStream
      - BufferedOutputStream
      - DataOutputStream

Przykład wysyłania danych przez gniazdo strumieniowe

```
try {
   DataOutputStream out;
   OutputStream out_sock;
   out_sock = socket.getOutputStream();
   out = new DataOutputStream ( out_sock );
   out.writeInt(87);
}
   catch (IOException e) {
      System.err.println( e.getMessage() );
      e.printStackTrace();
   }
```

Przykład odbierania danych przez gniazdo strumieniowe

```
try {
   int v;
   DataInputStream in;
   InputStream in_sock;
   in_sock = socket.getInputStream();
   in = new DataInputStream ( in_sock );
   v = in.readInt();
   System.out.println("Odebrano: " + v);
}
   catch (IOException e) {
      System.err.println( e.getMessage() );
      e.printStackTrace();
   }
```

Filtry

Filtry są szczególnym rodzajem strumieni. Umożliwiają one łączenie strumieni i tworzenie złożonych potoków dzięki specyficznym konstruktorom:

- `FilterInputStream(InputStream in)`
- `FilterOutputStream(OutputStream out)`

Buforowanie strumienia

- Filtry `BufferedInputStream` i `BufferedOutputStream` umożliwiają buforowanie danych (strumienia bajtów).
- Wielkość bufora można ustalić w czasie tworzenia obiektu z pomocą konstruktora: `BufferedInputStream(InputStream in, int size)`
- Dane z bufora są przekazywane do strumienia **wyjściowego** w wyniku zapełnienia bufora lub wywołania metody `flush()` obiektu klasy `FilterOutputStream` (lub pochodnej).

Przykład buforowania po stronie nadawczej

```
try {
   DataOutputStream out;
   OutputStream out_sock;
   BufferedOutputStream out_buf;
   out_sock = socket.getOutputStream();
   out_buf = new BufferedOutputStream( out_sock );
   out = new DataOutputStream ( out_buf );
   out.writeInt(87);
   out.flush();
}
   catch (IOException e) {
      System.err.println( e.getMessage() );
      e.printStackTrace();
   }
```

Przykład buforowania po stronie obiorczej

```
try {
   int v;
   DataInputStream in;
   InputStream in_sock;
   BufferedInputStream in_buf;
   in_sock = socket.getInputStream();
   in_buf = new BufferedInputStream( in_sock );
   in = new DataInputStream ( in_buf );
   v = in.readInt();
   System.out.println("Odebrano: " + v);
}
   catch (IOException e) {
      System.err.println( e.getMessage() );
      e.printStackTrace();
   }
```

Przesyłanie pakietów

Następujące metody gniazda klasy `DatagramSocket` służą do przekazywania komunikatów pomiędzy obiorcą na nadawcą:

- `void receive(DatagramPacket p)` — odbiór pakietu z gniazda z ewentualnym zablokowaniem odbiorcy w oczekiwaniu na dotarcie pakietu.
- `void send(DatagramPacket p)` — wysłanie pakietu przez gniazdo.

Wszelkie informacje adresowe przkazywane są wraz z transmitowanymi danymi w obiekcie klasy `DatagramPacket`.

Klasa `java.net.DatagramPacket`

Obiekt klasy `DatagramPacket` reprezentuje bufor do transmisji pakietowej. który może być używany zarówno jako bufor nadawczy, jak i bufor odbiorczy.
Dane w buforze interpretowane są jako ciąg bajtów.
Najważniejsze atrybuty bufora z punktu widzenia programisty:

- buf — tablica bajtów stanowiących transmitowane dane,
- length — liczba bajtów danych,
- offset — przesunięcie danych względem początku tablicy,
- address — adres gniazda źródłowego lub gniazda przeznaczenia.

Konstruktory obiektu klasy `DatagramPacket`

```
DatagramPacket(byte[] buf, int length)
DatagramPacket(byte[] buf, int length,
InetAddress address, int port)
DatagramPacket(byte[] buf, int offset, int
length)
DatagramPacket(byte[] buf, int offset, int
length, InetAddress address, int port)
DatagramPacket(byte[] buf, int offset, int
length, SocketAddress address)
DatagramPacket(byte[] buf, int length,
SocketAddress address)
```

Metody get* klasy DatagramPacket

```
InetAddress getAddress()
byte[] getData()
int getLength()
int getOffset()
int getPort()
SocketAddress getSocketAddress()
```

Metody `set*` klasy `DatagramPacket`

```
void setAddress(InetAddress iaddr)
void setData(byte[] buf)
void setData(byte[] buf, int offset, int
length)
void setLength(int length)
void setPort(int iport)
void setSocketAddress(SocketAddress address)
```