Networking

Zaawansowane technologie Javy 2019

Model klient-serwer

- W modelu klient-serwer (ang. *client-server*) dane trzymane są na serwerze, interfejs użytkownika i logika przetwarzania danych są realizowane na kliencie.
- Zadaniem serwera jest przetwarzanie i analizowanie danych przed odesłaniem ich do klienta.
- Przykłady takich modeli: FTP, WWW.
- Przeciwieństwem tego modelu jest model równorzędny (ang. *peer-to-peer*) takie jak gry sieciowe, system telefoniczny, itp.

Gniazda

- Transmitowanie danych w pakietach to bardzo skomplikowana czynność – na szczęście berkeleyowkie *gniazda* pozwalają traktować połączenie sieciowe jak strumień.
- Gniazdo obsługuje połączenie między dwoma hostami w sieci:
  - łączy się ze zdalną maszyną,
  - wysyła dane,
  - odbiera dane,
  - zamyka połączenie,
  - a także łączy się z portem, czeka na dane i odbiera połączenie o zdalnej maszyny na porcie granicznym.
- Klasa `Socket` jest wykorzystywana przez klienty i serwery – posiada metody pozwalające wykonywać pierwsze cztery z wymienionych operacji.

Klasa `Socket`

- `Socket` to podstawowa klasa do wykonywania operacji TCP po stronie klienta.
- W czasie konstrukcji obiektu klasy `Socket` od razu jest nawiązywane połączenie:

```
Socket (String host, int port) throws
  UnknownHostException, IOException
Socket (InetAddress host, int port)
 throws IOException
```

- Na końcu pracy z gniazdem należy zamknąć połączenie:

```
socket.close()
```

Klasa `Socket`

- Obiekt klasy `Socket` posiada kilka metod udostępniających informacje o gnieździe:
  `getInetAddress()`
  `getPort()`
  `getLocalPort()`
  `isClosed()`

Komunikacja z hostem za pomocą obiektu `Socket`

- Komunikacja z hostem jest realizowana za pomocą zwykłych strumieni bajtowych, do których dostęp uzyskuje się za pomocą metod:
  `getInputStream()`
  `getOutputStream()`
- Zwykle strumienie te są opakowywane:

```
Socket s = new Socket(…);
BufferedReader in = new BufferedReader(
  new InputStreamReader(s.getInputStream()));
PrintWriter out = new PrintWriter(
  new OutputStreamWriter(s.getOutputStream()));
```

Komunikacja z hostem za pomocą obiektu `Socket`

- Metoda `close()` gniazda zamyka automatycznie jego strumienie do komunikacji.
- Gdy jeden ze strumieni do komunikacji zostanie zamknięty, całe gniazdo jest zamykane.
- Gdy w czasie pracy z gniazdem chcemy zamknąć tylko strumień do czytania lub pisania należy użyć jednej z metod:
  `shutdownInput()`
  `shutdownOutput()`

Opcje gniazda `Socket`

- Opcje gniazda określają, w jaki sposób gniazda wysyłają i odbierają dane:
  - `TCP_NODELAY` – wartość `true` wyłącza schemat buforowania;
  - `SO_LINGER` – określa, co należy zrobić z datagramami, które nie zostały wysłane przed zamknięciem gniazda;
  - `SO_TIMEOUT` – wartość wyrażona w milisekundach powoduje, że gniazdo nie zablokuje się na dłużej w trakcie czytania (zgłaszany jest wyjątek `InterruptedException` ale gniazdo nie jest zamykane);
  - `SO_KEEPALIVE` – wartość `true` włącza system kontrolowania bezczynnych połączeń (raz na dwie godziny).

Serwer

- Pisząc serwer zawsze trzeba opracować protokół rozmowy z klientami.
- Cykl życiowy serwera:
  - serwer zajmuje port;
  - w pętli: serwer czeka na klienta, prowadzi z nim rozmowę a na końcu zamyka połączenie z klientem;
  - serwer zwalnia port po zakończonej pracy.

Klasa `ServerSocket`

- Gniazdo serwera zajmuje lokalny port i czeka na nadchodzące połączenia TCP.
- W czasie konstrukcji obiektu klasy `ServerSocket` od razu jest zajmowany lokalny port:

```
ServerSocket (int port)
  throws IOException
ServerSocket (int port, int queue)
  throws IOException
```

Klasa `ServerSocket`

- Przyjmowanie i zamykanie połączeń:

```
ServerSocket server =
  new ServerSocket(4444);
Socket s = server.accept();
// rozmowa z klientem
s.close();
```

- Dostęp do strumieni za pomocą gniazda roboczego:

```
OutputStream os =
  s.getOutputStream();
InputStream is =
  s.getInputStream();
```

Klasa `ServerSocket`

- Obiekt klasy `ServerSocket` posiada kilka metod udostępniających informacje o gnieździe:
  `getInetAddress()`
  `getLocalPort()`
- Opcje gniazd serwera:
  `SO_TIMEOUT` – wartość wyrażona w milisekundach określa czas akceptacji nowego połączenia (wartość 0 oznacza brak limitu).

Program `telnet`

- Przykład wysłania listu z konsoli:

```
prz@sunflower:~$ telnet swiatowit 25
Trying 192.168.3.1...
Connected to swiatowit.prac.ii.
Escape character is '^]'.
220 swiatowit.ii.uni.wroc.pl ESMTP Postfix
HELO neo.edu.pl
250 swiatowit.ii.uni.wroc.pl
MAIL FROM: stirlitz
250 2.1.0 Ok.
RCPT TO: prz@ii.uni.wroc.pl
250 2.1.5 Ok.
DATA
354 End data with <CR><LF>.<CR><LF>
to jest mail testowy wysłany z konsoli ;)

.
250 2.0.0 Ok: queued as C0E7E351A1
QUIT
221 2.0.0 Bye
Connection closed by foreign host.
prz@sunflower:~$
```

Program `telnet`

- Aby przetestować działanie różnych protokołów z wykorzystaniem gniazd można wykorzystać program **`telnet`**.
- Telnet łączy się ze wskazanym serwerem, czeka na polecenia użytkownika i zwraca odpowiedzi od serwera.
- Domyślnie telnet łączy się z portem 23, ale można mu wskazać dowolny port. Przykład:
  ```
  % telnet localhost 25
  ```
- Aby skorzystać z programu `telnet` i porozmawiać z serwerem jakieś usługi należy poznać podstawowe elementy danego protokołu.
- Można wykorzystać program `telnet` do udawania klienta (testowanie własnych serwerów).

Bezpieczne gniazda

- Istnieje możliwość szyfrowania danych przesyłanych przez sieć za pomocą gniazd używając protokołu SSL (ang. *Secure Sockets Layer*).
- Rozszerzenie zwykłych gniazd o szyfrowanie jest dostępne w pakiecie `javax.net.ssl` w postaci klas `SSLSocket` i `SSLServerSocket`.

Klasa `MulticastSocket`

- Konstruktory:

```
MulticastSocket ()
MulticastSocket (int port)
```

- Podstawowe metody:

```
void joinGroup (InetAddress ia)
void leaveGroup (InetAddress ia)
void setTimeToLive (int ttl)
int getTimeToLive ()
```

Protokół UDP w porównaniu z TCP

- Implementacja UDP w Javie składa się z dwóch klas:
  - klasa `DatagramPacket` pozwala umieścić albo odzyskać dane z pakietów UDP (czyli z datagramów);
  - klasa `DatagramSocket` wysyła i odbiera datagramy UDP.
- W UDP nie istnieje pojęcia gniazda serwera – używa się takich samych gniazd do wysyłania i odbierania datagtamów.
- W UDP nie pracuje się ze strumieniami – zawsze operuje na paczkach z danymi.
- Gnizado UDP nie jest związane z żadnym konkretnym hostem – może wysyłać i odbierać dane od wielu niezależnych hostów.

Klasa `DatagramPacket`

- Teoretycznie w datagramie UDP można zmieścić do 65507 bajtów danych, na wielu platformach jednak rzeczywistym ograniczeniem jest rozmiar 8192 bajtów ale naprawdę bezpiecznie jest wtedy, gdy rozmiar ten nie przekracza 512 bajtów.
- Datagram składa się z nagłówka IP (minimum 20 bajtów), nagłówka UDP (8 bajtów) i bloku danych (maksymalnie 65507).

Klasa `DatagramPacket`

- Konstruktory do odbierania datagramów:

```
DatagramPacket (byte[] buf,
  int length)
DatagramPacket (byte[] buf,
  int offset, int length)
```

- Przykład:

```
byte[] data = new byte[8192];
DatagramPacket dp =
  new DatagramPacket(data,data.length);
```

Klasa DatagramPacket

- Konstruktory do wysyłania datagramów:

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

- Przykład:

```
byte[] data = new byte[8192];
InetAddress ia = InetAddress.getByName("…");
int port = 7;
DatagramPacket dp =
  new DatagramPacket(data,data.length,ia,port);
```

Klasa `DatagramPacket`

- Klasa `DatagramPacket` zawiera kilka metod do odczytywania informacji z datagramów:

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

- Przykład:

```
DatagramPacket dp = …;
//…
String s = new String(dp.getData(),"utf-8");
//…
ByteArrayInputStream bis =
  new ByteArrayInputStream(
  dp.getData(),dp.getOffset(),dp.getLength());
DataInputStream dis = new DataInputStream(bis);
```

Klasa `DatagramPacket`

- Klasa `DatagramPacket` zawiera kilka metod do wpisywania danych do datagramów oraz wprowadzania zmian w nagłówku:

```
void setAddress (InetAddress ia)
void setPort (int port)
void setData (byte[] buf)
void setData (byte[] buf, int off, int len)
void setLength (int len)
```

- Przykład:

```
byte[] data = new byte[8192];
DatagramPacket dp = …;
//…
dp.setData(data);
//…
```

Klasa `DatagramSocket`

- Wszystkie gniazda datagramowe są powiązane z lokalnym portem.
- Jeśli piszesz serwer, klienty muszą wiedzieć, na którym porcie serwer czeka na przychodzące datagramy; wtedy używasz konstruktora:
  `DatagramSocket (int port)`
- Jeśli piszesz klienta, możesz użyć portu anonimowego; wtedy używasz konstruktora:
  `DatagramSocket ()`
- Aby odczytać numer portu zajętego przez gniazdo UDP należy się posłużyć metodą `getLocalPort()`.
- Aby zwolnić zajęty port UDP należy zamknąć gniazdo metodą `close()`.

Klasa `DatagramSocket`

- Po skonstruowaniu datagramu można go wysłać metodą `send`:

```
DatagramPacket dp = …;
DatagramSocket ds = …;
ds.send(dp);
```

- Metoda `receive` pozwala odczytać datagram (blokada bieżącego wątku aż do otrzymania datagramu) i po odczycie umieszcza dane w obiekcie datagramu:

```
DatagramPacket dp = …;
DatagramSocket ds = …;
ds.receive(dp);
```

Klasa `DatagramSocket`

- Połączeniami UDP można zarządzać za pomocą następujących metod:
  - `void connect (InetAddress host, int port)`
    określa wybrany host i port z którym będzie się komunikować gniazdo UDP;
  - `void disconnect ()`
    znosi ograniczenia nałożone przez metodę connect;
  - `int getPort ()`
    zwraca numer portu, do którego gniazdo jest podłączone (albo `-1`);
  - `InetAddress getInetAddress ()`
    zwraca adres hosta, do którego gniazdo jest podłączone (albo `null`).
- Opcje gniazd serwera:
  `SO_TIMEOUT` – wartość wyrażona w milisekundach określa czas akceptacji nowego połączenia (wartość 0 oznacza brak limitu).

Multicasting

- *Unicasting* zapewnia komunikację między dwoma punktami w sieci.
- *Multicasting* to transmisja grupowa realizowana przez dodatkowe protokoły warstwy aplikacji opierające się na TCP albo UDP.
- *Broadcasting* to komunikacja rozgłoszeniowa.

Multicasting

- Multicasting zaprojektowano z myślą o niewidocznym wpasowaniu go w strukturę Internetu – większość pracy wykonują rutery, programiści nie powinni mieć z nim styczności.
- Rutery gwarantują, że pakiet zostanie dostarczony wszystkim hostom w grupie multicast.

Multicasting

- Przy multicastingu w nagłówku datagramu znajduje się dodatkowe pole TTL (ang. *Time-To-Live*), które określa maksymalną liczbę ruterów, przez które może przejść pakiet.
- Adresy multicast to adresy IP z zakresu 224.0.0.0 – 239.255.255.255 (klasa D).
- Adres 224.0.0.1 jest zarezerwowany dla grupy multicast w sieci lokalnej

Multicasting

- Kiedy host chce przesłać dane do grupy multicast, umieszcza je w zwykłych datagramach UDP adresowanych do grupy multicast.
- Dane rozsyłane za pomocą multicastingu to przede wszystkim obraz lub dźwięk.

Klasa `MulticastSocket`

- Klasa `MulticastSocket` odpowiada za obsługę multicastingu w Javie.
- Gniazdo `MulticastSocket` zachowuje się podobnie do `DatagramSocket`, czyli wysyła i odbiera dane za pomocą obiektów `DatagramPacket`.

Klasa `MulticastSocket`

- Konstruktory:

```
MulticastSocket ()
MulticastSocket (int port)
```

- Podstawowe metody:

```
void joinGroup (InetAddress ia)
void leaveGroup (InetAddress ia)
void setTimeToLive (int ttl)
int getTimeToLive ()
```

Klasa `MulticastSocket`

- Odczytywanie pakietów przeznaczonych dla grupy:

```
MulticastSocket ms = new MulticastSocket(2468);
InetAddress ia =
  InetAddress.getByName("224.0.0.1");
byte[] buf = new byte[8192];
DatagramPacket dp =
  new DatagramPacket(buf,buf.length);
ms.joinGroup(ia);
while (true) {
    ms.receive(dp);
    String s =
      new String(dp.getData(),"utf-8");
    //…
}
```

Klasa `MulticastSocket`

- Wysyłanie pakietów do grupy:

```
byte[] buf = "dane multicast\r\n".getBytes();
InetAddress ia =
  InetAddress.getByName("experiment.mc.net");
int port = 2468;
DatagramPacket dp =
  new DatagramPacket(buf,buf.length,ia,port);
MulticastSocket ms = new MulticastSocket();
ms.send(dp);
//…
```

Literatura

- E.R.Harold: *Java. Programowanie sieciowe.* Wydawnictwo RM, Warszawa 2001.
- C.S.Horstmann, G.Cornell: *Java – techniki zaawansowane. Wydanie 9. Rozdział 3: Programowanie aplikacji sieciowych.* Wydawnictwo HELION, Gliwice 2013.
- Custom Networking (Java Tutorial): https://docs.oracle.com/javase/tutorial/networking/