Programowanie klient – serwer

UDP

Komunikacja UDP

Cechy:

- Brak kontroli przepływu – datagramy mogą przychodzić w nieokreślonej kolejności
- Niepewność transmisji – możliwość utraty datagramu
- Ograniczenie wielkości wiadomości – zawiera się w pojedynczym komunikacie UDP (max. ok. 60kB)
- Transmisja szybsza niż w przypadku TCP

Do obsługi komunikacji UDP służą klasy:

- **`DatagramSocket`** – gniazdo
- **`DatagramPacket`** – pakiet UDP (odpowiednik strumieni w TCP)

Komunikacja UDP

- **`DatagramSocket`** jest używane do wysyłania i odbierania **`DatagramPacket`**
- **`DatagramPacket`** jest nakładką dla tablicy bajtów, z której dane będą wysyłane lub do której dane będą przyjmowane. Obiekt tej klasy przechowuje także adres i port hosta, do którego dane będą wysłane.
- **`DatagramSocket`** jest połączeniem do portu, z którego nadawane i odbierane są przesyłki.
- Nie ma rozróżnienia między klientem a serwerem
- **`DatagramSocket`** może wysyłać do różnych odbiorców – adres odbiorcy zapisany jest w pakiecie, a nie w gnieździe.

DatagramPacket

- Nakładka dla tablicy bajtów
- Zawiera także port i adres hosta, do którego dane będą wysłane / z którego dane będą odbierane
- Do tworzenia pakietu służą konstruktory:

```
public DatagramPacket(byte[] data, int length)
public DatagramPacket(byte[] data, int length,
                      InetAddress host, int port)
```

- Pozwalają przekazać obiektowi **`DatagramPacket`** tablicę bajtów oraz liczbę bajtów, która ma być wysłana.

```
String s = "Wiadomosc do odbiorcy"
byte[] b = s.getBytes();
DatagramPacket dp = new DatagramPacket(b, b.length);
```

- Pierwszego używamy tylko wtedy, gdy gniazdo zostało przypisane do konkretnego odbiorcy metodą **`connect()`** (raczej rzadko)

DatagramPacket

Podajemy adres odbiorcy i port, na który wysyłamy:

```
try {
InetAddress host = InetAddress.getByName("192.168.0.2");
int port = 1111;
String s = "Wiadomosc do odbiorcy2"
byte[] b = s.getBytes();
DatagramPacket dp = new DatagramPacket(b, b.length, host, port);
}
catch (UnknownHostException ex) {
System.err.println(ex);
}
```

Tablica bajtów jest przekazywana do konstruktora przez referencję.

Jeśli jej zawartość zostanie zmieniona poza obiektem, to zmiany te dotyczyć będą również danych w **`DatagramPacket`**.

DatagramPacket

Obiekty są modyfikowalne, tzn. można zmienić ich parametry, takie jak dane, długość danych, port, adres:

```
public void setAddress(InetAddress host)
public void setPort(int port)
public void setData(byte buffer[])
public void setLength(int length)
```

Do odczytu właściwości można użyć metod:

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

Metody do odczytu właściwości są przydatne przy odbieraniu pakietów.

DatagramPacket – wysyłanie pakietu

- Do tworzenia datagramowych gniazd używamy konstruktorów:

```
public DatagramSocket() throws SocketException
public DatagramSocket(int port) throws SocketException
public DatagramSocket(int port, InetAddress laddr) throws
                                                SocketException
```

- Pierwszy używany jest do gniazd datagramowych działających w trybie klienta (tylko do wysyłania, z pierwszego wolnego portu)
- Dwa następne mają dodatkowo możliwość nasłuchiwania i odbierania pakietów na wskazanym porcie i opcjonalnie tylko ze wskazanego interfejsu. Wysyłanie takim gniazdem następuje z podanego portu.
- Tak jak w TCP, tylko jeden proces może nasłuchiwać na danym porcie (porty TCP i UDP są rozdzielne)

DatagramPacket – wysyłanie pakietu

1. Konwersja danych na tablicę bajtów.
2. Wywołanie konstruktora **`DatagramPacket`** – przekazujemy tablicę bajtów, długość danych, **`InetAddress`** z numerem portu, do którego będą wysyłane dane.
3. Wywołanie metody **`send()`** dla obiektu **`DatagramSocket:`**

```
try {
InetAddress host = InetAddress.getByName("192.168.0.2");
int port = 1111;
String s = "Wiadomosc do odbiorcy2"
byte[] b = s.getBytes();
DatagramPacket dp = new DatagramPacket(b, b.length, host, port);
DatagramSocket sender = new DatagramSocket();
sender.send(dp);
}
catch (UnknownHostException ex) { System.err.println(ex); }
catch (IOException e) { System.out.println(e); }
```

DatagramPacket – odbieranie danych

1. Tworzymy obiekt klasy **`DatagramSocket`** przekazując co najmniej numer portu, na którym będzie prowadzony nasłuch.
2. Przekazujemy pusty obiekt **`DatagramPacket`** do metody **`receive()`** gniazda **`DatagramSocket`**:

   `public void receive(DatagramPacket dp) throws IOException`

   Wątek wywołujący tą metodę zablokuje się, aż do chwili otrzymania pakietu.
3. Po otrzymaniu pakietu obiekt **`DatagramPacket`** wypełniony zostanie danymi (w tym nazwą hosta i portem nadawcy):

```
try{
DatagramSocket ds = new DatagramSocket(1111);
byte buffer[] = new byte[10];
DatagramPacket dp = new DatagramPacket(buffer,buffer.length);
ds.receive(dp);
byte[] data = dp.getData();
String s = new String(data, 0, data.length);
System.out.println(s);
System.out.println(dp.getPort());
}
catch(IOException e) { System.out.println(e); }
```

UDP – gniazda

- Istnieje możliwość przywiązania gniazda UDP do hosta i portu zdalnego za pomocą metody klasy **`DatagramSocket`**:

  `public void` **`connect(InetAddress address, int port)`**
- Użycie metody **`connect`** nie powoduje łączenia ze zdalnym hostem. Dzięki niej gniazdo ma możliwość wysyłania i odbierania pakietów UDP do/z podanego zdalnego hosta i portu.
- Używamy wówczas konstruktora pakietu:

  `public` **`DatagramPacket(byte[] data, int length)`**
- Próba wykorzystania drugiego (przekazując inny host lub port od tego związanego funkcją **`connect()`**) kończy się niepowodzeniem.

```
try{ DatagramSocket ds = new DatagramSocket(1234);
String s = "Komunikat" ;
byte[] b = s.getBytes();
InetAddress host = InetAddress.getByName("192.168.66.5");
ds.connect(host,1111);
DatagramPacket dp = new DatagramPacket(b, b.length);
ds.send(dp);
}
catch(IOException e){ System.out.println(e); }
```

UDP – rozgłaszanie

Jeśli chcemy wysłać pakiet do wszystkich hostów w sieci, adresujemy obiekt **`DatagramPacket`** adresem rozgłoszeniowym:

```
try{
DatagramSocket ds = new DatagramSocket();
String s = "Do wszystkich!!" ;
byte[] b = s.getBytes();
InetAddress host = InetAddress.getByName("192.168.0.255");
DatagramPacket dp = new DatagramPacket(b, b.length, host,
1111);
ds.send(dp);
}
catch(IOException e)
{
System.out.println(e);
}
```

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 często obrazy 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);
//…
```

Rozgłaszanie grupowe

- Do rozgłaszania grupowego korzystamy z klasy **`MulticastSocket`**, która jest pochodną **`DatagramSocket`.**
- Zdefiniowane są tu dodatkowe metody:

```
public void joinGroup(InetAddress mcastaddr) throws IOException
public void leaveGroup(InetAddress mcastaddr) throws IOException
```

- Służą one do zapisywania i wypisywania z grupy.
- W parametrze przekazujemy obiekt InetAddress reprezentujący adres grupy (adres klasy D)
- Jeśli następnie wyślemy pakiet na ten adres dotrze on do wszystkich hostów zapisanych:

```
String msg = "Hello";
InetAddress group = InetAddress.getByName("228.5.6.7");
MulticastSocket s = new MulticastSocket(6789);
s.joinGroup(group);
DatagramPacket hi = new DatagramPacket(msg.getBytes(), msg.length(), group, 6789);
s.send(hi);
```