Разработка клиент-серверных приложений с использованием UDP соединений.

**Цель:** разработка клиент-серверных приложений с использованием протокола UDP*.*

**Протокол пользовательских дейтаграмм (UDP)**

Протокол UDP намного проще, чем ТСР; он полезен в ситуациях, когда мощные механизмы обеспечения надежности протокола ТСР не обязательны. Заголовок UDP имеет всего четыре поля:

* + поле порта источника (source port),
  + поле порта пункта назначения (destination port),
  + поле длины (length)
  + поле контрольной суммы UDP (checksum UDP).

Поля порта источника и порта назначения выполняют те же функции, что и в заголовке ТСР. Поле длины обозначает длину заголовка UDP и данных; поле контрольной суммы обеспечивает проверку целостности пакета. Контрольная сумма UDP является факультативной возможностью.

**Дейтаграмма**

* Дейтаграмма- это пакет, передаваемый через сеть независимо от других пакетов без установления логического соединения и подтверждения приема. Дейтаграмма - совершенно самостоятельный пакет, поскольку сама содержит всю необходимую для ее передачи информацию. Ее передача происходит безо всякого предварения и подготовки.

Дейтаграммы, сами по себе, не содержат средств обнаружения и исправления ошибок передачи, поэтому при передаче данных с их помощью следует принимать меры по обеспечению надежности пересылки информации. Методы организации надежности могут быть самыми разными, обычно же используется метод подтверждения приема посылкой эхо-отклика при получении каждого пакета с дейтаграммой.

**Использование UDP**

В отличие от TCP, данные, отправляемые прикладным процессом через модуль UDP, достигают места назначения как единое целое. Так же, в отличие от UDP, TCP получает и обрабатывает данные в том порядке, в котором они пришли. Размер каждого записанного сообщения будет совпадать с размером соответствующего прочитанного. Протокол UDP сохраняет границы сообщений, определяемые прикладным процессом. Он никогда не объединяет несколько сообщений в одно целое и не делит одно сообщение на части.

* Если нужна надежная доставка, то лучше использовать TCP.
* Если нужна эффективность на быстрых сетях с короткими соединениями, то - UDP.
* UDP предоставляет ненадёжный сервис, и дейтаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа.
* Нет гарантии, что при отправке дейтаграммUDP она будет доставлена, и доставлена правильно. Если вы решили применить в своей программе дейтаграммы, вам придется иметь дело с утерянными или неупорядоченными пакетами.
* Если два сообщения отправлены одному получателю, то порядок их достижения цели не может быть предугадан.
* Контрольная сумма UDP охватывает UDP заголовок и UDP данные. Для UDP контрольная сумма необязательна, но для TCP она обязательна.
* Прикладные программы, конечно, могут устранять некоторые недостатки выбранного протокола. Например, если вы выбрали UDP, а вам необходима надежность, то прикладная программа должна обеспечить надежность сама, как описано выше: требовать подтверждения, пересылки утерянных или увечных пакетов и т.д. Если вы выбрали TCP, а вам нужно передавать записи, то прикладная программа должна вставлять метки в поток байтов так, чтобы можно было различить записи.

**Свойства протокола UDP**

* Ненадежный – UDP не имеет ни встроенного механизма обнаружения ошибок, ни средств повторной пересылки поврежденных или потерянных данных
* Как следствие ненадежности в UDP происходит отправка дейтаграмм без установления логического соединения – перед пересылкой данных UDP не устанавливает логического соединения. Информация пересылается в предположении, что принимающая сторона ее ожидает.
* Основанный на сообщениях – UDP позволяет приложениям пересылать информацию в виде сообщений, передаваемых посредством дейтаграмм, которые являются единицами передачи данных в UDP. Приложение должно самостоятельно распределить данные по отдельным дейтаграммам.

**UDP-сокеты в Java**

Java обеспечивает работу с дейтаграммами благодаря использованию классов DatagramSocket и DatagramPacket пакета java.net.

* С помощью объекта DatagramSocketв Java-программе создается сокет для обеспечения сетевого соединения и пересылки дейтаграмм.
* С другой стороны, сама дейтаграмма представляет собой объект класса DatagramPacket и используется группой методов класса DatagramSocket.

**Создание соединения**

Для того, чтобы создать новое соединение для пересылки дейтаграмм, следует объявить экземпляр класса DatagramSocket, используя один из следующих его конструкторов:

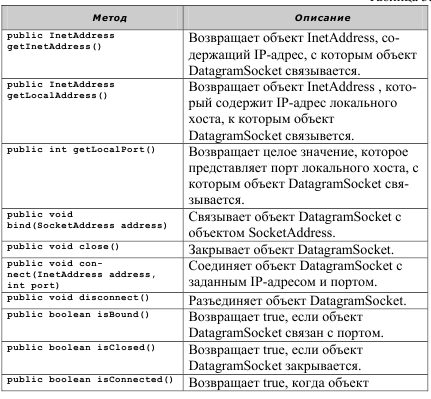
* + DatagramSocket();
  + DatagramSocket(int port);
  + DatagramSocket(int port, InetAddress laddr);

Где port – номер порта, laddr-адрес подключения.

**Методы класса DatagramSocket**

Для экземпляров класса DatagramSocket определена группа методов, позволяющих как управлять процессами сетевого соединения, так и осуществлять его информационную поддержку. Объекты DatagramPacket отправляют и принимают сохраненные данные, используя объект DatagramSocket.

В таблице представлены методы класса DatagramSocket, которые используются для получения информации из объекта DatagramSocket.

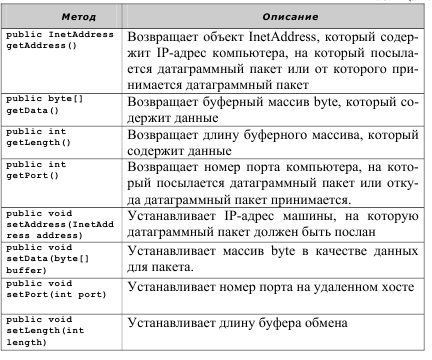


**Отправка и получение пакетов DatagramPacket**

После установки соединения, используя методы send() и receive(), можно выполнить соответственно отправку и получение пакетов DatagramPacket. Однако предварительно необходимо будет объявить экземпляры класса DatagramPacket виду того, что методы send() и receive() в качестве параметра принимают объект данного класса. В связи с этим для создания нового экземпляра класса DatagramPacket необходимо будет воспользоваться следующим из его конструкторов:

* DatagramPacket (byte[] buf, int length, InetAddress, int port), где buf – массив байт, представляющий пакет дейтаграммы, length – размер данного пакета, address-Internet-адрес отправки, port- порт компьютера, на который отправляется дейтаграмма.
* DatagramPacket(byte[] buffer, int buffer\_length): Создает объект DatagramPacket, который принимает и сохраняет данные в массиве byte. Длина буфера массива byte задается вторым параметром buffer\_length.

**Методы DatagramPacket**



В процессе работы с объектами дейтаграмм могут возникать исключительные ситуации, для чего нужно реализовать соответствующую обработку следующих исключений:

* SocketException – ошибка протокола при обращении к сокету
* SecurityException – ошибка доступа к системе безопасности
* UnknownHostException – неправильно указан адрес InetAddress

При завершении работы с сетевыми соединениями DatagramSocket его следует закрыть, используя метод close() данного класса.

**Пример создания дейтаграмм**

Рассмотрим пример, иллюстрирующий использование дейтаграмм. В представленных программах одно приложение выступает в роли сервера и ожидает поступления на порт 8008 пакетов дейтаграмм. Если таковые были получены, то выполняется их преобразование в строковый эквивалент и полученный таким образом текст печатается на экран.

Приложение-сервер опрашивает порт 8008 компьютера для получения дейтаграммы. После этого содержимое дейтаграммы выводится на экран.

**Создание сервера UDP**

Для работы с этим протоколом и на стороне клиента, и на стороне сервера используется класс DatagramSocket**.**

Сервер UDP представляет собой сетевое приложение, использующее протокол UDP для обслуживания запросов клиентских приложений. Для создания сервера UDP используется объект DatagramSocket, который принимает объекты DatagramPacket от клиентов.

Для создания сервера UDP необходимо выполнить следующие шаги:

* Создать сокет, используя объект DatagramSocket.
* Создать объект класса DatagramPacket и использовать метод receive() для получения сообщения клиента.
* Создать объект класса DatagramPacket и использовать метод send() для отправки сообщение клиенту.
* Запустить сервер, вызывая конструктор класса сервера UDP в методе main().

**Программа-сервер**

import java.net.\*;

public class DatagramServer extends Thread

{

byte[] buf = new byte[100];

int port=8008;

public void run()

{

try

{ // определяем локальный адрес

InetAddress local=InetAddress.getByName ("127.0.0.1");

// определяем сокет дейтаграммы

DatagramSocket srvsocket=new DatagramSocket (port,local);

// определяем пакет дейтаграммы

DatagramPacket pack=new DatagramPacket (buf,buf.length);

while (true)

{ //получаем дейтаграмму

srvsocket.receive(pack);

// конвертируем ее в строку

String str =new String (pack.getData(),0,pack.getLength());

System.out.println (str);

}

}

catch (Exception e)

{

System.out.println (e.getMessage());

}

}

public static void main(String arg[])

{

System.out.println("Datagram server running");

DatagramServer my=new DatagramServer();

my.start();

}

}

**Создание клиента UDP**

Клиент UDP представляет собой сетевое приложение, которое использует протокол UDP для отправки запросов на сервер и получения ответов от серверного приложения. В клиентском UDP-приложении, необходимо создать объект класса DatagramSocket, который принимает сообщения от сервера UDP, для чего необходимо выполнить следующие шаги:

* Создать сокет, использующий объект класса DatagramSocket для установки соединения с сервером
* Создать объект класса DatagramPacket и использовать метод send() для отправки сообщения на сервер
* Создать объект класса DatagramPacket и использовать метод receive() для получения сообщений, отправленных сервером.

**Программа-клиент**

import java.net.\*;

public class DatagramClient

{

public static void main(String arg[])

{

// объявляем массив байт для пакета дейтаграммы

byte[] buf=new byte[100];

//инициализация порта

int port=8008;

int cur=0;

int ch;

boolean flag=true;

while (flag)

{

try

{

// получаем символ с клавиатуры

ch=System.in.read();

// определяем локальный адрес

InetAddress local=InetAddress.getLocalHost();

// записываем символ в массив

buf[cur]=(byte)ch;

if (ch==-1) flag=false;

if (ch=='\n')

{ // определяем сокет дейтаграмм

DatagramSocket srvsocket=new DatagramSocket();

// определяем пакет дейтаграммы для отправки по адресу local и порту port

DatagramPacket pack=new DatagramPacket(buf,buf.length, local, port);

// посылаем пакет через сокет

srvsocket.send(pack);

cur=-1;

for (int i=0; i<buf.length;i++) buf[i]=0;

}

cur++;

}

catch(Exception e)

{

System.out.println(e.getMessage());

break;

}

}

}

}

**Задания для самостоятельного выполнения.**

Разработать приложение архитектуры «клиент сервер» на основе UDP-соединения, позволяющее осуществлять взаимодействие клиента и сервера по совместному решению задач обработки информации. Интерфейс клиента и сервера должен быть реализован, как GUI интерфейс на базе JavaFX. Приложение должно располагать возможностью передачи и модифицирования получаемых (передаваемых) данных. Возможности клиента: передать серверу исходные параметры (число а=2, число b=4 и число с=7 вводятся с клавиатуры) для расчета значения функции, а также получить расчетное значение функции. Возможности сервера: по полученным от клиента исходным параметрам рассчитать сумму первого ряда от а до b и сумму второго ряда от b до с. Расчет первой и второй сумм осуществляется в разных потоках.

**СЕРВЕР (приведена реализация БЕЗ JavaFX, ее необходимо реализовать самостоятельно):**

import java.oi.\*;

import java.net.\*;

public class Servermain

{

public static void main(String[] args) throws IOException, InterruptedException {

new Servermain();

}

double sum1=0,sum2=0;

DatagramSocket st;

byte [] buf=new byte[100];

DatagramPacket dp=new DatagramPacket(buf,100);

Servermain()throws IOException, InterruptedException{

st=new DatagramSocket(12345);

listen();

}

private void listen() throws IOException, InterruptedException

{

int a,b,c;

st.receive(dp);

String str=new String(dp.getData());

str=str.substring(0, str.indexOf('\n'));

System.out.println("number "+str+" recieved as a");

a=Integer.parseInt(str);

st.receive(dp);

str=new String(dp.getData());

str=str.substring(0, str.indexOf('\n'));

System.out.println("number "+str+" recieved as b");

b=Integer.parseInt(str);

st.receive(dp);

str=new String(dp.getData());

str=str.substring(0, str.indexOf('\n'));

System.out.println("number "+str+" recieved as c");

c=Integer.parseInt(str);

Thread t1=new Thread(new Runnable()

{

public void run()

{

for(int i=a;i<b;i++)

{

sum1+=(i-1)\*(i-1);

}

}

})

,t2=new Thread(new Runnable()

{

public void run()

{

for(int i=b;i<c;i++)

{

sum1+=2\*i/(7\*i+1);

}

}

});

t1.start();

t2.start();

t1.join();

t2.join();

sendBack();

}

private void sendBack() throws IOException

{

String str=String.valueOf(sum1-sum2);

byte [] send=str.getBytes();

dp=new DatagramPacket(send, send.length, InetAddress.getByName("localhost"),12346 );

st.send(dp);

st.close();

}

}

**КЛИЕНТ (приведена реализация БЕЗ JavaFX, ее необходимо реализовать самостоятельно):**

import java.oi.\*;

import java.net.\*;

public class Client {

public static void main(String[] args) throws IOException {

DatagramSocket st=new DatagramSocket(12346);

DatagramPacket dp = null;

InetAddress loc=InetAddress.getByName("localhost");

byte [] buf=new byte[100];

Scanner in = new Scanner(System.in);

int arr[] = new int [3];

for (int i = 0; i < arr.length; i++)

{

System.out.print("Введите элемент arr["+ i +"]:");

int k=System.in.read(buf);

dp=new DatagramPacket(buf, k, loc,12345 );

st.send(dp);

}

dp=new DatagramPacket(buf,100);

st.receive(dp);

System.out.println("the answer is "+new String(dp.getData()));

st.close();

}

}

**Варианты для выполнения задания.**