Java语言与系统设计

第12讲 网络编程

- 网络编程概述
- Socket套接字
- TCP网络编程
- UDP网络编程

1. Java网络编程概述

- Java是Internet 上的语言，它从语言级上提供了对网络应用程序的支持，程序员能够很容易开发常见的网络应用程序。
- Java提供的网络类库，可以方便调用其网络操作的方法。
- Java 实现了一个跨平台的网络库，程序员面对的是一个统一的网络编程环境。

1. Java网络编程概述

- IP 地址：标识Internet 上的计算机（通信实体）
- Internet还可以用域名(hostName)标识主机。
  - 域名容易记忆，域名服务器(DNS)负责将域名转化成IP地址，这样才能和主机建立连接。-------域名解析
- InetAddress类提供了如下几个方法来获取IP地址对象及其属性
  - public static InetAddress getByName(String host)： 获取IP地址
  - public static InetAddress getLocalHost()： 获取本机地址
  - public String getHostAddress()： 返回IP 地址字符串
  - public String getHostName()： 获取此IP 地址的主机名

```
import java.net.*;
public class IpTest{
    public static void main(String args[]){
  try{
  InetAddress inet1 = InetAddress.getByName("192.168.0.1");
  System.out.println(inet1);
  InetAddress inet2 = InetAddress.getByName("www.csu.edu.cn");
  System.out.println(inet2);
  InetAddress inet3 = InetAddress.getLocalHost();
  System.out.println(inet3);
  System.out.println(inet2.getHostAddress());
  System.out.println(inet2.getHostName());
    }catch(UnknownHostException e) {}
    }
}
```

1. Java网络编程概述

- 端口号：标识正在计算机上运行的进程（程序）
- 公认端口：0~1023。被预先定义的服务通信占用（如：HTTP占用端口80，FTP占用端口21，Telnet占用端口23）
- 注册端口：1024~65535。分配给用户进程或应用程序。（如：Tomcat占用端口8080，MySQL占用端口3306，Oracle占用端口1521等）。

端口号与IP地址的组合得出一个网络套接字：Socket

2. Socket套接字

- Socket原理
- 利用套接字(Socket)开发网络应用程序早已被广泛的采用，以至于成为事实上的标准。
- 网络上具有唯一标识的IP地址和端口号组合在一起才能构成唯一能识别的标识符套接字。通信的两端都要有Socket，是两台机器间通信的端点。网络通信其实就是Socket间的通信。
- Socket允许程序把网络连接当成一个流，数据在两个Socket间通过IO传输。一般主动发起通信的应用程序属客户端，等待通信请求的为服务端。

2. Socket套接字

- Socket分类：
- 流套接字（stream socket）：使用TCP提供可靠的字节流服务
- 数据报套接字（datagram socket）：使用UDP提供“尽力而为”的数据报服务
- Socket类的常用构造器：
- public Socket(InetAddress address, int port)创建一个流套接字并将其连接到指定IP地址的指定端口号。
- public Socket(String host, int port)创建一个流套接字并将其连接到指定主机上的指定端口号。

2. Socket套接字

- Socket类的常用方法：
- public InputStream getInputStream()返回此套接字的输入流。可以用于接收网络消息
- public OutputStream getOutputStream()返回此套接字的输出流。可以用于发送网络消息
- public InetAddress getInetAddress()此套接字连接到的远程IP地址；如果套接字是未连接的，则返回null。
- public InetAddress get LocalAddress()获取套接字绑定的本地地址。

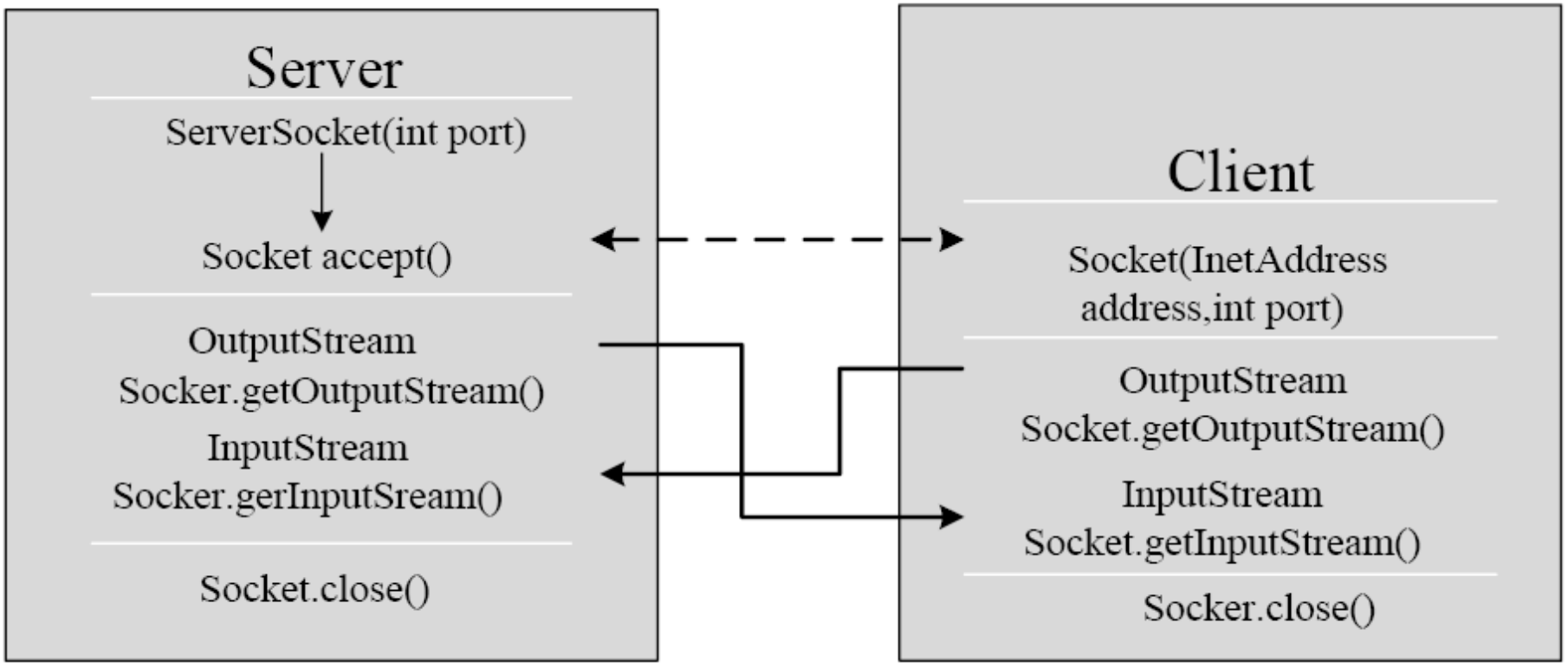
2. Socket套接字

- Socket类的常用方法：
- public int getPort()此套接字连接到的远程端口号；如果尚未连接套接字，则返回0。
- public int getLocalPort()返回此套接字绑定到的本地端口。如果尚未绑定套接字，则返回-1。即本端的端口号。
- public void close()关闭此套接字。套接字被关闭后，便不可在以后的网络连接中使用（即无法重新连接或重新绑定）需要创建新的套接字对象。关闭此套接字也将会关闭该套接字的InputStream和OutputStream。

3. TCP网络编程

- TCP的网络编程分为服务端套接字编程和客户端套接字编程

基于Socket的TCP通信模型



Server

```
import java.io.*;
import java.net.*;
public class MyServer{
   public static void main(String[] args) throws IOException{
        System.out.println("This is server");
        ServerSocket server=new ServerSocket(1111);
        Socket client=server.accept();
        InputStream is=client.getInputStream();
        InputStreamReader isr=new InputStreamReader(is);
        BufferedReader in=new BufferedReader(isr);
        OutputStream os= client.getOutputStream();
        PrintWriter out=new PrintWriter(os);
       while(true){
         String str=in.readLine();
         System.out.println(str);
         out.println("Server received "+ str);
         out.flush();
         if(str.equals("end")) break;
       }
       client.close();
       server.close();
    }
 }
```

Client

```
import java.net.*;
import  java.io.*;
public class MyClient{
    public static void main(String[] args) throws Exception{
      System.out.println("This is client");
      Socket server=new Socket("127.0.0.1",1111);
      InputStream is = server.getInputStream();
      InputStreamReader isr = new InputStreamReader(is);
      BufferedReader in=new BufferedReader(isr);
      OutputStream os = server.getOutputStream();
      PrintWriter out=new PrintWriter(os);
      InputStreamReader sysin = new InputStreamReader(System.in);
      BufferedReader wt=new BufferedReader(sysin);
      while(true){
        String str=wt.readLine();
         out.println(str);
         out.flush();
       if(str.equals("end")) break;
      System.out.println( in.readLine());
       }
     server.close();
     }
 }
```

3. TCP网络编程

- 服务器程序的工作过程包含以下四个基本的步骤：
- 调用ServerSocket(int port)：创建一个服务器端套接字，并绑定到指定端口上。用于监听客户端的请求。
- 调用accept()：监听连接请求，如果客户端请求连接，则接受连接，返回通信套接字对象。
- 调用该Socket类对象的getOutputStream() 和getInputStream()：获取输出流和输入流，开始网络数据的发送和接收。
- 关闭ServerSocket和Socket对象：客户端访问结束，关闭通信套接字。

3. TCP网络编程

- 服务器建立ServerSocket对象
- ServerSocket对象负责等待客户端请求建立套接字连接，类似邮局某个窗口中的业务员。也就是说，服务器必须事先建立一个等待客户请求建立套接字连接的ServerSocket对象。
- 所谓“接收”客户的套接字请求，就是accept()方法会返回一个Socket 对象

3. TCP网络编程

- 客户端Socket的工作过程包含以下四个基本的步骤：
- 创建Socket：根据指定服务端的IP 地址或端口号构造Socket 类对象。若服务器端响应，则建立客户端到服务器的通信线路。若连接失败，会出现异常。
- 打开连接到Socket 的输入/出流：使用getInputStream()方法获得输入流，使用getOutputStream()方法获得输出流，进行数据传输
- 按照一定的协议对Socket 进行读/写操作：通过输入流读取服务器放入线路的信息（但不能读取自己放入线路的信息），通过输出流写入信息。
- 关闭Socket：断开客户端到服务器的连接，释放线路

3. TCP网络编程

- 客户端创建Socket对象
- 客户端程序可以使用Socket类创建对象，创建的同时会自动向服务器方发起连接。
- Socket的构造器是：
- Socket(String host, int port)throws UnknownHostException, IOException：向服务器(域名是host，端口号为port)发起TCP连接，若成功，则创建Socket对象，否则抛出异常。
- Socket(InetAddress address, int port)throws IOException：根据InetAddress对象所表示的P地址以及端口号port发起连接。

4. UDP网络编程

- 类DatagramSocket和DatagramPacket实现了UDP 协议网络程序。
- UDP数据报通过数据报套接字DatagramSocket发送和接收，系统不保证UDP数据报一定能够安全送到目的地，也不能确定什么时候可以抵达。
- DatagramPacket 对象封装了UDP数据报，在数据报中包含了发送端的IP地址和端口号以及接收端的IP地址和端口号。
- UDP协议中每个数据报都给出了完整的地址信息，因此无须建立发送方和接收方的连接。如同发快递包裹一样。

4. UDP网络编程

- DatagramSocket 类的常用方法
- public DatagramSocket(int port)创建数据报套接字并将其绑定到本地主机上的指定端口。
- public void close()关闭此数据报套接字。
- public void send(DatagramPacket p)从此套接字发送数据报包。DatagramPacket包含的信息指示：将要发送的数据、其长度、远程主机的IP地址和远程主机的端口号。
- public void receive(DatagramPacket p)从此套接字接收数据报包。当此方法返回时，DatagramPacket的缓冲区填充了接收的数据。数据报包也包含发送方的IP地址和发送方机器上的端口号。此方法在接收到数据报前一直阻塞。

4. UDP网络编程

- DatagramPacket类的常用方法
- public DatagramPacket(byte[] buf, int length)构造DatagramPacket，用来接收长度为length的数据包。length参数必须小于等于buf.length。
- public DatagramPacket(byte[] buf, int length, InetAddress address, int port)构造数据报包，用来将长度为length的包发送到指定主机上的指定端口号。

4. UDP网络编程

- 发送端

```
DatagramSocket ds = null;
try {
    ds = new DatagramSocket();
    byte[] by = "hello,atguigu.com".getBytes();
    DatagramPacket dp = new DatagramPacket(by, 0, by.length,
InetAddress.getByName("127.0.0.1"), 10000);
    ds.send(dp);
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (ds != null)
        ds.close();
}
```

4. UDP网络编程

- 接收端

```
DatagramSocket ds = null;
try {
    ds = new DatagramSocket(10000);
    byte[] by = new byte[1024];
    DatagramPacket dp = new DatagramPacket(by, by.length);
    ds.receive(dp);
    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str + "--" + dp.getAddress());
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (ds != null)
        ds.close();
}
```