# 1.套接字(Socket)

## 1.1基本概念

套接字基于网络进行数据传输的API。实际上就是基于网络的流。

套接字（socket）是通信的基石，是支持TCP/IP协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：连接使用的协议，本地主机的IP地址，本地进程的协议端口，远地主机的IP地址，远地进程的协议端口。

## 1.2网络模型

通信协议划分为七层：物理层（Physics Layer）、数据链路层（Data Link Layer）、网络层（Network Layer）、传输层（Transport Layer）、会话层（Session Layer）、表示层（Presentation Layer）、应用层（Application Layer）。

会表应传网数物。

1.2.1网络层

通过IP地址标记网络设备。

1.2.2 IP协议

IPv4用四组数来标记一个IP地址，每组数的取值范围是0-255（232），将近能标记43亿个地址。

127.0.0.1 指向本机。

255.255.255.255 广播地址。

IPv6用了六组十六进制数来表示一个IP地址。

1.2.3端口

端口是计算机与外部来交换信息的媒介。

端口号：范围0-65535。0-1024被计算机占用。

1.2.4域名

二级域名：baidu.com

1.2.5 DNS解析服务器

将域名解析为对应的IP地址。

localhost 指向本机。

1.2.6传输层

UDP/TCP

1.2.7表示层

HTTP、POP3、FTP、

## 1.3套接字类

此类表示不带任何协议附件的 Socket Address。作为一个抽象类，应通过特定的、协议相关的实现为其创建子类。 它提供不可变对象，供套接字用于绑定、连接或用作返回值。

1.3.1 InetSocketAddress class

此类继承了SocketAddress class。

此类实现 IP 套接字地址（IP 地址 + 端口号）。它还可以是一个对（主机名 + 端口号），在此情况下，将尝试解析主机名。如果解析失败，则该地址将被视为未解析 地址，但是其在某些情形下仍然可以使用，比如通过代理连接。

它提供不可变对象，供套接字用于绑定、连接或用作返回值。

通配符 是一个特殊的本地 IP 地址。它通常表示“任何”，只能用于 bind 操作。

**构造函数**

InetSocketAddress(InetAddress addr,int port);

根据 IP 地址和端口号创建套接字地址。

InetSocketAddress(int port);

创建套接字地址，其中 IP 地址为通配符地址，端口号为指定值。

InetSocketAddress(String hostname,int Port);

根据主机名和端口号创建套接字地址。

**重要方法**

getAddress();

获取IP地址。

getHostName();

获取主机名或者域名。

getPort();

获取端口号。

toString();

## 1.4 UDP

基于流的。不建立连接，不可靠。数据传输的速度比较高，会对数据进行封包操作，每个包不超过64k。适用于相对速度要求比较高而对安全性要求比较低的场景，比如视频聊天。

先启动接收端，后启动发送端。

1.4.1实现类

DatagramSocket

此类表示用来发送和接收数据报包的套接字。

在DatagramSocket上总是启用UDP广播发送。为了接受广播包，应该将DatagramSocket绑定到通配符地址。在某些实现中，将DatagramSocket绑定到一个更加具体的地址时广播包也可以被接收。

**构造函数**

DatagramSocket();

构造数据报套接字并将其绑定到本地主机上任何可用的端口。

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。

**重要方法**

receive();

接收数据。

send();

发送数据。

DatagramPacket

此类表示数据包。

**构造函数**

DatagramPacket(byte[] buf,int length,InetSocketAddress itsa);

第一个参数表示实际数据，第二个参数表示数据的大小，第三个参数表示接收端的地址。

DatagramPacket(byte[] buf,int length);

第一个参数是一个字节数组，用于存放数据；第二个参数表示所能存储的最大长度。

**重要方法**

getData();

获取字节数组。

getLength();

获取数据的实际大小。

getAddress();

获取发送端地址。

getPort();

获取发送端的端口号。

1.4.2发送端：

第一步：创建套接字对象。

第二步：准备数据包：实际数据，指定数据的实际大小，指定接收端地址。

第三步：发送数据包。

第四步：关流。

代码实现

public class UDPSenderDemo {

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

// 创建套接字对象

DatagramSocket ds = new DatagramSocket();

// 准备数据包

// 第一个参数表示的是实际数据--- 你好~~~

// 第二个参数表示的是数据的大小

// 第三个参数表示的是要发往的地址

// 127.0.0.1---指向本机

// DNS服务器会自动的将localhost解析为127.0.0.1

DatagramPacket dp = new DatagramPacket("你好~~~".getBytes(), "你好~~~".getBytes().length,

new InetSocketAddress("localhost", 9999));

// 发送数据

ds.send(dp);

// 关流

ds.close();

}

}

1.4.3接收端

第一步：创建套接字对象，绑定端口号：需要和发送的时候指定的端口号一致。

第二步：准备数据包：指定一个字节数组作为实际存储数据的容器，指定这个容器所能使用的大小。

第三步：接收数据包。

第四步：关流。

解析数据：获取盛放数据的数组以及数据的实际大小。

public class UDPREceiverDemo {

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

// 创建套接字对象,绑定端口号

DatagramSocket ds = new DatagramSocket(9999);

// 准备一个数据包

// 第一个参数是一个字节数组，用于存放数据

// 第二个参数表示所能存储的最大长度

DatagramPacket dp = new DatagramPacket(new byte[1024], 1024);

// 接收数据

// 阻塞

ds.receive(dp);

// 关流

ds.close();

// 解析数据

// 获取字节数组

byte[] data = dp.getData();

// 获取数据的实际大小

int len = dp.getLength();

System.out.println(new String(data, 0, len));

// 获取发送端的地址

System.out.println(dp.getAddress());

// 获取发送用的端口号

System.out.println(dp.getPort());

}

}

1.4.4注意

如果是写在两个Java文件中，要先启动接收端，后启动发送端。

在接收端报错有可能是发送端出问题。

接收端receive();方法会产生阻塞。

发送端send();方法会产生阻塞。

## 1.5 UDP练习

1.5.1单人聊天

用一个线程作为发送端，一个线程作为接收端。

public class UDPChatDemo {

public static void main(String[] args) {

new Thread(new Sender()).start();

new Thread(new Receiver()).start();

}

}

class Sender implements Runnable {

@SuppressWarnings("resource")

@Override

public void run() {

try {

// 创建套接字对象

DatagramSocket ds = new DatagramSocket();

Scanner s = new Scanner(System.in);

while (true) {

// 从控制台获取数据

String str = s.nextLine();

// 将数据封包

DatagramPacket dp = new DatagramPacket(str.getBytes(), str.getBytes().length,

new InetSocketAddress("255.255.255.255", 9999));

// 发送数据包

ds.send(dp);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

class Receiver implements Runnable {

@SuppressWarnings("resource")

public void run() {

try {

// 创建套接字对象，绑定端口号

DatagramSocket ds = new DatagramSocket(9999);

// 准备数据包

DatagramPacket dp = new DatagramPacket(new byte[1024], 1024);

while (true) {

// 接收数据

ds.receive(dp);

System.out.println(dp.getAddress() + ":");

// 解析数据

System.out.println(new String(dp.getData(), 0, dp.getLength()));

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

## 1.6 TCP

基于流的。建立连接，经过三次握手，可靠。传输速度相对较低，理论上不对数据的大小进行限制。分为客户端和服务器端。

1.6.1 客户端（Client）

第一步：创建客户端套接字（Socket）对象。

第二步：向服务器端发起连接，绑定连接地址。这一步产生阻塞。

第三步：获取输出流，然后利用输出流来写出数据。

第四步：通知服务器端数据已经写出完毕。

第五步：关流

public class TCPClientDemo {

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

// 创建客户端的套接字对象

Socket s = new Socket();

// 发起连接

// 阻塞

s.connect(new InetSocketAddress("localhost", 8090));

// 获取输出流

OutputStream out = s.getOutputStream();

// 写出数据

out.write("abc".getBytes());

// 通知服务器端数据已经写出完毕

s.shutdownOutput();

// 关流

s.close();

}

}

**Socket class**

实现客户端的套接字。

**构造方法**

Socket();

**重要方法**

connect();

发起连接。

getOutputStream();

获取输出流。

shutdownOutput();

通知服务器数据传输完毕。

shutdownInput();

通知客户端数据接受完毕。

getInputStream();

获取输入流。

1.6.2 服务器端（Server）

第一步：创建服务器端套接字（ServerSocket）对象,绑定端口。

第二步：接收连接，获取一个Socket对象。这一步会产生阻塞。

第三步：获取输入流，然后利用输入流来读取数据。

第四步：通知客户端数据已经读取完毕。

第五步：关流。

public class TCPServerDemo {

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

// 创建服务器端的套接字对象，绑定端口

ServerSocket ss = new ServerSocket(8090);

// 接收连接

// 阻塞

Socket s = ss.accept();

System.out.println("连接成功~~~");

// 获取输入流

InputStream in = s.getInputStream();

// 读取数据

byte[] bs = new byte[1024];

int len = -1;

while ((len = in.read(bs)) != -1) {

System.out.println(new String(bs, 0, len));

}

// 通知客户端数据已经读取完毕

s.shutdownInput();

// 关流

ss.close();

}

}

**ServerSocket class**

实现服务器端的套接字。

**构造方法**

ServerSocket();

**重要方法**

accept();

接收连接。

**注意**

receive/connect/accept/read/write这些方法会产生阻塞。

**扩展**

BIO

BlockingIO同步式阻塞式。

NIO

NewIO同步式非阻塞式IO，JDK1.4出现的。别名：NonBlockingIO。

AIO

AsynchronousIO异步式非阻塞式IO，JDK1.8出现的。

## 1.7 练习

1.7.1 传输文件

将D:\\a目录作为服务器端存储目录。

public class FileClientDemo {

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

// 创建Socket对象

Socket s = new Socket();

// 发起连接

s.connect(new InetSocketAddress("localhost", 8090));

// 获取输出流

OutputStream out = s.getOutputStream();

// 创建File对象代表要传输的文件

File file = new File("F:\\java基础增强.txt");

// 记录文件名

byte[] name = file.getName().getBytes();

// 写出文件名的大小

out.write(name.length);

// 将文件名写出

out.write(name);

// 创建输入流读取文件

FileInputStream in = new FileInputStream(file);

// 读取数据

byte[] data = new byte[1024];

int len = -1;

while ((len = in.read(data)) != -1) {

// 将读取的内容写出

out.write(data, 0, len);

}

// 通知服务器端数据已经全部写出

s.shutdownOutput();

// 关流

s.close();

in.close();

}

}

public class FileServerDemo {

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

// 创建ServerSocket对象

ServerSocket ss = new ServerSocket(8090);

// 接收连接

Socket s = ss.accept();

// 获取输入流

InputStream in = s.getInputStream();

// 获取的是文件名的字节的个数

int i = in.read();

// 读取文件名

byte[] name = new byte[i];

in.read(name);

// 创建输出流

FileOutputStream out = new FileOutputStream("E:\\" + new String(name));

// 读取数据

byte[] data = new byte[1024];

int len = -1;

while ((len = in.read(data)) != -1) {

// 将读取的数据写出

out.write(data, 0, len);

}

// 通知客户端数据已经读取完毕

s.shutdownInput();

// 关流

ss.close();

out.close();

}

}