1、DatagramSocket API

UDP socket有两个核心的类：

1.DatagramSocket 描述一个 socket 对象

1. receive: 接收数据。如果没有接受到数据，receive 就会阻塞等待，如果接受到数据，receive 就返回 DatagramPacket 对象。

2. send：发送数据。以DatagramPacket 为单位进行发送。

2.DatagramPacket 描述一个 UDP 数据报

操作系统提供的网络编程 API 叫做 socket API，其中涉及到一个核心概念 socket，socket 本质是一个文件描述符。

文件描述符：某个进程被创建出来，进程就会对应一个 PCB ， PCB 中就对应了一个文件描述符表，每次打开一个文件，就会在文件描述符表中分配一个表项，文件描述符表类似一个数组，数组的下标就是文件描述符，数组的元素是一个内核结构 struct file。

1. DatagramSocket 构造方法：

`DatagramSocket` 构造方法：

方法签名	方法说明
DatagramSocket()	创建一个UDP数据报套接字的Socket，绑定到本机任意一个随机端口（一般用于客户端）
DatagramSocket(int port)	创建一个UDP数据报套接字的Socket，绑定到本机指定的端口（一般用于服务端）

2. DatagramSocket 方法

方法签名	方法说明
void receive(DatagramPacket p)	从此套接字接收数据报（如果没有接收到数据报，该方法会阻塞等待）
void send(DatagramPacket p)	从此套接字发送数据报包（不会阻塞等待，直接发送）
void close()	关闭此数据报套接字

2、DatagramPacket APID

DatagramPacket 是UDP Socket发送和接收的数据报。

1. DatagramPacket 构造方法

方法签名	方法说明
DatagramPacket(byte[] buf, int length)	构造一个DatagramPacket以用来接收数据报，接收的数据保存在字节数组（第一个参数buf）中，接收指定长度（第二个参数length）
DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)	构造一个DatagramPacket以用来发送数据报，发送的数据为字节数组（第一个参数buf）中，从0到指定长度（第二个参数length）。address指定目的主机的IP和端口号

2. DatagramPacket 方法

方法签名	方法说明
InetAddress getAddress()	从接收的数据报中，获取发送端主机IP地址；或从发送的数据报中，获取接收端主机IP地址
int getPort()	从接收的数据报中，获取发送端主机的端口号；或从发送的数据报中，获取接收端主机端口号
byte[] getData()	获取数据报中的数据

构造UDP发送的数据报时，需要传入 SocketAddress ，该对象可以使用 InetSocketAddress 来创建。

3. InetSocketAddress API

方法签名	方法说明
InetSocketAddress(InetAddress addr, int port)	创建一个Socket地址，包含IP地址和端口号

3、UDP 编程举例

正常的客户端/服务器的通信流程：

客户端	服务器
1. 构造请求并发送	2.读取请求并解析
5.获取响应并显示	3.根据请求计算响应（服务器的灵魂：最核心的操作）
	4.构造响应并返回

1. 简单的回显程序

服务端

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;

/**
 * @author wangyimu
 * @Program 简单的回显程序
 * @create 2021-10-25-21:18
 */
public class UdpEchoServer {
    private DatagramSocket socket = null;

    // port表示端口号
    // 服务器在启动的时候，需要关联（绑定）一个端口号
    // 收到数据的时候，就会根据这个端口号来决定把数据交给那个进程
    // 虽然此处 port 写的是 int，但是实际上端口号是一个两个字节的无符号整数
    // 范围 0 ~ 65535
    public UdpEchoServer(int port) throws SocketException {
        socket = new DatagramSocket(port);
    }

    // 通过这个方法来启动服务器
    public void start() throws IOException {
        System.out.println("服务器启动!");
        while (true) {
            // 1. 读取请求,当前服务器不知道客户啥时候发来请求,此时 receive() 也会阻塞等待
            //  如果此时有请求过来，receive 就会返回
            //  参数 DatagramPacket 是一个输出型参数,socket 中读到的数据会设置到这个参数的对象中
            //  DatagramPacket 在构造的时候,需要指定一个缓冲区(就是一段内存空间,需要使用 byte[])
            DatagramPacket requestPacket = new DatagramPacket(new byte[4096],4096);
            socket.receive(requestPacket);
            // 把 requestPacket 对象里面的内容读取出来,作为一个字符串
            String request = new String(requestPacket.getData(),0,requestPacket.getLength());
            // 2.根据请求来计算响应
            String response = process(request);
            // 3.把响应写到客户端,这时候也需要构造一个 DatagramPacket
            //   此时给 DatagramPacket 中设置的长度,必须是"字节的个数"
            //   如果直接取 response.length() 此处得到的是,字符串的长度,也就是"字符的个数"
            //   当前的 requestPacket 在构造的时候,还需要指定这个包发给谁
            //   其实发送给的目标,就是发请求的一方
            DatagramPacket responsePacket = new DatagramPacket(response.getBytes(),
response.getBytes().length,
```

```
                    requestPacket.getSocketAddress());
            socket.send(responsePacket);
            // 4.加上日志打印
            //   %d 表示要打印一个有符号十进制的整数，%s 表示要打印一个字符串
            String log = String.format("[%s:%d] req: %s; resp: %s",requestPacket.getAddress().toStrin
g(),
                    requestPacket.getPort(),request,response);
            System.out.println(log);
        }
    }

    private String process(String request) {
        return request;
    }

    public static void main(String[] args) throws IOException {
        UdpEchoServer server = new UdpEchoServer(8090);
        server.start();
    }

}
```

```
DatagramPacket requestPacket = new DatagramPacket(new byte[4096], length: 4096);
socket.receive(requestPacket);
```

1. 这个代码中，receive 得到的结果并没有作为一个返回值，而是作为一个输出型参数。这种操作在java 中识比较少见的，Java 中大部分情况都是用返回值表示输出，用参数表示输入，很少会用参数表示输出（引用类型的参数，也可以用来表示输出）

原因：为了能够让用户给 DatagramPacket 指定一个缓冲区。缓冲区的大小由用户去确定。

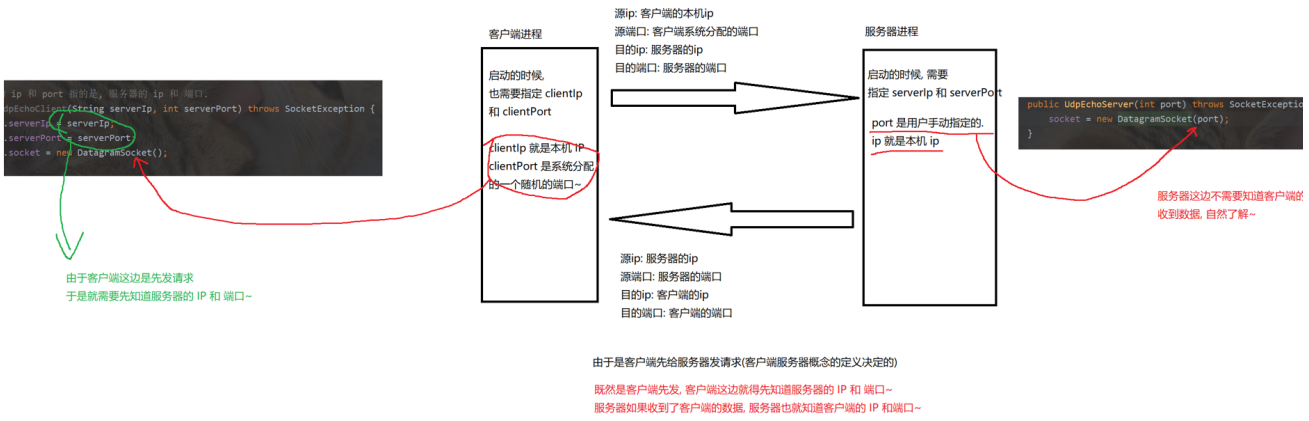
2. String.format：格式化构造字符串的方式

客户端

```
import java.io.IOException;
import java.net.*;
import java.util.Scanner;

/**
 * @author wangyimu
 * @Program 简单的回显程序
 * @create 2021-10-25-21:17
 */
public class UdpEchoClient {
    private DatagramSocket socket  = null;
    private String serverIp;
    private int serverPort;

    public UdpEchoClient(String serverIp, int serverPort) throws SocketException {
        this.serverIp = serverIp;
        this.serverPort = serverPort;
        this.socket = new DatagramSocket();
    }

    public void start() throws IOException {
        while (true) {
            Scanner scan = new Scanner(System.in);
```

```
            // 1.从标准输入一个数据
            System.out.println("->");
            String request = scan.nextLine()
            if(request .equals("exit")){
                System.out.println("exit!");
                return;
            }
            // 2.把字符串构成一个 UDP 请求,并发送数据
            //   这个 DatagramPacket 中,既要包含具体的数据,又要包含这个数据发给谁
            DatagramPacket requestPacket = new DatagramPacket(request.getBytes(),request.getBytes().length,
                    InetAddress.getByName(serverIp),serverPort);
            socket.send(requestPacket);
            // 3.尝试从服务器获取响应
            DatagramPacket responsePacket = new DatagramPacket(new byte[4096],4096);
            socket.receive(responsePacket);
            String response = new String(responsePacket.getData(),0,responsePacket.getLength());
            // 4.显示这个结果
            String log = String.format("req: %s; resp: %s",request,response);
            System.out.println(log);
        }
    }

    public static void main(String[] args) throws IOException {
        UdpEchoClient client = new UdpEchoClient("127.0.0.1",8090);
        client.start();
    }
}
```

回显程序中 IP地址 和 端口号：



DatagramPacket 的构造：

```
DatagramPacket requestPacket = new DatagramPacket (new byte[4096],4096);
```

构造了一个空的 Packet，同时指定了一个缓冲区

```
DatagramPacket responsePacket = new DatagramPacket(response.getBytes(),response.getBytes().length,
 requestPacket.getSocketAddress());
```

构造数据的同时也是加上 IP 和端口，InetSocketAddress

```
 DatagramPacket requestPacket = new DatagramPacket(request.getBytes(),request.getBytes().length,
                    InetAddress.getByName(serverIp),serverPort);
```

构造数据的同时加上 IP和端口，将IP 和端口分开放放置。