1、回顾协议

1. 回顾协议理解为什么需要协议

网络协议是分层的，应用层、传输层、网络层、数据链路层、物理层。

应用层的协议，是直接和程序相关的。

1.使用现成的应用层协议进行开发

2. 使用自己定制的协议，完成需求。

现成的应用协议：HTTP协议（使用最多的）、FTP、SSH、TELNET、DNS等等

对于客户端及服务端应用程序来说，请求和响应，需要约定一致的数据格式：

1. 客户端发送请求和服务端解析请求要使用相同的数据格式。

2. 服务器返回响应和客户端解析响应也要使用相同的数据格式

3. 请求格式和响应格式可以相同，也可以不同

4. 约定相同的数据格式，主要目的是为了让接收端在解析的时候明确如何解析数据中的各个字段

5.可以使用知名协议（广泛使用的协议），如果自己约定的协议，就属于自定义协议

2. 封装/分用和序列化/反序列化

一般来说，在网络数据传输中，发送端应用程序，发送数据时的数据转换（如java一般就是将对象转换为某种协议格式），即对**发送数据时的数据包装动作来说：**

- 如果使用的知名协议，就称为封装。
- 如果使用的小众协议（包含自定义协议），这个动作成为序列化，一般是将程序中的对象抓环卫特定的数据格式。

接收端应用程序，接收数据时的数据转换，即**对接收数据时的数据解析动作来说**：

- 如果使用的是知名协议，就称为分用。
- 如果使用的小众协议（包含自定义协议），这个动作成为反序列化，一般是基于接收数据特定的格式，转换为程序中的对象。

3. 如何设计协议

对于协议来说，在重点是约好如何解析，一般是根据字段的特点进行设计协议：

对于定长的字段：

- 可以基于长度约定，如 int 字段，约定好4个字节即可

对于不定长字段：

- 可以约定字符间的间隔符，或者最后一个字段的结束符，如换行符间隔，\3符合结束等等
- 除了该字段“数据”之外，在加上一个长度字节，用来标识“数据”长度，及总共使用两个字段：
  - “数据”字段本身，不定长，需要通过“长度”字段来解析；
  - “长度”字段，标识该“数据”的长度，即用于辅助解析“数据”字段；

实际开发中，如何来约定自定义协议呢？除了简单粗暴地文本+分隔符的方式，还有哪些更好的方式呢？

1. 文本格式（把请求当成字符串来处理，处理的基本单位是字符）

   xml；json

2. 二进制格式（把请求当成二进制数据处理，处理的基本单位是字节）

protobuffer；thift……

xml：格式化组织数据的方式，结构化的数据

```
// 请求 响应
<request> <response>
 <num1>10</num1> <result>30</result>
 <num2>10</num2> </response>
 <operator>+</operator>
</request>
```

json:

```
// 请求 响应
{ {
 num1:10, result:30
 num2:20, }
 operator:"+"
}
```

2、多线程 + 自定义协议

UDP 客户端：

```

/**
 * @author wangyimu
 * @Program 自定义协议练习---客户端
 * 请求：字符串，第一个操作数 ；  第二个操作数 ；  运算符
 * 响应：字符串 计算机结果
 * @create 2021-11-02-22:18
 */
public class UdpCalcClient {
  private DatagramSocket socket = null;
  private String serverIP;
  private int serverPort;

  public UdpCalcClient(String serverIP, int serverPort) throws S
ocketException {
```

```
15    this.serverIP = serverIP;
16    this.serverPort = serverPort;
17    this.socket = new DatagramSocket();
18    }
19    public void start() throws IOException {
20    Scanner scan = new Scanner(System.in);
21    while(true){
22    // 1.让用户进行输入
23    System.out.println("请输入操作数 num1:");
24    int num1 = scan.nextInt();
25    System.out.println("请输入操作数 num2:" );
26    int num2 = scan.nextInt();
27    System.out.println("请输入运算符(+ - * /):");
28    String operator = scan.next();
29    // 2.构造并发送请求
30    String request = num1 + ";" + num2 + ";" + operator;
31    DatagramPacket requestPacket = new DatagramPacket(request.getB
ytes(),request.getBytes().length,
32    InetAddress.getByName(serverIP),serverPort);
33    socket.send(requestPacket);
34    // 2.尝试连接服务器的响应
35    DatagramPacket responsePacket = new DatagramPacket(new byte[40
96],4096);
36    socket.receive(responsePacket);
37    String response = new String(responsePacket.getData(),0,respon
sePacket.getLength());
38    // 4.显示这个结果
39    System.out.println("计算结果为: " + response);
40    }
41    }
42
43    public static void main(String[] args) throws IOException {
44    UdpCalcClient client = new UdpCalcClient("127.0.0.1",9090);
45    client.start();
46    }
47  }
```

UDP 服务器

```
1
2  /**
3   * @author wangyimu
4   * @Program 自定义协议练习---服务器
5   * 请求：字符串，第一个操作数 ；  第二个操作数 ；  运算符
6   * 响应：字符串 计算机结果
7   * @create 2021-11-02-21:20
8   */
9  public class UdpCalcServer {
10   private DatagramSocket socket = null;
11
12   public UdpCalcServer(int port) throws SocketException {
13   this.socket = new DatagramSocket(port);
14   }
15
16   public void start() throws IOException {
17   System.out.println("服务器启动!");
18   while(true){
19   // 1.读取请求并解析
20   DatagramPacket requestPacket = new DatagramPacket(new byte[409
6], 4096);
21   socket.receive(requestPacket);
22   String request = new String(requestPacket.getData(),0,requestP
acket.getLength());
23   // 2.根据请求计算响应
24   String response = process(request);
25   // 3.把响应写回到客户端
26   DatagramPacket responsePacket = new DatagramPacket(response.ge
tBytes(),response.getBytes().length,
27   requestPacket.getSocketAddress());
28   socket.send(responsePacket);
29   // 4.打印日志
30   String log = String.format("[%s,%d] req: %s; resp: %s",request
Packet.getAddress().toString(),
31   requestPacket.getPort(),request,response);
```

```
32  System.out.println(log);
33  }
34  }
35
36  // process() 内部就是按照约定好的自定义协议进行具体的处理!
37  private String process(String request) {
38  // 1.把 request 还原成操作数和运算符
39  String[] tokens = request.split(";");
40  if(tokens.length != 3){
41  return "[请求格式出错!]";
42  }
43  int num1 = Integer.parseInt(tokens[0]);
44  int num2 = Integer.parseInt(tokens[1]);
45  String operator = tokens[2];
46  int result = 0 ;
47  if(operator.equals("+")){
48  result = num1 + num2;
49  }else if(operator.equals("-")){
50  result = num1 - num2;
51  }else if(operator.equals("*")){
52  result = num1 * num2;
53  }else if(operator.equals("/")){
54  result = num1 / num2;
55  }else{
56  return "[请求格式错误! 操作符不支持!]";
57  }
58  return result + "";
59  }
60
61  public static void main(String[] args) throws IOException {
62  UdpCalcServer server = new UdpCalcServer(9090);
63  server.start();
64  }
65 }
```