## 基于java的网络协议消息的构建和解析

姓名：吴超

学号：20121204057

年级：2013

专业：网络工程

## 基于java的网络协议消息的构建和解析

## 程序间达成的某种包含了信息交换的形式和意义的共识称为协议，用来实现特定应用程序的协议叫做应用程序协议。大部分应用程序协议是根据由字段序列组成的离散信息定义的，其中每个字段中都包含了一段以位序列编码（即二进制字节编码，也可以使用基于文本编码的方式，但常用协议如：TCP、UDP、HTTP等在传输数据时，都是以位序列编码的）的特定信息。应用程序协议中明确定义了信息的发送者应该如何排列和解释这些位序列，同时还要定义接收者应该如何解析，这样才能使信息的接收者能够抽取出每个字段的意义。TCP/IP协议唯一的约束：信息必须在块中发送和接收，而块的长度必须是8位的倍数，因此，我们可以认为TCP/IP协议中传输的信息是字节序列。

由于协议通常处理的是由一组字段组成的离散的信息，因此应用程序协议必须指定消息的接收者如何确定何时消息已被完整接收。成帧技术就是解决接收端如何定位消息首尾位置问题的，  由于协议通常处理的是由一组字段组成的离散的信息，因此应用程序协议必须指定消息的接收者如何确定何时消息已被完整。主要有两种技术使接收者能够准确地找到消息的结束位置：

1、基于定界符：消息的结束由一个唯一的标记指出，即发送者在传输完数据后显式添加的一个特定字节序列，这个特殊标记不能在传输的数据中出现（这也不是绝对的，应用填充技术能够对消息中出现的定界符进行修改，从而使接收者不将其识别为定界符）。该方法通常用在以文本方式编码的消息中。

2、显式长度：在变长字段或消息前附加一个固定大小的字段，用来指示该字段或消息中包含了多少字节。该方法主要用在以二进制字节方式编码的消息中。

由于UDP套接字保留了消息的边界信息，因此不需要进行成帧处理（实际上，主要是DatagramPacket负载的数据有一个确定的长度，接收者能够准确地知道消息的结束位置），而TCP协议中没有消息边界的概念，因此，在使用TCP套接字时，成帧就是一个非常重要的考虑因素（在TCP连接中，接收者读取完最后一条消息的最后一个字节后，将受到一个流结束标记，即read（）返回-1，该标记指示出已经读取到了消息的末尾，非严格意义上来讲，这也算是基于定界符方法的一种特殊情况）

下面是对定长传输的一个tcp数据传输的构建和解析

定义接口

package testframer;

import java.io.IOException;

import java.io.OutputStream;

public interface Framer {

void frameMsg(byte[] message,OutputStream out)throws IOException;

byte[] nextMsg() throws IOException;

}

接口实现：

package testframer;

import java.io.ByteArrayOutputStream;

import java.io.EOFException;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

public class DelimFramer

{

private static final int DELIMITER = '\n';

/\*\*

\* 添加成帧信息并将信息写入到输出流

\*

\* @param message

\* @param out

\* @throws IOException

\*/

public void frameMsg(byte[] message, OutputStream out) throws IOException

{

for (byte b : message)

{

if (b == DELIMITER)

{

throw new IOException("Message contains delimiter");

}

}

out.write(message);

out.write(DELIMITER);

out.flush();

}

public byte[] nextMsg(InputStream in) throws IOException

{

ByteArrayOutputStream messageBuffer = new ByteArrayOutputStream();

int nextByte;

while ((nextByte = in.read()) != DELIMITER)

{

if (nextByte == -1)

{

if (messageBuffer.size() == 0)

{

return null;

}

else

{

throw new EOFException("Non-empty message without delimiter");

}

}

messageBuffer.write(nextByte);

}

return messageBuffer.toByteArray();

}

}

Java client:

package testframer;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.InetAddress;

import java.net.Socket;

import java.net.UnknownHostException;

public class TCPClient

{

public static void main(String[] args) throws UnknownHostException,

IOException

{

Socket client = new Socket(InetAddress.getLocalHost(), 8188);

OutputStream output = client.getOutputStream();

InputStream input = client.getInputStream();

DelimFramer delimFramer = new DelimFramer();

byte[] msg = new String("Hello").getBytes();

// 发送消息

delimFramer.frameMsg(msg, output);

// 接收消息

byte[] receiveByte = delimFramer.nextMsg(input);

String receiveMsg = new String(receiveByte);

System.out.println("Client receive msg:" + receiveMsg);

input.close();

output.close();

client.close();

}

}

Java sever:

package testframer;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.ServerSocket;

import java.net.Socket;

public class TCPServer

{

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

{

DelimFramer delimFramer = new DelimFramer();

ServerSocket server = new ServerSocket(8188);

OutputStream output;

InputStream input;

while (true)

{

Socket client = server.accept();

System.out.println("Handing client at "

+ client.getRemoteSocketAddress());

output = client.getOutputStream();

input = client.getInputStream();

byte[] msg = delimFramer.nextMsg(input);

System.out.println("Server receive msg:" + new String(msg));

delimFramer.frameMsg(msg, output);

}

}

}

实验结果：

