第十三章：通过UDP广播事件

目录 [-]

本章介绍

- UDP介绍
- UDP程序结构和设计
- 日志事件POJO
- 编写广播器
- 编写监听者
- 使用广播器和监听者
- Summary

前面的章节都是在示例中使用TCP协议，这一章，我们将使用UDP。UDP是一种无连接协议，若需要很高的性能和对数据的完成性没有严格要求，那使用UDP是一个很好的方法。最著名的基于UDP协议的是用来域名解析的DNS。

Netty使用了统一的传输API，这使得编写基于UDP的应用程序很容易。可以重用现有的ChannelHandler和其他公共组件来编写另外的Netty程序。看完本章后，你就会知道什么事无连接协议以及为什么UDP可能适合你的应用程序。

13.1 UDP介绍

在深入探讨UDP之前，我们先了解UDP是什么，以及UDP有什么限制或问题。UDP是一种无连接的协议，也就是说客户端和服务器在交互数据之前不会像TCP那样事先建立连接。

UDP是User Datagram Protocol的简称，即用户数据报协议。UDP有不提供数据报分组、组装和不能对数据报进行排序的缺点，也就是说，当数据报发送之后是无法确认数据是否完整到达的。

UDP协议的主要作用是将网络数据流量压缩成数据包的形式。一个典型的数据包就是一个二进制数据的传输单位。每一个数据包的前8个字节用来包含报头信息，剩余字节则用来包含具体的传输数据。

在选择使用协议的时候，选择UDP必须要谨慎。在网络质量令人十分不满意的环境下，UDP协议数据包丢失会比较严重。但是由于UDP的特性：它不属于连接型协议，因而具有资源消耗小，处理速度快的优点，所以通常音频、视频和普通数据在传送时使用UDP较多，因为它们即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。比如我们聊天用的ICQ和QQ就是使用的UDP协议。

UDP就介绍到这里，更详细的资料可以百度或谷歌。

13.2 UDP程序结构和设计

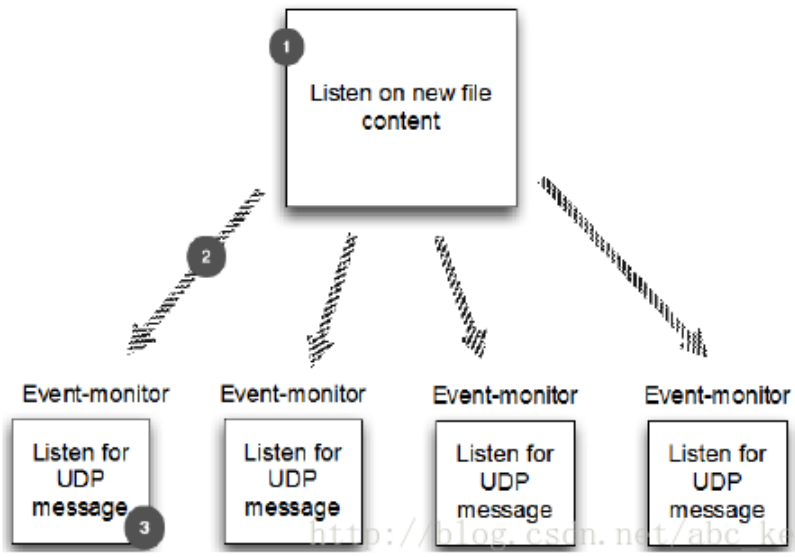
本章例子中，程序打开一个文件并将文件内容一行一行的通过UDP广播到其他的接收主机，这很像UNIX操作系统的日志系统。对于像发送日志的需求，UDP非常适合这样的应用程序，并可以使用UDP通过网络发送大量的“事件”。

使用UDP可以在同一个主机上启动多个应用程序并能独立的进行数据报的发送和接收，UDP使用底层的互联网协议来传送报文，同IP一样提供不可靠的无连接数据报传输服务，它不提供报文到达确认、排序、及流量控制等功能。每个UDP报文分UDP报头和UDP数据区两部分，报头由四个16位长（2字节）字段组成，分别说明该报文的源端口、目的端口、报文长度以及校验值；数据库就是传输的具体数据。

UDP最好在局域网内使用，这样可以大大减少丢包概率。UDP有如下特性：

1. UDP是一个无连接协议，传输数据之前源端和终端不建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。在发送端，UDP传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP把每个消息段放在队列中，应用程序每次从队列中读一个消息段。
2. 由于传输数据不建立连接，因此也就不需要维护连接状态，包括收发状态等，因此一台服务机可同时向多个客户机传输相同的消息。
3. UDP信息包的标题很短，只有8个字节，相对于TCP的20个字节信息包的额外开销很小。
4. 吞吐量不受拥挤控制算法的调节，只受应用软件生成数据的速率、传输带宽、源端和终端主机性能的限制。
5. UDP使用尽最大努力交付，即不保证可靠交付，因此主机不需要维持复杂的链接状态表（这里面有许多参数）。
6. UDP是面向报文的。发送方的UDP对应用程序交下来的报文，在添加首部后就向下交付给IP层。既不拆分，也不合并，而是保留这些报文的边界，因此，应用程序需要选择合适的报文大小。

本章UDP程序例子的示意图入如下：



从上图可以看出，例子程序由两部分组成：广播日志文件和“监控器”，监控器用于接收广播。为了简单，我们将不做任何形式的身份验证或加密。

13.3 日志事件POJO

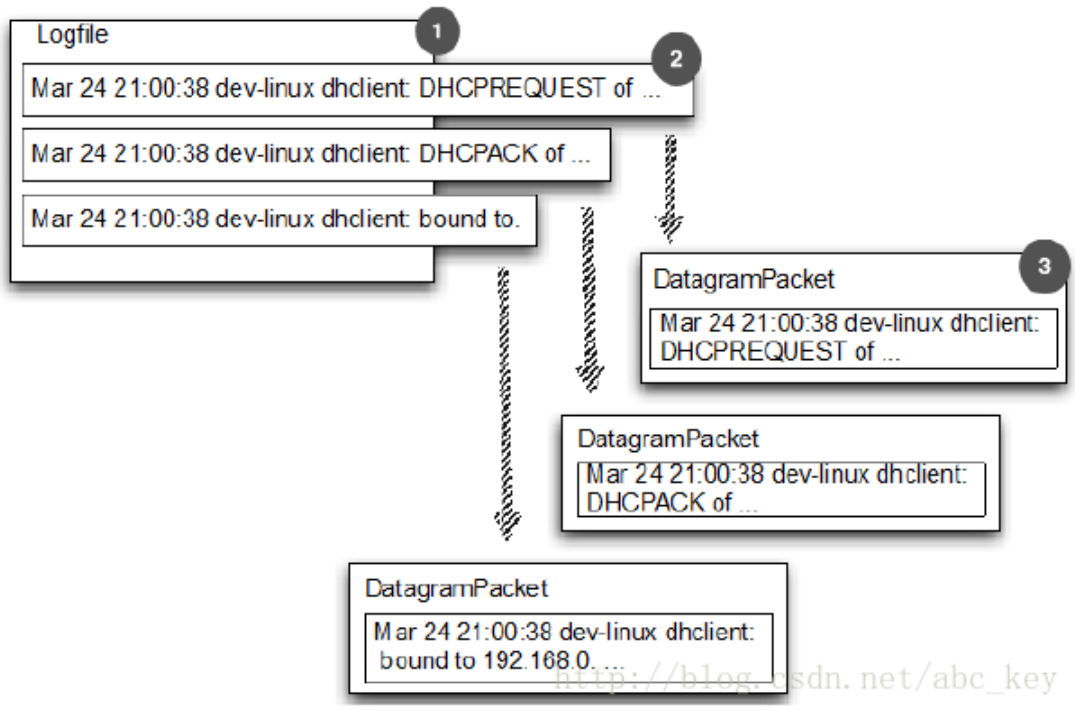
我们的应用程序通常需要某种“消息POJO”用于保存消息，我们把这个消息POJO看成是一个“事件消息”在本例子中我们也创建一个POJO叫做LogEvent，LogEvent用来存储事件数据，然后将数据输出到日志文件。看下面代码：

```java
package netty.in.action.udp;

import java.net.InetSocketAddress;

public class LogEvent {

    public static final byte SEPARATOR = (byte) '|';

    private final InetSocketAddress source;
    private final String logfile;
    private final String msg;
    private final long received;

    public LogEvent(String logfile, String msg) {
        this(null, -1, logfile, msg);
    }

    public LogEvent(InetSocketAddress source, long received, String logfile, String msg) {
        this.source = source;
        this.logfile = logfile;
        this.msg = msg;
        this.received = received;
    }

    public InetSocketAddress getSource() {
        return source;
    }

    public String getLogfile() {
        return logfile;
    }

    public String getMsg() {
        return msg;
    }

    public long getReceived() {
        return received;
    }

}
```

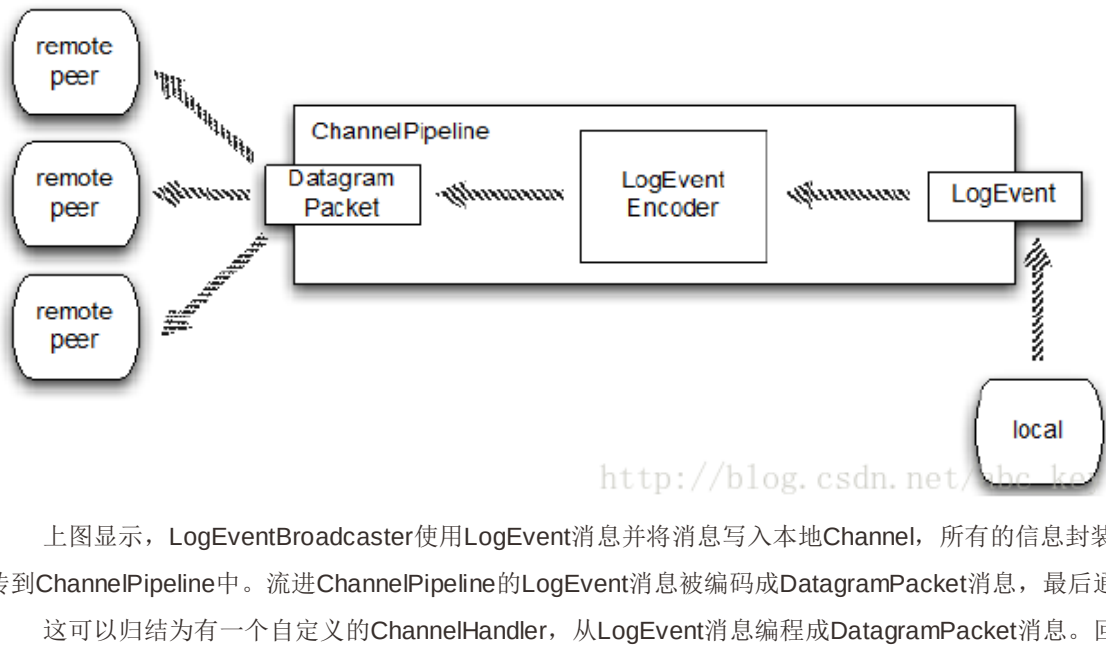
接下来的章节，我们将用这个POJO类来实现具体的逻辑。

13.4 编写广播器

我们要做的是广播一个DatagramPacket日志条目，如下图所示：



上图显示我们有一个从日志条路到DatagramPacket一对一的关系。如同所有的基于Netty的应用程序一样，它由一个或多个ChannelHandler和一些实体对象绑定，用于引导该应用程序。首先让我们来看看LogEventBroadcaster的ChannelPipeline以及作为数据载体的LogEvent的流向，看下图：



上图显示，LogEventBroadcaster使用LogEvent消息并将消息写入本地Channel，所有的信息封装在LogEvent消息中，这些消息被传到ChannelPipeline中。流进ChannelPipeline的LogEvent消息被编码成DatagramPacket消息，最后通过UDP广播到远程对等通道。

这可以归结为有一个自定义的ChannelHandler，从LogEvent消息编程成DatagramPacket消息。回忆我们在第七章讲解的编解码器，我们定义个LogEventEncoder，代码如下：

```java
package netty.in.action.udp;

import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.socket.DatagramPacket;
import io.netty.handler.codec.MessageToMessageEncoder;
import io.netty.util.CharsetUtil;

import java.net.InetSocketAddress;
import java.util.List;

public class LogEventEncoder extends MessageToMessageEncoder<LogEvent> {

    private final InetSocketAddress remoteAddress;

    public LogEventEncoder(InetSocketAddress remoteAddress){
        this.remoteAddress = remoteAddress;
    }

    @Override
    protected void encode(ChannelHandlerContext ctx, LogEvent msg, List<Object> out)
            throws Exception {
        ByteBuf buf = ctx.alloc().buffer();
        buf.writeBytes(msg.getLogfile().getBytes(CharsetUtil.UTF_8));
        buf.writeByte(LogEvent.SEPARATOR);
        buf.writeBytes(msg.getMsg().getBytes(CharsetUtil.UTF_8));
        out.add(new DatagramPacket(buf, remoteAddress));
    }
```

```
}
```

下面我们再编写一个广播器：

```java
package netty.in.action.udp;

import io.netty.bootstrap.Bootstrap;
import io.netty.channel.Channel;
import io.netty.channel.ChannelOption;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.nio.NioDatagramChannel;

import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.net.InetSocketAddress;

public class LogEventBroadcaster {

    private final EventLoopGroup group;
    private final Bootstrap bootstrap;
    private final File file;

    public LogEventBroadcaster(InetSocketAddress address, File file) {
        group = new NioEventLoopGroup();
        bootstrap = new Bootstrap();
        bootstrap.group(group).channel(NioDatagramChannel.class)
                .option(ChannelOption.SO_BROADCAST, true)
                .handler(new LogEventEncoder(address));
        this.file = file;
    }

    public void run() throws IOException {
        Channel ch = bootstrap.bind(0).syncUninterruptibly().channel();
        long pointer = 0;
        for (;;) {
            long len = file.length();
            if (len < pointer) {
                pointer = len;
            } else {
                RandomAccessFile raf = new RandomAccessFile(file, "r");
                raf.seek(pointer);
                String line;
                while ((line = raf.readLine()) != null) {
                    ch.write(new LogEvent(null, -1, file.getAbsolutePath(), line));
                }
                ch.flush();
                pointer = raf.getFilePointer();
                raf.close();
            }
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                Thread.interrupted();
                break;
            }
        }
    }

    public void stop() {
        group.shutdownGracefully();
    }

    public static void main(String[] args) throws Exception {
        int port = 4096;
        String path = System.getProperty("user.dir") + "/log.txt";
        LogEventBroadcaster broadcaster = new LogEventBroadcaster(new InetSocketAddress(
                "255.255.255.255", port), new File(path));
        try {
            broadcaster.run();
        } finally {
            broadcaster.stop();
        }
    }

```

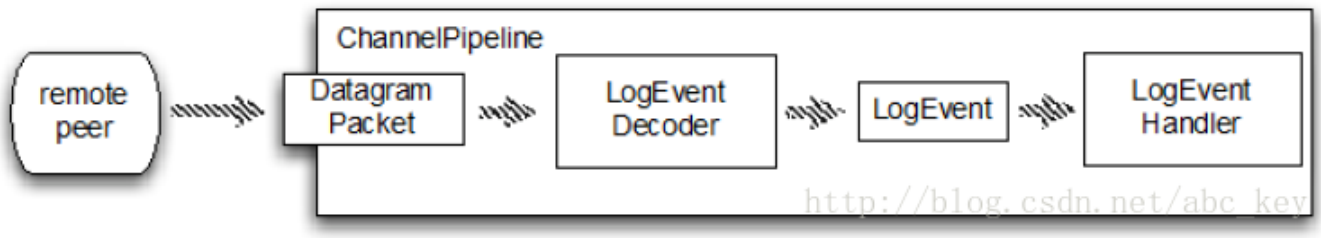
```
73. }
```

13.5 编写监听者

这一节我们编写一个监听者：EventLogMonitor，也就是用来接收数据的程序。EventLogMonitor做下面事情：

- 接收LogEventBroadcaster广播的DatagramPacket
- 解码LogEvent消息
- 输出LogEvent

EventLogMonitor的示意图如下：



解码器代码如下：

```java
package netty.in.action.udp;

import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.socket.DatagramPacket;
import io.netty.handler.codec.MessageToMessageDecoder;
import io.netty.util.CharsetUtil;

import java.util.List;

public class LogEventDecoder extends MessageToMessageDecoder<DatagramPacket> {

    @Override
    protected void decode(ChannelHandlerContext ctx, DatagramPacket msg, List<Object> out)
            throws Exception {
        ByteBuf buf = msg.content();
        int i = buf.indexOf(0, buf.readableBytes(), LogEvent.SEPARATOR);
        String filename = buf.slice(0, i).toString(CharsetUtil.UTF_8);
        String logMsg = buf.slice(i + 1, buf.readableBytes()).toString(CharsetUtil.UTF_8);
        LogEvent event = new LogEvent(msg.sender(),
                System.currentTimeMillis(), filename, logMsg);
        out.add(event);
    }

}
```

处理消息的Handler代码如下：

```java
package netty.in.action.udp;

import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.SimpleChannelInboundHandler;

public class LogEventHandler extends SimpleChannelInboundHandler<LogEvent> {

    @Override
    protected void channelRead0(ChannelHandlerContext ctx, LogEvent msg) throws Exception {
        StringBuilder builder = new StringBuilder();
        builder.append(msg.getReceived());
        builder.append(" [");
        builder.append(msg.getSource().toString());
        builder.append("] [");
        builder.append(msg.getLogfile());
        builder.append("] : ");
        builder.append(msg.getMsg());
        System.out.println(builder.toString());
    }
}
```

EventLogMonitor代码如下：

```java
```

```java
package netty.in.action.udp;

import io.netty.bootstrap.Bootstrap;
import io.netty.channel.Channel;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.ChannelOption;
import io.netty.channel.ChannelPipeline;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.nio.NioDatagramChannel;

import java.net.InetSocketAddress;

public class LogEventMonitor {

    private final EventLoopGroup group;
    private final Bootstrap bootstrap;

    public LogEventMonitor(InetSocketAddress address) {
        group = new NioEventLoopGroup();
        bootstrap = new Bootstrap();
        bootstrap.group(group).channel(NioDatagramChannel.class)
                .option(ChannelOption.SO_BROADCAST, true)
                .handler(new ChannelInitializer<Channel>() {
                    @Override
                    protected void initChannel(Channel channel) throws Exception {
                        ChannelPipeline pipeline = channel.pipeline();
                        pipeline.addLast(new LogEventDecoder());
                        pipeline.addLast(new LogEventHandler());
                    }
                }).localAddress(address);
    }

    public Channel bind() {
        return bootstrap.bind().syncUninterruptibly().channel();
    }

    public void stop() {
        group.shutdownGracefully();
    }

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

        LogEventMonitor monitor = new LogEventMonitor(new InetSocketAddress(4096));
        try {
            Channel channel = monitor.bind();
            System.out.println("LogEventMonitor running");
            channel.closeFuture().sync();
        } finally {
            monitor.stop();
        }
    }
}
```

13.6 使用LogEventBroadcaster和LogEventMonitor

为避免LogEventMonitor接收不到数据，我们必须先启动LogEventMonitor后，再启动LogEventBroadcaster，输出内容这么就不贴图了，读者可以自己运营本例子测试。

13.7 Summary

本章依然没按照原书中的来翻译，主要是以一个例子来说明UDP在Netty中的使用。概念性的东西都是从网上复制的，读者只需要了解UDP的概念再了解清楚例子代码的含义，并试着运行一些例子。