UDP

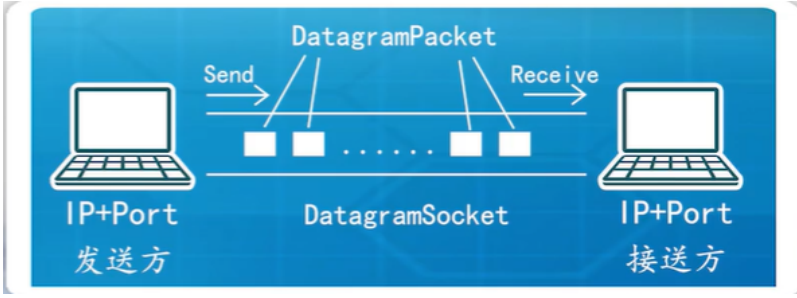
> 计算机通讯：数据从一个IP的port出发（发送方），运输到另外一个IP的port（接收方）

UDP：

– 无连接无状态的通讯协议，
– 发送方发送消息，如果接收方刚好在目的地，则可以接受。如果不在，那这个消息就丢失了
– 发送方也无法得知是否发送成功
– UDP的好处就是简单，节省，经济

实现UDP的类

• DatagramSocket：通讯的数据管道
  – send 和receive方法
  – (可选，多网卡)绑定一个IP和Port
• DatagramPacket
  – 集装箱：封装数据
  – 地址标签：目的地IP+Port



UDP接受方代码

```
import java.net.*;
public class UdpRecv
{
    public static void main(String[] args) throws Exception
    {
        DatagramSocket  ds=new DatagramSocket(3000); //捆绑在3000端口
        byte [] buf=new byte[1024];
        DatagramPacket dp=new DatagramPacket(buf,1024);

        System.out.println("UdpRecv: 我在等待信息");
        ds.receive(dp);
        System.out.println("UdpRecv: 我接收到信息");
        String strRecv=new String(dp.getData(),0,dp.getLength()) +
        " from " + dp.getAddress().getHostAddress()+":"+dp.getPort();
        System.out.println(strRecv);

        Thread.sleep(1000);
        System.out.println("UdpRecv: 我要发送信息");
        String str="hello world 222";
```

```
        DatagramPacket dp2=new DatagramPacket(str.getBytes(),str.length(),
                InetAddress.getByName("127.0.0.1"),dp.getPort());//因为成功的联系过
所以可以直接获得端口
        ds.send(dp2);
        System.out.println("UdpRecv: 我发送信息结束");
        ds.close();
    }
}
```

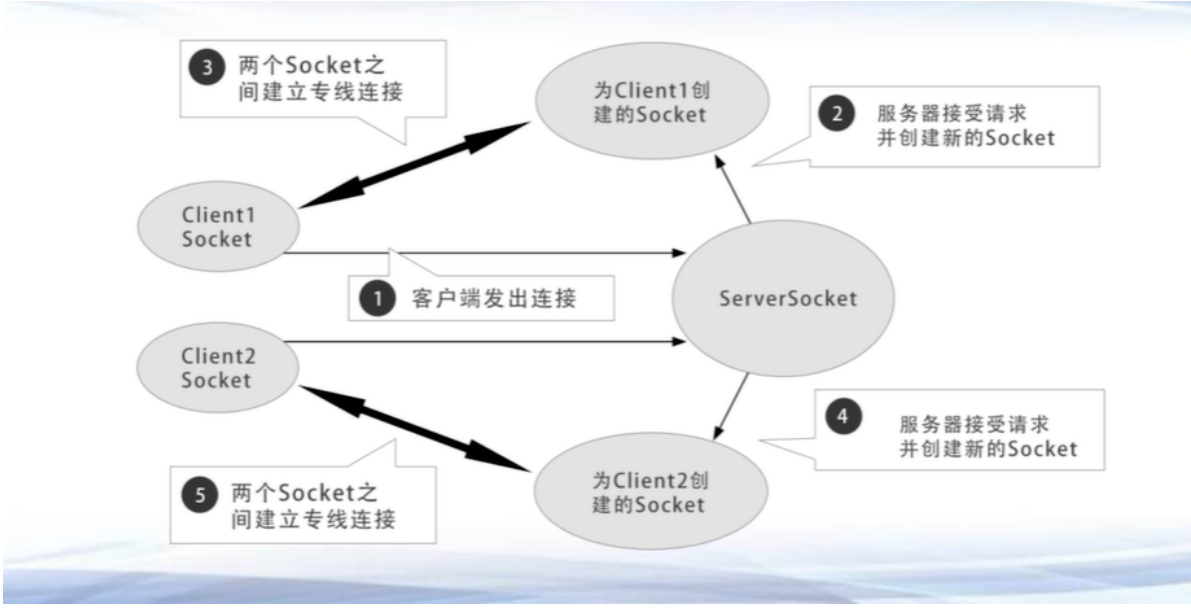
UDP发送方代码

```
import java.net.*;
public class UdpSend
{
    public static void main(String [] args) throws Exception
    {
        DatagramSocket ds=new DatagramSocket();
        String str="hello world";
        DatagramPacket dp=new DatagramPacket(str.getBytes(),str.length(),
                InetAddress.getByName("127.0.0.1"),3000); //内容 长度 目标IP地址 目
标端口

        System.out.println("UdpSend: 我要发送信息");
        ds.send(dp); //解决receive的阻塞
        System.out.println("UdpSend: 我发送信息结束");

        Thread.sleep(1000);
        byte [] buf=new byte[1024];
        DatagramPacket dp2=new DatagramPacket(buf,1024);
        System.out.println("UdpSend: 我在等待信息");
        ds.receive(dp2);
        System.out.println("UdpSend: 我接收到信息");
        String str2=new String(dp2.getData(),0,dp2.getLength()) +
                " from " + dp2.getAddress().getHostAddress()+":"+dp2.getPort();
        System.out.println(str2);

        ds.close();
    }
}
```

TCP

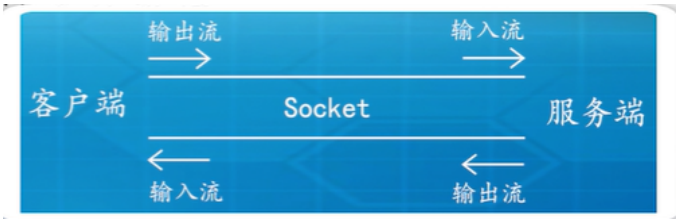
TCP协议

- 有链接、保证可靠的无误差通讯
- ①服务器：创建一个ServerSocket，等待连接
- ②客户机：创建一个Socket，连接到服务器
- ③服务器：ServerSocket接收到连接，创建一个Socket和客户的Socket建立专线连接，后续服务器和客户机的对话(这一对Socket)会在一个单独的线程（服务器端）上运行
- ④服务器的ServerSocket继续等待连接，返回①



TCP编程需要的类

- ServerSocket: 服务器码头
  - 需要绑定port
  - 如果有多块网卡，需要绑定一个IP地址
- Socket: 运输通道
  - 客户端需要绑定服务器的地址和Port
  - 客户端往Socket输入流写入数据，送到服务端
  - 客户端从Socket输出流取服务器端过来的数据
  - 服务端反之亦然



功能特点

- 服务端等待响应时，处于阻塞状态
- 服务端可以同时响应多个客户端
- 服务端每接受一个客户端，就启动一个独立的线程与之对应
- 客户端或者服务端都可以选择关闭这对Socket的通道
- 实例
  - 服务端先启动，且一直保留
  - 客户端后启动，可以先退出

代码实现

单线程服务端

```
import java.net.*;
import java.io.*;
public class TcpServer
{
    public static void main(String [] args)
    {
        try
        {
            ServerSocket ss=new ServerSocket(8001); //驻守在8001端口
            Socket s=ss.accept();                   //阻塞，等到有客户端连接上来
            System.out.println("welcome to the java world");
            InputStream ips=s.getInputStream();     //有人连上来，打开输入流
            OutputStream ops=s.getOutputStream();   //打开输出流
            //同一个通道，服务端的输出流就是客户端的输入流；服务端的输入流就是客户端的输出流

            ops.write("Hello, Client!".getBytes());  //输出一句话给客户端


            BufferedReader br = new BufferedReader(new InputStreamReader(ips));
            //从客户端读取一句话
            System.out.println("Client said: " + br.readLine());


            ips.close();
            ops.close();
            s.close();
            ss.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

多线程服务端

```
// TcpServer2.java
import java.net.*;
public class TcpServer2
{
    public static void main(String [] args)
    {
        try
        {
            ServerSocket ss=new ServerSocket(8001);
            while(true)
            {
                Socket s=ss.accept();
                System.out.println("来了一个client");
                new Thread(new Worker(s)).start();
```

```
            }
            //ss.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}

//Worker.java

import java.net.*;
import java.io.*;

class Worker implements Runnable {
    Socket s;

    public Worker(Socket s) {
        this.s = s;
    }

    public void run() {
        try {
            System.out.println("服务人员已经启动");
            InputStream ips = s.getInputStream();
            OutputStream ops = s.getOutputStream();

            BufferedReader br = new BufferedReader(new InputStreamReader(ips));
            DataOutputStream dos = new DataOutputStream(ops);
            while (true) {
                String strWord = br.readLine();
                System.out.println("client said:" + strWord +":" +
strWord.length());
                if (strWord.equalsIgnoreCase("quit"))
                    break;
                String strEcho = strWord + " 666";
                // dos.writeBytes(strWord +"---->"+ strEcho +"\r\n");
                System.out.println("server said:" + strWord + "---->" +
strEcho);
                dos.writeBytes(strWord + "---->" + strEcho +
System.getProperty("line.separator"));
            }
            br.close();
            // 关闭包装类，会自动关闭包装类中所包装的底层类。所以不用调用ips.close()
            dos.close();
            s.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

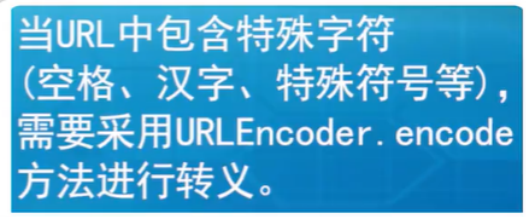
客户端

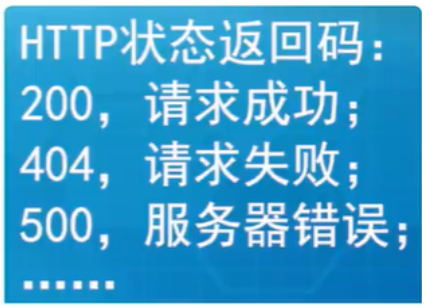
```
import java.net.*;
import java.io.*;

public class TcpClient {
    public static void main(String[] args) {
        try {
            Socket s = new Socket(InetAddress.getByName("127.0.0.1"), 8001); //需
要服务端先开启

            //同一个通道，服务端的输出流就是客户端的输入流；服务端的输入流就是客户端的输出流
            InputStream ips = s.getInputStream();    //开启通道的输入流
            BufferedReader brNet = new BufferedReader(new
InputStreamReader(ips));

            OutputStream ops = s.getOutputStream();  //开启通道的输出流
            DataOutputStream dos = new DataOutputStream(ops);

            BufferedReader brKey = new BufferedReader(new
InputStreamReader(System.in));
            while (true)
            {
                String strWord = brKey.readLine();
                if (strWord.equalsIgnoreCase("quit"))
                {
                    break;
                }
                else
                {
                    System.out.println("I want to send: " + strWord);
                    dos.writeBytes(strWord +
System.getProperty("line.separator"));

                    System.out.println("Server said: " + brNet.readLine());
                }

            }

            dos.close();
            brNet.close();
            brKey.close();
            s.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

HTTP

HTTP

– 超文本传输协议(HyperText Transfer Protocol)
– 用于从WWW（World Wide Web）服务器传输超文本到本地浏览器的传输协议
– 1989年蒂姆•伯纳斯•李（Tim Berners Lee）提出了一种能让远隔两地的研究者们共享知识的设想
– 借助多文档之间相互关联形成的超文本 （HyperText），连成可相互参阅的 WWW
– 1990年问世，1997年发布版本1.1，2015年发布版本2.0
– 资源文件采用HTML编写，以URL形式对外提供





代码实现

post.properties

```
# https://www.ssa.gov/locator
# url=https://secure.ssa.gov/ICON/ic001.do
# zipCodeSearched=94118
url=https://tools.usps.com/go/ZipLookupAction.action
User-Agent=HTTPie/0.9.2
redirects=10
tCompany=
tAddress=1 Market Street
tApt=
tCity=San Francisco
sState=CA
mode=1
```

get功能

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


public class URLConnectionGetTest
{
   public static void main(String[] args)
   {
```

```
      try
      {
         String urlName = "http://www.baidu.com";

         //与百度建立好联系通道
         URL url = new URL(urlName);
         URLConnection connection = url.openConnection();
         connection.connect();

         // 打印http的头部信息

         Map<String, List<String>> headers = connection.getHeaderFields();
         for (Map.Entry<String, List<String>> entry : headers.entrySet())
         {
            String key = entry.getKey();
            for (String value : entry.getValue())
               System.out.println(key + ": " + value);
         }

         // 输出将要收到的内容属性信息

         System.out.println("----------");
         System.out.println("getContentType: " + connection.getContentType());
         System.out.println("getContentLength: " +
connection.getContentLength());
         System.out.println("getContentEncoding: " +
connection.getContentEncoding());
         System.out.println("getDate: " + connection.getDate());
         System.out.println("getExpiration: " + connection.getExpiration());
         System.out.println("getLastModifed: " + connection.getLastModified());
         System.out.println("----------");

         BufferedReader br = new BufferedReader(new
InputStreamReader(connection.getInputStream(), "UTF-8"));

         // 输出收到的内容
         String line = "";
         while((line=br.readLine()) != null)
         {
             System.out.println(line);
         }
         br.close();
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

post功能

```
import java.io.*;
import java.net.*;
import java.nio.file.*;
import java.util.*;

public class URLConnectionPostTest
{
   public static void main(String[] args) throws IOException
   {
      String urlString = "https://tools.usps.com/go/ZipLookupAction.action";
      Object userAgent = "HTTPie/0.9.2";
      Object redirects = "1";
      CookieHandler.setDefault(new CookieManager(null,
CookiePolicy.ACCEPT_ALL));

      Map<String, String> params = new HashMap<String, String>();
      params.put("tAddress", "1 Market Street");
      params.put("tCity", "San Francisco");
      params.put("sState", "CA");
      String result = doPost(new URL(urlString), params,
         userAgent == null ? null : userAgent.toString(),
         redirects == null ? -1 : Integer.parseInt(redirects.toString()));
      System.out.println(result);
   }

   public static String doPost(URL url, Map<String, String> nameValuePairs,
String userAgent, int redirects)
         throws IOException
   {
      HttpURLConnection connection = (HttpURLConnection) url.openConnection();
      if (userAgent != null)
         connection.setRequestProperty("User-Agent", userAgent);

      if (redirects >= 0)
         connection.setInstanceFollowRedirects(false);

      connection.setDoOutput(true);

      //输出请求的参数
      try (PrintWriter out = new PrintWriter(connection.getOutputStream()))
      {
         boolean first = true;
         for (Map.Entry<String, String> pair : nameValuePairs.entrySet())
         {
            //参数必须这样拼接 a=1&b=2&c=3
            if (first)
            {
                first = false;
            }
            else
            {
                out.print('&');
            }
            String name = pair.getKey();
```

```
            String value = pair.getValue();
            out.print(name);
            out.print('=');
            out.print(URLEncoder.encode(value, "UTF-8"));
         }
      }
      String encoding = connection.getContentEncoding();
      if (encoding == null)
      {
          encoding = "UTF-8";
      }

      if (redirects > 0)
      {
         int responseCode = connection.getResponseCode();
         System.out.println("responseCode: " + responseCode);
         if (responseCode == HttpURLConnection.HTTP_MOVED_PERM
               || responseCode == HttpURLConnection.HTTP_MOVED_TEMP
               || responseCode == HttpURLConnection.HTTP_SEE_OTHER)
         {
            String location = connection.getHeaderField("Location");
            if (location != null)
            {
               URL base = connection.getURL();
               connection.disconnect();
               return doPost(new URL(base, location), nameValuePairs, userAgent,
redirects - 1);
            }

         }
      }
      else if (redirects == 0)
      {
         throw new IOException("Too many redirects");
      }

      //接下来获取html 内容
      StringBuilder response = new StringBuilder();
      try (Scanner in = new Scanner(connection.getInputStream(), encoding))
      {
         while (in.hasNextLine())
         {
            response.append(in.nextLine());
            response.append("\n");
         }
      }
      catch (IOException e)
      {
         InputStream err = connection.getErrorStream();
         if (err == null) throw e;
         try (Scanner in = new Scanner(err))
         {
            response.append(in.nextLine());
            response.append("\n");
         }
      }

      return response.toString();
```

```
    }
}
```

HttpClient

JDK HttpClient

• JDK 9 新增，JDK10更新，JDK11正式发布
• java.net.http包
• 取代URLConnection
• 支持HTTP/1.1和HTTP/2
• 实现大部分HTTP方法
• 主要类
– HttpClient
– HttpRequest
– HttpResponse

get功能

```
import java.io.IOException;
import java.net.URI;
import java.net.URLEncoder;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.charset.Charset;

public class JDKHttpClientGetTest {

    public static void main(String[] args) throws IOException,
InterruptedException {
        doGet();
    }

    public static void doGet() {
        try{
            HttpClient client = HttpClient.newHttpClient();
            HttpRequest request =
HttpRequest.newBuilder(URI.create("http://www.baidu.com")).build();
            HttpResponse response = client.send(request,
HttpResponse.BodyHandlers.ofString());
            System.out.println(response.body());
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

post功能

```
import java.io.IOException;
import java.net.URI;
import java.net.URLEncoder;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

public class JDKHttpClientPostTest {

    public static void main(String[] args) throws IOException,
InterruptedException {
        doPost();
    }

    public static void doPost() {
        try {
            HttpClient client = HttpClient.newBuilder().build();
            HttpRequest request = HttpRequest.newBuilder()

.uri(URI.create("https://tools.usps.com/go/ZipLookupAction.action"))
                    //.header("Content-Type","application/x-www-form-
urlencoded")
                    .header("User-Agent", "HTTPie/0.9.2")
                    .header("Content-Type","application/x-www-form-
urlencoded;charset=utf-8")
                    //.method("POST",
HttpRequest.BodyPublishers.ofString("tAddress=1 Market Street&tCity=San
Francisco&sState=CA"))
                    //.version(Version.HTTP_1_1)
                    .POST(HttpRequest.BodyPublishers.ofString("tAddress="
                        + URLEncoder.encode("1 Market Street", "UTF-8")
                        + "&tCity=" + URLEncoder.encode("San Francisco", "UTF-
8") + "&sState=CA"))
                    //.POST(HttpRequest.BodyPublishers.ofString("tAddress=" +
URLEncoder.encode("1 Market Street", "UTF-8") + "&tCity=" +
URLEncoder.encode("San Francisco", "UTF-8") + "&sState=CA"))
                    .build();
            HttpResponse response = client.send(request,
HttpResponse.BodyHandlers.ofString());
            System.out.println(response.statusCode());
            System.out.println(response.headers());
            System.out.println(response.body().toString());

        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

HttpComponents

• hc.apache.org, Apache出品
• 从HttpClient进化而来
• 是一个集成的Java HTTP工具包
– 实现所有HTTP方法：get/post/put/delete
– 支持自动转向
– 支持https协议
– 支持代理服务器等

pom.xml

```
  <dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient
-->
    <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpclient</artifactId>
        <version>4.5.6</version>
    </dependency>

  </dependencies>
```

get方法

```
import java.io.IOException;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.ResponseHandler;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class HttpComponentsGetTest {

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

        CloseableHttpClient httpClient = HttpClients.createDefault();
        RequestConfig requestConfig = RequestConfig.custom()
                .setConnectTimeout(5000)   //设置连接超时时间
                .setConnectionRequestTimeout(5000) // 设置请求超时时间
                .setSocketTimeout(5000)
                .setRedirectsEnabled(true)//默认允许自动重定向
                .build();

        HttpGet httpGet = new HttpGet("http://www.baidu.com");
        httpGet.setConfig(requestConfig);
        String srtResult = "";
        try {
            HttpResponse httpResponse = httpClient.execute(httpGet);
            if(httpResponse.getStatusLine().getStatusCode() == 200){
```

```
                srtResult = EntityUtils.toString(httpResponse.getEntity(), "UTF-
8");//获得返回的结果
                System.out.println(srtResult);
            }else
            {
                //异常处理
            }
        } catch (IOException e) {
            e.printStackTrace();
        }finally {
            try {
                httpClient.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

post方法

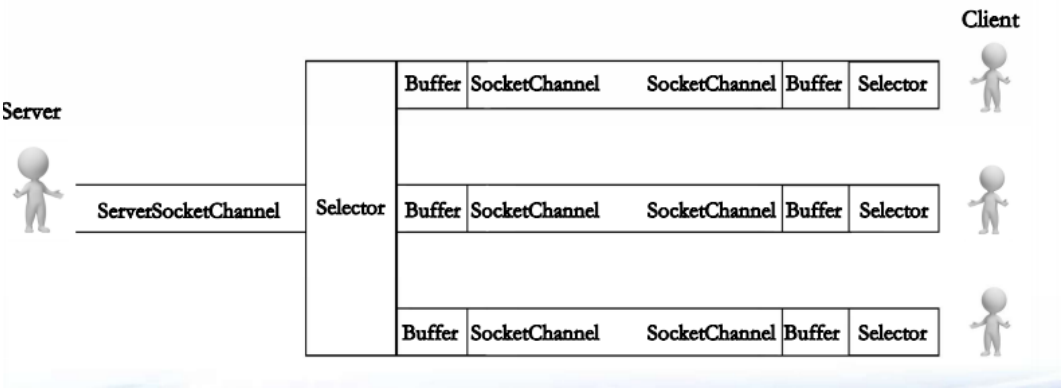
```
import java.io.IOException;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.List;

import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.client.LaxRedirectStrategy;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;

public class HttpComponentsPostTest {

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

        //获取可关闭的 httpCilent
        //CloseableHttpClient httpClient = HttpClients.createDefault();
        CloseableHttpClient httpClient =
HttpClientBuilder.create().setRedirectStrategy(new
LaxRedirectStrategy()).build();
        //配置超时时间
        RequestConfig requestConfig = RequestConfig.custom().
                setConnectTimeout(10000).setConnectionRequestTimeout(10000)
                .setSocketTimeout(10000).setRedirectsEnabled(false).build();

        HttpPost httpPost = new
HttpPost("https://tools.usps.com/go/ZipLookupAction.action");
        //设置超时时间
```

```
        httpPost.setConfig(requestConfig);

        //装配post请求参数
        List<BasicNameValuePair> list = new ArrayList<BasicNameValuePair>();
        list.add(new BasicNameValuePair("tAddress", URLEncoder.encode("1 Market
Street", "UTF-8")));  //请求参数
        list.add(new BasicNameValuePair("tCity", URLEncoder.encode("San
Francisco", "UTF-8"))); //请求参数
        list.add(new BasicNameValuePair("sState", "CA")); //请求参数
        try {
            UrlEncodedFormEntity entity = new UrlEncodedFormEntity(list,"UTF-
8");
            //设置post求情参数
            httpPost.setEntity(entity);
            httpPost.setHeader("User-Agent", "HTTPie/0.9.2");
            //httpPost.setHeader("Content-Type","application/form-data");
            HttpResponse httpResponse = httpClient.execute(httpPost);
            String strResult = "";
            if(httpResponse != null){

 System.out.println(httpResponse.getStatusLine().getStatusCode());
                if (httpResponse.getStatusLine().getStatusCode() == 200) {
                    strResult = EntityUtils.toString(httpResponse.getEntity());
                }
                else {
                    strResult = "Error Response: " +
httpResponse.getStatusLine().toString();
                }
            }else{

            }
            System.out.println(strResult);
        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            try {
                if(httpClient != null){
                    httpClient.close(); //释放资源
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

NIO

Non-Blocking I/O, 非阻塞I/O

• JDK 1.4引入，1.7升级NIO 2.0 (包括了AIO)
• 主要在java.nio包中
• 主要类
– Buffer 缓存区
– Channel 通道
– Selector多路选择器

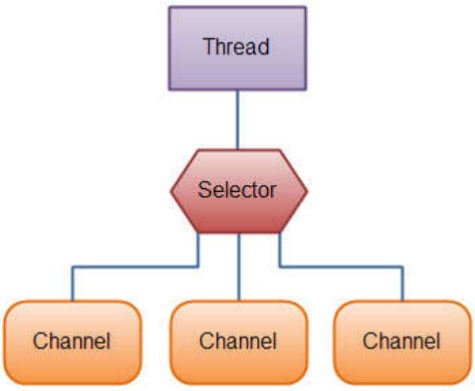


主要功能

• Buffer 缓冲区，一个可以读写的内存区域
  – ByteBuffer, CharBuffer, DoubleBuffer, IntBuffer, LongBuffer, ShortBuffer (StringBuffer 不是Buffer缓冲区)
  • 四个主要属性
    – capacity 容量， position 读写位置
    – limit 界限， mark 标记，用于重复一个读/写操作

• Channel 通道
  – 全双工的，支持读/写(而Stream流是单向的)
  – 支持异步读写
  – 和Buffer配合，提高效率
  – ServerSocketChannel 服务器TCP Socket 接入通道，接收客户端
  – SocketChannel TCP Socket通道，可支持阻塞/非阻塞通讯
  – DatagramChannel UDP 通道
  – FileChannel 文件通道

• Selector多路选择器
  – 每隔一段时间，不断轮询注册在其上的Channel
  – 如果有一个Channel有接入、读、写操作，就会被轮询出来
  – 根据SelectionKey可以获取相应的Channel，进行后续IO操作
  – 避免过多的线程
  – SelectionKey四种类型
    • OP_CONNECT
    • OP_ACCEPT
    • OP_READ
    • OP_WRITE



代码实现

服务端

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.util.Iterator;
import java.util.Set;

public class NioServer {

    public static void main(String[] args) throws IOException {
        int port = 8001;
        Selector selector = null;
        ServerSocketChannel servChannel = null;

        //服务器初始化
        try {
            selector = Selector.open();
            servChannel = ServerSocketChannel.open();
            //设置为非阻塞状态
            servChannel.configureBlocking(false);
            //服务器channel驻守在本机的8001端口
            servChannel.socket().bind(new InetSocketAddress(port), 1024);
            //绑定多路选择器和channel
            servChannel.register(selector, SelectionKey.OP_ACCEPT);
            System.out.println("服务器在8001端口守候");
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(1);
        }

        while(true)
        {
            try {
                //轮询channel
                selector.select(1000);
                //如果获取到有数据响应，那么channel就获得了所有有数据响应的SelectionKey
```

```
                Set<SelectionKey> selectedKeys = selector.selectedKeys();
                //对key进行遍历
                Iterator<SelectionKey> it = selectedKeys.iterator();
                SelectionKey key = null;
                while (it.hasNext()) {
                    key = it.next();
                    it.remove();
                    try {
                        //如果有数据 就放到这里来处理
                        handleInput(selector,key);
                    } catch (Exception e) {
                        if (key != null) {
                            key.cancel();
                            if (key.channel() != null)
                                key.channel().close();
                        }
                    }
                }
            }
            catch(Exception ex)
            {
                ex.printStackTrace();
            }

            try
            {
                Thread.sleep(500);
            }
            catch(Exception ex)
            {
                ex.printStackTrace();
            }
        }
    }

    public static void handleInput(Selector selector, SelectionKey key) throws
IOException {

        if (key.isValid()) {
            // 处理新接入的请求消息
            if (key.isAcceptable()) {
                // Accept the new connection
                ServerSocketChannel ssc = (ServerSocketChannel) key.channel();
                SocketChannel sc = ssc.accept();
                sc.configureBlocking(false);
                // Add the new connection to the selector
                sc.register(selector, SelectionKey.OP_READ);
            }
            if (key.isReadable()) {
                // Read the data
                SocketChannel sc = (SocketChannel) key.channel();
                ByteBuffer readBuffer = ByteBuffer.allocate(1024);
                int readBytes = sc.read(readBuffer);
                if (readBytes > 0) {
                    readBuffer.flip();
                    byte[] bytes = new byte[readBuffer.remaining()];
                    readBuffer.get(bytes);
                    String request = new String(bytes, "UTF-8"); //接收到的输入
```

```
                    System.out.println("client said: " + request);

                    String response = request + " 666";
                    doWrite(sc, response);
                } else if (readBytes < 0) {
                    // 对端链路关闭
                    key.cancel();
                    sc.close();
                } else
                    ; // 读到0字节，忽略
            }
        }
    }

    public static void doWrite(SocketChannel channel, String response) throws
IOException {
        if (response != null && response.trim().length() > 0) {
            byte[] bytes = response.getBytes();
            ByteBuffer writeBuffer = ByteBuffer.allocate(bytes.length);
            writeBuffer.put(bytes);
            writeBuffer.flip();
            channel.write(writeBuffer);
        }
    }
}
```

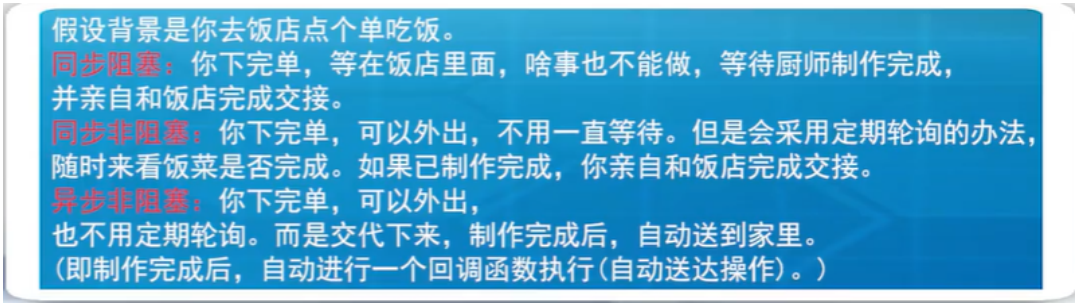
客户端

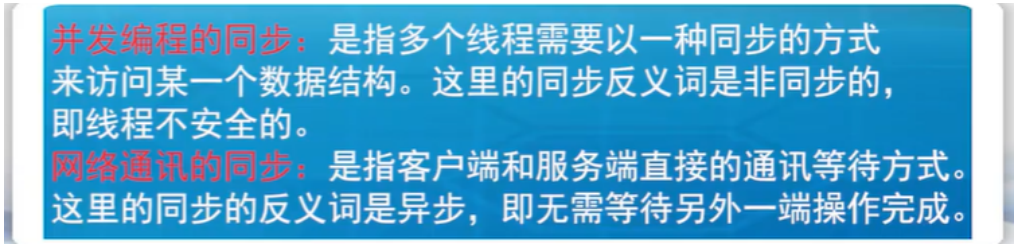
```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.SocketChannel;
import java.util.Iterator;
import java.util.Set;
import java.util.UUID;

public class NioClient {

    public static void main(String[] args) {

        String host = "127.0.0.1";
        int port = 8001;

        Selector selector = null;
        SocketChannel socketChannel = null;

        try
        {
            selector = Selector.open();
            socketChannel = SocketChannel.open();
            socketChannel.configureBlocking(false); // 非阻塞

            // 如果直接连接成功，则注册到多路复用器上，发送请求消息，读应答
            if (socketChannel.connect(new InetSocketAddress(host, port)))
```

```
            {
                socketChannel.register(selector, SelectionKey.OP_READ);
                doWrite(socketChannel);
            }
            else
            {
                //client的selector和channel进行绑定
                socketChannel.register(selector, SelectionKey.OP_CONNECT);
            }

        } catch (IOException e) {
            e.printStackTrace();
            System.exit(1);
        }

        while (true)
        {
            try
            {
                selector.select(1000);
                Set<SelectionKey> selectedKeys = selector.selectedKeys();
                Iterator<SelectionKey> it = selectedKeys.iterator();
                SelectionKey key = null;
                while (it.hasNext())
                {
                    key = it.next();
                    it.remove();
                    try
                    {
                        //处理每一个channel
                        handleInput(selector, key);
                    }
                    catch (Exception e) {
                        if (key != null) {
                            key.cancel();
                            if (key.channel() != null)
                                key.channel().close();
                        }
                    }
                }
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }


        // 多路复用器关闭后，所有注册在上面的Channel资源都会被自动去注册并关闭
//      if (selector != null)
//          try {
//              selector.close();
//          } catch (IOException e) {
//              e.printStackTrace();
//          }
//
//      }
    }
```

```
    public static void doWrite(SocketChannel sc) throws IOException {
        byte[] str = UUID.randomUUID().toString().getBytes();
        ByteBuffer writeBuffer = ByteBuffer.allocate(str.length);
        writeBuffer.put(str);
        writeBuffer.flip();
        sc.write(writeBuffer);
    }

    public static void handleInput(Selector selector, SelectionKey key) throws
Exception {

        if (key.isValid()) {
            // 判断是否连接成功
            SocketChannel sc = (SocketChannel) key.channel();
            if (key.isConnectable()) {
                if (sc.finishConnect()) {
                    sc.register(selector, SelectionKey.OP_READ);
                }
            }
            if (key.isReadable()) {
                ByteBuffer readBuffer = ByteBuffer.allocate(1024);
                int readBytes = sc.read(readBuffer);
                if (readBytes > 0) {
                    readBuffer.flip();
                    byte[] bytes = new byte[readBuffer.remaining()];
                    readBuffer.get(bytes);
                    String body = new String(bytes, "UTF-8");
                    System.out.println("Server said : " + body);
                } else if (readBytes < 0) {
                    // 对端链路关闭
                    key.cancel();
                    sc.close();
                } else
                    ; // 读到0字节，忽略
            }
            Thread.sleep(3000);
            doWrite(sc);
        }
    }
}
```

AIO





Asynchronous I/O, 异步I/O

- JDK 1.7引入，主要在java.nio包中
- 异步I/O，采用回调方法进行处理读写操作
- 主要类
  - AsynchronousServerSocketChannel 服务器接受请求通道
    - bind 绑定在某一个端口 accept 接受客户端请求
  - AsynchronousSocketChannel Socket通讯通道
    - read 读数据 write 写数据
  - CompletionHandler 异步处理类
    - completed 操作完成后异步调用方法 failed 操作失败后异步调用方法

代码实现

AIO服务端

```
package aio;

import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.CharBuffer;
import java.nio.channels.AsynchronousChannelGroup;
import java.nio.channels.AsynchronousServerSocketChannel;
import java.nio.channels.AsynchronousSocketChannel;
import java.nio.channels.CompletionHandler;
import java.nio.charset.Charset;
import java.nio.charset.CharsetDecoder;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class AioServer {

    public static void main(String[] args) throws IOException {
        AsynchronousServerSocketChannel server =
AsynchronousServerSocketChannel.open();
        server.bind(new InetSocketAddress("localhost", 8001));
        System.out.println("服务器在8001端口守候");

        //开始等待客户端连接，一旦有连接，做26行任务
        server.accept(null, new CompletionHandler<AsynchronousSocketChannel,
Object>() {
            @Override
```

```
            public void completed(AsynchronousSocketChannel channel, Object
attachment) {
                 server.accept(null, this); //持续接收新的客户端请求

                 ByteBuffer buffer = ByteBuffer.allocate(1024); //准备读取空间
                 //开始读取客户端内容，一旦读取结束，做33行任务
                 channel.read(buffer, buffer, new CompletionHandler<Integer,
ByteBuffer>() {
                    //读取完毕后该做什么
                     @Override
                     public void completed(Integer result_num, ByteBuffer
attachment) {
                         attachment.flip(); //反转此Buffer
                         CharBuffer charBuffer = CharBuffer.allocate(1024);
                         CharsetDecoder decoder =
Charset.defaultCharset().newDecoder();
                         decoder.decode(attachment,charBuffer,false);
                         charBuffer.flip();
                         String data = new String(charBuffer.array(),0,
charBuffer.limit());
                         System.out.println("client said: " + data);
                         channel.write(ByteBuffer.wrap((data + "
666").getBytes())); //返回结果给客户端
                         try{
                             channel.close();
                         }catch (Exception e){
                             e.printStackTrace();
                         }
                     }

                     @Override
                     public void failed(Throwable exc, ByteBuffer attachment) {
                         System.out.println("read error "+exc.getMessage());
                     }
                 });


            }

            @Override
            public void failed(Throwable exc, Object attachment) {
                System.out.print("failed: " + exc.getMessage());
            }
        });

        while(true){
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

AIO客户端

```
package aio;

import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.CharBuffer;
import java.nio.channels.AsynchronousSocketChannel;
import java.nio.channels.CompletionHandler;
import java.nio.charset.Charset;
import java.nio.charset.CharsetDecoder;
import java.util.UUID;


public class AioClient {

    public static void main(String[] a) {
        try
        {
            AsynchronousSocketChannel channel =
AsynchronousSocketChannel.open();

            //18行连接成功后，自动做20行任务
            channel.connect(new InetSocketAddress("localhost", 8001), null, new
CompletionHandler<Void, Void>() {

                public void completed(Void result, Void attachment) {
                    String str = UUID.randomUUID().toString();

                    //24行向服务器写数据成功后，自动做28行任务
                    channel.write(ByteBuffer.wrap(str.getBytes()), null,
                            new CompletionHandler<Integer, Object>() {

                                @Override
                                public void completed(Integer result, Object
attachment) {
                                    try {
                                        System.out.println("write " + str + ",
and wait response");
                                        //等待服务器响应
                                        ByteBuffer buffer =
ByteBuffer.allocate(1024); //准备读取空间
                                         //开始读取服务器反馈内容，一旦读取结束，做39行
任务
                                        channel.read(buffer, buffer, new
CompletionHandler<Integer, ByteBuffer>() {
                                            @Override
                                            public void completed(Integer
result_num, ByteBuffer attachment) {
                                                attachment.flip(); //反转此Buffer
                                                CharBuffer charBuffer =
CharBuffer.allocate(1024);
                                                CharsetDecoder decoder =
Charset.defaultCharset().newDecoder();
```

```
decoder.decode(attachment,charBuffer,false);
                                                    charBuffer.flip();
                                                    String data = new
String(charBuffer.array(),0, charBuffer.limit());
                                                    System.out.println("server
said: " + data);
                                                    try{
                                                        channel.close();
                                                    }catch (Exception e){
                                                        e.printStackTrace();
                                                    }
                                                }

                                                @Override
                                                public void failed(Throwable exc,
ByteBuffer attachment) {
                                                    System.out.println("read error
"+exc.getMessage());
                                                }
                                            });

                                            channel.close();
                                        } catch (Exception e) {
                                            e.printStackTrace();
                                        }
                                    }

                                    @Override
                                    public void failed(Throwable exc, Object
attachment) {
                                        System.out.println("write error");
                                    }

                                });
                    }

                    public void failed(Throwable exc, Void attachment) {
                        System.out.println("fail");
                    }

                });
                Thread.sleep(10000);
            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

每一个CompletionHandler都可以定义两个方法：completed和failed方法。当操作成功完成，将自动回调completed方法；
如果操作发生异常，那么将自动回调failed方法。

不同方式的比较

	BIO	NIO	AIO
阻塞方式	阻塞	非阻塞	非阻塞
同步方式	同步	同步	异步
编程难度	简单	较难	困难
客户机/服务器线程对比	1:1	N:1	N:1
性能	低	高	高

Netty编程

Netty

- 最早由韩国Trustin Lee 设计开发的
- 后来由JBoss接手开发，现在是独立的Netty Project
- 一个非阻塞的客户端-服务端网络通讯框架
- 基于异步事件驱动模型
- 简化Java的TCP和UDP编程
- 支持HTTP/2， SSL等多种协议
- 支持多种数据格式，如JSON等

关键技术

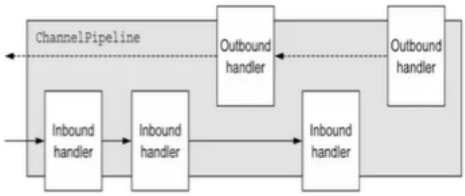
– 通道 Channel
  - ServerSocketChannel/NioServerSocketChannel/...
  - SocketChannel/NioSocketChannel

– 事件 EventLoop
  - 为每个通道定义一个EventLoop，处理所有的I/O事件
  - EventLoop注册事件
  - EventLoop将事件派发给ChannelHandler
  - EventLoop安排进一步操作

– 事件
  - 事件按照数据流向进行分类
  - 入站事件：连接激活/数据读取/.....
  - 出站事件：打开到远程连接/写数据/.....

– 事件处理 ChannelHandler
  - Channel通道发生数据或状态改变
  - EventLoop会将事件分类，并调用ChannelHandler的回调函数
  - 程序员需要实现ChannelHandler内的回调函数
  - ChannelInboundHandler/ChannelOutboundHandler

- ChannelHandler工作模式：责任链
  - 责任链模式
    - 将请求的接收者连成一条链
    - 在链上传递请求，直到有一个接收者处理该请求
    - 避免请求者和接收者的耦合
  - ChannelHandler可以有多个，依次进行调用
  - ChannelPipeline作为容器，承载多个ChannelHandler
    - ByteBuf
      - 强大的字节容器，提供丰富API进行操作



代码实现

Netty服务端

```
package netty1;

import io.netty.bootstrap.ServerBootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioServerSocketChannel;

import java.net.InetSocketAddress;

public class EchoServer {
    public static void main(String[] args) throws Exception {
        int port = 8001;
        final EchoServerHandler serverHandler = new EchoServerHandler();
        EventLoopGroup group = new NioEventLoopGroup();
        try {
            //ServerBootstrap是netty中的一个服务器引导类
            ServerBootstrap b = new ServerBootstrap();
            //配置好ServerBootstrap
            b.group(group)
                .channel(NioServerSocketChannel.class)  //设置通道类型
                .localAddress(new InetSocketAddress(port))  //设置监听端口
                .childHandler(new ChannelInitializer<SocketChannel>() { //初始化责任链
                    @Override
                    public void initChannel(SocketChannel ch) throws Exception {
                        ch.pipeline().addLast(serverHandler); //添加处理类
                    }
                });

            ChannelFuture f = b.bind().sync();  //开启监听
            if(f.isSuccess()){
```

```
                System.out.println(EchoServer.class.getName() +
                        " started and listening for connections on " +
f.channel().localAddress());
            }

            f.channel().closeFuture().sync();
        } finally {
            group.shutdownGracefully().sync();
        }
    }
}
```

Netty服务端处理函数

```
package netty1;

import io.netty.buffer.ByteBuf;
import io.netty.buffer.Unpooled;
import io.netty.channel.ChannelFutureListener;
import io.netty.channel.ChannelHandler.Sharable;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;
import io.netty.util.CharsetUtil;

public class EchoServerHandler extends ChannelInboundHandlerAdapter {
   //完成服务器接受数据并返回数据的过程
    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) {
        ByteBuf in = (ByteBuf) msg;
        String content = in.toString(CharsetUtil.UTF_8);
        System.out.println("Server received: " + content);

        ByteBuf out = ctx.alloc().buffer(1024);
        out.writeBytes((content + " 666").getBytes());
        ctx.write(out);
    }

    @Override
    public void channelReadComplete(ChannelHandlerContext ctx)
            throws Exception {
        ctx.writeAndFlush(Unpooled.EMPTY_BUFFER)
                .addListener(ChannelFutureListener.CLOSE);
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx,
        Throwable cause) {
        cause.printStackTrace();
        ctx.close();
    }
}
```

Netty客户端

```
package netty1;

import io.netty.bootstrap.Bootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioSocketChannel;

import java.net.InetSocketAddress;

public class EchoClient {

    public static void main(String[] args) throws Exception {
        String host = "localhost";
        int port = 8001;

        EventLoopGroup group = new NioEventLoopGroup();
        try {
            Bootstrap b = new Bootstrap();
            b.group(group)
                .channel(NioSocketChannel.class) //设置channel类型
                .remoteAddress(new InetSocketAddress(host, port)) //配置远程地址
                .handler(new ChannelInitializer<SocketChannel>() {
                    @Override
                    public void initChannel(SocketChannel ch)
                        throws Exception {
                        ch.pipeline().addLast(new EchoClientHandler());
                    }
                });
            ChannelFuture f = b.connect().sync(); //连接到服务器
            f.channel().closeFuture().sync();
        } finally {
            group.shutdownGracefully().sync();
        }
    }
}
```

Netty客户端处理函数

```
package netty1;

import io.netty.buffer.ByteBuf;
import io.netty.buffer.Unpooled;
import io.netty.channel.ChannelHandler.Sharable;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.SimpleChannelInboundHandler;
import io.netty.util.CharsetUtil;


//不同的事情放到不同的函数处理
```

```
public class EchoClientHandler
    extends SimpleChannelInboundHandler<ByteBuf> {
    @Override
    public void channelActive(ChannelHandlerContext ctx) {
        ctx.writeAndFlush(Unpooled.copiedBuffer("Netty rocks!",
                CharsetUtil.UTF_8));
    }

    @Override
    public void channelRead0(ChannelHandlerContext ctx, ByteBuf in) {
        System.out.println(
                "Client received: " + in.toString(CharsetUtil.UTF_8));
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx,
        Throwable cause) {
        cause.printStackTrace();
        ctx.close();
    }
}
```