网络编程套接字

本节重点

> - 认识IP地址, 端口号, 网络字节序等网络编程中的基本概念;
> - 学习socket api的基本用法;
> - 能够实现一个简单的udp客户端/服务器;
> - 能够实现一个简单的tcp客户端/服务器(单连接版本, 多线程版本);
> - 理解tcp服务器建立连接, 发送数据, 断开连接的流程;

1 预备知识

理解源IP地址和目的IP地址

在IP数据包头部中, 有两个IP地址, 分别叫做源IP地址, 和目的IP地址.

思考: 我们光有IP地址就可以完成通信了嘛? 想象一下发qq消息的例子, 有了IP地址能够把消息发送到对方的机器上, 但是还需要有一个其他的标识来区分出, 这个数据要给哪个程序进行解析.

认识端口号

端口号(port)是传输层协议的内容.

> - 端口号是一个32位的整数;
> - 端口号用来标识一个进程, 告诉操作系统, 当前的这个数据要交给哪一个进程来处理;
> - IP地址 + 端口号能够标识网络上的某一台主机的某一个进程;
> - 一个端口号只能被一个进程占用.

理解 "端口号" 和 "进程ID"

我们之前在学习系统编程的时候, 学习了 pid 表示唯一一个进程; 此处我们的端口号也是唯一表示一个进程. 那么这两者之间是怎样的关系?

10086例子

另外, 一个进程可以绑定多个端口号; 但是一个端口号不能被多个进程绑定;

理解源端口号和目的端口号

传输层协议(TCP和UDP)的数据段中有两个端口号, 分别叫做源端口号和目的端口号. 就是在描述 "数据是谁发的, 要发给谁";

认识TCP协议

此处我们先对TCP(Transmission Control Protocol 传输控制协议)有一个直观的认识; 后面我们再详细讨论TCP的一些细节问题.

> - 传输层协议
> - 有连接

> - 可靠传输
> - 面向字节流

认识UDP协议

此处我们也是对UDP(User Datagram Protocol 用户数据报协议)有一个直观的认识; 后面再详细讨论.

> - 传输层协议
> - 无连接
> - 不可靠传输
> - 面向数据报

2 socket编程接口

2.1socket 常见API

DatagramSocket类:

方法签名	方法说明
DatagramSocket(int port,InetAddress laddr)	创建一个数据报套接字，绑定到指定的本地地址
DatagramSocket(SocketAddress bindaddr)	创建一个数据报套接字，绑定到指定的本地套接字地址
void bind(SocketAddress addr)	将此DatagramSocket绑定到特定的地址和端口
void connect(InetAddress address, int port)	将套接字连接到此套接字的远程地址
void receive(DatagramPacket p)	从此套接字接收数据报包
void close()	关闭此数据报套接字
void send(DatagramPacket p)	从此套接字发送数据报包

简单的UDP网络程序

实现一个简单的服务端接收客户端输入的信息然后在服务端显示。

UDP服务端

```
package com.store.frank.udp;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketAddress;

/**
 * @author zhangmingming  braveheart1115@163.com
 * @date 2019年10月08日 11:45
 * @Description: udp 示例，接收客户端传入的字符串，然后在服务端输出
 */
```

```java
public class UdpServer {

    public static void main(String[] args) throws Exception {
        new UdpServer().start();
    }

    // 定义udp端口号
    public static final int PORT = 30000;
    // 每个数据报最大为4K
    private static final int DATA_LEN = 4096;
    // 接收网络数据的字节数组
    byte[] buff_in = new byte[DATA_LEN];
    // 以指定字节数组创建准备接受数据的DatagramPacket对象
    private DatagramPacket packet_in = new DatagramPacket(buff_in, buff_in.length);

    // 定义一个用于发送的DatagramPacket对象
    private DatagramPacket packet_out;

    public void start() throws Exception {
        try(DatagramSocket socket = new DatagramSocket(PORT)){
            String word = null;  // 客户端输入的英文单词
            SocketAddress address=null; // 获取客户端对象,通过该对象将数据写给对方，否则没有目的地
            byte[] reviceData=null; // 返回给客户端的数据
            System.out.println("服务端启动了.....");
            while (true){
               // 读取socket中的数据，然后将数据封装到 packet_in 中
                socket.receive(packet_in);
               // 获取客户端输入的数据
                buff_in=packet_in.getData();
               // 将byte数组转换为字符串  需要去掉后面的空格
                word=new String(buff_in,0,buff_in.length).trim();
                System.out.println("客户端输入的是:"+word);
                address=packet_in.getSocketAddress();
                // 检查退出条件
                if("down".equals(word)){
                    reviceData="服务器关闭了，请重试".getBytes();
                    packet_out=new DatagramPacket(reviceData,reviceData.length,address);
                    socket.send(packet_out);
                    break;
                }else {
                 // 构建服务端发送给客户端的数据
                    reviceData=("你输入的英文单词是:"+word).getBytes();
                 // 构建 packet_out 发送数据
                    packet_out=new DatagramPacket(reviceData,reviceData.length,address);
                    socket.send(packet_out);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("=====服务端关闭=====");
        }
    }
}
```

运行后控制台输入如下：

```
服务端启动了.....
```

UDP服务客户端

```
package com.store.frank.udp;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

/**
 * @author zhangmingming  braveheart1115@163.com
 * @date 2019年09月27日 19:50
 * @Description:
 */
public class UdpClient {
    // 定义发送数据报的目的地
    public static final int DEST_PORT = 30000;

    public static final String DEST_IP = "127.0.0.1";
    // 定义每个数据报的最大大小为4K
    private static final int DATA_LEN = 4096;
    // 定义接收网络数据的字节数组
    byte[] inBuff = new byte[DATA_LEN];


    // 创建接受回复数据的DatagramPacket对象
    private DatagramPacket packet_in = new DatagramPacket(inBuff, inBuff.length);

    // 定义一个用于发送的DatagramPacket对象
    private DatagramPacket packet_out = null;



    public void start() throws IOException {
        try (DatagramSocket socket = new DatagramSocket()) {
            // 初始化发送用的DatagramSocket
            InetAddress ip = InetAddress.getByName(DEST_IP);
            packet_out = new DatagramPacket(new byte[0], 0, ip, DEST_PORT);
            // 创建键盘输入流
            Scanner sc = new Scanner(System.in);
            System.out.println("请输入数据");
            // 不断读取键盘输入
            String key =null;
            // 键盘输入字符对应的byte数组
            byte[] keyBuff = null;
            while (sc.hasNextLine()) {
                key = sc.nextLine();
```

```
                if ("exit".equals(key)) {
                    break;
                }
                // 输入的字符串→字节数组
                keyBuff = key.getBytes();
                // 设置发送用的DatagramPacket里的字节数据
                packet_out.setData(keyBuff);
                // 发送数据报
                socket.send(packet_out);
                // 读取Socket中的数据，读到的数据放在inPacket所封装的字节数组里。
                socket.receive(packet_in);
                System.out.println(new String(inBuff, 0, packet_in.getLength()));
                System.out.println("请输入数据:");
            }
            System.out.println("=== 客户端退出 ===");
        }
    }

    public static void main(String[] args) throws IOException {
        new UdpClient().start();
    }
}
```

客户端启动后显示如下：

```
请输入数据
```

客户端输入数据后，客户端和服务端显示如下：

```
-- 客户端
请输入数据
cat
你输入的英文单词是:cat
请输入数据:

-- 服务端
服务端启动了......
客户端输入的是:cat
```

实现英译汉服务器

通过上面的UDP服务端进行修改就可以实现英汉互译的服务器，服务端代码需要调整，客户端则不变，代码如下：

```
package com.store.frank.udp;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketAddress;
```

```java
import java.util.HashMap;
import java.util.Map;

/**
 * @author zhangmingming  braveheart1115@163.com
 * @date 2019年10月16日 17:54
 * @Description: 英译汉服务器
 */
public class UdpDictServer {


    public static void main(String[] args) throws Exception {
        new UdpDictServer().start();
    }

    // 定义udp端口号
    public static final int PORT = 30000;
    // 每个数据报最大为4K
    private static final int DATA_LEN = 4096;
    // 接收网络数据的字节数组
    byte[] buff_in = new byte[DATA_LEN];
    // 以指定字节数组创建准备接受数据的DatagramPacket对象
    private DatagramPacket packet_in = new DatagramPacket(buff_in, buff_in.length);

    // 定义一个用于发送的DatagramPacket对象
    private DatagramPacket packet_out;

    public void start() throws Exception {
        try(DatagramSocket socket = new DatagramSocket(PORT)){
            String key = null;  // 客户端输入的英文单词
            String value=null;  // 服务端翻译的中文内容
            SocketAddress address=null; // 获取客户端对象,通过该对象将数据写给对方，否则没有目的地
            byte[] reviceData=null; // 返回给客户端的数据
            System.out.println("英译汉服务器启动了.....");
            while (true){
                // 读取socket中的数据，然后将数据封装到 packet_in 中
                socket.receive(packet_in);
                // 获取客户端输入的数据
                buff_in=packet_in.getData();
                // 将byte数组转换为字符串  需要去掉后面的空格
                key=new String(buff_in,0,buff_in.length).trim();

                // 根据map的key获取value
                value=maps.get(key);
                if(null == value){
                    value="默认值";
                }
                address=packet_in.getSocketAddress();
                // 检查退出条件
                if("down".equals(key)){
                    System.out.println("客户端输入的是:"+key);
                    reviceData="服务器关闭了，请重试".getBytes();
                    packet_out=new DatagramPacket(reviceData,reviceData.length,address);
```

```
                    socket.send(packet_out);
                    break;
                }else {
                    System.out.println("客户端输入的是:"+key+",翻译后的结果是"+value);
                    // 构建服务端发送给客户端的数据
                    reviceData=("你输入的英文单词翻译成中文是:"+value).getBytes();
                    // 构建 packet_out 发送数据
                    packet_out=new DatagramPacket(reviceData,reviceData.length,address);
                    socket.send(packet_out);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("=====服务器关闭=====");
        }
    }

    // 定义map集合，用来存放需要翻译的单词。
    private static Map<String, String> maps = new HashMap<>();
    // 向map中添加数据
    static {
        maps.put("dog", "狗");
        maps.put("cat", "猫");
        maps.put("fish", "鱼");
        maps.put("bird", "鸟");
        maps.put("pig", "猪");
    }
}
```

客户端输入数据后，客户端和服务端显示如下：

```
-- 客户端
请输入数据
cat
你输入的英文单词翻译成中文是:猫
请输入数据:

-- 服务端
英译汉服务器启动了.....
客户端输入的是:cat,翻译后的结果是猫
```

2.2简单的TCP网络程序

和刚才UDP类似. 实现一个简单的英译汉的功能

TCP socket API 详解

ServerSocket类

方法签名	方法说明
ServerSocket(int port)	创建绑定到指定端口的服务器套接字
ServerSocket(int port, int backlog)	创建服务器套接字并将其绑定到指定的本地端口号，并指定了积压。
Socket accept()	侦听要连接到此套接字并接受它
bind(SocketAddress endpoint)	将ServerSocket绑定到特定地址（IP地址和端口号）
InetAddress getInetAddress()	返回此服务器套接字的本地地址
void close()	关闭此套接字
int getLocalPort()	返回此套接字正在侦听的端口号

bind():

> - 服务器程序所监听的网络地址和端口号通常是固定不变的,客户端程序得知服务器程序的地址和端口号后就可以向服务器发起连接; 服务器需要调用bind绑定一个固定的网络地址和端口号;
> - 如果地址为`null`，则系统将接收临时端口和有效的本地地址来绑定套接字。

accept():

> - 三次握手完成后, 服务器调用accept()接受连接;
> - 如果服务器调用accept()时还没有客户端的连接请求,就阻塞等待直到有客户端连接上来;
> - Socket 是一个返回值,代表网络的套接字;

Socket类

方法签名	方法说明
Socket(InetAddress address, int port)	创建流套接字并将其连接到指定IP地址的指定端口号
Socket(String host, int port)	创建流套接字并将其连接到指定主机上的指定端口号
void bind(SocketAddress bindpoint)	将套接字绑定到本地地址
void connect(SocketAddress endpoint)	将此套接字连接到服务器
InetAddress getInetAddress()	返回套接字所连接的地址
InputStream getInputStream()	返回此套接字的输入流
OutputStream getOutputStream()	返回此套接字的输出流

TCP通用服务器

TCP服务端，接收客户端内容，然后输出

```
package com.store.frank.http;

import java.io.BufferedReader;
```

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * @author zhangmingming  braveheart1115@163.com
 * @date 2019年10月08日 14:12
 * @Description: HTTP 请求服务端.实现将客户端输入的英文单词翻译为中文。
 */
public class HttpServer {

    public static void main(String[] args) throws IOException {
        ServerSocket server = new ServerSocket(8888);
        System.out.println("服务器启动了....");
        boolean isShow=true; // 设置第一次输出[服务器启动了....], 启动后不再显示。
        while (true){
            try {
                Socket client = server.accept();
                String clientName=client.getInetAddress().getLocalHost().toString();
                if(isShow){
                    System.out.println("客户端:"+clientName+"已连接到服务器");
                }
                isShow=false;
                BufferedReader br = new BufferedReader(new
InputStreamReader(client.getInputStream()));
                String word = br.readLine(); // word 客户端发送来的消息
                if(null != word){
                    System.out.println("客户端输入的是:"+word+",服务器响应的是:"+word);
                    BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(client.getOutputStream()));
                    bw.write(word+"\n");
                    bw.flush();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

TCP客户端

```
package com.store.frank.http;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
```

```
import java.net.Socket;
import java.util.Scanner;

/**
 * @author zhangmingming  braveheart1115@163.com
 * @date 2019年10月08日 14:13
 * @Description: HTTP客户端
 */
public class HttpClient {

    public static void main(String[] args) {
        try {
            Socket s = new Socket("127.0.0.1",8888);
            //构建IO
            InputStream is = s.getInputStream();
            OutputStream os = s.getOutputStream();
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(os));
            BufferedReader br = new BufferedReader(new InputStreamReader(is));
            Scanner scanner=new Scanner(System.in);
            String key = null;
            String value = null;
            while (true){
                System.out.println("请输入数据");
                if(scanner.hasNext()) {
                    key = scanner.nextLine();
                    //向服务器端发送一条消息
                    bw.write(key+"\n");
                    bw.flush();
                    //读取服务器返回的消息
                    value = br.readLine();
                    System.out.println("客户端输入的是:"+key+",服务器响应的是: "+value);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

服务端与客户端启动后，客户端输入cat后，客户端和服务端显示如下：

```
-- 客户端
请输入数据
cat
客户端输入的是:cat,服务器响应的是: cat
请输入数据

-- 服务端
服务器启动了....
客户端:frank/192.168.3.240已连接到服务器
客户端输入的是:cat,服务器响应的是:cat
```

英译汉服务器

```
package com.store.frank.http;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashMap;
import java.util.Map;

/**
 * @author zhangmingming  braveheart1115@163.com
 * @date 2019年10月16日 23:25
 * @Description: HTTP 请求服务端.实现将客户端输入的英文单词翻译为中文。
 */
public class HttpDictServer {

    // 定义map集合，用来存放需要翻译的单词。
    private static Map<String, String> maps = new HashMap<>();
    // 向map中添加数据
    static {
        maps.put("dog", "狗");
        maps.put("cat", "猫");
        maps.put("fish", "鱼");
        maps.put("bird", "鸟");
        maps.put("pig", "猪");
    }


    public static void main(String[] args) throws IOException {
        ServerSocket server = new ServerSocket(8888);
        System.out.println("服务器启动了....");
        boolean isShow=true; // 设置第一次输出[服务器启动了....]，启动后不再显示。
        while (true){
            try {
                Socket client = server.accept();
                String clientName=client.getInetAddress().getLocalHost().toString();
                if(isShow){
                    System.out.println("客户端:"+clientName+"已连接到服务器");
                }
                isShow=false;
                BufferedReader br = new BufferedReader(new
InputStreamReader(client.getInputStream()));
                // key 客户端发送来的消息
                String key = br.readLine();
                // 服务器翻译后的结果
                String value=maps.get(key);
```

```
                if(null != key){
                    if(null == value){
                        value="default";
                        System.out.println("客户端输入的是:"+key+",服务端返回的是默认值: default");
                        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(client.getOutputStream()));
                        bw.write(value+"\n");
                        bw.flush();
                    }else{
                        System.out.println("客户端输入的是:"+key+",服务器翻译后是:"+value);
                        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(client.getOutputStream()));
                        bw.write(value+"\n");
                        bw.flush();
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

英译汉客户端

英译汉的客户端与通用服务器的客户端代码是相同的。

服务端与客户端启动后，客户端输入cat后，客户端和服务端显示如下：

```
-- 客户端
请输入数据
cat
客户端输入的是:cat,服务器响应的是: 猫
请输入数据

-- 服务端
服务器启动了....
客户端:frank/192.168.3.240已连接到服务器
客户端输入的是:cat,服务器翻译后是:猫
```

测试多个连接的情况

再启动一个客户端, 尝试连接服务器, 发现第二个客户端, 不能正确的和服务器进行通信.
分析原因, 是因为我们accecpt了一个请求之后, 就在一直while循环尝试read, 没有继续调用到accecpt, 导致不能接受新的请求.

我们当前的这个TCP, 只能处理一个连接, 这是不科学的.

客户端输入后，没有后续的显示。

2.3 多线程版本网络程序

TCP服务端

通过每个请求, 创建子进程的方式来支持多连接;

ServerThread 类为多线程类，**StringServer**类为调用类。

```
package com.store.frank.thread;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

/**
 * @author zhangmingming  braveheart1115@163.com
 * @date 2019年10月22日 14:56
 * @Description: 多线程HTTP具体实现类
 */
public class ServerThread implements Runnable {

    private int index;
    private Socket socket;

    public ServerThread(Socket socket, int i) {
        this.socket = socket;
        this.index = i;
    }

    // 任务是为一个用户提供服务
    @Override
    public void run() {
        try {
            try {
                // 读取客户端传过来信息的DataInputStream
                DataInputStream in = new DataInputStream(socket.getInputStream());
                // 向客户端发送信息的DataOutputStream
                DataOutputStream out = new DataOutputStream(socket.getOutputStream());
```

```
                while (true) {
                    // 读取来自客户端的信息
                    String accpet = in.readUTF();
                    System.out.println("第" + index + "个客户端发出消息: " + accpet);
                }
            } finally {// 建立连接失败的话不会执行socket.close();
                socket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
package com.store.frank.thread;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * @author zhangmingming  braveheart1115@163.com
 * @date 2019年10月22日 14:53
 * @Description: 多线程HTTP版本的服务端
 */
public class StringServer {

    // 提供服务
    public void service() {
        int i = 1;
        try {
            // 建立服务器连接,设定客户连接请求队列的长度
            ServerSocket server = new ServerSocket(8080, 3);
            while (true) {
                if (i <= 3) {
                    // 等待客户连接
                    Socket socket = server.accept();
                    System.out.println("第" + i + "个客户连接成功! ");
                    new Thread(new ServerThread(socket, i)).start();
                    i++;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        new StringServer().service();
    }
}
```

TCP客户端

```
package com.store.frank.thread;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.Random;
import java.util.Scanner;

/**
 * @author zhangmingming  braveheart1115@163.com
 * @date 2019年10月22日 14:59
 * @Description: 多线程HTTP版本的客户端
 */
public class StringClient {

    private String name = String.valueOf(new Random().nextInt(999999));//客户端的名字

    public void chat() {
        try {
            // 连接到服务器
            Socket socket = new Socket("localhost", 8080);
            // 读取服务器端传过来信息的DataInputStream
            DataInputStream in = new DataInputStream(socket.getInputStream());
            // 向服务器端发送信息的DataOutputStream
            DataOutputStream out = new DataOutputStream(socket.getOutputStream());
            // 装饰标准输入流，用于从控制台输入
            Scanner scanner = new Scanner(System.in);
            while (true) {
                System.out.println("请输入数据");
                String send = scanner.nextLine();
                // 把从控制台得到的信息传送给服务器
                out.writeUTF("客户端[" + name + "]:" + send);
                // 读取来自服务器的信息
                String accpet = in.readUTF();
                System.out.println(accpet);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        new StringClient().chat();
    }
}
```

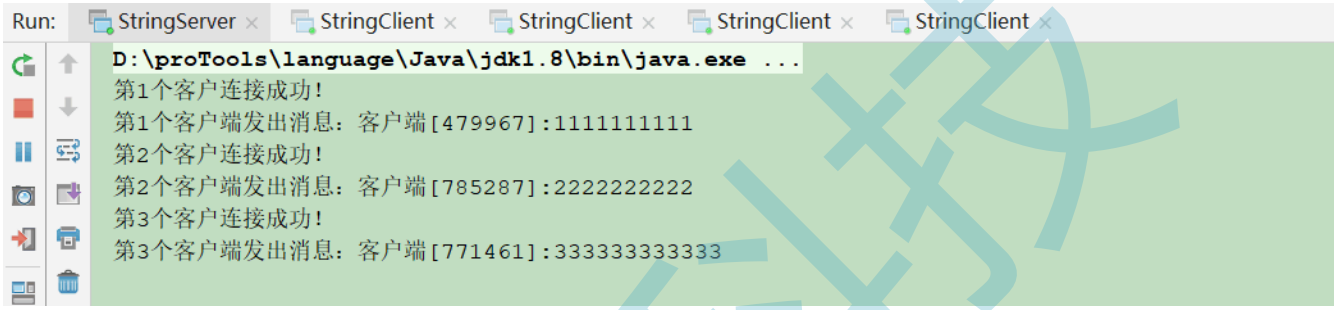
客户端启动多个与服务端显示如下：

```
-- 服务端
第1个客户连接成功!
第1个客户端发出消息：客户端[479967]:1111111111
第2个客户连接成功!
第2个客户端发出消息：客户端[785287]:2222222222
第3个客户连接成功!
第3个客户端发出消息：客户端[771461]:3333333333

-- 客户端
客户端第一个输入 1111111111，第二个输入 2222222222，第三个输入 3333333333
```

当打开第四个客户端输入的时候，服务端不显示任何信息。表明只能接收3个客户端的连接。

截图如下：



2.4 线程池版本的 TCP 服务器

同学们根据之前学过的线程池, 自行修改服务器为线程池版本. 下面是线程池版的服务端。

TCP服务端

```
package com.store.frank.threadPool;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/**
 * @author zhangmingming  braveheart1115@163.com
 * @date 2019年10月09日 14:53
 * @Description: 线程池版本的http请求服务端
 */
public class ThreadPoolServer {


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

```
        new ThreadPoolServer().run();
    }


    private void run() throws IOException {
        ServerSocket server = new ServerSocket(9999);
        ExecutorService executorService = Executors.newFixedThreadPool(10);
        // System.out.println("服务器启动了....");
        while (true) {
            Socket client = server.accept();
            executorService.execute(new Runnable() {
                public void run() {
                    try {
                        System.out.println(this.getClass().toString());
                        execute(client);
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            });
        }
    }

    /**
     * @author zhangmingming  braveheart1115@163.com
     * @date 2019/10/9 15:21
     * @Description: 服务器运行代码
     */
    public void execute(Socket client) throws IOException {
        String clientName = client.getInetAddress().getLocalHost().toString();
        System.out.println("客户端:" + clientName + "已连接到服务器");
        BufferedReader br = new BufferedReader(new
InputStreamReader(client.getInputStream()));
        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(client.getOutputStream()));
        String key = null; // key 客户端发送来的消息
        String value = null; // 服务器翻译后的结果
        while (true) {
            try {
                key = br.readLine();
                value = maps.get(key);
                if (null != key) {
                    if (null == value) {
                        value = "default";
                        System.out.println("客户端输入的是:" + key + ",服务端返回的是默认值:" +
value);

                    } else {
                        System.out.println("客户端输入的是:" + key + ",服务器翻译后是:" + value);
                    }
                    bw.write(value + "\n");
                    bw.flush();
                }
```

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


    // 定义map集合，用来存放需要翻译的单词。
    private static Map<String, String> maps = new HashMap<>();

    // 向map中添加数据
    static {
        maps.put("dog", "狗");
        maps.put("cat", "猫");
        maps.put("fish", "鱼");
        maps.put("bird", "鸟");
        maps.put("pig", "猪");
    }
}
```

TCP客户端

```
package com.store.frank.threadPool;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.net.Socket;
import java.util.Scanner;

/**
 * @author zhangmingming  braveheart1115@163.com
 * @date 2019年10月09日 14:52
 * @Description: 线程版本server对应的http请求客户端，与通用TCP客户端没有区别。
 */
public class ThreadPoolClient {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("127.0.0.1",9999);
            //构建IO
            InputStream is = s.getInputStream();
            OutputStream os = s.getOutputStream();
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(os));
            Scanner scanner=new Scanner(System.in);
            BufferedReader br = new BufferedReader(new InputStreamReader(is));
            String key = null; // 输入的单词
            String value = null; // 单词的翻译结果
            while (true){
                System.out.println("请输入数据");
                if(scanner.hasNext()) {
```

```
                    key = scanner.nextLine();
                    //向服务器端发送一条消息
                    bw.write(key+"\n");
                    bw.flush();
                    //读取服务器返回的消息
                    value = br.readLine();
                    System.out.println("客户端输入的是:"+key+",服务器翻译后是: "+value);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

线程池 优点

- 利用线程池管理并复用线程、控制最大并发数等。
- 实现任务线程队列缓存策略和拒绝机制。
- 实现某些与时间相关的功能，如定时执行、周期执行等。
- 隔离线程环境。比如，交易服务和搜索服务在同一台服务器上，分别开启两个线程池，交易线程的资源消耗明显要大；因此，通过配置独立的线程池，将较慢的交易服务与搜索服务隔开，避免个服务线程互相影响。

线程池 缺点

- 前期需要创建多个线程示例对象。
- 如果客户端连接少，会造成线程资源浪费。

TCP 和 UDP 对比

> - 可靠传输 vs 不可靠传输
> - 有连接 vs 无连接
> - 字节流 vs 数据报

更多细节 网络原理细讲