# Java Socket编程入门

## 1.必备知识

**TCP**是Tranfer Control Protocol的简称，即传输控制协议，基于TCP协议，可以进行有顺序的，无差错的数据流传输，当然，发送方和接收方之间必须建立连接，在建立连接后，双方可以双向通信。**UDP**是User Datagram Protocol的简称，即用户数据报协议，该协议是无连接的，每个数据报都是个独立的信息体，包含源地址和目的地址，它是无序的，不安全的，能否到达目的地，什么时候到达目的地，都无法保证，但它比TCP高效，多用于网络稳定的局域网通信。

Socket通常也称作"套接字"，用于描述IP地址和端口，是一个通信链的句柄。[应用程序](http://baike.so.com/doc/3417785.html)通常通过"套接字"向网络发出请求或者应答网络请求。当然，Java中的Socket并不是只支持TCP/IP，还包括SMTP甚至是自定义协议传输。

基于Java自身包实现消息方式的系统间通信的方式有四种：TCP/IP+BIO,TCP/IP+NIO ,UDP/IP+BIO,UDP/IP+NIO。本篇文章讲解的就是TCP/IP+BIO。

## 2.简单例子

为了实践，我们先建立两个Java项目，一个为客户端，项目名称为**JavaClientProject**，一个为服务端，项目名称为**JavaServerProject**。

下面先展现一个比较简单的Socket例子**：**

1.客户端项目ClientTest.java中的main方法:

**public** **static** **void** main(String[] args) {

Socket socket = **null**;

**try** {

// InetAddress.getLocalHost()为服务端IP地址,因为服务器就部署在本地，1314为服务端开启的端口号

socket = **new** Socket(InetAddress.*getLocalHost*(), 1314);

// socket.setSoTimeout(60000); // 设置超时时间为60秒

// 创建向服务器写入流的PrintWriter

PrintWriter send = **new** PrintWriter(socket.getOutputStream(), **true**);

// 创建读取服务器端返回流的BufferedReader

BufferedReader receive = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

// 向服务器发送字符串信息

send.println("Is me, Client!!");

// 阻塞读取服务端的返回信息。如果希望一段时间后就不阻塞了，那么要在创建Socket对象后调用socket.setSoTimeout(毫秒单位的超时时间)

System.*out*.println("接收到服务端的响应：" + receive.readLine());

System.*out*.println("客户端处理完毕！");

socket.close();

} **catch** (UnknownHostException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

**if**(socket != **null**)

socket.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

2.服务端项目ServerTest.java中的main方法:

**public** **static** **void** main(String[] args) {

ServerSocket serverSocket = **null**;

**try** {

// 创建对本地指定端口的监听，如果端口冲突，则抛出SocketException，并指定队列里面最多只能存放20个Socket。

serverSocket = **new** ServerSocket(1314, 20, InetAddress.*getLocalHost*());

// serverSocket.setSoTimeout(60000);

// 接收客户端建立连接的请求，并返回Socket对象，以便和客户端进行交互。

// 交互方式与客户端相同，也是通过Socket.getInputStream和Socket.getOutputStream

// 来进行读写操作，此方法会一直阻塞到有客户端发送建立连接的请求，如果希望次方法阻塞一定的时间，

// 则要在创建ServerSocket后调用setSoTimeout设置超时时间

Socket socket = serverSocket.accept();

// 创建读取客户端发送流的BufferedReader

BufferedReader receive = **new** BufferedReader(**new** InputStreamReader(

socket.getInputStream()));

// 创建向客户端写入流的PrintWriter

PrintWriter send = **new** PrintWriter(socket.getOutputStream(), **true**);

// 阻塞读取服务端的返回信息。如果希望一段时间后就不阻塞了，那么要在创建ServerSocket对象后调用其setSoTimeout(毫秒单位的超时时间)

System.*out*.println("接收到客户端的信息：" + receive.readLine());

// 向客户端发送字符串信息

send.println("I know you are Client!");

System.*out*.println("服务端处理完毕！");

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

**if**(serverSocket != **null**)

serverSocket.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

为了更方便的看到效果，最好客户端项目和服务端项目各用一个eclipse运行。先运行服务端的ServerTest.java，开始对1314端口进行监听（如果实在windows下，可以在CMD中输入netstat –ano 来查看已经打开的端口，其中肯定有1314），然再运行客户端项目的ClientTest.java，最后就可以看到如下控制台效果：

**--服务端控制台--**

接收到客户端的信息：Is me, Client!!

服务端处理完毕！

**--客户端控制台—**

接收到服务端的响应：I know you are Client!

客户端处理完毕！

这里需要注意的是，客户端用的是Socket。而服务端得先用ServerSocket来建立起端口的监听，而且可以用ServerSocket的accept()方法来获取一个Socket，当然accept()方法是阻塞的，除非客户端那边有新的Socket来连接到服务端监听的端口，accept()方法才会返回新的Socket，客户端向已经连接上的Socket发送消息accept()是不会被触发的。其实在服务端有个Socket队列，新连接上的Scoekt会依次放在里面，只要队列里面还有没有取出来的Socket，accept()每触发一次，就会从队列中取出一个Socket。

## 3.用连接池来管理开销，提高性能

由于服务端资源是有限的，如果客户端发来大量Socket连接，必定会导致客户端等待消息处理超时，因为上面的例子服务端处理Socket用的是单线程。但也不能每来一个Socket连接服务端就建立一个新的线程，这样很快会导致服务端资源耗尽。于是乎，我们可以在服务器端使用**线程池**来对线程进行管理。

而对于客户端，频繁的新建Socket和关闭Socket也是一笔不小的资源开销，最好的办法是建立一个**Socket连接池**，用完的Socket不是直接关闭，而是放入连接池中，当有新的消息需要发送，再从Socket连接池中把空闲的Socket连接取出来。

## 4.连接池的简单例子

下面，我们给出一个最最最简单的连接池的实现。

在客户端项目**JavaClientProject**中新建**com.manzhizhen.tcpip**包，该包下有如下几个类：

**SocketData.java** – 客户端和服务端沟通的数据对象

**TcpIpBIOClient.java** – 该类用来模拟客户端向服务端发送并发Socket请求

**TcpIpBIOClientSocketPool.java** – 客户端的Socket连接池的最最最简单的实现

**TcpIpBIOClientThread.java** – 客户端连接的线程实现（因为需要模拟并发，所以需要线程）

在服务端项目**JavaServerProject**中新建**com.manzhizhen.tcpip**包，该包下有如下几个类：

**SocketData.java** – 客户端和服务端沟通的数据对象（代码和客户端的一样）

**TcpIpBIOServer.java** – 该类用来模拟服务端监听客户端发送的请求，并做出回应

**TcpIpBIOServerThreadPool.java** – 服务端的线程连接池的最最最简单的实现

**TcpIpBIOServerThread.java** – 服务端的线程实现，由线程池进行统一管理

为了方便测试，我们可以把这两个项目用不同的Eclipse运行。先运行**JavaServerProject**中**TcpIpBIOServer.java**的main方法，来开启服务端的对本地1314端口的监听。然后再运行**JavaClientProject**中**TcpIpBIOClient.java**的main方法，可以发现服务端和客户端的控制台都有信息输出，说明两者之间通信成功。因为我们在客户端的**TcpIpBIOClient**的main方法中加入了Thread.*sleep*(9999999);代码段，这是因为如果客户端运行完了main方法，由于服务端还在不断的读取，却发现客户端未响应了（因为运行完main方法客户端就关了），服务端会抛出Connect reset的Socket异常，加入该代码段只是为了暂时避免。如果想多次尝试，可以关闭客户端应用，再次运行客户端**TcpIpBIOServer.java**的main方法来测试，因为服务端还未关闭，一直在监听。可以发现，如果客户端设置的并发数量大于Socket连接池的Socket数量，会有部分客户端线程暂时获取不到空闲可用的Socket，于是我们通过循环来每隔一段时间去重新调用Socket连接池的方法来尝试获取空闲Socket，直到超过Socket连接池设置的最大尝试次数为止，如果到了最大尝试次数还未获取到空闲的Socket，则报超时。

## 5.附录（客户端和服务端代码）

**客户端JavaClientProject代码如下：**

**--SocketData.java—-**

**package** com.manzhizhen.tcpip;

/\*\*

\* 客户端和服务器之间传输的数据对象

\* **@author** Manzhizhen

\*/

**public** **class** SocketData {

**private** **int** clientNum; // 客户端的初始化数据

**private** **int** serverNum; // 被服务端处理后的数据

**public** **int** getClientNum() {

**return** clientNum;

}

**public** **void** setClientNum(**int** clientNum) {

**this**.clientNum = clientNum;

}

**public** **int** getServerNum() {

**return** serverNum;

}

**public** **void** setServerNum(**int** serverNum) {

**this**.serverNum = serverNum;

}

@Override

**public** String toString() {

**return** "(" + clientNum + "~" + serverNum +")";

}

/\*\*

\* 将字符串转换为SocketData对象

\* Socket是不能发送对象的，只能将对象转化成字符串发送

\* **@param** socketData

\* **@return**

\*/

**public** String getStrFromObj() {

**return** clientNum + "~" + serverNum;

}

/\*\*

\* 将字符串转换为SocketData对象

\* 该方法用于把经过getStrFromObj(SocketData socketData)

\* 的字符串转化成SocketData对象

\* **@param** str

\* **@return**

\*/

**public** **static** SocketData getObjFromStr(String str) {

SocketData socketData = **new** SocketData();

String[] strs = str.trim().split("~");

socketData.setClientNum(Integer.*valueOf*(strs[0]));

socketData.setServerNum(Integer.*valueOf*(strs[1]));

**return** socketData;

}

}

**--TcpIpBIOClient.java—-**

**package** com.manzhizhen.tcpip;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.PrintWriter;

**import** java.net.InetAddress;

**import** java.net.Socket;

**import** java.net.UnknownHostException;

/\*\*

\* 客户端的入口

\* TCP/IP+BIO

\* 基于Java自身包实现消息方式的系统间通信的方式有四种：

\* TCP/IP+BIO TCP/IP+NIO UDP/IP+BIO UDP/IP+NIO

\* **@author** Manzhizhen

\*

\*/

**public** **class** TcpIpBIOClient {

**public** **static** **void** client() {

**try** {

// 初始化Socket连接池

TcpIpBIOClientSocketPool.*init*();

// 自增用来作为发送的数据

**int** startNum = 10000;

// 模拟300个并发请求

**for**(**int** i = 0; i < 300; i++) {

SocketData socketData = **new** SocketData();

socketData.setClientNum(startNum + i);

// 每次请求都会创建一个新的线程

Runnable runnable = **new** TcpIpBIOClientThread(socketData);

**new** Thread(runnable).start();

}

**try** {

// 处理完休息一下

Thread.*sleep*(9999999);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

} **catch** (UnknownHostException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**public** **static** **void** main(String[] args) {

*client*();

}

}

**-- TcpIpBIOClientSocketPool.java—-**

**package** com.manzhizhen.tcpip;

**import** java.io.IOException;

**import** java.net.InetAddress;

**import** java.net.Socket;

**import** java.net.UnknownHostException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.Hashtable;

**import** java.util.Map;

/\*\*

\* 客户端Socket连接池

\* **@author** Manzhizhen

\*

\*/

**public** **class** TcpIpBIOClientSocketPool {

**private** **static** **int** *MAX\_POOL\_NUM* = 50; // Socket连接池中Socket最大数量

**public** **static** **int** *TIMEOUT* = 60000; // 设置超时间为60秒

**public** **static** **int** *MAX\_TRY\_TIMES* = 10; // 如果客户端没有获取到空闲的Scoket，尝试获取Scoket的次数

**private** **static** **boolean** *INIT\_STATUS* = **false**; // 连接池的初始化状态

**private** **static** InetAddress *IP\_ADDRESS*; // 服务器端IP地址

**private** **static** **int** *PORT*; // 服务器端口号

**public** **static** Map<Socket, Boolean> *socketMap* = **new** Hashtable<Socket, Boolean>(*MAX\_POOL\_NUM*); // true表示该Socket可以使用

**private** TcpIpBIOClientSocketPool() {

}

/\*\*

\* 初始化Socket连接池

\* **@throws** IOException

\* **@throws** UnknownHostException

\*/

**public** **static** **void** init() **throws** UnknownHostException, IOException {

**if**(*INIT\_STATUS*) {

**return** ;

}

*IP\_ADDRESS* = InetAddress.*getLocalHost*(); // 设置服务器端IP地址，因为服务器就在本地，所以使用本地IP地址。

*PORT* = 1314; // 设置服务器端口号

**for**(**int** i = *MAX\_POOL\_NUM*; i > 0; i--) {

Socket socket = **new** Socket(*IP\_ADDRESS*, *PORT*);

*socketMap*.put(socket, **true**);

}

// 如果没抛异常，设置初始化状态为true

*INIT\_STATUS* = **true**;

}

/\*\*

\* 释放一个Socket，该Socket可以供其他客户端使用

\* **@param** socket

\*/

**public** **static** **void** freeSocket(Socket socket) {

**if**(!*INIT\_STATUS*) {

System.*out*.println("连接池未初始化！");

**return** ;

}

**if**(socket == **null**) {

**return** ;

}

**if**(*socketMap*.get(socket) != **null**) {

*socketMap*.put(socket, **true**);

System.*out*.println(TcpIpBIOClientSocketPool.*currentTime*() + " 已经释放了一个Socket");

}

}

/\*\*

\* 获得一个可用的Socket，如果没有可用Socket，则返回null

\* **@param** socket

\*/

**public** **static** **synchronized** Socket getSocket() {

**if**(!*INIT\_STATUS*) {

System.*out*.println("连接池未初始化！");

**return** **null**;

}

**for**(Socket socket : *socketMap*.keySet()) {

**if**(*socketMap*.get(socket)) {

*socketMap*.put(socket, **false**);

**return** socket;

}

}

**return** **null**;

}

/\*\*

\* 获取当前时间

\* **@return**

\*/

**public** **static** String currentTime() {

**return** **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(**new** Date());

}

}

**-- TcpIpBIOClientThread.java—-**

**package** com.manzhizhen.tcpip;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.PrintWriter;

**import** java.net.Socket;

/\*\*

\* 客户端线程

\* **@author** Manzhizhen

\*

\*/

**public** **class** TcpIpBIOClientThread **implements** Runnable {

**private** SocketData socketData;

**private** Socket socket = **null**;

**public** TcpIpBIOClientThread(SocketData socketData) {

**this**.socketData = socketData;

}

@Override

**public** **void** run() {

**try** {

**boolean** isGet = **false**;

**for**(**int** i = 0; i < TcpIpBIOClientSocketPool.*MAX\_TRY\_TIMES*; i++) {

**if**((socket = TcpIpBIOClientSocketPool.*getSocket*()) == **null**) {

Thread.*sleep*(1000); // 如果没获取到空闲的Socket，则1秒后再尝试获取

System.*out*.println(TcpIpBIOClientSocketPool.*currentTime*() + **this** + " 尝试再次获取Socket！");

} **else** {

System.*out*.println(TcpIpBIOClientSocketPool.*currentTime*() + **this** + " 成功获取Socket！");

isGet = **true**;

**break** ;

}

}

**if**(!isGet) {

System.*err*.println(TcpIpBIOClientSocketPool.*currentTime*() + " 客户端获取Socket超时！");

**return** ;

}

// socket.setSoTimeout(TcpIpBIOClientSocketPool.TIMEOUT);

// 创建向服务器写入流的PrintWriter

PrintWriter send = **new** PrintWriter(socket.getOutputStream(), **true**);

// 创建读取服务器端返回流的BufferedReader

BufferedReader receive = **new** BufferedReader(**new** InputStreamReader(

socket.getInputStream()));

// 向服务器发送数据对象

send.println(socketData.getStrFromObj());

// 阻塞读取服务端的返回信息。如果希望一段时间后就不阻塞了，那么要在创建Socket对象后调用socket.setSoTimeout(毫秒单位的超时时间)

socketData = SocketData.*getObjFromStr*(receive.readLine());

System.*out*.println(TcpIpBIOClientSocketPool.*currentTime*() + **this** + " 客户端接收完毕：" + socketData);

} **catch** (IOException e) {

System.*err*.println(TcpIpBIOClientSocketPool.*currentTime*() + **this** + " 客户端异常：" + socketData);

e.printStackTrace();

} **catch** (InterruptedException e) {

System.*err*.println(TcpIpBIOClientSocketPool.*currentTime*() + **this** + " 客户端异常：" + socketData);

e.printStackTrace();

} **finally** {

// 释放客户端那有限的Socket资源

TcpIpBIOClientSocketPool.*freeSocket*(socket);

}

}

**public** SocketData getSocketData() {

**return** socketData;

}

**public** Socket getSocket() {

**return** socket;

}

}

**服务端JavaClientProject代码如下：**

**--SocketData.java—- （同客户端）**

**--TcpIpBIOServer.java—-**

**package** com.manzhizhen.tcpip;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.PrintWriter;

**import** java.net.InetAddress;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**import** java.net.UnknownHostException;

**public** **class** TcpIpBIOServer {

**public** **static** **void** server() {

**try** {

TcpIpBIOServerThreadPool.*init*();

// 创建对本地指定端口的监听，如果端口冲突，则抛出SocketException。

ServerSocket serverSocket = **new** ServerSocket(1314);

// serverSocket.setSoTimeout(60000); // 这里不设置超时时间，以便可以不停的运行客户端来测试

**boolean** isDeal;

// 因为不知道什么时候有会有Socket来连接，所以不断循环

**while**(**true**) {

// 只有当新的Socket来连接，就会触发accept()

Socket socket = serverSocket.accept();

TcpIpBIOServerThread runnable = **null**;

isDeal = **false**;

**for**(**int** i = 0; i < TcpIpBIOServerThreadPool.*MAX\_TRY\_TIMES*; i++) {

**if**((runnable = TcpIpBIOServerThreadPool.*getThread*()) == **null**) {

Thread.*sleep*(500); // 如果没获取到空闲的线程，则0.5秒后再尝试获取

System.*out*.println(TcpIpBIOServerThreadPool.*currentTime*() + " 尝试再次获取线程！");

} **else** {

// 给获取到的Socket分配到线程

runnable.setSocket(socket);

System.*out*.println(TcpIpBIOServerThreadPool.*currentTime*() + " 获取线程成功！");

isDeal = **true**;

**break** ;

}

}

// 如果在尝试了指定次数后，仍然无法获取到空闲的线程，则报超时。

**if**(!isDeal) {

System.*err*.println(TcpIpBIOServerThreadPool.*currentTime*() + " 服务端获取Thread超时！");

**continue**;

}

// 处理客户端请求

**new** Thread(runnable).start();

}

} **catch** (UnknownHostException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**public** **static** **void** main(String[] args) {

*server*();

}

}

**--TcpIpBIOServerThread.java—-**

**package** com.manzhizhen.tcpip;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.PrintWriter;

**import** java.net.Socket;

**import** java.net.SocketException;

/\*\*

\* 服务端处理Socket的线程

\*

\* **@author** Administrator

\*

\*/

**public** **class** TcpIpBIOServerThread **implements** Runnable {

**private** SocketData socketData = **new** SocketData();

**private** Socket socket = **null**;

**public** TcpIpBIOServerThread() {

}

@Override

**public** **void** run() {

**try** {

// socket.setSoTimeout(TcpIpBIOServerThreadPool.READ\_TIMEOUT);

// 创建读取客户端发送的BufferedReader

BufferedReader read = **new** BufferedReader(**new** InputStreamReader(

socket.getInputStream()));

// 创建给客户端响应写入流的PrintWriter

PrintWriter sendOut = **new** PrintWriter(socket.getOutputStream(), **true**);

// 阻塞读取客户端的发送的信息。如果希望一段时间后就不阻塞了，那么要在创建Socket对象后调用socket.setSoTimeout(毫秒单位的超时时间)

String readStr = **null**;

**while**(**true**) {

// 一旦客户端项目的TcpIpBIOClient.java的main方法运行完，客户端会失去响应，

// 此时下面的语句会抛出java.net.SocketException: Connection reset异常

// 为了避免这样发生，可以在客户端的main方法加入sleep()让客户端进程睡眠。

readStr = read.readLine();

**if**(readStr != **null**) {

socketData = SocketData.*getObjFromStr*(readStr);

// 服务端处理数据

dealData(socketData);

// 向客户端发送响应数字信息

sendOut.println(socketData.getStrFromObj());

System.*out*.println(TcpIpBIOServerThreadPool.*currentTime*() + " 服务端处理完毕完毕：" + socketData);

}

}

} **catch** (SocketException e) {

e.printStackTrace();

} **catch** (IOException e) {

System.*err*.println(TcpIpBIOServerThreadPool.*currentTime*() + " 服务端异常：" + socketData);

e.printStackTrace();

} **finally** {

// 使用完后释放自己

TcpIpBIOServerThreadPool.*freeThread*(**this**);

// 注意，这里不必关闭Socket，因为这个Socket是客户端Socket连接池维护的，

// 如果这时候关了，客户端就无法通过这个Socket来和服务端通信了

}

}

/\*\*

\* 对数据进行业务处理方法

\* **@param** socketData2

\*/

**private** **void** dealData(SocketData socketData) {

// 将客户端发送过来的数据进行简单处理

socketData.setServerNum(socketData.getClientNum() \* 2);

}

**public** **void** setSocket(Socket socket) {

**this**.socket = socket;

}

}

**--TcpIpBIOServerThreadPool.java—-**

**package** com.manzhizhen.tcpip;

**import** java.io.IOException;

**import** java.net.UnknownHostException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.Hashtable;

**import** java.util.Map;

/\*\*

\* 服务端线程连接池

\* **@author** Administrator

\*

\*/

**public** **class** TcpIpBIOServerThreadPool {

**public** **static** **int** *MAX\_POOL\_NUM* = 50; // 连接池数量

**public** **static** **int** *ACCEPT\_TIMEOUT* = 99999999; // 设置ServerSocket监听的超时时间

**public** **static** **int** *READ\_TIMEOUT* = 60000; // 设置Socket读取数据的超时时间为60秒

**public** **static** **int** *MAX\_TRY\_TIMES* = 20; // 如果服务端没有获取到空闲的线程，尝试获取线程的次数

**private** **static** **boolean** *INIT\_STATUS* = **false**; // 连接池的初始化状态

**public** **static** Map<TcpIpBIOServerThread, Boolean> *threadMap* = **new** Hashtable<TcpIpBIOServerThread, Boolean>(*MAX\_POOL\_NUM*); // true表示该线程可以使用

**private** TcpIpBIOServerThreadPool() {

}

/\*\*

\* 初始化线程连接池

\* **@throws** IOException

\* **@throws** UnknownHostException

\*/

**public** **static** **void** init() **throws** UnknownHostException, IOException {

**if**(*INIT\_STATUS*) {

**return** ;

}

**for**(**int** i = *MAX\_POOL\_NUM*; i >= 0; i--) {

TcpIpBIOServerThread runnable = **new** TcpIpBIOServerThread();

*threadMap*.put(runnable, **true**);

}

// 如果没抛异常，设置初始化状态为true

*INIT\_STATUS* = **true**;

}

/\*\*

\* 释放一个线程，该线程可以供其他请求使用

\* **@param** socket

\*/

**public** **static** **void** freeThread(TcpIpBIOServerThread runnable) {

**if**(!*INIT\_STATUS*) {

System.*out*.println("连接池未初始化！");

**return** ;

}

**if**(runnable == **null**) {

**return** ;

}

**if**(*threadMap*.get(runnable) != **null**) {

System.*out*.println(*currentTime*() + runnable + " 释放线程成功！ ");

*threadMap*.put(runnable, **true**);

}

}

/\*\*

\* 获得一个可用的线程，如果没有可用线程，则返回null

\* **@param** socket

\*/

**public** **static** TcpIpBIOServerThread getThread() {

**if**(!*INIT\_STATUS*) {

System.*out*.println("连接池未初始化！");

**return** **null**;

}

**for**(TcpIpBIOServerThread runnable : *threadMap*.keySet()) {

**if**(*threadMap*.get(runnable)) {

*threadMap*.put(runnable, **false**);

**return** runnable;

}

}

**return** **null**;

}

/\*\*

\* 获取当前时间

\* **@return**

\*/

**public** **static** String currentTime() {

**return** **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(**new** Date());

}

}