ServerSocket与Socket

**ServerSocket** This class implements **server sockets.**

**Socket** This class implements **client sockets** (also called just "sockets").

# ServerSocket服务端套接字

## 继承关系

public class **ServerSocket** extends Object implements **Closeable**



**Since:JDK1.0**

All Implemented Interfaces: Closeable, AutoCloseable

Direct Known Subclasses: **SSLServerSocket**

## 功能特性

ServerSocket实现了服务器端的socket。等待请求，并根据请求执行相应操作然后返回结果。

This class implements **server sockets**. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.

The actual work of the server socket is performed by an instance of the **SocketImpl** class. An application can change **the socket factory** that creates the socket implementation to configure itself to create sockets appropriate to the local firewall.

## 构造方法

### **ServerSocket**()

Creates an unbound server socket.

### **ServerSocket**(int port)

Creates a server socket, bound to the specified port.

### **ServerSocket**(int port, int backlog)

Creates a server socket and binds it to the specified local port number, with the specified backlog.

### **ServerSocket**(int port, int backlog, InetAddress bindAddr)

Create a server with the specified port, listen backlog, and local IP address to bind to.

## 一般方法

### **accept()：最最重要的方法**

public **Socket** **accept()** throws IOException

Listens for a connection to be made to this socket and accepts it. The method blocks until a connection is made.(**阻塞式方法**)

A new Socket s is created and, if there is a security manager, the security manager's **checkAccept** method is called with **s.getInetAddress().getHostAddress()** and **s.getPort()** as its arguments to ensure the operation is allowed. This could result in a **SecurityException**.

Returns: the new Socket

Throws:

IOException - if an I/O error occurs when waiting for a connection.

SecurityException - if a security manager exists and its checkAccept method doesn't allow the operation.

SocketTimeoutException - if a timeout was previously set with setSoTimeout and the timeout has been reached.

IllegalBlockingModeException - if this socket has an associated channel, the channel is in non-blocking mode, and there is no connection ready to be accepted

See Also: SecurityManager.checkAccept(java.lang.String, int)

### bind方法

void **bind**(SocketAddress endpoint)

Binds the ServerSocket to a specific address (IP address and port number).

void **bind**(SocketAddress endpoint, int backlog)

Binds the ServerSocket to a specific address (IP address and port number).

### getXxx方法

ServerSocketChannel getChannel()

Returns the unique **ServerSocketChannel** object associated with this socket, if any.

InetAddress getInetAddress()

Returns the local address of this server socket.

int getLocalPort()

Returns the port number on which this socket is listening.

SocketAddress getLocalSocketAddress()

Returns the address of the endpoint this socket is bound to.

int getReceiveBufferSize()

Gets the value of the SO\_**RCVBUF** option for this ServerSocket, that is the proposed buffer size that will be used for Sockets accepted from this ServerSocket.

boolean **getReuseAddress**()

Tests if SO\_REUSEADDR is enabled.

int getSoTimeout()

Retrieve setting for SO\_**TIMEOUT**.

### setXxx方法

void **setPerformancePreferences**(int connectionTime, int latency, int bandwidth)

Sets performance preferences for this ServerSocket.

void **setReceiveBufferSize**(int size)

Sets a default proposed value for the SO\_RCVBUF option for sockets accepted from this ServerSocket.

void **setReuseAddress**(boolean on)

Enable/disable the SO\_REUSEADDR socket option.

static void **setSocketFactory**(SocketImplFactory fac)

Sets the server socket implementation factory for the application.

void **setSoTimeout**(int timeout)

Enable/disable SO\_TIMEOUT with the specified timeout, in milliseconds.

### void close() Closes this socket.

### isXxx

boolean isBound()

Returns the binding state of the **ServerSocket**.

boolean isClosed()

Returns the closed state of the ServerSocket.

# Socket客户端套接字

## 继承关系

public class **Socket** extends Object implements Closeable

All Implemented Interfaces: Closeable, AutoCloseable

Direct Known Subclasses: SSLSocket



## 功能特性

This class implements **client sockets** (also called just "sockets"). A socket is an endpoint for communication **between two machines**.

The actual work of the socket is performed by an instance of the **SocketImpl** class. An application, by changing the socket factory that creates the socket implementation, can configure itself to create sockets appropriate to the local firewall.

## 构造方法

### Socket()

Creates an unconnected socket, with the system-default type of SocketImpl.

### Socket(**InetAddress address, int port**)

Creates a stream socket and connects it to the specified port number at the specified IP address.

### Socket(InetAddress host, int port, boolean stream)

**Deprecated**. Use **DatagramSocket** instead for UDP transport.

### Socket(InetAddress address, int port, *InetAddress localAddr, int localPort)*

Creates a socket and connects it to the specified remote address on the specified remote port.

### Socket(**Proxy proxy**)

Creates an unconnected socket, specifying the type of proxy, if any, that should be used regardless of any other settings.

### protected Socket(*SocketImpl impl*)

Creates an unconnected Socket with a user-specified SocketImpl.

### **Socket(String host, int port)**

Creates a stream socket and connects it to the specified port number on the named host.

### **Socket**(String host, int port, boolean stream)

**Deprecated**. Use DatagramSocket instead for UDP transport.

### **Socket**(String host, int port, InetAddress localAddr, int localPort)

Creates a socket and connects it to the specified remote host on the specified remote port.

## 方法介绍

### **connect连接服务器**

void **connect**(SocketAddress endpoint)

Connects this socket to the server.

void connect(SocketAddress endpoint, int timeout)

Connects this socket to the server with a specified timeout value.

### **getInputStream与getOutputStream**(最重要的方法)

InputStream getInputStream()

Returns an input stream for this socket.

OutputStream getOutputStream()

Returns an output stream for this socket.

### getChannel

SocketChannel getChannel()

Returns the unique **SocketChannel** object associated with this socket, if any.

### bind方法

void bind(SocketAddress bindpoint)

Binds the socket to a local address.

### shutdownInput与shutdownOutput

void **shutdownInput**()

Places the input stream for this socket at "end of stream".

void **shutdownOutput**()

Disables the output stream for this socket.

### getXxx方法

### setXxx方法

### isXxx方法

# ServerSocket与Socket示例:

## 几点注意：

**客户端Client**直接利用Socket的**getInputStream和getOutputStream**方法获取对应的输入流和输出流，服务端Server需要通过accept()方法获取socket，然后利用socket再获取输入流和输出流。无论输入端和输出端，一般都需要进行包装：

对于**输入流**：

reader = new **BufferedReader**(new **InputStreamReader**(socket.getInputStream()))

对于**输出流**：

pw = new **PrintWriter**(socket.**getOutputStream**(),**true**);//自动刷新

注意：若没有设置autoFlush为true，需要手动调用flush方法刷新，否则对方收不到。

在BufferedReader读取数据的时候，不能通过while((line = br.readLine()) != null){}来跳出循环，因为readLine方法会一直等待对方写内容，一直阻塞不会返回null。那怎么判断结束呢？人为设定结束标志，如当读取到end的时候自动退出。

while(true){

System.out.println("waiting....");

**line = reader.readLine();**

System.out.println("get the data!");

//*if(line == null) break;//readLine始终不会返回null，只会一直等待*

**if(line.equals("end")) break;//读取到end表示结束**

System.out.println("收到的内容："+ line);

}

另外，**流的关闭问题**：一方写完数据，不能立刻关闭，因为对方可能还未读取。也就是说读和写必须同时存在。

不管是客户端还是服务端在通过Socket读取数据的时候，在你的数据传输完成后，发送一个结束标记(比如:end)，这样在你的 while循环里面判断一下数据是否已经传输完毕，完毕后跳出循环，否则由于你的程序就会一直阻塞在 readLine()这里，因为你的 socket还没有断开，会一直等待你写数据。

## 示例1：简单实现

客户端：

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

public class MyClient {

public static void main(String[] args) {

Socket client = null;

BufferedReader reader = null;

PrintWriter pw = null;

try {

**client = new Socket("127.0.0.1", 9090);**

pw = new PrintWriter(client.getOutputStream(),true);

reader = new BufferedReader(new InputStreamReader(client.getInputStream()));

pw.println("AAAAAAA");

pw.println("BBBBBBBB");

pw.println("end");

String line = null;

while(true){

System.out.println("waiting....");

line = reader.readLine();

System.out.println("get the data!");

if("end".equals(line)) break;

System.out.println("br.readLine() = " + line);

}

}catch(Exception e){

System.out.println("异常了");

}finally{

try {

reader.close();

pw.close();

client.close();

}catch(Exception e){

System.out.println(e.toString());

}

}

}

}

服务端：

import java.io.\*;

import java.net.ServerSocket;

import java.net.Socket;

public class MyServer {

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

ServerSocket server = null;

try {

server = new ServerSocket(9090);

Socket socket = null;

BufferedReader reader = null;

PrintWriter pw = null;

**while (true) {//服务端一直等待**

**socket = server.accept();**

System.out.println("收到请求了");

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

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

String line = null;

while(true){

System.out.println("waiting....");

line = reader.readLine();

System.out.println("get the data!");

if(line.equals("end")) break;//读取到end表示结束

System.out.println("收到的内容："+ line);

}

System.out.println("跳出循环了");

pw.println("你好，收到了:这是给你的Response!");

pw.println("end");

System.out.println("结束了一次请求一次回复");

reader.close();

pw.close();

socket.close();

}

}catch(Exception e){

if(!server.isClosed())

server.close();

System.out.println("服务器关闭");

}finally{

}

}

}

## 改进：一个请求对应一个线程

客户端：不变。

服务端：**原理就是把具体的处理放到Runnable的run方法中**。

存在问题：如果请求过多，就会创建很多线程，这样就会导致服务器崩溃，因此利用线程池改造可以得到较好的效果。

package July5;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class MyServerNewThread {

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

ServerSocket server = null;

try {

server = new ServerSocket(9090);

System.out.println("服务器开启了......");

Socket socket = null;

while (true) {

socket = server.accept();

System.out.println("收到一个请求");

new Thread(new Task(socket)).start();//开启一个线程去处理这个请求

}

} catch (Exception e) {

} finally {

}

}

}

class Task implements Runnable {

**private Socket socket;**

public Task(Socket socket) {

this.socket = socket;

}

@Override

public void run() {

BufferedReader reader = null;

PrintWriter pw = null;

try {

reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

pw = new PrintWriter(socket.getOutputStream(), true);

String line = null;

while (true) {

System.out.println("waiting....");

line = reader.readLine();

System.out.println("get the data!");

if (line.equals("end")) break;//读取到end表示结束

System.out.println("收到的内容：" + line);

}

pw.println("你好，收到了:这是给你的Response!");

pw.println("end");//发送结束标志

System.out.println("结束了一次请求一次回复");

} catch (Exception e) {

System.out.println("服务器关闭");

} finally {

try {

reader.close();

pw.close();

socket.close();

} catch (Exception e) {

System.out.println(e.toString());

}

}

}

}

## 利用线程池ThreadPool实现：

存在问题：如果请求过多，就会创建很多线程，这样就会导致服务器崩溃，因此利用线程池改造可以得到较好的效果。

**Task和上述的一样。**

import java.net.ServerSocket;

import java.net.Socket;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class ThreadPoolServer {

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

ServerSocket server = null;

ExecutorService threadPool = null;

try {

threadPool = Executors.newFixedThreadPool(100,Executors.defaultThreadFactory());

server = new ServerSocket(9090);

System.out.println("服务器开启了......");

Socket socket = null;

while (true) {

socket = server.accept();

System.out.println("收到一个请求");

**threadPool.submit(new Task(socket));//从线程池中获取一个线程去处理这个请求**

}

} catch (Exception e) {

} finally {

}

}

}

# SocketImpl

## 继承关系

public abstract class **SocketImpl** extends **Object** implements **SocketOptions**



**All Implemented Interfaces: SocketOptions**

## 功能特性

The abstract class SocketImpl is **a common superclass** of all classes that actually implement sockets. It is used to create both client and server sockets.

对于Socket和ServerSocket都是基于SocketImpl抽象类实现的。

A "plain" socket implements these methods exactly as described, without attempting to go through a firewall or proxy.

Since: JDK1.0

## 属性字段

protected **InetAddress** address

The IP address of the remote end of this socket.

protected **FileDescriptor** fd

The file descriptor object for this socket.

protected int **localport**

The local port number to which this socket is connected.

protected int **port**

The port number on the remote host to which this socket is connected.

## 大部分方法都是protected abstract

Modifier and Type Method and Description

protected abstract void accept(SocketImpl s)

Accepts a connection.

protected abstract int **available**()

Returns the number of bytes that can be read from this socket without blocking.

protected abstract void bind(InetAddress host, int port)

Binds this socket to the specified local IP address and port number.

protected abstract void close()

Closes this socket.

protected abstract void connect(InetAddress address, int port)

Connects this socket to the specified port number on the specified host.

protected abstract void connect(SocketAddress address, int timeout)

Connects this socket to the specified port number on the specified host.

protected abstract void connect(String host, int port)

Connects this socket to the specified port on the named host.

protected abstract void create(boolean stream)

Creates either a stream or a datagram socket.

protected FileDescriptor getFileDescriptor()

Returns the value of this socket's fd field.

protected InetAddress getInetAddress()

Returns the value of this socket's address field.

protected abstract InputStream getInputStream()

Returns an input stream for this socket.

protected int getLocalPort()

Returns the value of this socket's localport field.

protected abstract OutputStream getOutputStream()

Returns an output stream for this socket.

protected int getPort()

Returns the value of this socket's port field.

protected abstract void listen(int backlog)

Sets the maximum queue length for incoming connection indications (a request to connect) to the count argument.

protected abstract void sendUrgentData(int data)

Send one byte of urgent data on the socket.

protected void setPerformancePreferences(int connectionTime, int latency, int bandwidth)

Sets performance preferences for this socket.

protected void shutdownInput()

Places the input stream for this socket at "end of stream".

protected void shutdownOutput()

Disables the output stream for this socket.

protected boolean supportsUrgentData()

Returns whether or not this SocketImpl supports sending urgent data.

String toString()

Returns the address and port of this socket as a String.