基于TCP协议的Socket通信

1. TCP协议：面向连接的、安全可靠、效率低。

既然是面向连接的，则就会有请求与响应即**Request和Response**。

1. Socket编程中用到的类：

Socket、ServerSocket

**服务器端：利用ServerSocket。**

**客户端：利用Socket。**

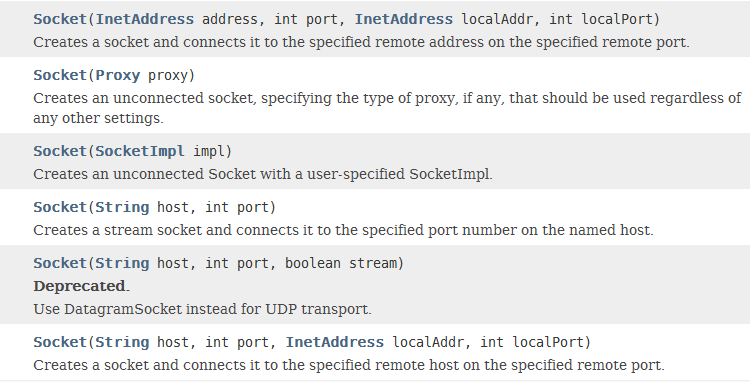
1. **Socket类**存在于java.net包中，直接继承于java.lang.Object，有一个直接子类SSLSocket类，**Socket类用于实现客户端的套接字，ServerSocket用于实现服务器的套接字**。
2. IP地址标识Internet上的一台主机，端口号标识一台计算机上正在运行的**进程**，IP地址和端口号组合得出一个**网络套接字**。
3. **Socket类的构造方法**：
4. **Socket（）**：创建一个无连接的Socket对象；之后通过**connect方法**进行连接。



1. 利用**网络地址对象InetAddress**和**端口号**建立一个连接的**套接字对象**。（比较常用）



1. 其他的构造方法：



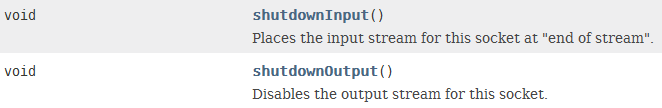
1. **Socket类对应两个流**，一个是InputStream，一个是OutputStream。

分别通过getInputStream方法和getOutputStream方法获取。  

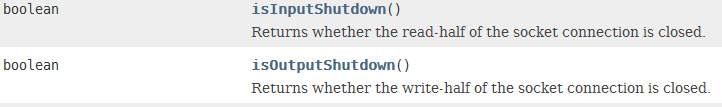



**关于流的其他方法：**

**关闭流：**



**判断是否关闭流：**



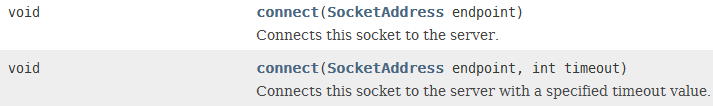
1. **Socket的方法**：
2. **setSoTimeout(itn timeout)**:设置请求连接的等待时间值，超过时间会抛出SocketTimeoutException异常。如果不设置请求时间，则会一直阻塞式等待，直到建立连接。



有设置方法，就会有获取方法：



1. **connect方法**：需要传入一个**SocketAddress对象**（第二个指定了等待时间）



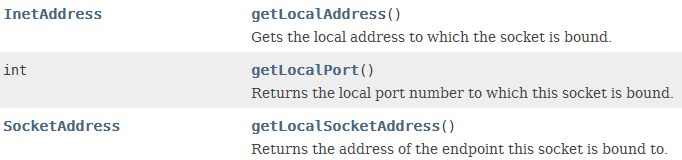
1. bind方法：



1. getXxx方法:









1. setXxx方法：。。。。。。

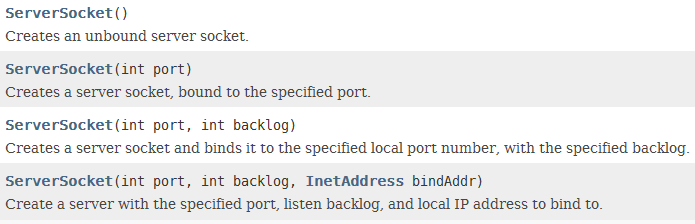


1. ServerSocket类：存在于java,net包中，直接继承于java.lang.Object类。具有一个直接子类SSLServerSocket。**ServerSocket类用于实现服务器的套接字。**

注意：**Socket类与ServerSocket类**是并列关系，不是继承关系。

1. **ServerSocket类**的**构造方法**：

**需要指定一个端口号。**



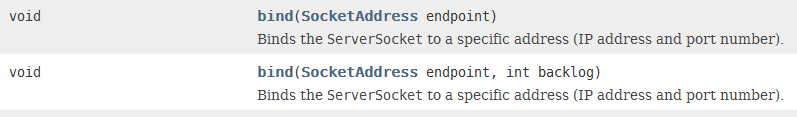
1. ServetSocket类的静态方法：



1. **ServetSocket类**的成员方法：
2. **accept（）方法**：监听客户端的请求连接，此方法为**阻塞式方法**。如果设置了等待请求时间，则时间一到，抛出异常SocketTimeoutException，否则一直等待。



1. **bind方法**：



1. setSoTimeout和getSoTimeout方法：



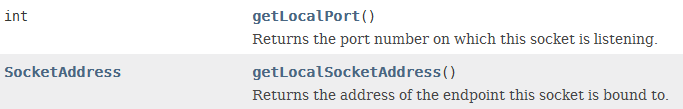


1. close方法

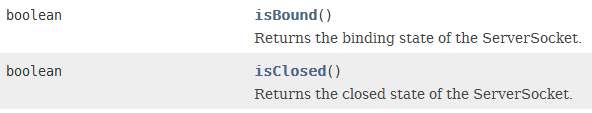


1. getXxx方法





1. 判断方法：



1. **套接字的连接**：

就是客户机的套接字对象（Socket）和服务器的套接字对象**（ServerSocket）**通过**输入输出流**连接在一起，主要分三个步骤：

1. 服务器端建立ServerSocket对象
2. 服务器端采用**ServerSocket对象**，ServerSocket对象负责等待客户机请求建立套接字连接。也就是说，服务器首先需要建立一个等待客户机请求建立套接字连接的ServerSocket对象。一般利用构造方法ServerSocket（int port）创建一个ServerSocket对象。（可能会抛出IOException异常）
3. 再调用**accept方法**创建与客户端的套接字连接对象。accept（）方法返回一个**Socket对象（不是ServerSocket对象）**。accept（）方法是一个**阻塞式方法**，会抛出多个异常。
4. 客户机创建Socket对象

客户机端利用Socket对象，**客户机建立Socket对象的过程就是向服务器端发出套接字连接请求**，如服务器端相应的端口上有ServerSocket对象正在调用accept方法等待客户机，那么双方法的套接字对象Socket也就建立成功了。也可以先通过空参数构造方法建立一个未连接的Socket对象，其次**再调用connect方法**，建立连接。调用**connect方法**，需要传入**InetSocketAddress对象**。

1. **建立流连接**

客户机和服务器的套接字对象分别创建完成之后，还必须进行输入输出流的连接。**服务器端的Socket对象（accept方法返回的Socket对象）调用方法getOutputStream（）方法获得输出流指向客户端Socket对象调用getInputStream（）获得的那个输入流。**同理，服务器端的Socket对象（accept方法返回的Socket对象）调用方法getInputStream（）方法获得输入流指向客户端Socket对象调用getOutputStream（）获得的那个输出流。

**当服务器通过向这个输出流写入信息时，客户机就会通过相应的输入流读取信息，反之亦然。**

**至此，套接字连接建立完毕**。

1. 套接字连接建立后，服务端的套接字对象调用getInetAddress（）方法可以获取一个 InetAddress对象，该对象可以获取客户机的IP地址和域名。

同样，客户机的套接字对象调用getInetAddress（）方法获取一个InetAddress对象，该对象包含服务端的IP地址和域名。



1. **Socket对象**可以调用close（）方法关闭套接字连接，一方关闭，就会导致对方发生IOException异常。
2. **利用多线程处理套接字连接**：套接字连接中涉及到输入流和输出流操作，读取数据时可能会引起堵塞，应该把读取数据放在一个单独的线程中去进行。另外，服务器收到一个客户机的套接字后，就应该启动一个专门为该客户机服务的线程。
3. 示例：

服务端程序：

public static void main(String[] args) {

ServerSocket ss = null;

DataInputStream dis = null;

DataOutputStream dos = null;

Socket socketAtServer = null;

try {

ss = new ServerSocket(2222);

socketAtServer = ss.accept();

dos = new DataOutputStream(socketAtServer.getOutputStream());

dis = new DataInputStream(socketAtServer.getInputStream());

while (true) {

int m = 0;

m = dis.readInt();

dos.writeUTF("你说的字符是" + (char) m);

System.out.println("服务器收到：" + m);

Thread.sleep(500);

}

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

客户端程序：

public class ClientDemo {

public static void main(String[] args) {

Socket socket = null;

DataInputStream dis = null;

DataOutputStream dos = null;

try {

**socket = new Socket("localhost", 2222);**

dos = new DataOutputStream(socket.getOutputStream());

dis = new DataInputStream(socket.getInputStream());

int i = 1;

String s = null;

dos.writeInt(i);

while(true) {

i = (i+1)%128;

s = dis.readUTF();

dos.writeInt(i);

System.out.println("客户收到："+s);

Thread.sleep(500);

}

} catch (UnknownHostException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

结果：成功。

