**服务器**：使用ServerSocket监听指定的端口，端口可以随意指定（由于1024以下的端口通常属于保留端口，在一些操作系统中不可以随意使用，所以建议使用大于1024的端口），等待客户连接请求，客户连接后，会话产生；在完成会话后，关闭连接。

**客户端**：使用Socket对网络上某一个服务器的某一个端口发出连接请求，一旦连接成功，打开会话；会话完成后，关闭Socket。客户端不需要指定打开的端口，通常临时的、动态的分配一个1024以上的端口。

# 服务器端代码

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**public** **class** Myserver {

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

**try** {

ServerSocket ser = **new** ServerSocket(8765);

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

Socket s = ser.accept();// 阻塞方法---每握手成功一次，就会返回与对方通讯的socket对象

// 读取

InputStream in = s.getInputStream();

**byte** buf[] = **new** **byte**[20];

**int** len = in.read(buf);

System.***out***.println("服务器读取客户端的信息:" + **new** String(buf, 0, len));

// 回话

OutputStream out = s.getOutputStream();

DataOutputStream dout = **new** DataOutputStream(out);

dout.writeInt(666);

s.close();

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

# Java客户端代码

**public** **class** SocketClient {

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

**try** {

Socket s = **new** Socket("127.0.0.1",8765);//指定服务器的ip+端口号

//输出

OutputStream out = s.getOutputStream();

out.write("are you server 是吗？".getBytes());

//读取

InputStream in = s.getInputStream();

DataInputStream din = **new** DataInputStream(in);

**int** n = din.readInt();

System.*out*.println(n);

s.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

# 客户端代码1

#include "lrs.h"

Action()

{

int recvlen=0;

int rc=0;

int respcode;

int length;

int ActucalNumber;

rc = lrs\_create\_socket("socket0", "TCP", "LocalHost=0", "RemoteHost=10.7.10.41:8765", LrsLastArg);

if (rc==0)

lr\_output\_message("Socket was successfully created ");

else{

lr\_output\_message("An error occurred while creating the socket, Error Code: %d", rc);

return 0;

}

lrs\_send("socket0", "buf0", LrsLastArg); //往"socket0"发送"buf0"中的数据

lrs\_receive("socket0", "buf1", LrsLastArg);//将"socke0"中返回的数据存放到"buf1"中

lrs\_save\_param\_ex("socket0","received",NULL,0,4,"ascii","param2");

length=strlen(lr\_eval\_string("<param2>"));

lrs\_hex\_string\_to\_int(lr\_eval\_string("<param2>"),length,&respcode);

lr\_output\_message("服务器返回内容：%i", respcode);

lrs\_close\_socket("socket0");

return 0;

}

# data.ws

送的是“00100312303456”这一串字符，那么我就直接把它写到data.ws中，脚本如下：

**;WSRData 2 1**

**send buf0 10**

**"00100312303456" //注意要加""**

**recv buf1 128**

**-1**

# lrs\_save\_param\_ex

lrs\_save\_param\_ex是lrs\_save\_param的扩展函数，包含了lrs\_save\_param的基本功能。其函数语法结构如下：

|  |
| --- |
| int **lrs\_save\_param\_ex** ( char \**s\_desc,* char \**type*, char \**buff*, int *offset*, int *length*, char \**encoding*, char \**para*m ); |

*s\_desc*表示指定的Socket名称；

*type*表示要存储的数据类型，有“user”、“static”、“received”三种，分别代表用户数据（脚本中自己定义的字符串）、静态数据（data.ws中录制的数据）、最后接收的缓冲区数据；

*buff*表示要保存哪个缓冲区的数据。这个参数与*type*有对应关系，如果*type*是“user”、“static”两种，则需要指定具体的buffer名称；如果是“received”则可以忽略这个参数；

*offset*表示在buffer中的偏移量；

*length*表示要从buffer中保存的字节数；

*encoding*表示的是解码方式。对于“user”类型的buffer，需要指定其为ascii或者ebcdic中的一种，NULL值表示默认的格式为ascii。对于“static”与“received”两种类型的buffer，则可以忽略这个参数，使用客户端的原始编码格式来进行解码；

*para*m表示要保存的参数名称。

例3-11演示了这三种用法：UserBuffer是脚本中自定义的字符串，存储的内容是"AAAAAThis is a userbuffer test!"，执行代码：

|  |
| --- |
| lrs\_save\_param\_ex("socket0","user",UserBuffer,5,26,NULL,"Param\_User"); |

后将会把第5个字符后的26个字符即“This is a userbuffer test!”保存到Param\_User中；

如果buf1在data.ws中的内容如下：

|  |
| --- |
| recv  buf1 38         "Get your command:Hello client!,长度13." |

执行代码：

|  |
| --- |
| lrs\_save\_param\_ex("socket0","static","buf1",0,30,NULL,"Param\_Static"); |

后，将会把buf1中从0开始的30个字符即“Get your command:Hello client!”保存到参数 Param\_Static中。

如果lrs\_receive最后接收到的数据包内容为“Length=11Successful.AAAAA”，则执行代码：

|  |
| --- |
| lrs\_save\_param\_ex("socket0","received",NULL,9,11,NULL,"Param\_Received"); |

后，将会把从"socket0"收到的缓冲区数据中第9个字节后的11个字节即“Successful.”保存到参数Param\_Received中。