Java 网络编程

1、基于 TCP 的 socket 编程

1）服务器程序编写

（1）调用 ServerSocket(int port) 创建一个服务器端套接字，并绑定到指定端口上；
（2）调用 accept()，监听连接请求，如果客户端请求连接，则接受连接，返回通信套接字。
（3）调用通信套接字的 getOutputStream() 和 getInputStream() 获取输出流和输入流，开始网络数据的发送和接收。
（4）最后关闭输入输出流以及通信套接字和服务器套接字。

2）客户端程序编写

（1）调用 Socket(InetAddress address, int port) 创建一个通信套接字，并连接到服务器端；
（2）调用通信套接字的 getOutputStream() 和 getInputStream() 获取输出流和输入流，开始网络数据的发送和接收。
（3）最后关闭通输入输出流以及信套接字。

3）示例代码

```
//服务器端
public class MyServer extends Thread
{
    private Socket s;

    public MyServer(Socket s)
    {
        this.s = s;
    }

    @Override
    public void run()
    {
        OutputStream os = null;
        InputStream is = null;
        try
        {
            os = s.getOutputStream();
```

```
                os.write("Hello,welcome you!".getBytes());

                is = s.getInputStream();
                byte[] buf = new byte[100];
                int len = is.read(buf);
                System.out.println(new String(buf, 0, len));
            }
            catch (Exception ex)
            {
                ex.printStackTrace();
            }
            finally
            {
                if (is != null)
                    try
                    {
                        is.close();
                    }
                    catch (IOException e1)
                    {
                        e1.printStackTrace();
                    }
                if (os != null)
                    try
                    {
                        os.close();
                    }
                    catch (IOException e)
                    {
                        e.printStackTrace();
                    }
                if (s != null)
                    try
                    {
                        s.close();
                    }
                    catch (IOException e)
                    {
                        e.printStackTrace();
                    }
            }
        }

        public static void server()
```

```
    {
        ServerSocket ss = null;
        try
        {
            ss = new ServerSocket(6666);
            while (true)
            {
                Socket s = ss.accept();
                new MyServer(s).start();
            }
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
        finally
        {
            if (ss != null)
                try
                {
                    ss.close();
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
        }
    }

    public static void main(String[] args)
    {
        MyServer.server();
    }
}

//客户端
public class MyClient
{
    public static void client()
    {
        Socket s = null;
        OutputStream os = null;
        InputStream is = null;
        try
```

```
        {
            s = new Socket(InetAddress.getByName(null), 6666);
            os = s.getOutputStream();
            is = s.getInputStream();
            byte[] buf = new byte[100];
            int len = is.read(buf);
            System.out.println(new String(buf, 0, len));
            os.write("Hello,this is Tom".getBytes());
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
        finally
        {
            if (is != null)
                try
                {
                    is.close();
                }
                catch (IOException e)
                {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            if (os != null)
                try
                {
                    os.close();
                }
                catch (IOException e)
                {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            if (s != null)
                try
                {
                    s.close();
                }
                catch (IOException e)
                {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
```

```
                }
            }
        }
        public static void main(String[] args)
        {
            MyClient.client();
        }
}
```

2、基于 UDP 的 socket 编程

1）接收端程序编写

（1）调用 DatagramSocket(int port)创建一个数据报套接字，并绑定到指定端口上；
（2）调用 DatagramPacket(byte[] buf, int length)，建立 UDP 包用于接受数据。
（3）调用数据报套接字的 receive() 方法，接收 UDP 包。
（4）最后关闭数据报套接字。

2）发送端程序编写

（1）调用 DatagramSocket() 创建一个数据报套接字；
（2）调用 DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)，建立要发送的 UDP 包。
（3）调用数据报套接字的 send()，发送 UDP 包。
（4）最后关闭数据报套接字。

3）示例代码

```
//接收端
public class MyRecv
{
    public static void recv()
    {
        DatagramSocket ds = null;
        try
        {
            ds = new DatagramSocket(6000);
            byte[] buf = new byte[100];
            DatagramPacket dpRecv = new DatagramPacket(buf, 100);
            ds.receive(dpRecv); //接收数据
            System.out.println(new String(buf, 0, dpRecv.getLength()));
```

```
                String str = "Welcome you!";
                DatagramPacket dpSend = new DatagramPacket(str.getBytes(),
                        str.length(), dpRecv.getAddress(), dpRecv.getPort());
                ds.send(dpSend); //回复数据
            }
            catch (Exception ex)
            {
                ex.printStackTrace();
            }
            finally
            {
                if (ds != null)
                    ds.close();
            }
        }
        public static void main(String[] args)
        {
            MyRecv.recv();
        }
}

//发送端
public class MySend
{
        public static void send()
        {
            DatagramSocket ds = null;
            try
            {
                ds = new DatagramSocket();
                String str = "Hello,this is Tom";
                DatagramPacket dp = new DatagramPacket(str.getBytes(),
                        str.length(), InetAddress.getByName("localhost"), 6000);
                ds.send(dp); //发送数据
                byte[] buf = new byte[100];
                DatagramPacket dpRecv = new DatagramPacket(buf, 100);
                ds.receive(dpRecv); //接收数据
                System.out.println(new String(buf, 0, dpRecv.getLength()));
            }
            catch (Exception ex)
            {
                ex.printStackTrace();
            }
            finally
```

```
        {
            if (ds != null)
                ds.close();
        }
    }
    public static void main(String[] args)
    {
        MySend.send();
    }
}
```

3、URL 与 URI

1）基本概念

（1）URL(Uniform Resource Locator)，通用资源定位符。URI(Uniform Resource Identifier)，通用资源标识符。
（2）URI 纯粹是个符号结构，用于指定构成 Web 资源的字符串的各个不同部分。URL 是一种特殊类型的 URI，它包含了用于查找某个资源的足够信息，能够准确定位到 Web 上的某个资源。其它的 URI 不能够准确定位到 Web 上的某个资源，这种 URI 称为 URN(通用资源名)。
（3）在 Java 库中，URI 类不包含用于访问通用资源的任何方法，它的唯一作用是进行分析。而 URL 类则可以打开到达资源的一个连接，从而获取资源的一些信息以及下载该资源。

2）示例代码

```
public class Test
{
    public static void main(String[] args) throws UnsupportedEncodingException, IOException
    {
        URL url = new URL("https://www.baidu.com/");
        BufferedReader br = new BufferedReader(new InputStreamReader(url.openStream(),
"utf-8"));
        BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream("baidu.html"), "utf-8"));

        String buf;
        while ((buf = br.readLine()) != null)
        {
            bw.append(buf);
            bw.newLine();
        }

```

```
            bw.close();
            br.close();
        }
}
```