****

分布式系统与云计算

**数据科学与大数据技术**专业

**大数据1802** 班

指导老师\_余腊生\_

姓名 **刘爱芳**

学号 **8208181116**

电话 **17877780889**

**2021** 年**12**月**23**日

# 实验一 数据包socket应用

# 实验目的

1. 理解数据包socket的应用

2. 实现数据包socket通信

3. 了解JAVA并行编程的基本方法

# 实验环境

Windows 操作系统 ， Jdk 工具包，Java语言，eclipse

# 实验原理

1. 分布式计算的核心是进程通信。 操作系统、网卡驱动程序等应用从不同抽象 层面提供了对进程通信的支持，例如

Winsock、java.net.\*。Socket API 是一种作为 IPC 提供对系统低层抽象的机制。尽管应用人 员很少需要在该层编写代码，但理解 socket API 非常重要，因为：1，高层设施是构建于 socket API 之上的，即他们是利用 socket API 提供的操作来实现；2，对于以响应时间要求较高或 运行于有限资源平台上的应用来说，socket API 可能是最适合的。

在 Internet 网络协议体系结构中，传输层上有 UDP 和 TCP 两种主要协议，UDP 允许在 传送层使用无连接通信传送，被传输报文称为数据包。（是否存在面向连接的数据包 socket？）因此数据包 socket 是基于 UDP 的不可靠 IPC。Java 为数据包 socket API 提供两个 类：

（1）针对 socket 的 datagramSocket 类

（2）针对数据包交换的 datagramPacket 类

希望使用该 API 发送和接收数据的进程须实例化一个 datagramSocket 对象，每个 socekt

被绑定到该进程所在及其的某个 UDP 端口上。为了向其他进程发送数据包，进程必须创建

一个代表数据包本身的对象。该对象通过实例化一个 datagram socket 对象创建。 在接收者进程中，datagramPacket 对象也必须被实例化并绑定到一个本地端口上，该端

口必须与发送者数据包的定义一致。接收进程创建一个指向字节数组的 DatagramPacket，并 调用 datagramSocket 对象的 receive 方法，将 DatagramPacket 对象指针作为参数定义。

2. 并行编程（以 Java 为例1）

一个线程是比进程更小的执行粒度。 Java 虚拟机允许应用程序有多个执行线程同时运 行。有两种方法来创建一个新线程的执行。一个是声明一个类是一个线程的子类。这个子类 应重写 Thread 类的 run 方法。一个子类的实例可以被分配和启动。另一种方法创建一个线 程，并同时声明一个类实现了 Runnable 接口（这个类要实现 run 方法）。一个类的实例可以 被分配并作为参数传递给创建的线程，并启动线程。例如：

 创建一个类是 Thread 的子类：

class SomeThread extends Thread {

SomeThread() {

}

public void run() {

. . .

}

}

SomeThread p = new SomeThread(); p.start();

 创建一个实现 Runnable 接口的类并传递给线程：

class SomeRun implements Runnable { SomeRun() {

}

public void run() {

. . .

}

}

SomeRun p = new SomeRun(143); new Thread(p).start();

当一个实现 Runnable 接口的类被执行时，可以没有子类。实例化一个 Thread 实例，并 通过自身作为目标线程。在大多数情况下，如果你只打算重写的 run（）方法，并没有其它的线程方法，应使用 Runnable 接口。因为类不应该被继承，除非程序员有意修改或增强类 的基本行为。

# 实验内容

## 构建客户端程序

（1）构建 DatagramSocket 对象实例

1. DatagramSocket()
2. //创建一个数据包套接字，绑定到本机上任意一个可用的端口
3. DatagramSocket(DatagramSocketImpl)
4. //创建一个绑定的数据报套接字， 与指定的datagramsocketimpl相关（一般不使用）
5. DatagramSocket(SocketAddress)
6. //通过制定的套接字地址来创建一个数据包套接字
7. DatagramSocket(int)
8. //指定本地的一个端口，以此来创建数据包套接字
9. DatagramSocket(int, InetAddress)
10. //创建一个DatagramSocket对象，并绑定到指定的地址和端口上

（2）构建 DatagramPacket 对象实例，并包含接收者主机地址、接收端口号等信息

1. DatagramPacket packet = new DatagramPacket(data, data.length, new InetSocketAddress("127.0.0.1", 10000));

（3）调用 datagramSocket 对象实例的 send 方法，将 DatagramPacket对象实例作为参数发送。

client.send(packet);

## 2.2 构建服务器端程序

（1）构建 datagramSocket 对象实例，指定接收的端口号。

DatagramSocket server=new DatagramSocket(8888);

（2）构建 DatagramPacket 对象实例，用于重组接收到的消息。

DatagramPacket datagramPacket=new DatagramPacket(container, container.length);

（3）调用datagramSocket对 象 实 例 的receive方 法 ， 进 行 消 息 接 收 ， 并 将DatagramPacket 对象实例作为参数。

server.receive(packet);

# 实验结果



# 源代码清单

## 客户端

1. package Internet;
3. import java.io.IOException;
4. import java.net.DatagramPacket;
5. import java.net.DatagramSocket;
6. import java.net.InetSocketAddress;
7. import java.net.SocketException;
9. public class Udp\_chat\_client {
10. public static void main(String[] args) throws IOException {
11. DatagramSocket datagramSocket=new DatagramSocket(9999);
12. String meString="Hello,World";
13. byte[] bs=meString.getBytes();
14. DatagramPacket datagramPacket=new DatagramPacket(bs, bs.length,new InetSocketAddress("localhost",8888));
15. datagramSocket.send(datagramPacket);
16. datagramSocket.close();
17. }

## 

## }服务器端

1. package Internet;
3. import java.io.IOException;
4. import java.net.DatagramPacket;
5. import java.net.DatagramSocket;
6. import java.net.SocketException;
8. public class UDP\_chat {
9. public static void main(String[] args) throws IOException {
10. //1、 构件服务端 +端口
12. DatagramSocket server=new DatagramSocket(8888);
13. //2、封装成包 DatagramPacket(byte[] buf,int lenth) 相当于构建接受池
14. byte[] container=new byte[1024];
15. DatagramPacket datagramPacket=new DatagramPacket(container, container.length);
16. //接收客户端发来的数据
17. server.receive(datagramPacket);
18. //获取接收到的数据
19. byte [] data=datagramPacket.getData();
20. //获取接收到的数据的长度
21. int len=datagramPacket.getLength();
22. //输出相关的信息
23. System.out.println(new String(data,0,len));
24. //关闭接收流
25. server.close();
26. }
27. }