中南大学

分布式系统实验报告

目录

（我选做 4 题，按住 ctrl　点击目录条可直达，wps 下有效）

实验一 数据包 socket 应用

一、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

二、预习与实验要求

1. 预习实验指导书及教材的有关内容，了解数据包 socket 的通信原理；
2. 熟悉一种 java IDE 和程序开发过程；
3. 了解下列 Java API ： Thread 、Runnable ；
4. 尽可能独立思考并完成实验。

三、实验环境

a) 独立计算机或计算机网络；

b) Windows 操作系统。

c) Jdk 工具包

d) JCreator or others

四、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、.*。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();
();
```

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。

五、实验内容

1. 构建客户端程序

（1） 构建 datagramSocket 对象实例

（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

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

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。

（2） 构建 DatagramPacket 对象实例，用于重组接收到的消息。

（3） 调用 datagramSocket 对象实例大家 receive 方法，进行消息接收，并将 DatagramPacket 对象实例作为参数。

六、实验报告

1. 客户端和服务器端程序的伪代码；

客户端：

```
import      class Client {
   public static void main(String[] args) throws IOException
       {
            etBytes();
```

如何避免数据包丢失而造成的无限等待问题

答：我认为可在发包时设定一个定时器，若发出去的包在一定时间内没有收到答应，则再发一次。为了避免接受者接到重复的包，可以给数据包加个序号，接受者收包时查看序号即可。

2. 如何实现全双工的数据包通信

答：利用端口套接字之间的通信功能。

实验二 流式 socket 应用

一、实验目的

1. 理解流式 socket 的原理

2. 实现流式 socket 通信

二、预习与实验要求

1. 预习实验指导书及教材的有关内容，了解流式 socket 的通信原理；

2. 熟悉 java 环境和程序开发过程；

3. 尽可能独立思考并完成实验。

三、实验环境

a) 独立计算机；

b) Windows 操作系统；

c) Jdk 工具包

四、实验原理

Socket API 是一种作为 IPC 提供低层抽象的机制。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允许使用无连接通信传送，被传输报文称为数据包。而 TCP 则允许面向连接的可靠通信，这种 IPC 称为流式 socket　。Java　为流式 socket API　提供两类 socket　（1）式用于连接的连接 socket（2）式用于数据交换的数据 socket　。

五、实验内容

1. 构建客户端程序和服务器端程序都需要的 MystreamSocket　类，定义继承自 java Socket

的 sendMessage　和 receiveMessage　方法

2. 构建客户端程序

（1）　创建一个 MyStreamsocket　的实例对象，并将其指定接收服务器和端口号

（2）　调用该 socket　的 receiveMessage　方法读取从服务器端获得的消息

3. 构建服务器端程序

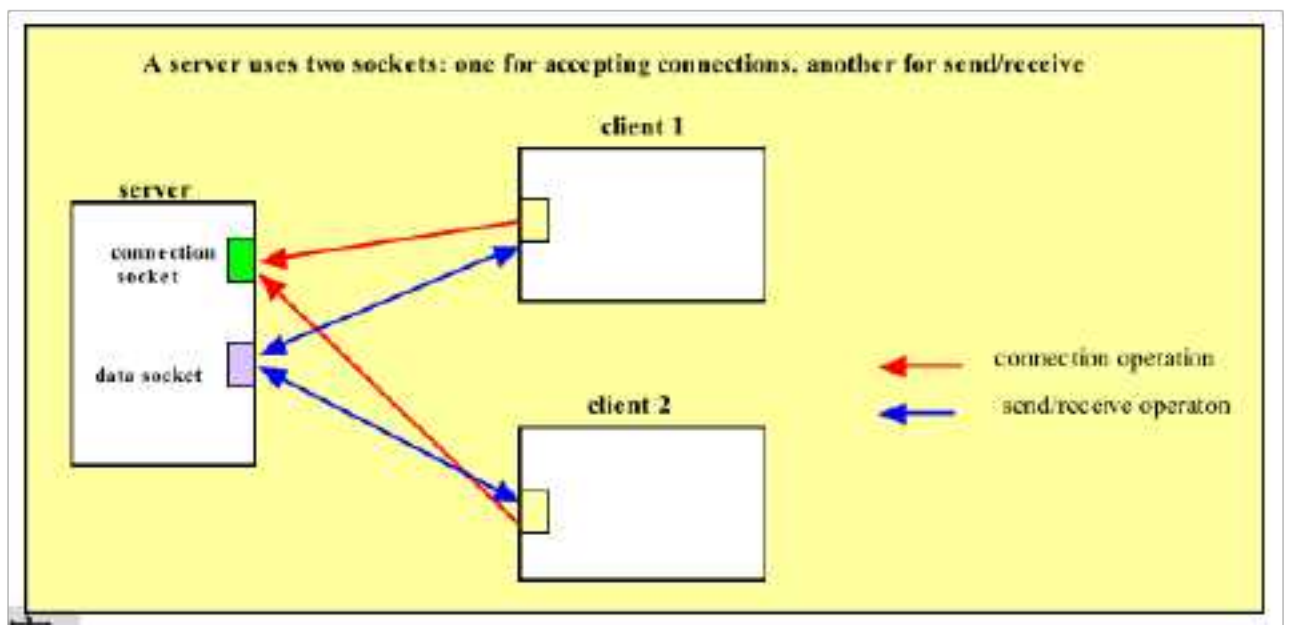
（1）　构建连接 socket　实例，并与指定的端口号绑定，该连接 socket　随时侦听客户端的连接请求

（2）　创建一个 MyStreamsocket　的实例对象

（3）　调用 MyStreamsocket　的实例对象的 sendMessage　方法，进行消息反馈。

六、实验报告

1. 应用程序的结构图，说明程序之间的关系；



程序的伪代码。

公用服务功能：

```
import .*;

import .*;


public class MystreamSocket extends Socket {

  private Socket socket;

  private BufferedReader input;

  private PrintWriter output;

  ava   :

import   class Client {

      public static void main(String args[]){

            try {

      MystreamSocket mss=new MystreamSocket(hostname,12345);

         我是客户端，我请求连接!

                  ();

} catch (IOException e) {

      tart();

      }

} catch (IOException e) {
```

如何实现全双工的流式 socket　通信

答：服务端监听端口，每当有一个连接请求发来时，就与其建立新的连接，然后利用其提供的功能进行通信。

2. 如何实现安全 socket API

答：注意在通信过程中的各种异常情况的捕获与处理。

3. 如何实现 1 对多的并发

答：在服务端使用多线程。

实验三 客户/ 服务器应用开发

一、实验目的

1. 验证 daytime　和 echo　程序，

2. 实现包 socket　支撑的 C/S　模式 IPC　机制

3. 实现流式 socket　支撑的 C/S　模式 IPC　机制

二、预习与实验要求

1. 预习实验指导书及教材的有关内容，了解 daytime　和 echo　要提供的具体服务内容；

2. 复习包 socket　和流式 socket　的实现原理；

3. 实验前认真听讲，服从安排。尽可能独立思考并完成实验。

三、实验环境

a) 独立计算机；

b) Windows 操作系统。

c) Jdk　工具包

C/S 模式是主要的分布式应用范型，其设计的目的是提供网络服务。网络服务指如 daytime 、telnet 、ftp 和 WWW 之类的允许网络用户共享资源的服务。要构建 C/S 范型的应用就必须解决以下一些关键问题：

（1） 如何通过会话实现多个用户的并发问题

（2） 如何定义客户和服务器在服务会话期间必须遵守的协议

（3） 服务定位问题

（4） 进程间通信和事件同步问题：语法、语义和响应

（5） 数据表示问题

在解决了这些问题的基础上，C/S 范型必须遵从 3 层结构的软件体系结构：

（1） 表示层，提供与客户端进行交互的界面

（2） 应用逻辑层，定义服务器和客户端要处理的主要事务的业务逻辑

（3） 服务层，定义应用逻辑层所需要的底层支持技术，例如定义其 IPC 机制里的 receive 方法和 send 方法等。

五、实验内容

1. 构建用数据包 socket 实现的 daytime 客户端程序

（1） 构建表示层程序

（2） 构建应用逻辑层程序

（3） 构建服务层程序

2. 构建用数据包 socket 实现的 daytime 服务器端程序

（1） 构建表示层和应用逻辑层程序

（2） 构建服务层程序

（3） 构建服务层程序所需要的下层程序（它封装了客户端的消息和地址）

3. 构建用流式 socket 实现的 daytime 应用程序包

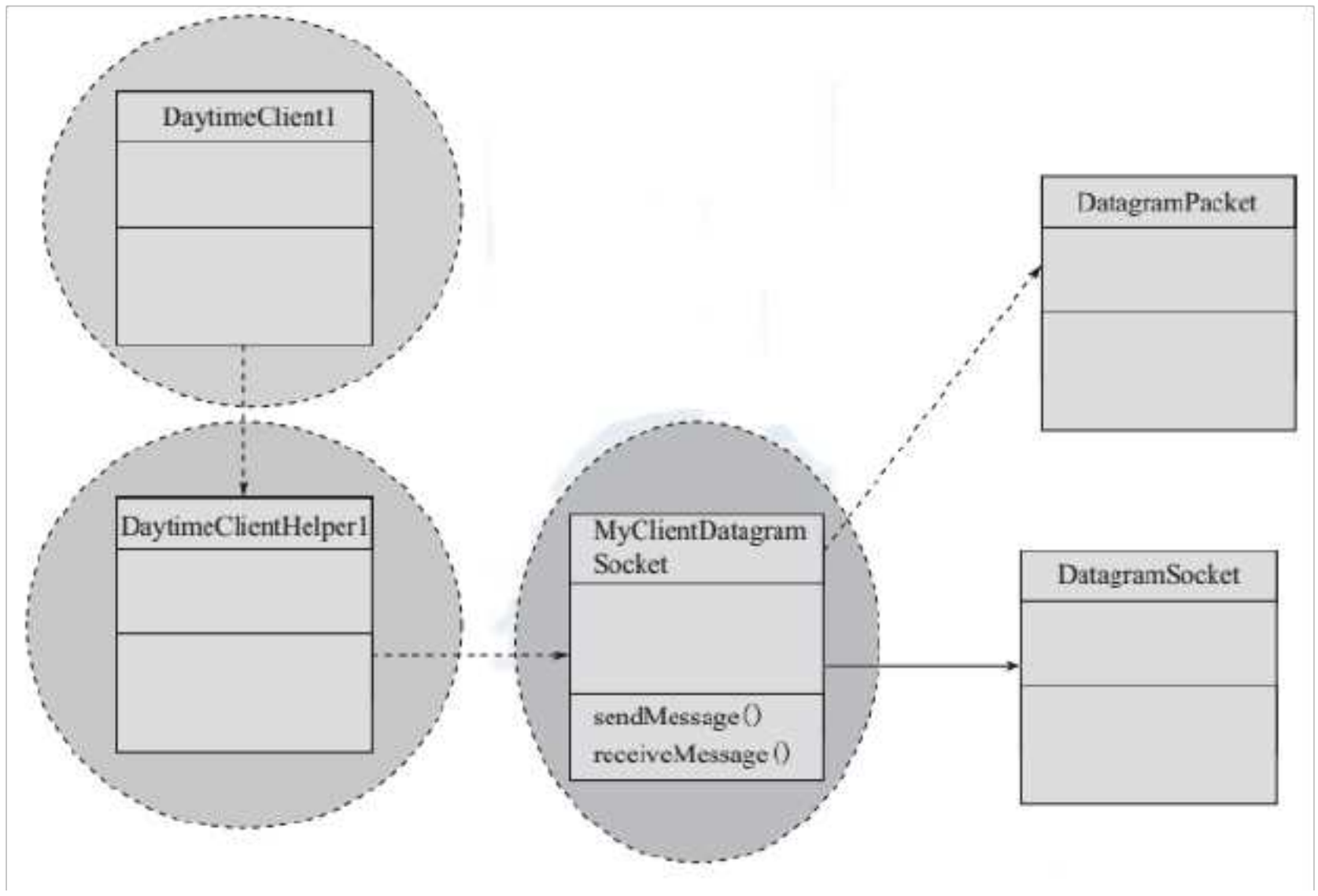
构建用数据包 socket　实现的 echo　应用程序包

5. 构建用流式 socket　实现的 echo　应用程序包

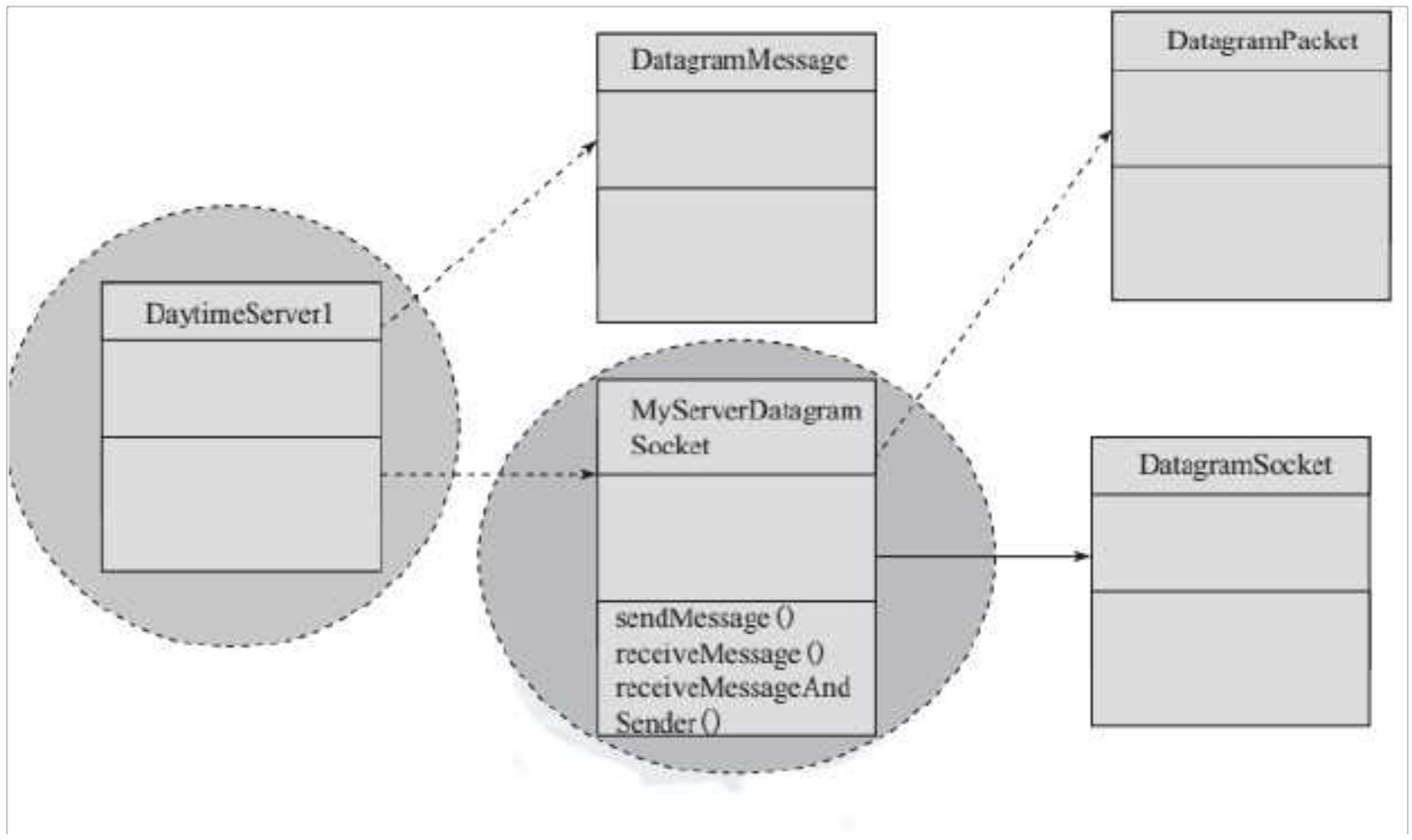
六、实验报告

1. 用数据包 socket　实现的 daytime　应用程序包的构架，列明各程序之间的关系；

客户端：



服务端：



客户端：

```
import .*;

public class DaytimeClient1 {
        public static void main(String[] args) {
                InputStreamReader is = new InputStreamReader;
                BufferedReader br = new BufferedReader(is);
                try {
                        String hostName = ();
                        if () == 0)
                                                                ;
public class DaytimeClientHelper1 {
```

```
        public static String getTimestamp(String hostName, String portNum){


                try {
                                InetAddress serverHost = (hostName);
                                int serverPort = (portNum);
                                ;
import .*;
public class MyClientDatagramSocket extends DatagramSocket {
static final int MAX_LEN = 100;
        MyClientDatagramSocket() throws SocketException{
                super();
        }
        MyClientDatagramSocket(int portNo) throws SocketException{
                super(portNo);
        }
        public void sendMessage(InetAddress receiverHost, int receiverPort, String
message)
                                                throws IOException {
                        byte[                ]          sendBuffer                =
() ;
                                DatagramPacket datagram = new DatagramPacket(sendBuffer, ,
                                                                                receiverHos
t, receiverPort);
                                (datagram);
        }
        public String receiveMessage()
                        throws IOException {
```

```
                byte[ ] receiveBuffer = new byte[MAX_LEN];

                DatagramPacket  datagram  =  new  DatagramPacket(receiveBuffer,
MAX_LEN);

                (datagram);

                String message = new String(receiveBuffer);

                return message;

        }

}
```

服务端：

```
import .*;

import

public class DaytimeServer1 {

        public static void main(String[] args) {

                int serverPort = 13;

                if  == 1 )

                        serverPort = (args[0]);

                try {

                        MyServerDatagramSocket          mySocket          =          new
MyServerDatagramSocket(serverPort);

                while (true) {

                        DatagramMessage request = ();

                        Date timestamp = new Date ();

                        () ,
```

() , ());

}

}

catch (Exception ex) {

}

}

}

import .*;

import .*;

public class MyServerDatagramSocket extends DatagramSocket {

static final int MAX_LEN = 100;

MyServerDatagramSocket(int portNo) throws SocketException{

super(portNo);

}

public void sendMessage(InetAddress receiverHost, int receiverPort, String message)

throws IOException {

byte[ ] sendBuffer = () ;

DatagramPacket datagram =

new DatagramPacket(sendBuffer, , receiverHost, receiverPort);

(datagram);

}

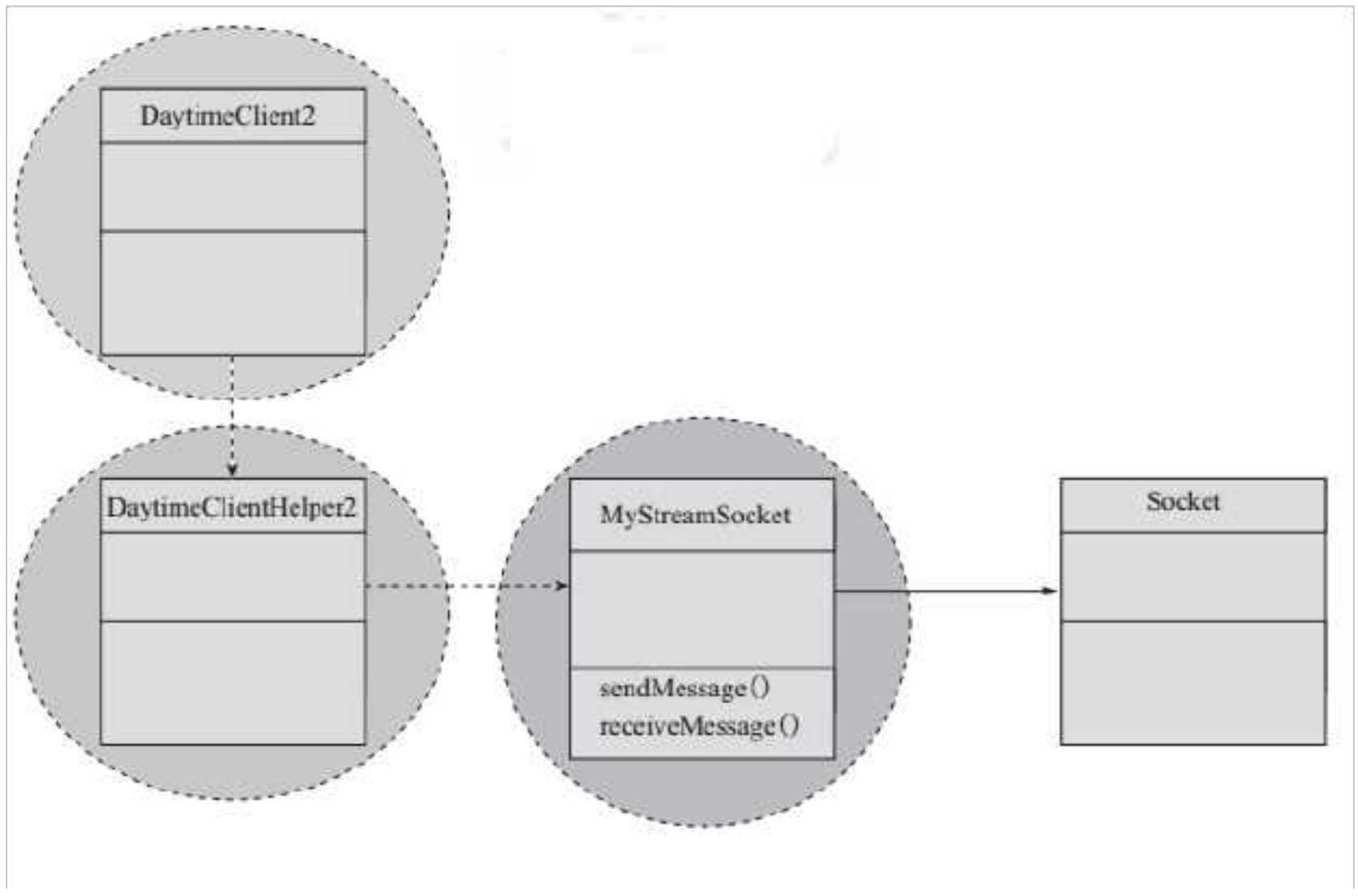
```
        public String receiveMessage()
                        throws IOException {
                    byte[ ] receiveBuffer = new byte[MAX_LEN];
                    DatagramPacket  datagram  =  new  DatagramPacket(receiveBuffer,
MAX_LEN);
                    (datagram);
                    String message = new String(receiveBuffer);
                    return message;
        }
        public DatagramMessage receiveMessageAndSender()
                        throws IOException {
                    byte[ ] receiveBuffer = new byte[MAX_LEN];
                    DatagramPacket  datagram  =  new  DatagramPacket(receiveBuffer,
MAX_LEN);
                    (datagram);
                    DatagramMessage returnVal = new DatagramMessage();
                    (new String(receiveBuffer), (),
                                                                ());
                    return returnVal;
        } }


import .*;
public class DatagramMessage{
        private String message;
        private InetAddress senderAddress;
        private int senderPort;
```

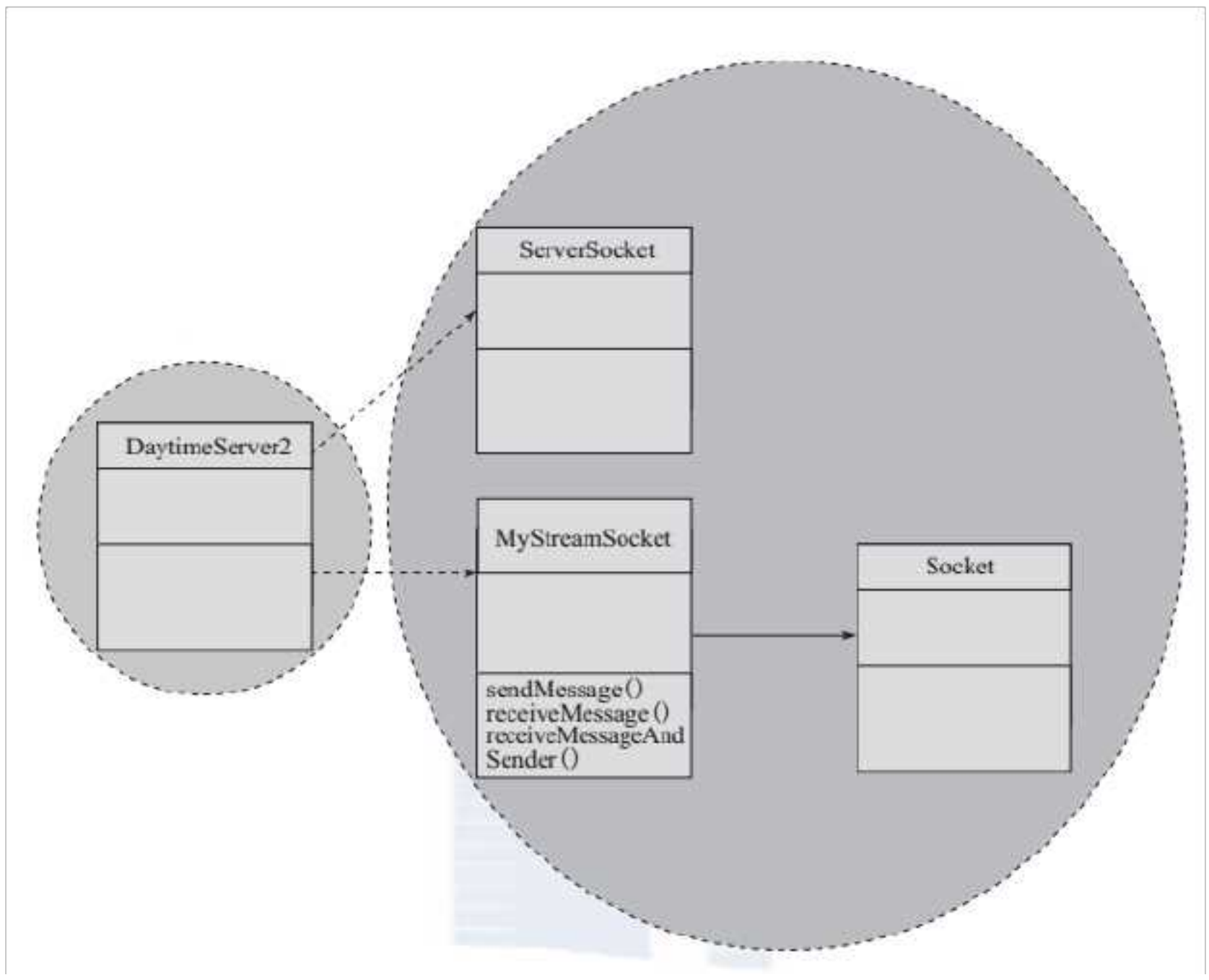
```
public void putVal(String message, InetAddress addr, int port) {
        = message;
        = addr;
        = port;
}
public String getMessage() {
        return ;
}
        public InetAddress getAddress() {
                return ;
        }
        public int getPort() {
                return ;
        }
}
```

2. 用流式 socket　实现的 daytime　应用程序包的构架，列明各程序之间的关系；

客户端：


DaytimeClient2
DaytimeClientHelper2
MyStreamSocket
sendMessage()
receiveMessage()
Socket

服务端：


ServerSocket
DaytimeServer2
MyStreamSocket
sendMessage()
receiveMessage()
receiveMessageAnd
Sender()
Socket

共有的：