云南大学数学与统计学院  
《计算机网络实验》上机实践报告

|  |  |  |
| --- | --- | --- |
| 课程名称：计算机网络实验 | 年级：2015级 | 上机实践成绩： |
| 指导教师：陆正福 | 姓名：刘鹏 | 专业：信息与计算科学 |
| 上机实践名称：计算机网络平台预备实验 | 学号：20151910042 | 上机实践日期：2018-08-27 |
| 上机实践编号：No.01 | 组号： |  |

# 实验目的

1. 熟悉本学期计算机网络编程实验的平台，为计算机网络实验课程的后继实验奠定基础；
2. 熟悉教材第一章的基本概念。

# 实验内容

1. 查阅本机IP地址（CLI运行ipconfig）
2. 测试连通性（CLI运行ping 127.0.0.1）
3. 查阅网络文档，建立主要网络服务（telnet，ftp，ssh等）
4. 查阅网络文档，找到与Java网络编程有关的类库包（net，io，nio等），做简单分析.
5. 查阅网络文档，安装配置虚拟机平台，形成宿主机与虚拟机之间的联机调试网络程序的环境。
6. 两人或者三人成组，形成两台或多台物理机之间的联机调试网络程序的环境。

# 实验平台

Windows 10 Pro 1803；

Cygwin GCC编译器。

# 算法设计

# 程序代码

## 查阅本机IP地址

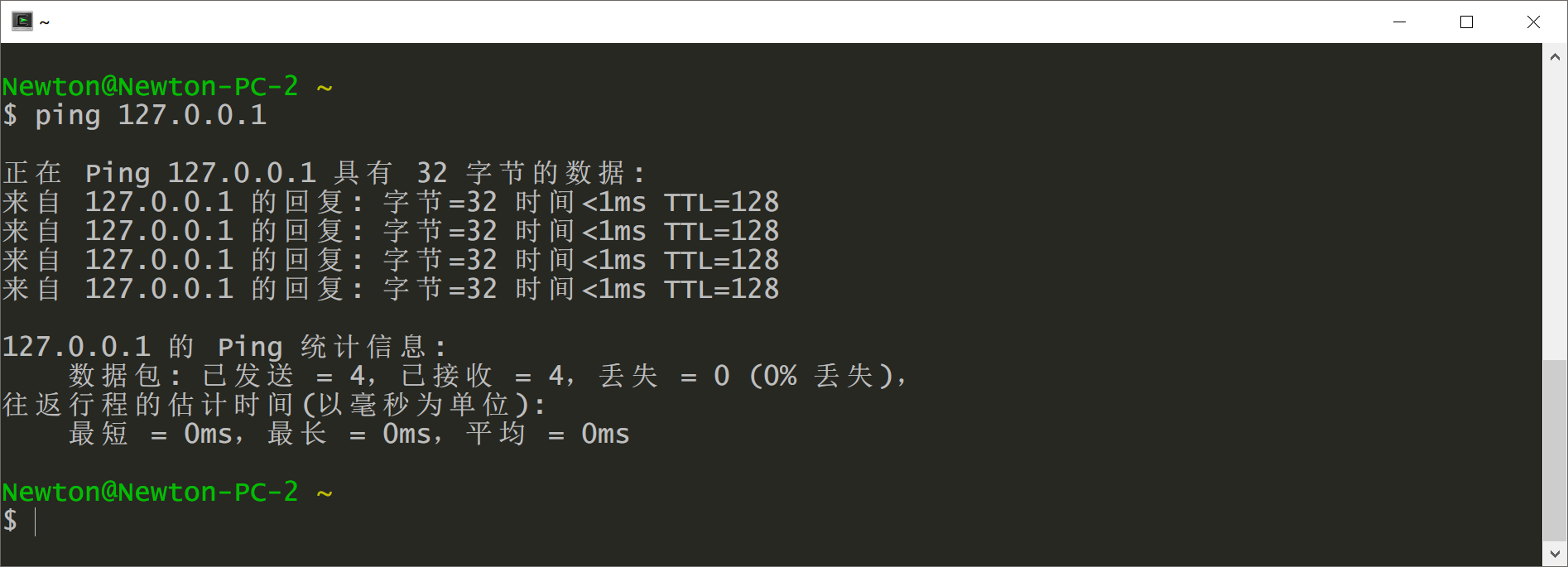
由于计算机网络主要在UNIX环境下完成了初期开发，所以为了模拟一个UNIX环境，这里采用Cygwin进行实验。

### Shell命令与输出

图片包含 文字

已生成高可信度的说明

## 测试连通性



## FTP、SSH的服务建立

查阅网络文档，建立主要网络服务（telnet，ftp，ssh等）

如果采用安装了Ubuntu系统的PC作为host，那么建立这些网络服务非常简单。在另一台Windows系统上，安装Xshell，可以快速建立ssh（Secure Shell）连接

### SSH

SSH为Secure Shell的缩写，由IETF的网络小组（Network Working Group）所制定；SSH为建立在应用层基础上的安全协议。SSH是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议。利用SSH协议可以有效防止远程管理过程中的信息泄露问题。SSH最初是UNIX系统上的一个程序，后来又迅速扩展到其他操作平台。SSH在正确使用时可弥补网络中的漏洞。SSH客户端适用于多种平台。几乎所有UNIX平台—包括HP-UX、Linux、AIX、Solaris、Digital UNIX、Irix，以及其他平台，都可运行SSH。

传统的网络服务程序，如：ftp、pop和telnet在本质上都是不安全的，因为它们在网络上用明文传送口令和数据，别有用心的人非常容易就可以截获这些口令和数据。而且，这些服务程序的安全验证方式也是有其弱点的，就是很容易受到“中间人”（man-in-the-middle）这种方式的攻击。所谓“中间人”的攻击方式，就是“中间人”冒充真正的服务器接收你传给服务器的数据，然后再冒充你把数据传给真正的服务器。服务器和你之间的数据传送被“中间人”一转手做了手脚之后，就会出现很严重的问题。通过使用SSH，你可以把所有传输的数据进行加密，这样“中间人”这种攻击方式就不可能实现了，而且也能够防止DNS欺骗和IP欺骗。使用SSH，还有一个额外的好处就是传输的数据是经过压缩的，所以可以加快传输的速度。SSH有很多功能，它既可以代替Telnet，又可以为FTP、PoP、甚至为PPP提供一个安全的“通道”。

由于系统支持，所以直接用Xshell在Windows平台，填写Ubuntu PC的IP地址，用户以及密钥，就可以快速进行远程Shell登录。SSH 主要由三部分组成：

传输层协议 [SSH-TRANS]

提供了服务器认证，保密性及完整性。此外它有时还提供压缩功能。 SSH-TRANS 通常运行在TCP/IP连接上，也可能用于其它可靠数据流上。 SSH-TRANS 提供了强力的加密技术、密码主机认证及完整性保护。该协议中的认证基于主机，并且该协议不执行用户认证。更高层的用户认证协议可以设计为在此协议之上。

用户认证协议 [SSH-USERAUTH]

用于向服务器提供客户端用户鉴别功能。它运行在传输层协议 SSH-TRANS 上面。当SSH-USERAUTH 开始后，它从低层协议那里接收会话标识符（从第一次密钥交换中的交换哈希H）。会话标识符唯一标识此会话并且适用于标记以证明私钥的所有权。 SSH-USERAUTH 也需要知道低层协议是否提供保密性保护。

连接协议 [SSH-CONNECT]

将多个加密隧道分成逻辑通道。它运行在用户认证协议上。它提供了交互式登录话路、远程命令执行、转发 TCP/IP 连接和转发X11连接。

图片包含 屏幕截图

已生成极高可信度的说明

### ftp

ftp也比较简单，通过WinSCP这款软件也可以实现FTP上传下载的服务。

### telnet

Telnet是电信（Telecommunications）和网络（Networks）的联合缩写，这是一种在UNIX平台上最为人所熟知的网络协议。Telnet使用端口23，它是专门为局域网设计的。Telnet不是一种安全通信协议，因为它并不使用任何安全机制，通过网络/互联网传输明文格式的数据，包括密码，所以谁都能嗅探数据包，获得这个重要信息。Telnet中没有使用任何验证策略及数据加密方法，因而带来了巨大的安全威胁，这就是为什么telnet不再用于通过公共网络访问网络设备和服务器。因为不安全，而且SSH更加先进，这里仅作描述，不再进行实验。

## Java网络编程类库包的简单分析

### net包

java.net提供了一些用于网络编程的类的实现。java.net包可以大致分为两个部分，首先是低级API部分，它完成了IP地址、套接字、网络接口之类的抽象，高级API解决了对URIs、URLs，Connections等对象的抽象。由于还没有深入了解网络编程，所以这里仅作了解。

### io包

Java语言中的io包支持Java的基本I/O（输入/输出）系统，包括文件的输入/输出。Java所有的I/O机制都是基于数据流进行输入输出，这些数据流表示了字符或者字节数据的流动序列。Java的I/O流提供了读写数据的标准方法。任何Java中表示数据源的对象都会提供以数据流的方式读写它的数据的方法。

java中将输入输出抽象成流，流通过输入输出系统与物理设备连接，尽管与它们链接的物理设备不尽相同，所有流的行为具有同样的方式。将数据从外部（包括磁盘文件、键盘、套接字）读入到内存中的流称为输入流，将从内存写入到外部设备（控制台、磁盘文件或者网络）的称为输出流。

流序列中的数据既可以是未经加工的原始二进制数据，也可以是经一定编码处理后符合某种格式规定的特定数据。因此Java中的流分为两种：

* 字节流：数据流中最小的数据单元是字节 多用于读取或书写二进制数据
* 字符流：数据流中最小的数据单元是字符，Java中的字符是Unicode编码，一个字符占用两个字节。

在最底层，所有的输入/输出都是字节形式的。基于字符的流只为处理字符提供方便有效的方法。

#### 程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | **import** java**.**io**.\*;**  //byte streams are used to perform input and output of 8-bit bytes  public class CopyFileByte **{**  public static void main**(**String**[]** args**)** **throws** IOException **{**  FileInputStream in **=** **null;**  FileOutputStream out **=** **null;**  **try** **{**  in **=** **new** FileInputStream**(**"input.txt"**);**  out **=** **new** FileOutputStream**(**"output.txt"**);**  int c**;**  **while** **((**c **=** in**.**read**())** **!=** **-**1**)** **{**  out**.**write**(**c**);**  **}**  **}** **finally** **{**  **if** **(**in **!=** **null)** **{**  in**.**close**();**  **}**  **if** **(**out **!=** **null)** **{**  out**.**close**();**  **}**  **}**  **}**  **}** |

### nio包

在JDK 1.4以前，Java的IO操作集中在java.io这个包中，是基于流的阻塞（blocking）API。对于大多数应用来说，这样的API使用很方便，然而，一些对性能要求较高的应用，尤其是服务端应用，往往需要一个更为有效的方式来处理IO。从JDK 1.4起，NIO API作为一个基于缓冲区，并能提供非阻塞（non-blocking）IO操作的API被引入。NIO API主要集中在java.nio和它的subpackages中：

***java.nio***定义了Buffer及其数据类型相关的子类。其中被java.nio.channels中的类用来进行IO操作的ByteBuffer的作用非常重要。

***java.nio.channels***定义了一系列处理IO的Channel接口以及这些接口在文件系统和网络通讯上的实现。通过Selector这个类，还提供了进行非阻塞IO操作的办法。这个包可以说是NIO API的核心。

***java.nio.channels.spi***定义了可用来实现channel和selector API的抽象类。

***java.nio.charset***定义了处理字符编码和解码的类。

***java.nio.charset.spi***定义了可用来实现charset API的抽象类。

***java.nio.channels.spi***和***java.nio.charset.spi***这两个包主要被用来对现有NIO API进行扩展，在实际的使用中，我们一般只和另外的3个包打交道。下面将对这3个包一一介绍。

***java.nio***这个包主要定义了Buffer及其子类。Buffer定义了一个线性存放primitive type数据的容器接口。对于除boolean以外的其他primitive type，都有一个相应的Buffer子类，ByteBuffer是其中最重要的一个子类。

下面这张UML类图描述了***java.nio***中的类的关系：



***Buffer***定义了一个可以线性存放primitive type数据的容器接口。***Buffer***主要包含了与类型（byte, char）无关的功能。值得注意的是***Buffer***及其子类都不是线程安全的。每个***Buffer***都有以下的属性：

* ***capacity*** 这个Buffer最多能放多少数据。***capacity***一般在buffer被创建的时候指定。
* ***limit*** 在Buffer上进行的读写操作都不能越过这个下标。当写数据到buffer中时，limit一般和capacity相等，当读数据时，limit代表buffer中有效数据的长度。
* ***position*** 读/写操作的当前下标。当使用buffer的相对位置进行读/写操作时，读/写会从这个下标进行，并在操作完成后，buffer会更新下标的值。
* ***mark*** 一个临时存放的位置下标。调用***mark***()会将***mark***设为当前的***position***的值，以后调用***reset***()会将***position***属性设置为***mark***的值。***mark***的值总是小于等于***position***的值，如果将***position***的值设的比***mark***小，当前的***mark***值会被抛弃掉。

这些属性总是满足以下条件：0 <= ***mark*** <= ***position*** <= ***limit*** <= ***capacity***

***limit***和***position***的值除了通过***limit***()和***position***()函数来设置，也可以通过下面这些函数来改变：

***Buffer*** ***clear***() 把position设为0，把limit设为capacity，一般在把数据写入Buffer前调用。

***Buffer*** ***flip***() 把limit设为当前position，把position设为0，一般在从Buffer读出数据前调用。

***Buffer*** ***rewind***() 把position设为0，limit不变，一般在把数据重写入Buffer前调用。

Buffer对象有可能是只读的，这时，任何对该对象的写操作都会触发一个***ReadOnlyBufferException***。***isReadOnly***()方法可以用来判断一个***Buffer***是否只读。

***ByteBuffer***在Buffer的子类中，ByteBuffer是一个地位较为特殊的类，因为在***java.io.channels***中定义的各种channel的IO操作基本上都是围绕ByteBuffer展开的。ByteBuffer定义了4个static方法来做创建工作：

ByteBuffer allocate(int capacity)创建一个指定capacity的ByteBuffer。***ByteBuffer allocateDirect***(***int capacity***)创建一个direct的***ByteBuffer***，这样的***ByteBuffer***在参与IO操作时性能会更好（很有可能是在底层的实现使用了DMA技术），相应的，创建和回收direct的ByteBuffer的代价也会高一些。***isDirect***()方法可以检查一个buffer是否是direct的。ByteBuffer wrap(byte [] array)，ByteBuffer wrap(byte [] array, int offset, int length)把一个byte数组或byte数组的一部分包装成ByteBuffer。ByteBuffer定义了一系列get和put操作来从中读写byte数据，如下面几个：

***byte get***()

***ByteBuffer get***(***byte*** [] dst)

***byte get***(***int*** index)

***ByteBuffer put***(***byte*** b)

***ByteBuffer put***(***byte*** [] src)

***ByteBuffer put***(***int*** index, ***byte*** b)

这些操作可分为绝对定位和相对定为两种，相对定位的读写操作依靠***position***来定位***Buffer***中的位置，并在操作完成后会更新***position***的值。在其它类型的***buffer***中，也定义了相同的函数来读写数据，唯一不同的就是一些参数和返回值的类型。

除了读写***byte***类型数据的函数，***ByteBuffer***的一个特别之处是它还定义了读写其它***primitive***数据的方法，如：***int getInt***() 从***ByteBuffer***中读出一个int值。***ByteBuffer putInt***(***int*** value)写入一个***int***值到***ByteBuffer***中。读写其它类型的数据牵涉到字节序问题，***ByteBuffer***会按其字节序（大字节序或小字节序）写入或读出一个其它类型的数据（***int***，***long***…）。字节序可以用***order***方法来取得和设置：***ByteOrder order***()返回***ByteBuffer***的字节序。***ByteBuffer order***(***ByteOrder*** bo)设置***ByteBuffer***的字节序。***ByteBuffer***另一个特别的地方是可以在它的基础上得到其它类型的***buffer***。如：***CharBuffer asCharBuffer***()为当前的***ByteBuffer***创建一个***CharBuffer***的视图。在该视图***buffer***中的读写操作会按照***ByteBuffer***的字节序作用到***ByteBuffer***中的数据上。

#### 程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | **import** java**.**nio**.\*;**  **import** java**.**io**.**IOException**;**  public class ByteBuffer\_read **{**  public static void main**(**String**[]** args**)** **throws** IOException **{**  ByteBuffer buf **=** ByteBuffer**.**allocate**(**256**);**  **while** **(true)** **{**  int c **=** System**.**in**.**read**();**  **if** **(**c **==** **-**1**)**  **break;**  buf**.**put**((**byte**)** c**);**  **if** **(**c **==** '\n'**)** **{**  buf**.**flip**();**  byte**[]** content **=** **new** byte**[**buf**.**limit**()];**  buf**.**get**(**content**);**  System**.**out**.**print**(new** String**(**content**));**  buf**.**clear**();**  **}**  **}**  **}**  **}** |

## 联机调试环境的搭建

通过手头的三台设备（一台纯Windows 10 Pro台式机，一台Windows 10 Pro与Ubuntu 18.04 Bionic双系统笔记本，一台纯Ubuntu 18.04 Bionic台式机），加上部署在Ubuntu端的SSH server程序以及部署在Windows端的SSH client程序，可以通过Vim编辑器在命令行交互环境下进行联机编程与调试。虚拟机环境与此类似，这里不再赘述。

# 实验体会

通过本次实验，基本搞清楚Telnet、SSH、FTP这三种网络协议的内涵以及如今的使用场景，完成了联机环境的搭建。对于Java网络编程类库还存在一些问题，Java语言之前并没有深入接触过，对于一些类库的API更是无从下手。

# 参考文献

[1] **林锐**. 高质量 C++/C 编程指南 [M]. 1.0 ed., 2001.

[2] java IO: https://zhuanlan.zhihu.com/p/21444494

[3] java NIO: https://www.jianshu.com/p/093b7c408dba

[4] java NIO: https://docs.oracle.com/javase/7/docs/api/java/nio/package-summary.html

[5] java NET: https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html