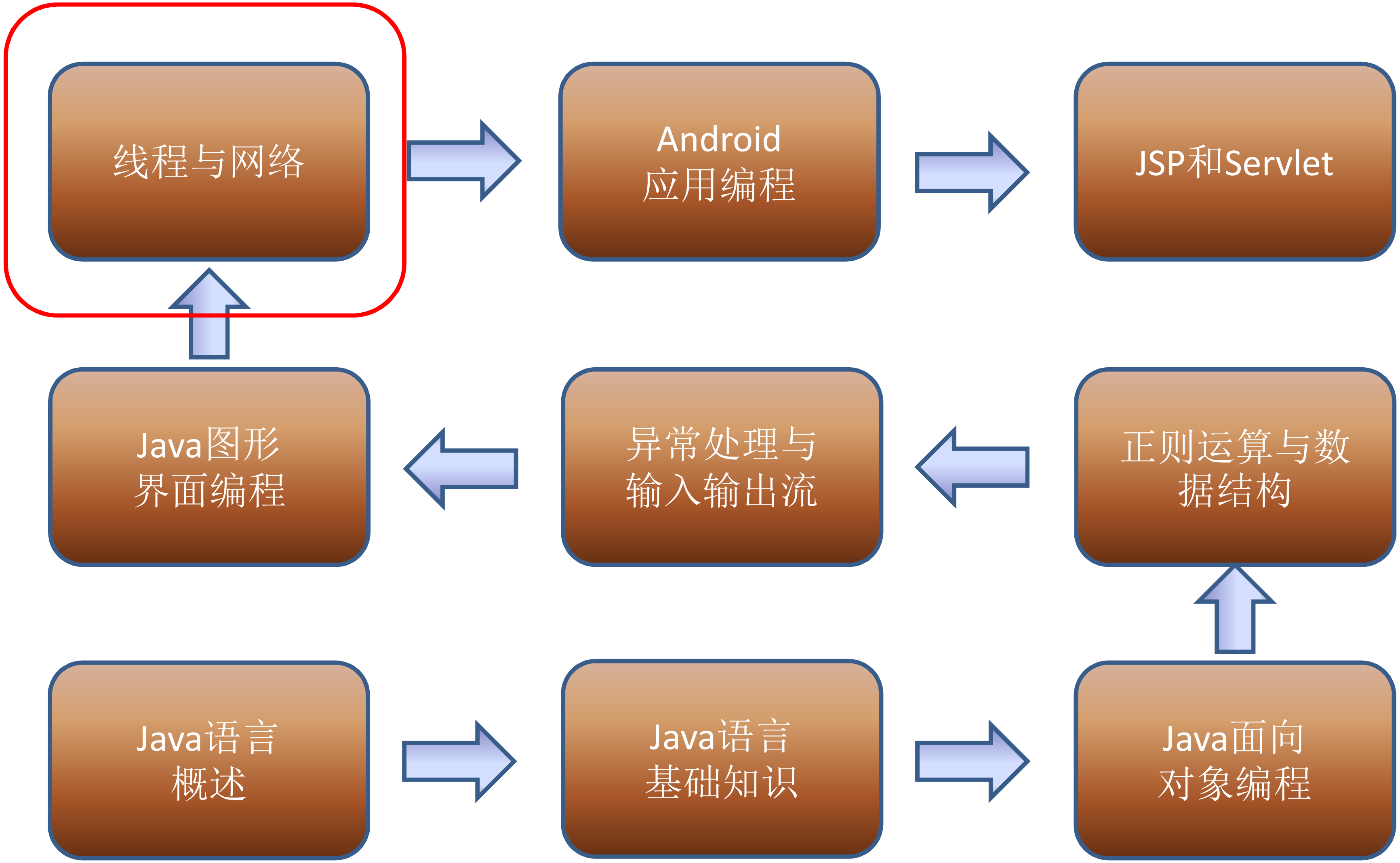
第七讲 线程与网络

课程内容安排


线程与网络
Android 应用编程
JSP和Servlet
Java图形界面编程
异常处理与输入输出流
正则运算与数据结构
Java语言概述
Java语言基础知识
Java面向对象编程

课前思考

1. 如何让程序在运行的过程中，实现多个程序段的同时运行？
2. 如何把新浪的网页抓下来？
3. 如何编写网络聊天程序？

学习目标

学习java中线程的使用，掌握线程的调度和控制方法，清楚地理解多线程的互斥和同步的实现原理，以及多线程的应用。

理解计算机网络编程的概念，掌握如何使用Java在一台或多台计算机之间进行基于TCP/IP协议的网络通讯。

难点和重点

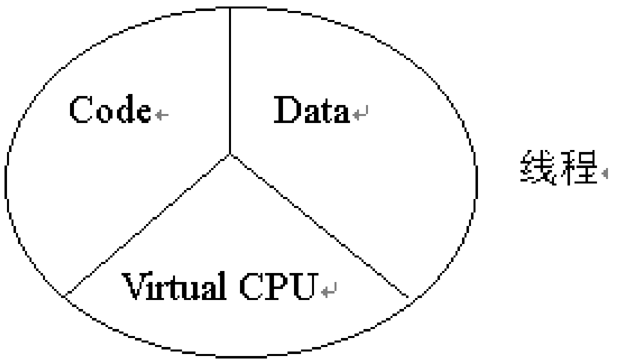
1. 多线程的调度和控制
2. 多线程的互斥和同步
3. 基于URL的网络编程（主要针对WWW资源）
4. 基于TCP的C/S网络编程（单客户，多客户）
5. 基于UDP的C/S网络编程

线程

- 一个线程是一个程序内部的顺序控制流。
- 线程和进程
  - 每个进程都有独立的代码和数据空间(进程上下文)，进程切换的开销大。
  - 线程：轻量的进程，同一类线程共享代码和数据空间，每个线程有独立的运行栈和程序计数器(PC)，线程切换的开销小。
  - 多进程：在操作系统中，能同时运行多个任务(程序)。
  - 多线程：在同一应用程序中，有多个顺序流同时执行。

线程的概念模型

- 虚拟的CPU，封装在java.lang.Thread类中。
- CPU所执行的代码，传递给Thread类。
- CPU所处理的数据，传递给Thread类。



线程体

- java的线程是通过java.lang.Thread类来实现的。
- 每个线程都是通过某个特定Thread对象的方法run( )来完成其操作的，方法run( )称为线程体。

通过继承类Thread构造线程体

```
class SimpleThread extends Thread {
  public SimpleThread(String str) {
    super(str);
  }
  public void run() {
    for (int i = 0; i < 10; i++) {
      System.out.println(i + " " + getName());
      try {
        sleep((int)(Math.random() * 1000));
      } catch (InterruptedException e) {}
    }
```

```
      System.out.println("DONE! " +getName());
    }
  }

public class TwoThreadsTest {
    public static void main (String args[]) {
     new SimpleThread("First").start();
     new SimpleThread("Second").start();
    }
 }
```

RUN

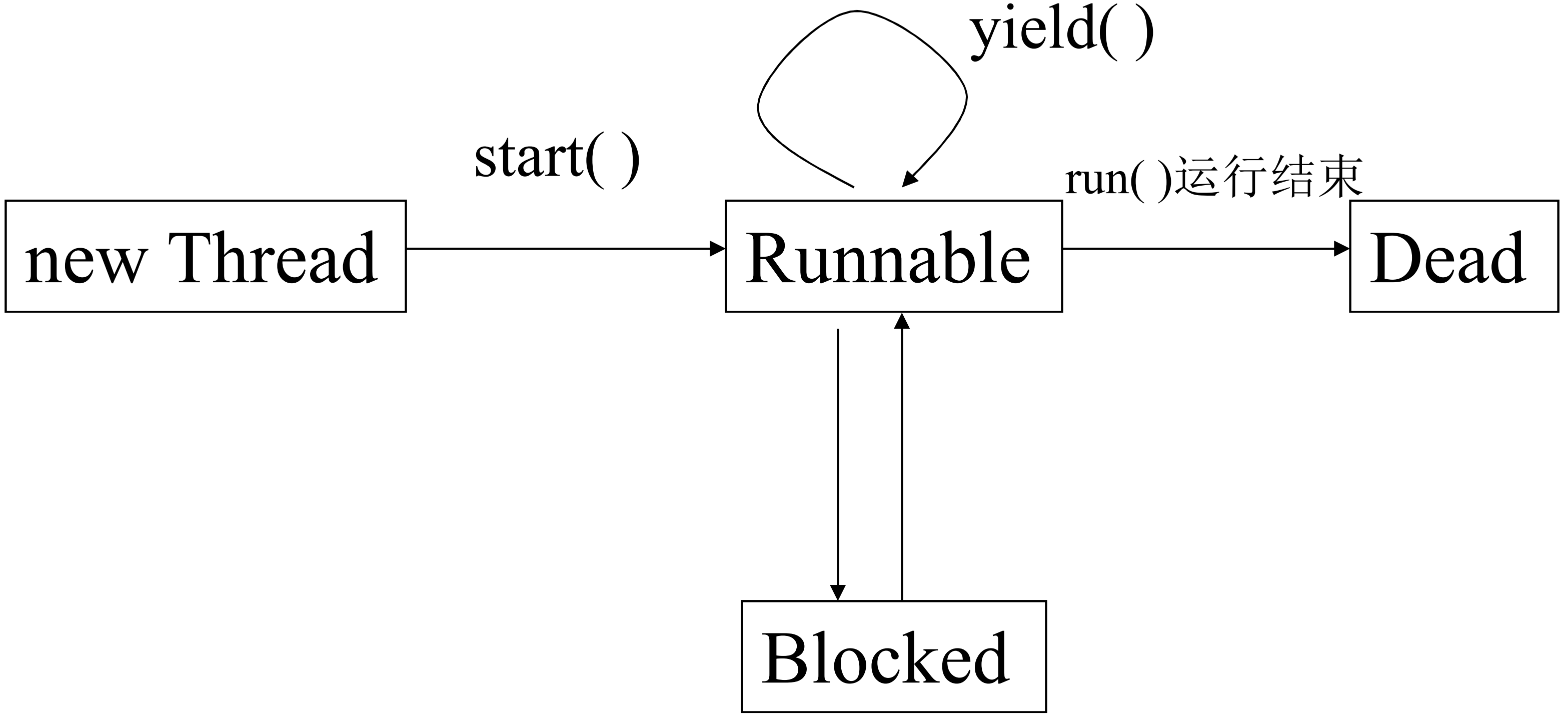
```
 0 First
 0 Second
 1 Second
 1 First
 2 First
 2 Second
 3 Second
 3 First
 4 First
 4 Second
 5 First
 5 Second
6 Second
```

```
 6 First
 7 First
 7 Second
 8 Second
 9 Second
 8 First
 DONE! Second
 9 First
DONE! First
```

线程的状态

- 创建状态（new）：线程对象已经创建，但尚未启动，所以不可运行。
- 可运行状态（runnable)：所有资源都准备好，就差CPU资源，一旦线程调度器分配CPU资源给该线程，立刻开始执行。
- 死亡状态（dead）：线程体执行完毕，即run( )方法运行结束。
- 堵塞状态（blocked）：不仅缺乏CPU资源，还缺乏其他资源

线程的状态



- 创建状态(new Thread)

  Thread myThread = new MyThreadClass( );

（注意：MyThreadClass是Thread的子类）

- 可运行状态(Runnable)

  Thread myThread = new MyThreadClass( );

  myThread.start( );

- 阻塞状态（Blocked）

  调用了sleep（）方法;

  为等候一个条件变量，线程调用wait（）方法;

  输入输出流中发生线程阻塞;
- 死亡状态（Dead）

  自然撤消（线程执行完）

- 下面几种情况下，当前线程会放弃CPU，进入阻塞状态（blocked):

1. 线程调用sleep()方法主动放弃；
2. 由于当前线程进行I/O访问，外存读写，等待用户输入等操作，导致线程阻塞；
3. 为等候一个条件变量，线程调用wait（）方法；
4. 线程试图调用另一个对象的“同步”方法，但那个对象处于锁定状态，暂时无法使用。

线程体的构造

- public Thread(ThreadGroup group, Runnable target, String name)；
- 任何实现接口Runnable的对象都可以作为一个线程的目标对象；

- 构造线程体的2种方法
  - 定义一个线程类，它继承类Thread并重写其中的方法run( )；
  - 提供一个实现接口Runnable的类作为线程的目标对象，在初始化一个Thread类或者Thread子类的线程对象时，把目标对象传递给这个线程实例，由该目标对象提供线程体run( )。

通过接口构造线程体

```
import java.util.*;
import java.awt.*;
import java.applet.*;
public class Clock extends Applet implements Runnable {
        Thread clockThread;
        public void start() {
         if (clockThread == null) {
             clockThread = new Thread(this, "Clock");
             clockThread.start();
          }
        }
```

```
public void run() {
    while (clockThread != null) {
     repaint();
     try {
          clockThread.sleep(1000);
       } catch (InterruptedException e){}
     }
}
```

```
public void paint(Graphics g) {
        Date now = new Date();
        Font f=new Font(“Italian”,Font.PLAIN,20);
        g.drawString(now.getHours() + ":" +
   now.getMinutes() + ":"        +now.getSeconds(), 5, 10);
   }
   public void stop() {
        clockThread.stop();
        clockThread = null;
   }
}
```

RUN

两种方法的比较

- 使用Runnable接口

  可以将CPU，代码和数据分开，形成清晰的模型;还可以从其他类继承;保持程序风格的一致性。

- 直接继承Thread类

  不能再从其他类继承;

  编写简单，可以直接操纵线程，无需使用Thread.currentThread()。

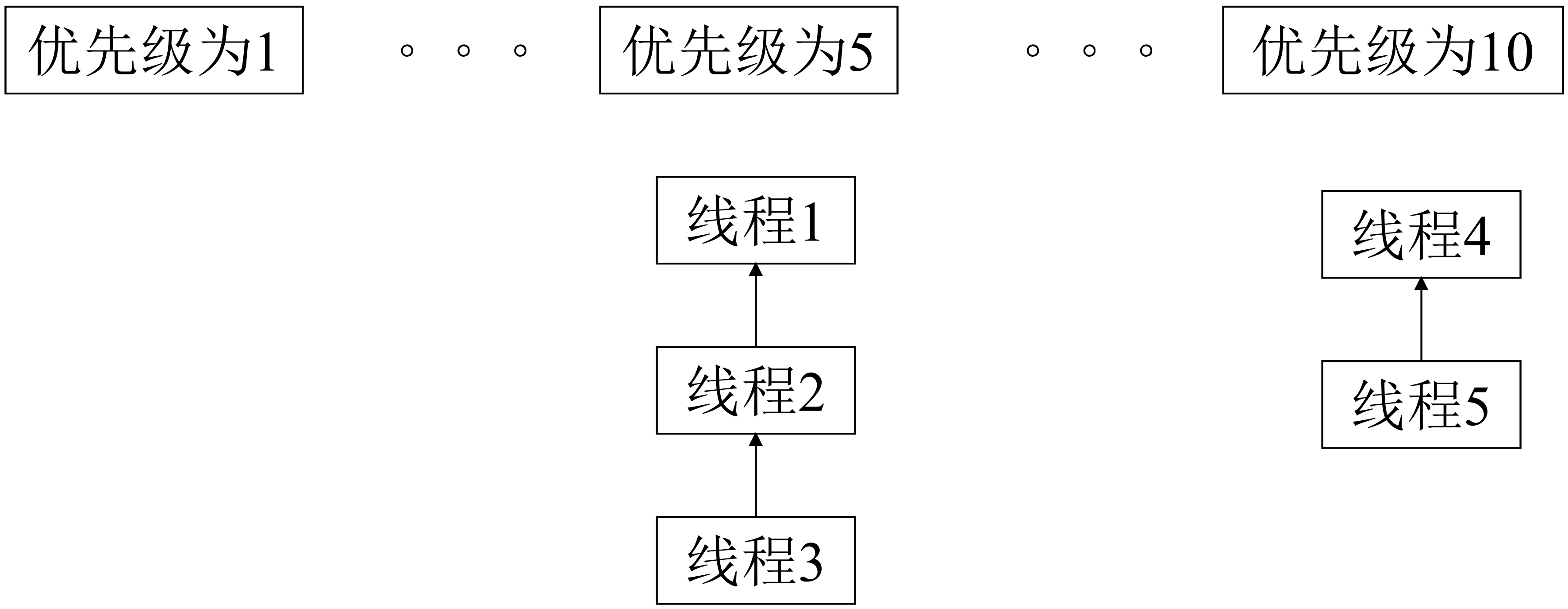
线程的调度

- java提供一个线程调度器来监控程序中启动后进入就绪状态的所有线程。线程调度器按照线程的优先级决定应调度哪些线程来执行。
- 线程的调度是抢先式的，按照优先级来调度：
  - 时间片方式
  - 非时间片方式

线程的优先级

- 线程的优先级用数字来表示，范围从1到10，即Thread.MIN_PRIORITY到Thread.MAX_PRIORITY。一个线程的缺省优先级是5，即Thread.NORM_PRIORITY。
- int getPriority();
- void setPriority(int newPriority);

线程调度队列（Runnable队列）



优先级为1 ○ ○ ○ 优先级为5 ○ ○ ○ 优先级为10

线程1

线程2

线程3

线程4

线程5

```
class ThreadTest{
    public static void main( String args [] ) {
     Thread t1 = new MyThread("T1");
     t1.setPriority( Thread.MIN_PRIORITY );
     t1.start( );
     Thread t2 = new MyThread("T2");
     t2.setPriority( Thread.MAX_PRIORITY );
     t2.start( );
     Thread t3 = new MyThread("T3");
      t3.setPriority( Thread.MAX_PRIORITY );
     t3.start( );
    }
}
```

```
class MyThread extends Thread {
    String message;
    MyThread ( String message ) {
        this.message = message;
    }
    public void run() {
        for ( int i=0; i<3; i++ )
          System.out.println( message+"   "+getPriority() );
    }
}
```

运行结果：

```
T2  10
T2  10
T2  10
T3  10
T3  10
T3  10
T1  1
T1  1
T1  1
```

- 注意：并不是在所有系统中运行Java程序时都采用时间片策略调度线程，所以一个线程在空闲时应该主动放弃CPU，以使其他同优先级（调用yield( )方法）和低优先级（调用sleep( )方法)的线程得到执行。

基本的线程控制

- 终止线程
  - 线程执行完其run()方法后，会自然终止。
- 测试线程状态
  - 可以通过Thread中的isAlive()方法来获取线程是否处于活动状态；
  - 线程由start()方法启动后，直到其被终止之间的任何时刻，都处于‘Alive’状态。

- 线程的暂停和恢复
  - sleep()方法

yield()方法

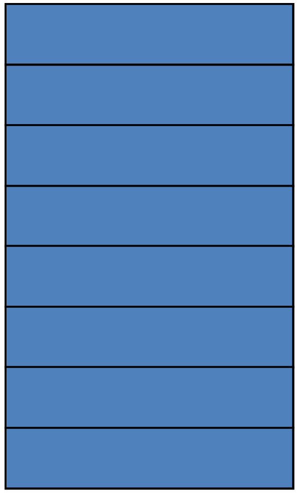
- 调用该方法的线程把自己的控制权让出来，线程调度器把该线程放到同一优先级的Runnable队列的最后，然后从该队列中取出下一个线程执行。该方法是给同优先级的线程以执行的机会，如果同优先级的Runnable队列中没有其它线程，则该线程继续执行。

多线程互斥与同步

线程 T1

```
class T1 extends Thread{
 Stack s;
 public T1(Stack s){
   this.s=s;
 }
 public void run(){
   s.pop();
   ……
 }
 ……
}
```

Stack对象s



线程 T2

```
class T2 extends Thread{
 Stack s;
 public T2(Stack s){
   this.s=s;
 }
 public void run(){
   s.push(char c);
   ……
 }
 ……
}
```

多线程的互斥与同步

- 临界资源问题

```
class Stack{
    int idx=0;
    char[ ] data = new char[6];

    public void push(char c){
      data[idx] = c;
      idx++;
    }
```

```
  public char pop(){
          idx--;
          return data[idx];
  }
}
```

两个线程A和B在同时使用Stack的同一个实例对象，A正在往堆栈里push一个数据，B则要从堆栈中pop一个数据。

1) 操作之前 data = | p | q | | | | | idx=2

2) A执行push中的第一个语句，将r推入堆栈；

data = | p | q | r | | | | idx=2

3) A还未执行idx++语句，A的执行被B中断，B执行pop方法，返回q：

data = | p | q | r | | | | idx=1

4〕A继续执行push的第二个语句：

data = | p | q | r | | , | | idx=2

最后的结果相当于r没有入栈。

- 产生这种问题的原因在于对共享数据访问的操作的不完整性。

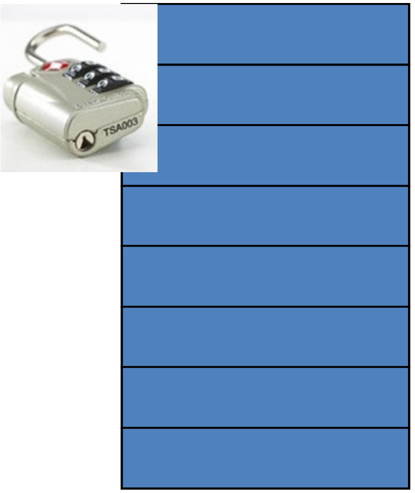
- 在Java 语言中，引入了对象互斥锁的概念，来保证共享数据操作的完整性。
  - 每个对象都对应于一个可称为“互斥锁”的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。
  - 关键字synchronized 来与对象的互斥锁联系。当某个方法用synchronized修饰时，表明该对象在任一时刻只能由一个线程访问。
  - 对象没被访问时候，其锁是打开的；当对象被某个线程访问时，锁就被该线程关上，其他线程就无法访问该对象，直到该线程访问完毕打开锁。

```
public void push(char c){
        synchronized(this){
        data[idx]=c;
        idx++;
        }
}
public char pop(){
synchronized(this){
        idx--;
        return data[idx];
        }
}
```

线程T1、T2未访问s前

线程 T1

```
class T1 extends Thread{
 Stack s;
 public T1(Stack s){
   this.s=s;
 }
 public void run(){
   s.pop();
   ……
 }
 ……
}
```



Stack对象s

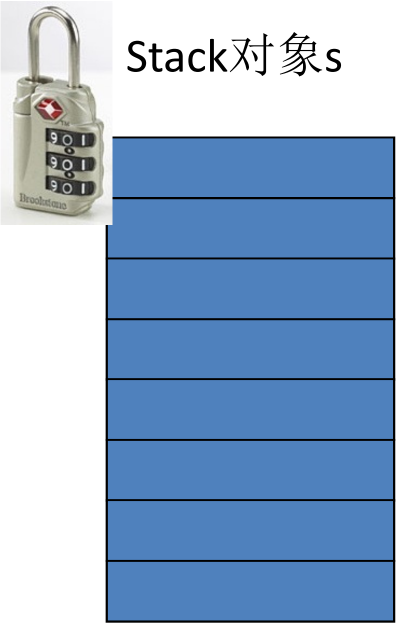
线程 T2

```
class T2 extends Thread{
 Stack s;
 public T2(Stack s){
   this.s=s;
 }
 public void run(){
   s.push(char c);
   ……
 }
 ……
}
```

线程T1拿到s的锁，关闭

线程 T1

```
class T1 extends Thread{
 Stack s;
 public T1(Stack s){
   this.s=s;
 }
 public void run(){
   s.pop();
   ……
 }
……
}
```



线程 T2

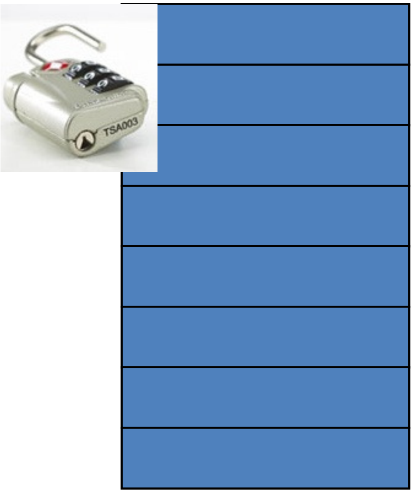
```
class T2 extends Thread{
 Stack s;
 public T2(Stack s){
   this.s=s;
 }
 public void run(){
   s.push(char c);
   ……
 }
……
}
```

线程T1访问s结束，释放锁

线程 T1

```
class T1 extends Thread{
 Stack s;
 public T1(Stack s){
   this.s=s;
 }
 public void run(){
   s.pop();
   ……
 }
 ……
}
```

Stack对象s



线程 T2

```
class T2 extends Thread{
 Stack s;
 public T2(Stack s){
   this.s=s;
 }
 public void run(){
   s.push(char c);
   ……
 }
 ……
}
```

- synchronized 除了象上面讲的放在对象前面限制一段代码的执行外，还可以放在方法声明中，表示整个方法为同步方法。

```
public synchronized void push(char c){
…
}
```

多线程的同步

```
class SyncStack{
  private int index = 0;
  private char []buffer = new char[6];

  public synchronized void push(char c){
        while(index == buffer.length){
        try{
                this.wait();
                }catch(InterruptedException e){}
}
```

```
    this.notify();
    buffer[index] = c;
    index++;
}

public synchronized char pop(){
    while(index ==0){
            try{
                        this.wait();
            }catch(InterruptedException e){}
    }
```

```
      this.notify();
      index- -;
      return buffer[index];
    }
}
```

生产者-消费者问题

```
class  Producer implements Runnable{
        SyncStack theStack;

        public Producer(SyncStack s){
                theStack = s;
        }
```

```
    public void run(){
      char c;
      for(int i=0; i<20; i++){
            c =(char)(Math.random()*26+'A');
            theStack.push(c);
            System.out.println("Produced: "+c);
            try{
            Thread.sleep((int)(Math.random()*1000));
            }catch(InterruptedException e){}
      }
    }
}
```

```
class Consumer implements Runnable{
    SyncStack theStack;

    public Consumer(SyncStack s){
        theStack = s;
    }
    public void run(){
      char c;
      for(int i=0;i<20;i++){
     c = theStack.pop();
     System.out.println("Consumed: "+c);
     try{
            Thread.sleep((int)(Math.random()*1000));
            }catch(InterruptedException e){}
      }
    }
}
```

```
public class SyncTest{
  public static void main(String args[]){
        SyncStack stack = new SyncStack();
        Runnable source=new Producer(stack);
        Runnable sink = new Consumer(stack);
        Thread t1 = new Thread(source);
        Thread t2 = new Thread(sink);
        t1.start();
        t2.start();
        }
}
```

RUN

程序执行结果

```
Produced:V
Consumed:V
Produced:E
Consumed:E
Produced:P
Produced:L
...
Consumed:L
Consumed:P
```

wait(),notify(),notifyAll()

(1) wait,nofity,notifyAll必须在已经持有锁的情况下执行,所以它们只能出现在synchronized作用的范围内.

(2) wait的作用:释放已持有的锁,进入wait队列.

(3) notify的作用:唤醒wait队列中的第一个线程并把它移入锁申请队列.

(4) notifyAll的作用:唤醒wait队列中的所有的线程并把它们移入锁申请队列.

网络编程

网络基础知识

- IP地址（32位，4个字节）

  如：166.111.136.3 , 166.111.52.80
- 主机名(hostname)

  如：www.tsinghua.edu.cn

  www.sun.com
- 端口号(port number)

  如:80，21，23，25,1~1024为保留端口号
- 服务类型(service)

  http, telnet, ftp, smtp

两类传输协议

- TCP (Transport Control Protocol)

  面向连接的能够提供可靠的流式数据传输的协议。类似于打电话的过程。

  URL, URLConnection, Socket, ServerSocket等类都使用TCP协议进行网络通讯。

- UDP (User Datagram Protocol)

  非面向连接、提供不可靠的数据包式数据传输的协议。类似于从邮局发送信件的过程。

  DatagramPacket, DatagramSocket, MulticastSocket等类使用UDP协议进行网络通讯。

通过URL读取WWW信息

```
import java.net.*;
import java.io.*;
public class URLReader {
  public static void main(String[] args) throws Exception {
        URL cs = new  URL("http://www.sina.com/");
        BufferedReader in = new BufferedReader(new
        InputStreamReader(cs.openStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
                System.out.println(inputLine);
        in.close();
 }
}
```

Run

URL类

- URL(Uniform Resource Locator)

  一致资源定位器的简称，它表示Internet上某一资源的地址。
- URL的组成

  protocol:resourceName

  协议名指明获取资源所使用的传输协议，如http、ftp、gopher、file等，资源名则应该是资源的完整地址，包括主机名、端口号、文件名或文件内部的一个引用。

- http://www.sun.com/
- http://home.netscape.com/home/welcome.html
- http://www.gamelan.com:80/Gamelan/network.html#BOTTOM
- file:///e:\download\Fop.htm

构造URL对象

- public URL(String spec)

```
URL urlBase = new URL( “http://www.gamelan.com/” );
```

- public URL(URL context, String spec)

```
URL gamelan =
        new URL("http://www.gamelan.com/pages/");
URL gamelanGames =
           new URL(gamelan, "Gamelan.game.html");
URL gamelanNetwork =
           new URL(gamelan, "Gamelan.net.html");
```

```
public URL(String  protocol, String  host, String  file);
new URL("http", "www.gamelan.com", "/pages/Gamelan.net.html");

public URL(String  protocol, String  host, int  port, String  file);
URL gamelan =new
URL("http","www.gamelan.com",80,"pages/Gamelan.network.html");
```

例外处理

```
try {
    URL myURL = new URL(. . .)
} catch (MalformedURLException e) {
    . . .
    // exception handler code here
    . . .
}
```

获取URL对象属性

- public String getProtocol( )
- public String getHost( )
- public String getPort( )
- public String getFile( )
- public String getRef( )

通过URLConnection读写WWW资源

```
import java.net.*;
import java.io.*;
public class URLConnector {
   public static void main(String[] args){
         try {
            URL cs = new URL("http://www.sina.com/");
            URLConnection tc = cs.openConnection();
            BufferedReader in = new BufferedReader(new
    InputStreamReader(tc.getInputStream()));
            String inputLine;
```

通过URLConnection读写WWW资源

```
while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
            in.close();
         }catch (MalformedURLException e)
   {System.out.println("MalformedURLException");}
catch (IOException e) {System.out.println("IOException");}
  }
}
```

Run

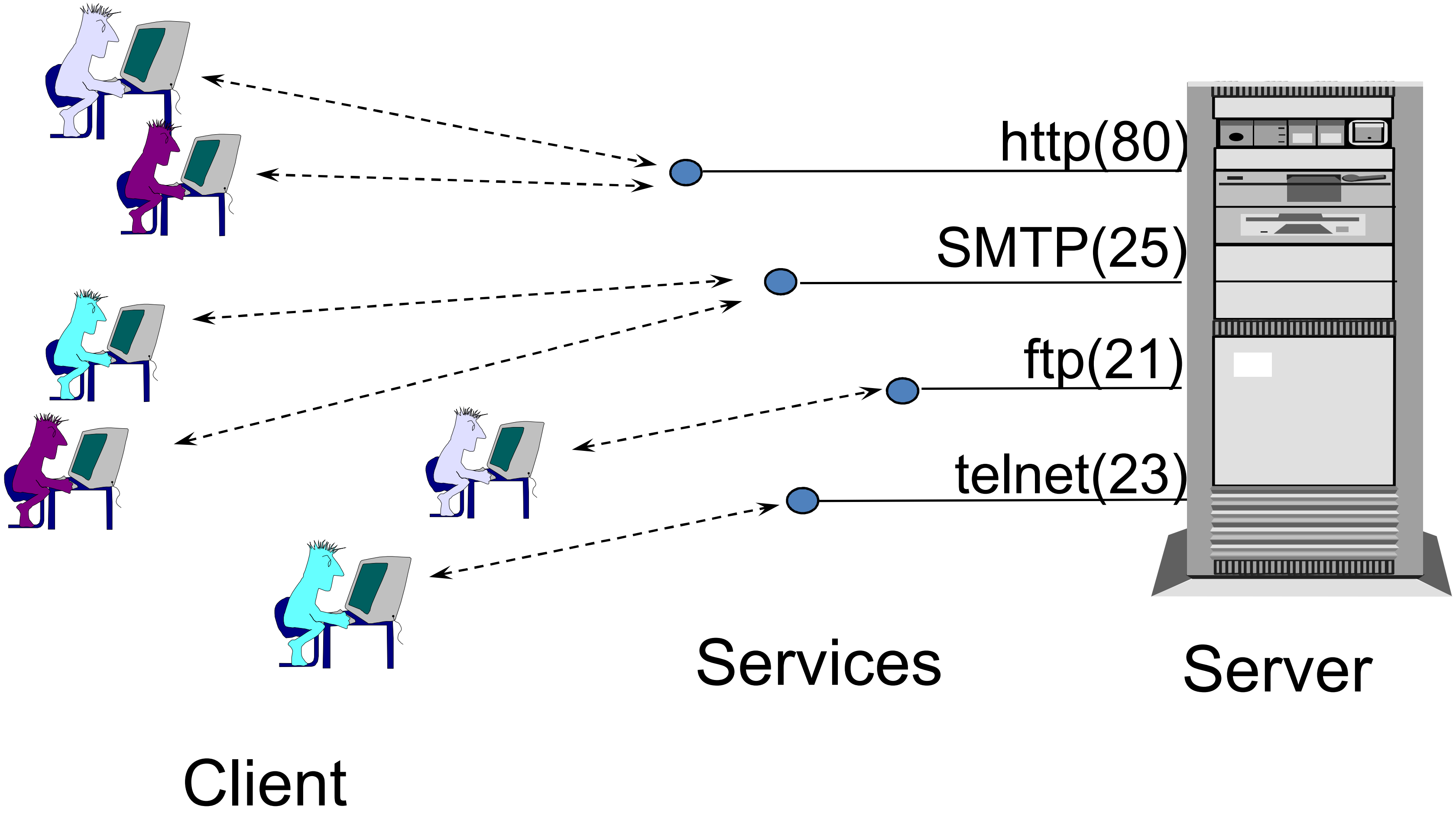
- 得到URLConnection对象之后，可以用如下方法得到相应的输入/输出流：
  - getInputStream( )
  - getOutputStream( )

  之后就可以读写输入/输出流，完成数据的读写。

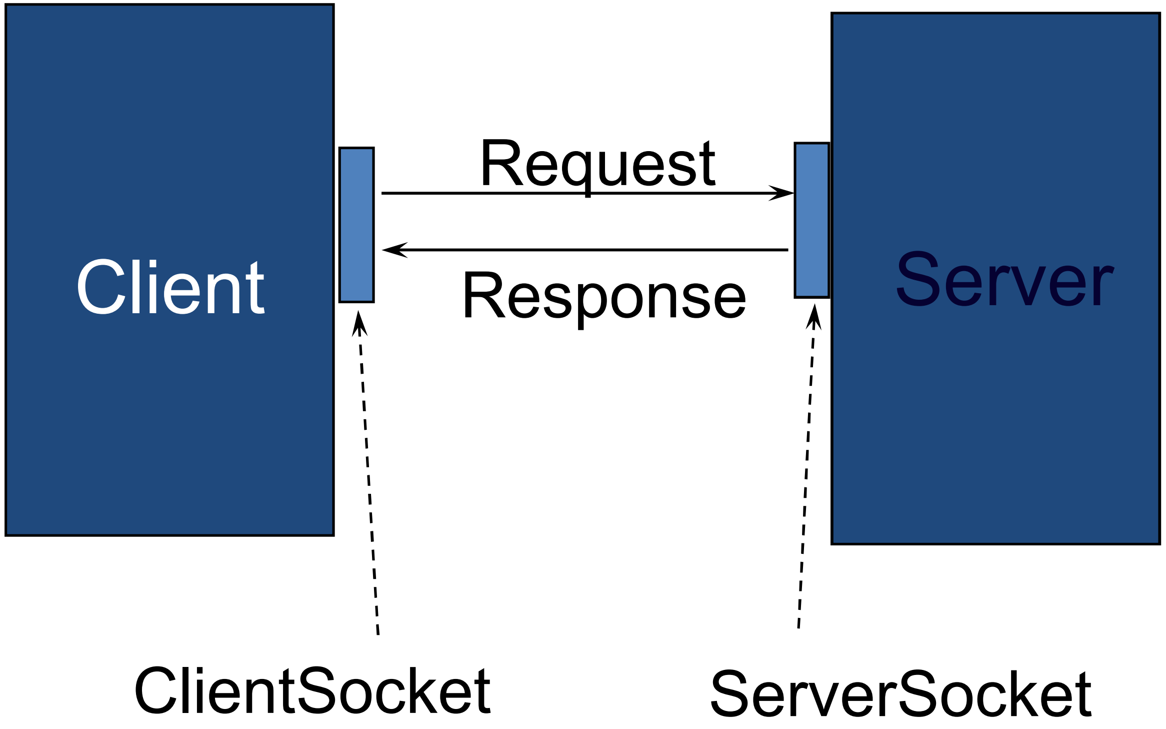
socket通讯

- 网络上的两个程序通过一个双向的通讯连接实现数据的交换，这个双向链路的一端称为一个socket。
- socket通常用来实现客户方和服务方的连接。

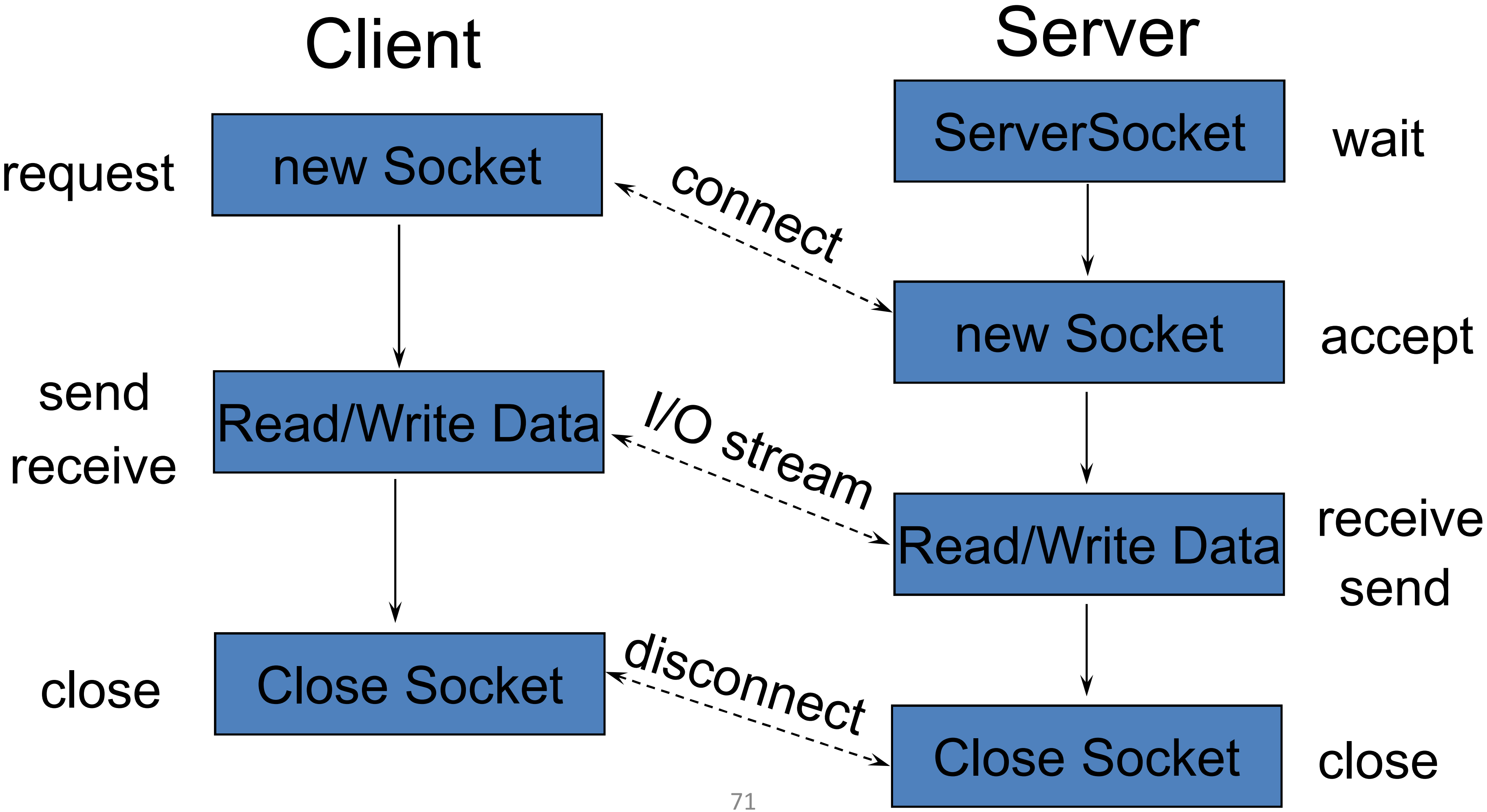
Client-server and Service



Socket Communication



Socket Programming Model



创建socket

- Socket( )
- Socket(InetAddress address, int port);
- Socket(String host, int port);
- Socket(InetAddress host, int port, InetAddress localAddr, int localPort)
- Socket(String host, int port, InetAddress localAddr, int localPort)

- 客户端Socket的建立

```
try{
  Socket socket=new Socket(”127.0.0.1",2000);
}catch(IOException e){
      System.out.println("Error:"+e);
}
```

- 服务器端Socket的建立

```
ServerSocket server=null;
try {
     server=new ServerSocket(2000);
}catch(IOException e){
    System.out.println("can not listen to :"+e);
}
Socket socket=null;
try {
   socket=server.accept();
}catch(IOException e){
    System.out.println("Error:"+e);
}
```

打开输入/输出流

```
PrintStream os=new PrintStream(new
   BufferedOutputStream(socket.getOutputStream()));
DataInputStream is=new
        DataInputStream(socket.getInputStream());
PrintWriter out=new
        PrintWriter(socket.getOutputStream(),true);
BufferedReader in=new
        BufferedReader(new
          InputStreamReader(socket.getInputStream()));
```

关闭socket

- os.close();
- is.close();
- socket.close();

- 注意关闭的顺序

```
import java.io.*;
import java.net.*;
public class TalkClient {
  public static void main(String args[]) {
        try{
          Socket socket=new Socket(“127.0.0.1”,4700);
          BufferedReader sin=new BufferedReader(new
  InputStreamReader(System.in));
          PrintWriter os=new
  PrintWriter(socket.getOutputStream());
```

```
BufferedReader is=new BufferedReader( new
    InputStreamReader(socket.getInputStream()));
String readline;
readline=sin.readLine();
while(!readline.equals(“bye”))
    {os.println(readline);
     os.flush();
     System.out.println(“Client:”+readline);
     System.out.println(“Server:”+is.readLine());
     readline=sin.readLine();
    }
    os.close();
    is.close();
```

```
socket.close();
}catch(Exception e) {
        System.out.println(“Error”+e);
        }
    }
}
```

服务器端程序:

```
import java.io.*;
import java.net.*;
import java.applet.Applet;
public class TalkServer{
    public static void main(String args[]) {
          try{
                    ServerSocket server=null;
                    try{
                        server=new ServerSocket(4700);
                    }catch(Exception e) {
                      System.out.println(“can not listen to:”
                                                             +e);}
```

```
Socket socket=null;
try{
    socket=server.accept();
}catch(Exception e) {
    System.out.println(“Error.”+e);
}
String line;
BufferedReader is=new BufferedReader(new
    InputStreamReader(socket.getInputStream()));
PrintWriter os=new
    PrintWriter(socket.getOutputStream());
BufferedReader sin=new BufferedReader(new
    InputStreamReader(System.in));
```

```
System.out.println(“Client:”+is.readLine());
line=sin.readLine();
while(!line.equals(“bye”))
{
    os.println(line);
    os.flush();
    System.out.println(“Server:”+line);
    System.out.println(“Client:”+is.readLine());
    line=sin.readLine();
}
os.close();
is.close();
socket.close();
```

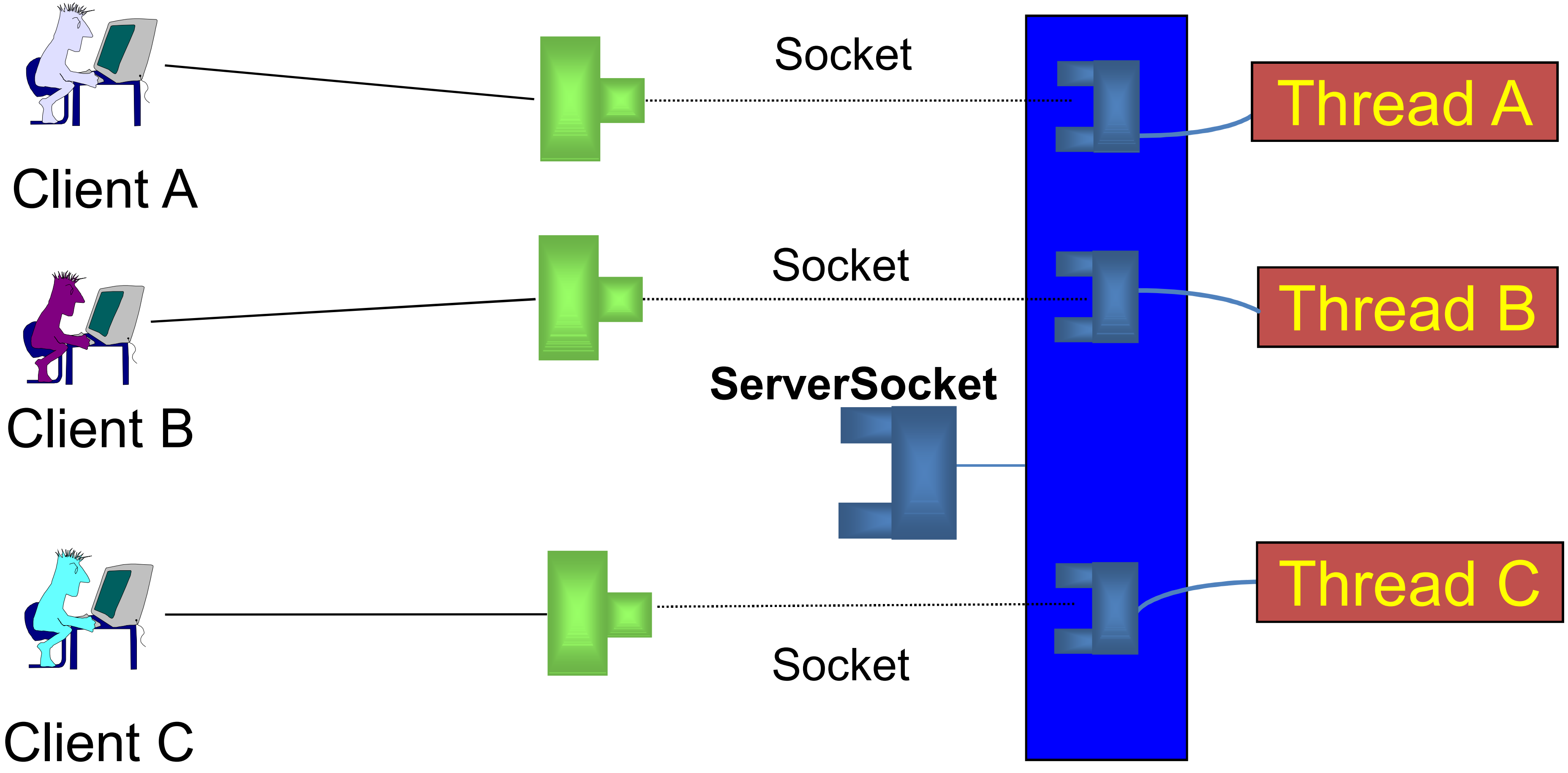
```
server.close();
}catch(Exception e){
        System.out.println(“Error:”+e);
        }
    }
}
```

运行结果

```
Client:hello!
Server:hello!
Client:how are you?
Server:I am fine,thank you!
…
Client:bye.
Server:bye.
```

Run Server　　　　*Run Client*

多客户机制


Socket
Thread A
Client A
Socket
Thread B
ServerSocket
Client B
Thread C
Socket
Client C

```
import java.io.*;
import java.net.*;
public class MultiTalkClient {
    int num;
    public static void main(String args[]) {
          try{
                    Socket socket=new Socket(“127.0.0.1”,4700);
                    BufferedReader sin=new BufferedReader(new
    InputStreamReader(System.in));
                    PrintWriter os=new
    PrintWriter(socket.getOutputStream());
```

```
BufferedReader is=new BufferedReader( new
   InputStreamReader(socket.getInputStream()));
String readline;
readline=sin.readLine();
while(!readline.equals(“bye”))
   {os.println(readline);
    os.flush();
    System.out.println(“Client:”+readline);
    System.out.println(“Server:”+is.readLine());
    readline=sin.readLine();
   }
   os.close();
   is.close();
```

```
socket.close();
}catch(Exception e) {
          System.out.println(“Error”+e);
          }
    }
}
```

服务器端程序: MultiTalkServer.java

```
import java.io.*;
import java.net.*;

public class MultiTalkServer{
   static int clientnum=0;
   public static void main(String args[]) throws IOException {
         ServerSocket serverSocket=null;
         boolean listening=true;
         try{   serverSocket=new ServerSocket(4700);
                  }catch(IOException e) {
                     System.out.println(“Could not listen
                                                    on port:4700.”);
System.exit(-1);
}
while(listening)
{
   new ServerThread(serverSocket.accept(),clientnum).start();
   clientnum++;
}
serverSocket.close();
   }
}
```

程序ServerThread.java

```
import java.io.*;
import java.net.*;
public class ServerThread extends Thread{
    Socket socket=null;
    int clientnum;
    public ServerThread(Socket socket,int num) {
        this.socket=socket;
        clientnum=num+1;
    }
    public void run() {
        try{
            String line;
```

```
BufferedReader is=new BufferedReader(new
    InputStreamReader(socket.getInputStream()));
PrintWriter os=new
    PrintWriter(socket.getOutputStream());
BufferedReader sin=new BufferedReader(new
    InputStreamReader(System.in));
System.out.println(“Client”+clientnum+ ”:” +
    is.readLine());
line=sin.readLine();
while(!line.equals(“bye”))
{
    os.println(line);
    os.flush();
```

```
     System.out.println(“Server:”+line);
     System.out.println(“Client:”+ clientnum + “:”
    +is.readLine());
     line=sin.readLine();
}
os.close();
is.close();
socket.close();
}catch(Exception e) {
    System.out.println(“Error:”+e);
    }
}
}
```

Run Server   *Client1*   *Client2*

数据报通信

- TCP (Transport Control Protocol)

  面向连接的能够提供可靠的流式数据传输的协议。类似于打电话的过程。

  URL, URLConnection, Socket, ServerSocket等类都使用TCP协议进行网络通讯。

- UDP (User Datagram Protocol)

  非面向连接的提供不可靠的数据包式的数据传输的协议。类似于从邮局发送信件的过程。

  DatagramPacket, DatagramSocket, MulticastSocket等类使用UDP协议进行网络通讯。

数据报通信

- TCP有建立时间
- UDP传输有大小限制:64K以内
- TCP的应用:Telnet, ftp
- UDP的应用:ping

数据报通信

- DatagramSocket( )
- DatagramSocket(int port)

- DatagramPacket(byte ibuf[],int ilength) //接收
- DatagramPacket(byte ibuf[],int ilength,
  InetAddress iaddr, int iport) ; //发送

数据报通信

- 收数据报:

```
DatagramPacket packet=new DatagramPacket (buf,256);
socket.receive(packet);
```

- 发数据报

```
DatagramPacket packet=new DatagramPacket
    (buf,buf.length,address,port);
socket.send(packet);
```

客户方程序 QuoteClient.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class QuoteClient {
  public static void main(String[] args) throws IOException {
  if(args.length!=1) {
      System.out.println(“Usage:java QuoteClient <hostname>”);
      return;
  }
DatagramSocket socket=new DatagramSocket();
```

```
//send request
byte[] buf=new byte[256];
InetAddress address=InetAddress.getByName(args
   [0]);
DatagramPacket packet=new DatagramPacket
   (buf,buf.length,address,4445);
socket.send(packet);
//get response
packet=new DatagramPacket(buf,buf.length);
socket.receive(packet);
//display response
String received=new String(packet.getData());
System.out.println(“Quote of the
   Moment:”+received );
```

```
        socket.close();
    }
}
```

服务器方程序:QuoteServer.java

```
public class QuoteServer{
    public static void main(String args[]) throws java.io.IOException {
    new QuoteServerThread().start();
    }
}
```

程序QuoteServerThread.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class QuoteServerThread extends Thread {
```

```
protected DatagramSocket socket=null;
protected BufferedReader in=null;
protected boolean moreQuotes=true;
public QuoteServerThread() throws IOException {
    this(“QuoteServerThread”);
}
public QuoteServerThread(String name) throws
    IOException {
    super(name);
    socket=new DatagramSocket(4445);
    try{ in= new BufferedReader(new FileReader(“ one-
    liners.txt”));
    }catch(FileNotFoundException e) {
```

```
System.err.println(“Could not open quote file.
    Serving time instead.”);
    }
}
public void run() {
while(moreQuotes) {
    try{
           byte[] buf=new byte[256];
           DatagramPacket packet=new
    DatagramPacket(buf,buf.length);
socket.receive(packet);
String dString=null;
if(in==null)   dString=new Date().toString();
else  dString=getNextQuote();
buf=dString.getBytes();
//send the response to the client at ”address” and
 //“port”
InetAddress address=packet.getAddress();
int port=packet.getPort();
packet=new
   DatagramPacket(buf,buf.length,address,port);
   socket.send(packet);
   }catch(IOException e) {
        e.printStackTrace();
        moreQuotes=false;
        }
```

```
    }
    socket.close();
}
protected String getNextQuote(){
    String returnValue=null;
    try {
        if((returnValue=in.readLine())==null) {
                in.close( );
                moreQuotes=false;
                returnValue="No more    quotes.Goodbye.";
                }
    }catch(IOException e) {
        returnValue=“IOException occurred in server”;
        }
    return returnValue;
    }
}
```

Run Server   *Run Client*

使用数据报进行广播通信

- DatagramSocket只允许数据报发往一个目的地址
- MulticastSocket将数据报以广播方式发送到该端口的所有客户.
- MulticastSocket用在客户端,监听服务器广播来的数据

客户方程序:MulticastClient.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class MulticastClient {
    public static void main(String args[]) throws IOException {
    MulticastSocket socket=new MulticastSocket(4446);
    InetAddress address=InetAddress.getByName(“230.0.0.1”);
    socket.joinGroup(address);
    DatagramPacket packet;
//get a few quotes
for(int i=0;i<5;i++) {
    byte[] buf=new byte[256];
    packet=new DatagramPacket(buf,buf.length);
    socket.receive(packet);
    String received=new String(packet.getData());
    System.out.println(“Quote of theMoment:”+received);
    }
    socket.leaveGroup(address);
    socket.close();
    }
}
```

服务器方程序:MulticastServer.java

```
public class MulticastServer{
    public static void main(String args[]) throws
    java.io.IOException {
    new MulticastServerThread().start();
    }
}
```

程序MulticastServerThread.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class MulticastServerThread extends
    QuoteServerThread {
```

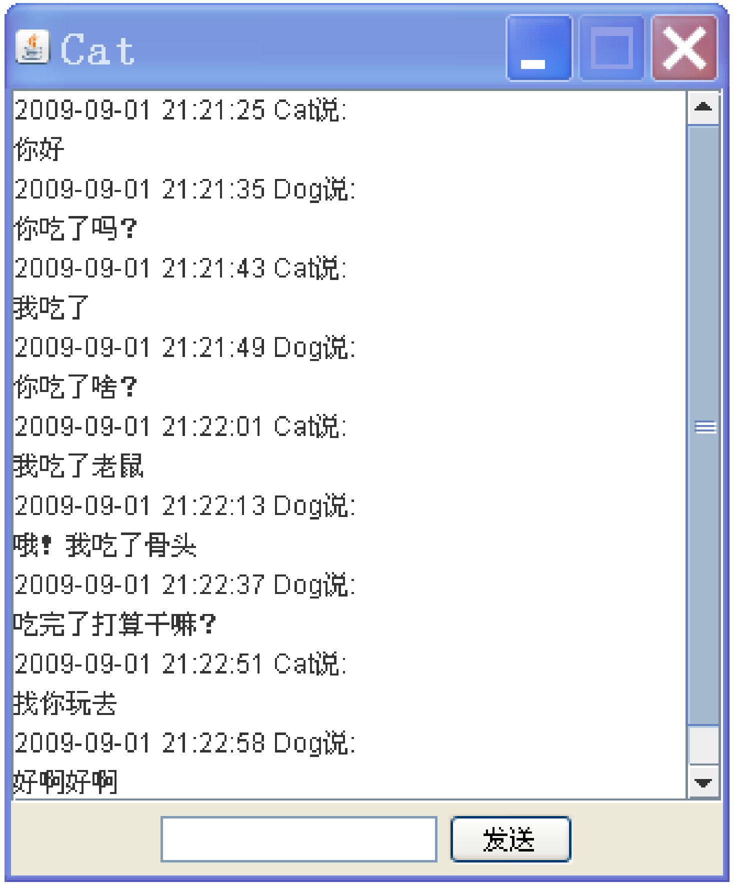
```
private long FIVE_SECOND=5000;
public MulticastServerThread() throws IOException {
   super(“MulticastServerThread”);
   }
public void run() {
while(moreQuotes) {
   try{
        byte[] buf=new byte[256];
   //construct quote
String dString=null;
if(in==null)   dString=new Date().toString();
   else  dString=getNextQuote();
```

```
buf=dString.getBytes();
//send it
InetAddress
    group=InetAddress.getByName(“230.0.0.1”);
DatagramPacket packet=new
    DatagramPacket(buf,buf.length,group,4446);
socket.send(packet);
//sleep for a while
    try{
    sleep((long)(Math.random()*FIVE_SECOND));
    }catch(InterruptedException e) { }
}catch(IOException e){
    e.printStackTrace( );
```

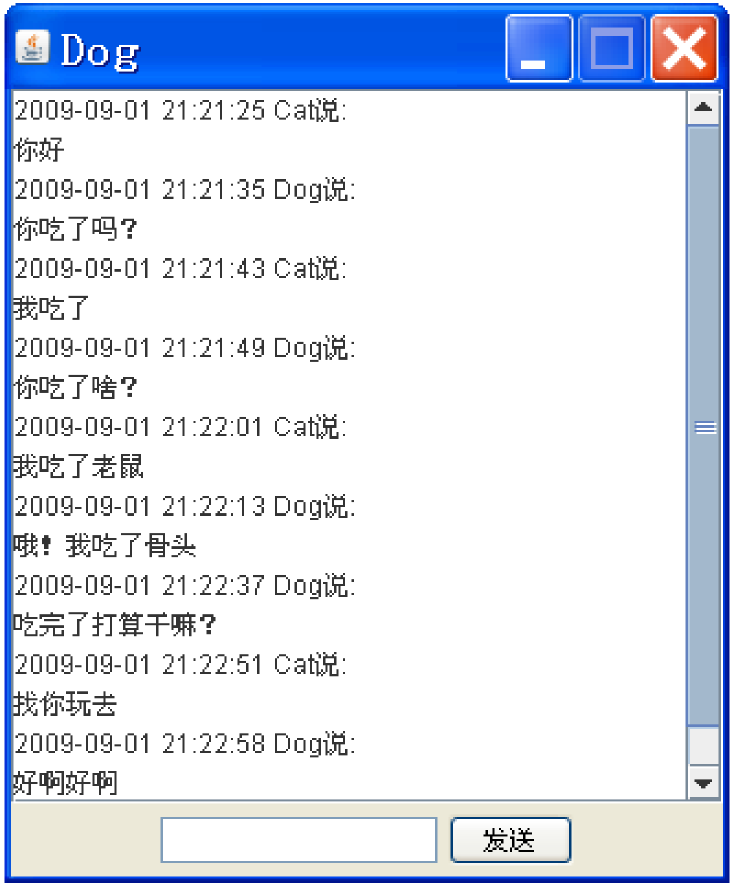
```
            moreQuotes=false;
            }
     }
     socket.close( );
     }
}
```

Client 1   *Client 2*   *Server*

带界面的聊天程序



```
2009-09-01 21:21:25 Cat说:
你好
2009-09-01 21:21:35 Dog说:
你吃了吗?
2009-09-01 21:21:43 Cat说:
我吃了
2009-09-01 21:21:49 Dog说:
你吃了啥?
2009-09-01 21:22:01 Cat说:
我吃了老鼠
2009-09-01 21:22:13 Dog说:
哦! 我吃了骨头
2009-09-01 21:22:37 Dog说:
吃完了打算干嘛?
2009-09-01 21:22:51 Cat说:
找你玩去
2009-09-01 21:22:58 Dog说:
好啊好啊
```



```
2009-09-01 21:21:25 Cat说:
你好
2009-09-01 21:21:35 Dog说:
你吃了吗?
2009-09-01 21:21:43 Cat说:
我吃了
2009-09-01 21:21:49 Dog说:
你吃了啥?
2009-09-01 21:22:01 Cat说:
我吃了老鼠
2009-09-01 21:22:13 Dog说:
哦! 我吃了骨头
2009-09-01 21:22:37 Dog说:
吃完了打算干嘛?
2009-09-01 21:22:51 Cat说:
找你玩去
2009-09-01 21:22:58 Dog说:
好啊好啊
```

Run Server

Run Client

```
import javax.swing.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;
import java.text.*;

public class ChatFrame extends JFrame implements
   ActionListener{
   JTextField tf;
   JTextArea ta;
   JScrollPane sp;
   JButton send;
   JPanel p;
```

```
 int port;
String s="";
String myID;
Date date;
ServerSocket server;
Socket mySocket;
BufferedReader is;
PrintWriter os;
String line;
```

```
public ChatFrame(String ID,String remoteID,String IP, int port, boolean
   isServer){
      super(ID);
      myID=ID;
      this.port=port;
      ta=new JTextArea();
      ta.setEditable(false);
      sp=new JScrollPane(ta);
      this.setSize(330,400);
      this.setResizable(false);
      try {

   UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClass
   Name());
      }catch(Exception e){
         System.out.println("UI error");
      }
```

```
this.getContentPane().add(sp,"Center");
p=new JPanel();
this.getContentPane().add(p,"South");
send=new JButton("发送");
tf=new JTextField(20);
tf.requestFocus();
p.add(tf);
p.add(send);
this.setDefaultCloseOperation(EXIT_ON_CLOSE);
this.setVisible(true);
send.addActionListener(this);
tf.addActionListener(this);
```

```
if (isServer){
       try{
               server=null;
               try{
                    server=new ServerSocket(port);
               }catch(Exception e) {
                  System.out.println("can not listen to:"+e);
            }
           mySocket=null;
           try{
               mySocket=server.accept();
               }catch(Exception e) {
                  System.out.println("Error."+e);
               }
            is=new BufferedReader(new
                    InputStreamReader(mySocket.getInputStream()));
            os=new PrintWriter(mySocket.getOutputStream());
           }catch(Exception e){
               System.out.println("Error: in server client socket"+e);
               }
       }//end of if
```

```
else {
        try{
                mySocket=new Socket(IP,port);
           os=new PrintWriter(mySocket.getOutputStream());
           is=new BufferedReader(new
      InputStreamReader(mySocket.getInputStream()));
        }catch(Exception e){
           System.out.println("Error: in client socket"+e);
           }
     }
```

```
while(true){
        try{
           line=is.readLine();
           date=new Date();
           SimpleDateFormat formatter = new
   SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
           String  currentTime= formatter.format(date);
           s+=currentTime+" "+remoteID+"说:\n"+line+"\n";
           ta.setText(s);
        }catch(Exception e){
         System.out.println("Error: in receive remote information"+e);
           }
     }
  }
```

```
 public void actionPerformed(ActionEvent e){
       date=new Date();
       SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
       String  currentTime= formatter.format(date);
       s+=currentTime+" "+myID+"说:\n"+tf.getText()+"\n";
       ta.setText(s);
       os.println(tf.getText());
        os.flush();
       tf.setText("");
       tf.requestFocus();
    }

}
```

```
public class ChatServerFrame {
    public static void main(String args[]){
        ChatFrame cserver=new
    ChatFrame("Cat","Dog","127.0.0.1",2009,
    true);
    }
}
```

```
public class ChatClientFrame {
    public static void main(String args[]){
        ChatFrame cclient=new
    ChatFrame("Dog","Cat","127.0.0.1",2009,
    false);
    }
}
```

小结

- 线程的互斥与同步
- 基于TCP和UDP的网络编程

谢谢！