Java面向对象编程

本节概要

- 抽象类与接口
- 异常处理
- 网络编程
- 进程与线程

清華大學 Tsinghua University

抽象类与接口类

抽象类的作用

◆ 为什么要定义抽象类?

- 抽象定义:
  - 抽象就是从多个事物中将共性的，本质的内容抽取出来。
  - 例如：狼和狗共性都是犬科，犬科就是抽象出来的概念。
- 抽象方法的由来:
  - 多个对象都具备相同的功能，但是功能具体内容有所不同，那么在抽取过程中，只抽取了功能定义，并未抽取功能主体，那么只有功能声明，没有功能主体的方法称为抽象方法。
  - 例如：狼和狗都有吼叫的方法，可是吼叫内容是不一样的。所以抽象出来的犬科虽然有吼叫功能，但是并不明确吼叫的细节。
- 在有良好定义的抽象类组成的程序中，逻辑更清晰，代码可读性更强

特殊的类：抽象类

- “抽象类”是使用关键字abstract修饰的类，它是一种特殊的类，**不能实例化对象，只能被继承用来声明子类**

java.lang.InstantiationException

- 如果子类没有给出抽象方法（abstract method ）的实现，这个子类仍然为抽象类，它被再次继承的时候对应方法仍然保持为抽象方法

```
abstract class Account {
    public String accountID;
    private double amount;
    public double interest;
    public abstract void calInterest();
    public void setBalance(double a) {…//函数体}
    public double getBalance() {{…//函数体}
}
```

在上面的例子中，calInterest是抽象方法，所有非抽象子类必须实现这个函数；setBalance和getBalance是普通的成员方法

特殊的类：抽象类

- 然后我们就可以定义SavingAccount（存款账户）来具体化这个抽象类

```
class SavingAccount extends Account {
    public boolean haveCard;
    public SavingAccount (String id, double amount, double interest,
                          boolean haveCard) {……}
    public void calInterest(){
        double amount = getBalance();
        System.out.println("利息：" + (amount * interest));
    }
}
```

- 在这个子类中，我们实现了父类中的抽象方法，因此可以使用这个子类来进行对象的实例化

抽象类的进一步抽象：接口

- 接口是抽象类的一种，只包含常量和方法的定义，而没有变量和方法的实现，且其方法都是抽象方法。它的用处体现在下面几个方面
  - 通过接口实现不相关类的相同行为,而无需考虑这些类之间的关系
  - 通过接口指明多个类需要实现的方法
  - 通过接口了解对象的交互界面,而无需了解对象所对应的类

- 抽象类是通过类继承关系对类的抽象
- 接口是对行为的抽象

抽象类的进一步抽象：接口

- 声明接口格式：
  interface {}
- 实现接口关键字 implements
- 接口中的成员修饰符是固定的。
  - 成员常量：public static final
  - 成员函数：public abstract
  - 接口中的成员都是public的。

- A继承父类B：A is a B
- A实现接口B：A can do B

```
public interface Iflying{
    public void fly(){}
}
```

```
public class bird implements Iflying{
    public void fly(){
        System.out.println("The bird is
flying");
    }
}
```

```
public class airplane implements Iflying{
    public void fly(){
        System.out.println("The airplane
is flying");
    }
}
```

接口与抽象类

- 接口中没有任何一个方法有缺省的实现
- 抽象类中有的方法有缺省的实现
- 一个类只能继承一个其他的类，而一个类可以实现多个接口

——抽象类更多的是使用父类提供的缺省方法以及相关扩展方法

——接口主要强调各实现类的区别，同一方法名有不同的表现

课堂练习（1）

- 编写Java程序，在程序中定义一个抽象类Person和它的两个子类Student和Employee类。要求：
  - Person抽象类只包含一个抽象方法total()；
  - Student子类拥有3次考试成绩，total()方法可以计算总分；
  - Employee子类拥有hours的每月工作时数，total()方法可以计算每日工时（一月24天）和工资，每一小时800元。

清華大學 Tsinghua University

异常处理

Java 中的异常处理

- Java中的“异常”(Exception)是指异常对象，也就是一种异常事件。在程序运行时，有时候程序会进入一些不正常的执行状态（比如数组下标访问越界）。

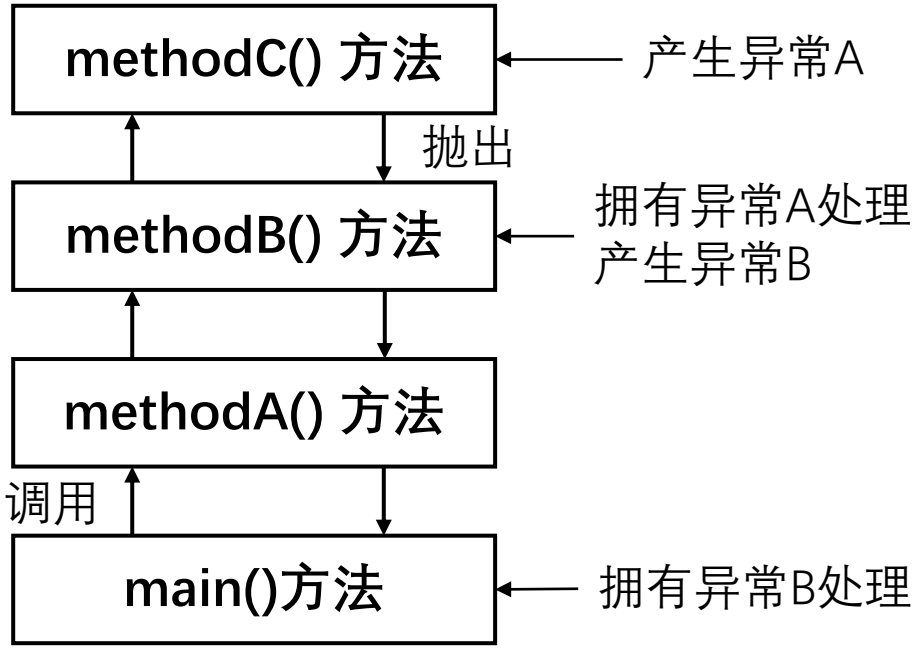
  java HelloWorld xxx arg[1]

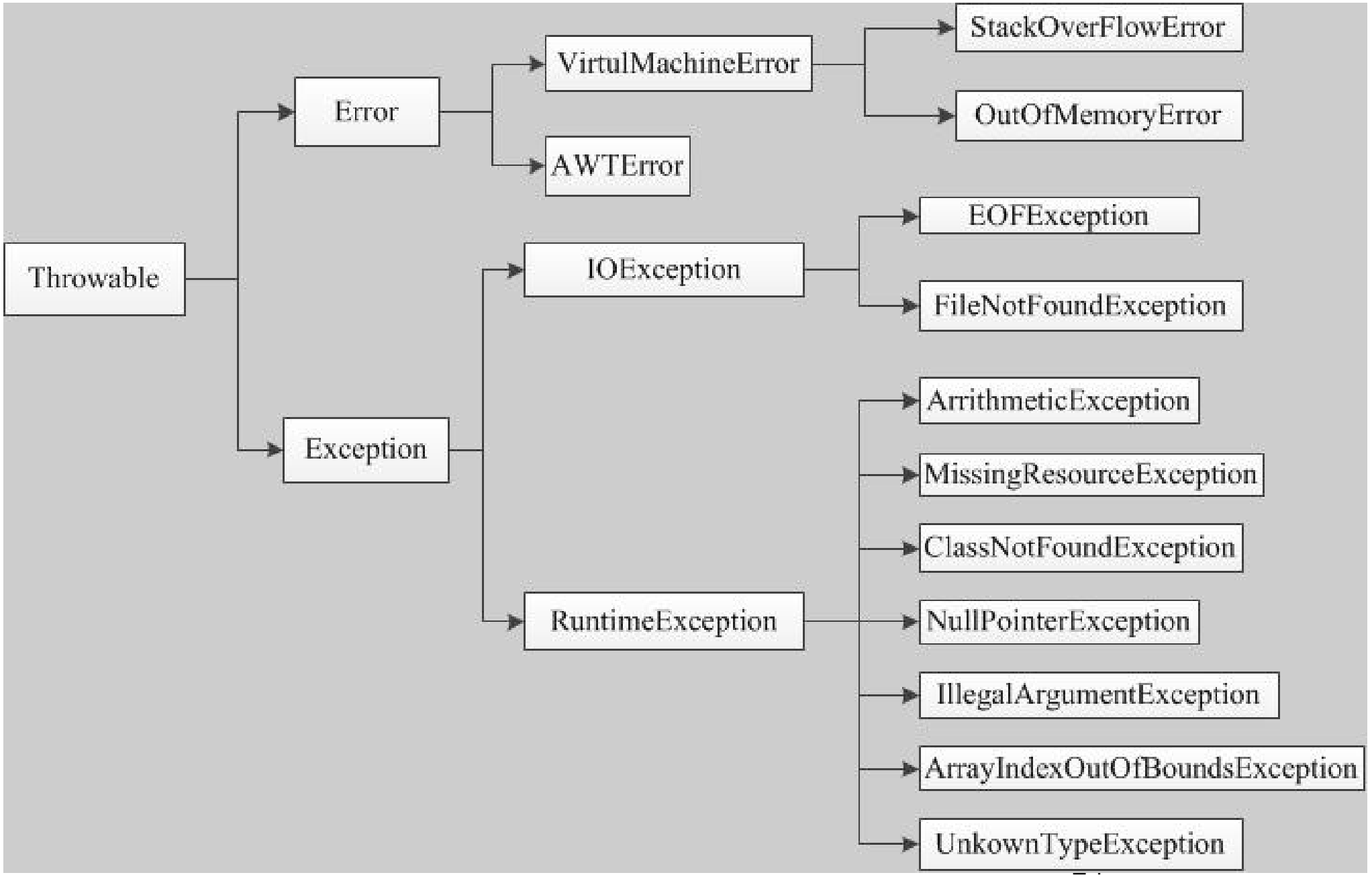
- “异常处理”(Handling Exceptions)就是用来处理程序产生的异常事件。
- “异常处理”的目的是为了让程序更加健壮(Robust)，防止程序在异常状态中“崩溃”

异常处理的架构

- Java异常处理架构是一种“你丢我捡”的架构
- 当JVM执行Java程序发生错误时，就会产生异常对象。
- JVM处理异常的两种方法：
  - 在发生异常的地方寻找异常处理语句进行处理
  - 抛出异常，由调用该函数的函数进行处理

如右图所示，在mechodC和methodB方法中发生的异常，会层层抛出，直至到找到对应的异常处理语句，如果最上层的main函数都没有异常处理语句，程序会直接终止，Console打印异常信息（调试定位错误类型的技巧）




Throwable
Error
VirtulMachineError
StackOverFlowError
OutOfMemoryError
AWTError
Exception
IOException
EOFException
FileNotFoundException
RuntimeException
ArrithmeticException
MissingResourceException
ClassNotFoundException
NullPointerException
IllegalArgumentException
ArrayIndexOutOfBoundsException
UnkownTypeException

异常处理语句

◆在Java中，异常处理语句是让函数具备异常处理功能的程序块。其格式如下

```
try {
    ……
} catch (ExceptionType e) {
    // 异常处理语句
    ……
} finally {
    ……
}
```

例如：

```
public int divide(int a, int b){
    int c = 0;
    try {
        int c = a/b;
    } catch (ArithmeticException e) {
        System.out.println(‘错误！除数
    不能为0’);
    } finally {
        return c;
    }
}
```

- try程序块：业务代码，可能会产生异常
- catch程序块：当try程序块的程序抛出异常e，就会执行这一块的代码
- finnaly程序块(可省略)：不论异常是否发生，都会执行的代码

异常处理语句

- 在上面的代码中，ExceptionType可以是Java已经定义好的多种Exception（都是Exception的子类）
- 如果try中的语句可能会产生不止一种Exception，而每种Exception的处理方法不一样，那么可以写多个Exception块进行分别处理，如：
- 我们也可以不指定具体的Exception子类，直接捕捉Exception类的对象（也就是捕捉所有类型的异常）
- 在catch块中，e是异常对象本身，它具备以下方法

方法	说明
String getMessage()	返回异常说明的字符串
void printStackTree()	显示程序调用的执行过程

```
try {
    ……
} catch (ExceptionType1 e) {
    // 异常1处理语句
    ……
} catch (ExceptionType2 e) {
    // 异常2处理语句
    ……
} finally {
    ……
}
```

异常处理语句

◆例程

```
public class ExceptionTest {
   // 类方法: 显示异常信息
   static void printErrMsg(Exception e) {
      System.out.println("异常说明: " +
e.getMessage());
      System.out.println("异常原因: ");
      e.printStackTrace();
   }
   // 主程序
   public static void main(String[] args) {
      int i;
      int[] data = {22, 14, 36, 68, 87};
      // 异常处理程序语句
      try {
         int index = (int)(Math.random()*10);
         i = data[index];  // 产生超过数组范围异常
         // 产生零除的异常
         for ( i = 2; i > -1; i-- )
            System.out.println("计算结果: " +10/i);
      }
      catch ( ArithmeticException e ) {
         // 处理零除的异常
         printErrMsg(e);
      }
      catch ( ArrayIndexOutOfBoundsException e ) {
         // 处理超过数组范围异常
         printErrMsg(e);
      }
      finally {
         System.out.println("异常处理结束");
      }
   }
}
```

练一练

抛出异常

- 前面所述的异常是JVM在运行时产生的异常。而在我们自己写的代码中，我们也可以手动抛出异常

1. 使用throw关键字手动抛出异常

```
throw new ExceptionType(“异常说明”);
```

比如：

```
int index = (int)(Math.random() * 10);
try{
    if (index < 5) {
        throw new Exception(“值小于5”);
    }
}catch (Exception e){
    System.out.println(“异常原因” + e.getMessage());
}
```

抛出异常

- 前面所述的异常是JVM在运行时产生的异常。而在我们自己写的代码中，我们也可以手动抛出异常

2. 在方法抛出异常

有的时候虽然方法产生了异常，但是我们并不希望在当前函数中处理（比如在上层的某一个地方集中处理），我们就需要将当前的异常抛出到上层

比如：

```
public void test() throws Exception{
    int index = (int)(Math.random() * 10);
    if (index < 5) {
        throw new Exception(“值小于5”);
    }
    ……
}
```

如果if条件语句被触发，程序会往上层（即调用当前函数的函数）抛出Exception，当前函数提前终止。

自定义异常

◆自定义Exception类

我们可以自行继承Exception类，进行异常的自定义，比如：

◆ Exception类的自定义

```
class UserException extends Exception {
   private static final long serialVersionUID = 1L;
   // 变量声明
   int data;
   // 构造函数
   public UserException(int data) {
      this.data = data;
   }
   // 重写getMessaeg()方法
   public String getMessage() {
      return ("出价次数太多: " + data);
   }
}
```

抛出异常与自定义异常

◆自定义Exception类

我们可以自行继承Exception类，进行异常的自定义，比如：

◆测试主函数

```
public class Ch10_2_3 {
    // 主程序
    public static void main(String[] args) {
       try {
          int i;
          for ( i = 0; i < 10; i++ ) {
             if ( i == 5 ) {
                // 丢出自定义的异常
                throw new UserException(5);
             }
             System.out.println("出价次数: " + i);
          }
       }
       catch( UserException e ) {
          // 处理自定义的异常
          System.out.println("异常说明: "+e.getMessage()); }
       finally {
          System.out.println("异常处理结束!");
       }
    }
 }
```

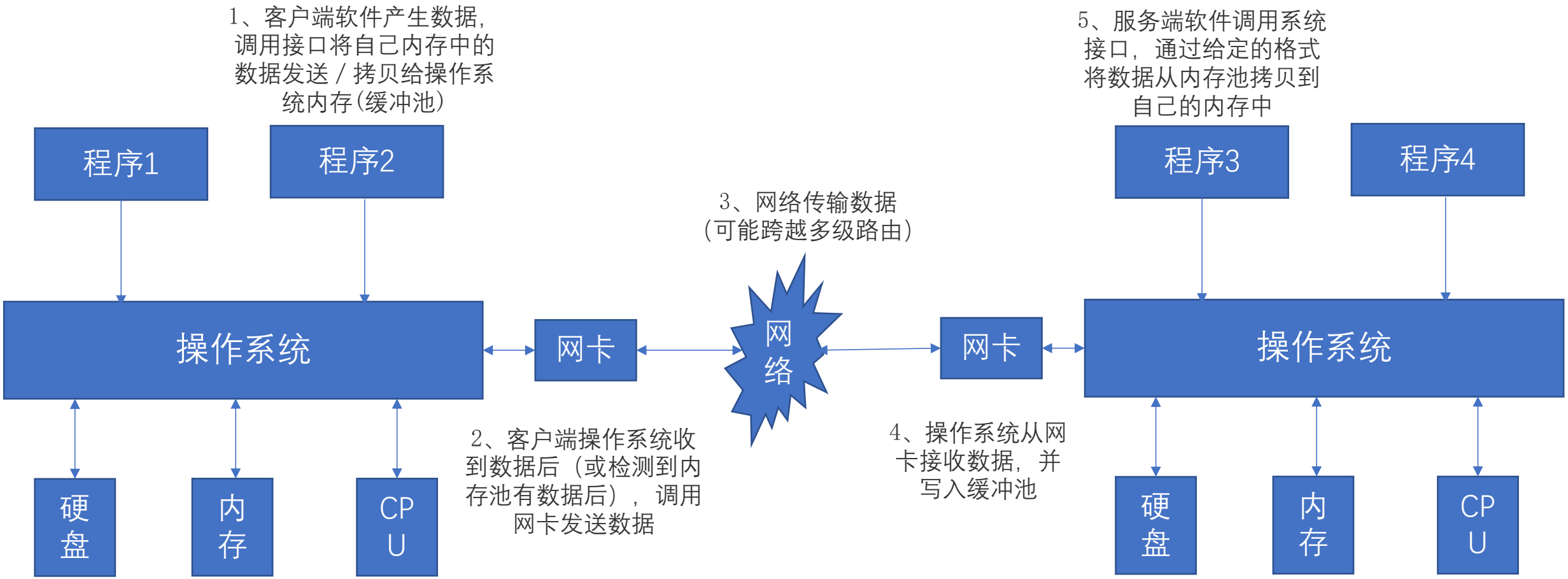
练一练

课堂练习（2）

- 请写出printNum(int)方法来显示2n+1的数列，例如：1、3、5、7……，其参数是整数int且实现抛出下列异常对象：
  - IllegalArgumentException，当参数小于0。
  - ArithmeticException，当参数大于100。

网络编程

计算机网络通讯基本流程



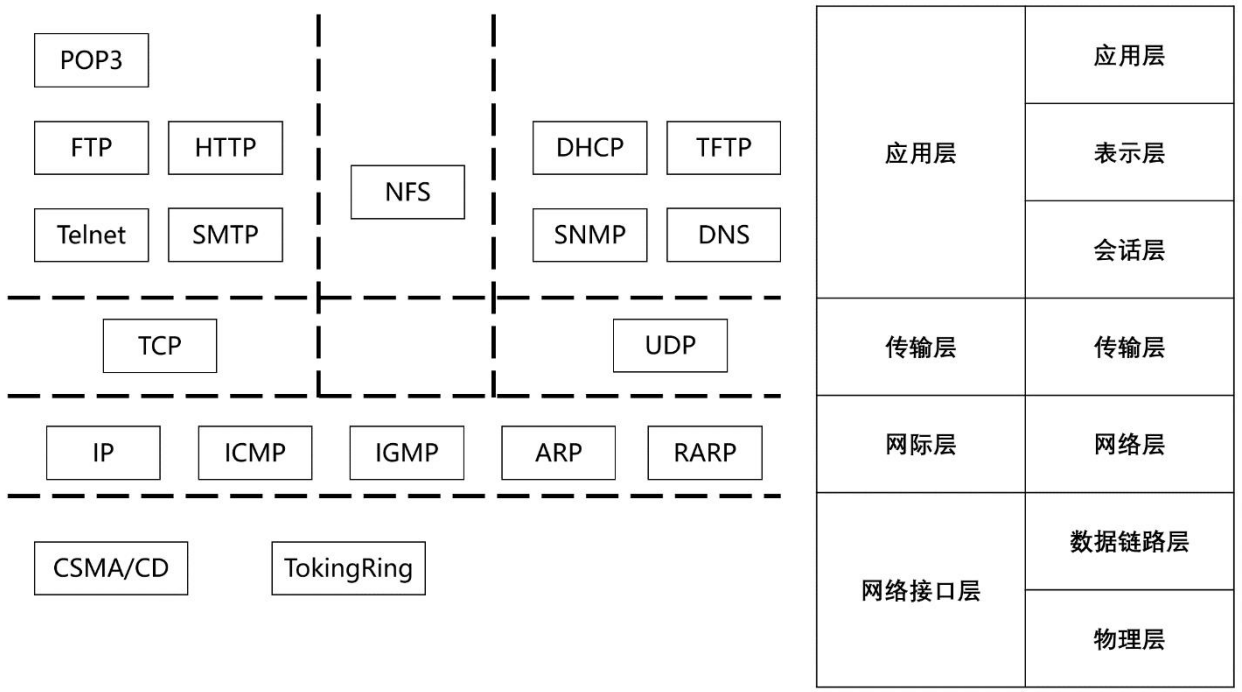
网络编程的基本概念

- OSI网络参考模型

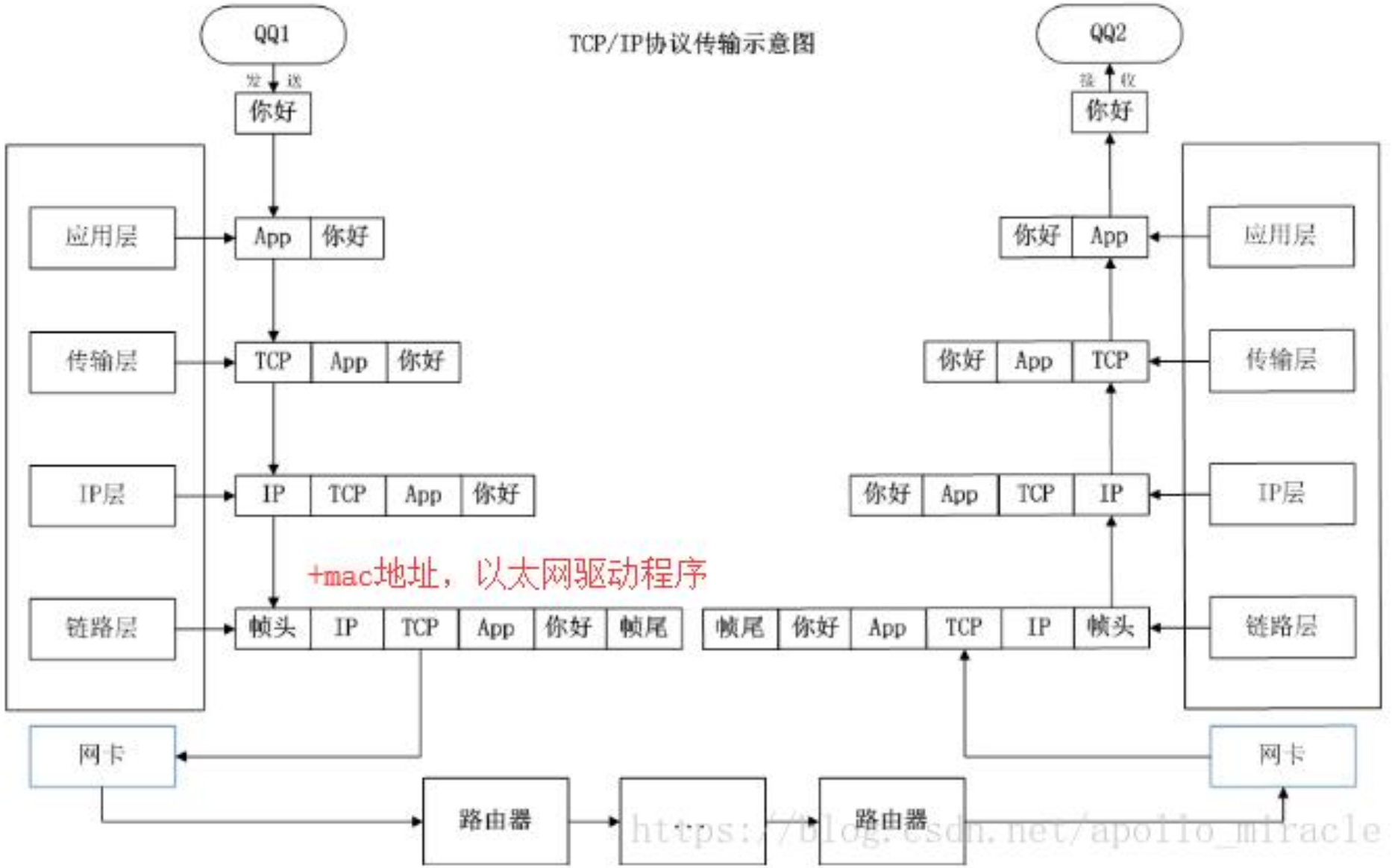
  包括七个层次：应用层、表示层、会话层、传输层、网络层、链路层、物理层

- TCP/IP（Transmission Control Protocol/Internet Protocol，传输控制协议/网间协议）

  包括五个层次：应用层、传输层、网络层、链路层、物理层

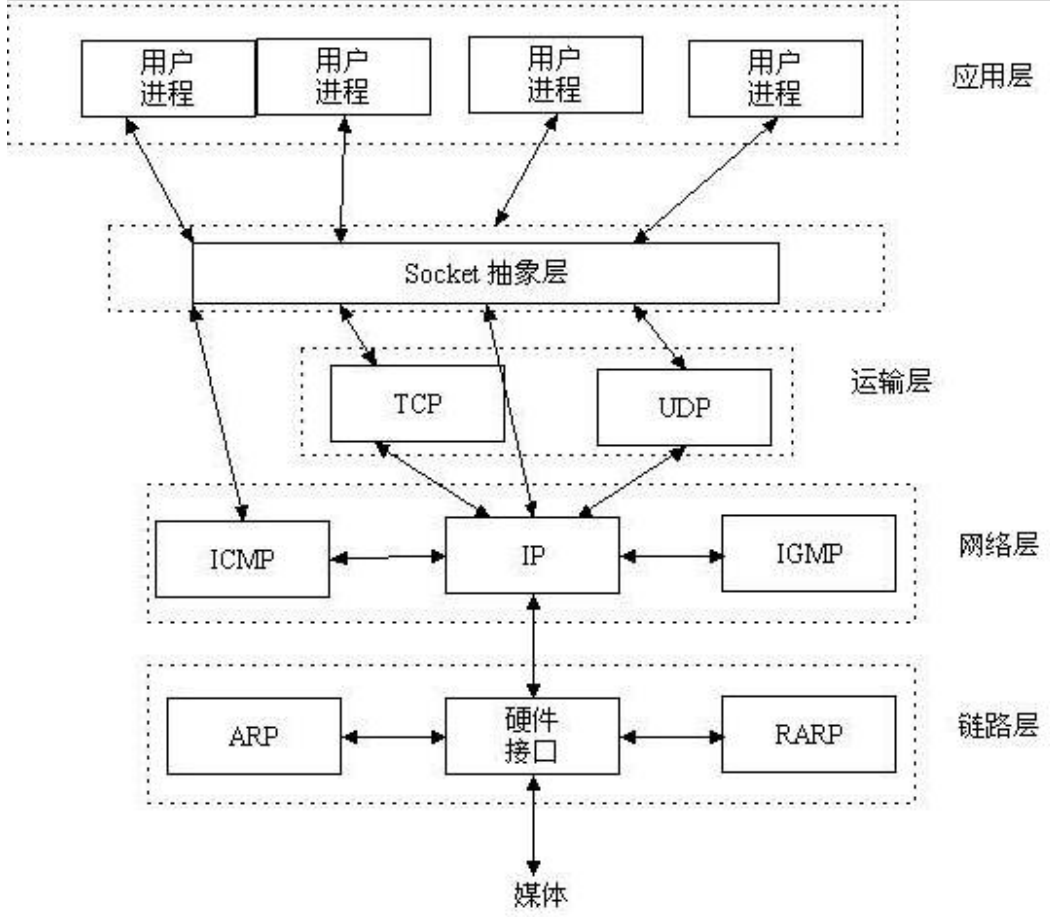


网络通讯分层协议模型


TCP/IP协议传输示意图
QQ1
发送
你好
应用层
传输层
IP层
链路层
App
你好
TCP
App
你好
IP
TCP
App
你好
帧头
IP
TCP
App
你好
帧尾
+mac地址，以太网驱动程序
网卡
路由器
...
路由器
网卡
QQ2
接收
你好
你好
App
应用层
你好
App
TCP
传输层
你好
App
TCP
IP
IP层
帧尾
你好
App
TCP
IP
帧头
链路层
https://blog.csdn.net/apollo_miracle

网络编程socket

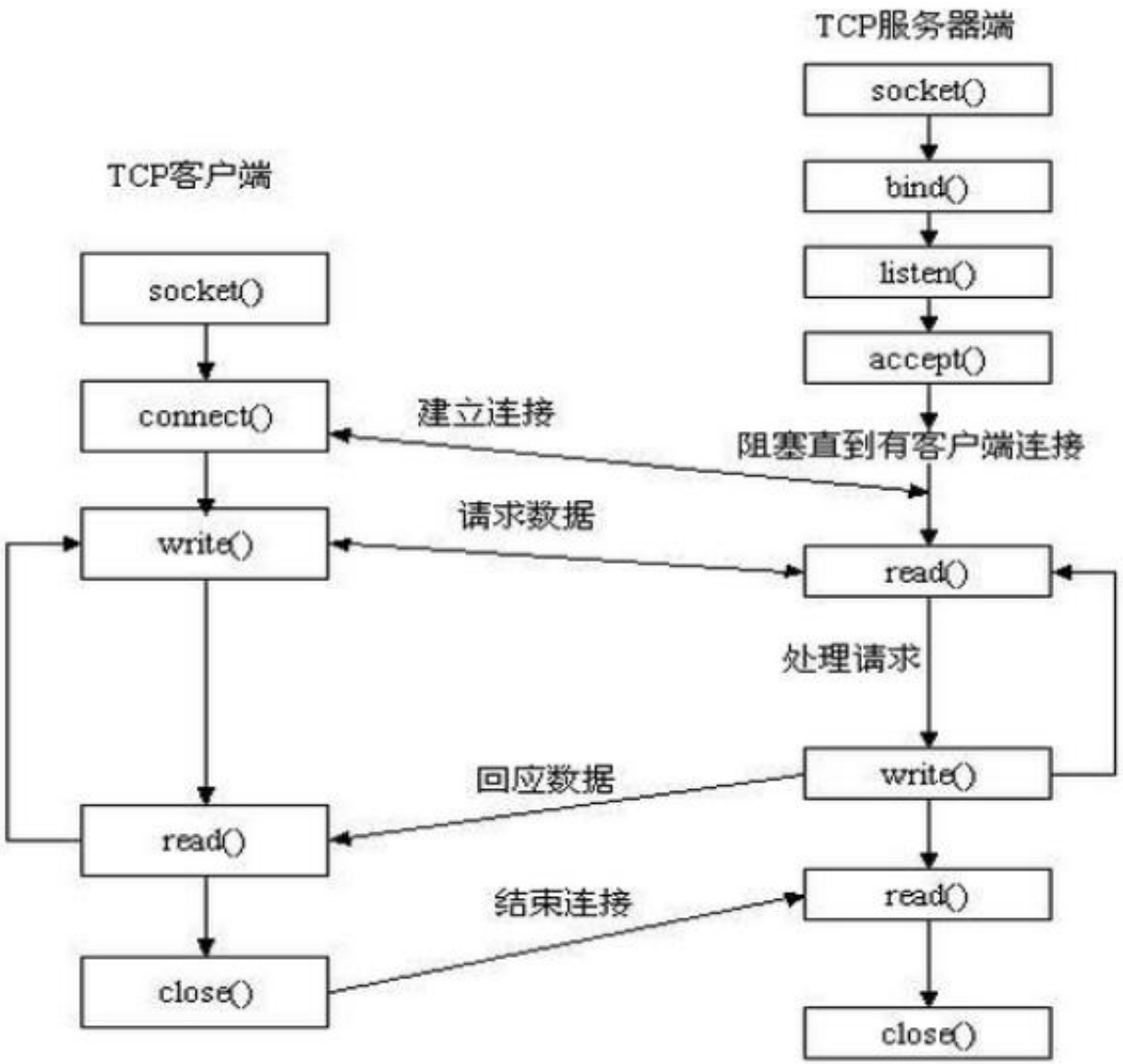
- 假设我现在要写一个程序通过TCP/IP与另外一台服务器上的服务端程序通讯，我应该怎么操作才能把数据封装成tcp/ip的包，又执行什么指令才能把数据发到对端机器上呢?
- Socket：数据封装、数据发送、数据接收、数据解包，只需要调用几行代码，就可以完成通信功能。



Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket其实就是一个门面模式，它把复杂的TCP/IP协议族隐藏在Socket接口后面，对用户来说，一组简单的接口就是全部，让Socket去组织数据，以符合指定的协议。

Socket最初是加利福尼亚大学Berkeley分校为Unix系统开发的本机进程通信接口。随着互联网发展，Socket成为在Internet上进行应用开发最为通用的API，后来又扩展到Windows下的网络编程接口。

网络编程socket-tcp过程



本质上socket只干2件事，一是收数据，一是发数据，没数据时就等着。类比于打电话为例：

接电话方(服务器端)：

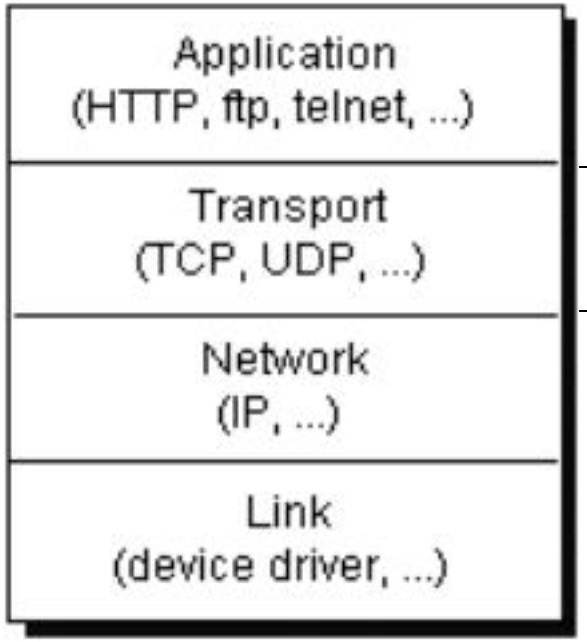
- 首先你得有个电话\(生成socket对象\)
- 你的电话要有号码\(绑定本机ip+port\)
- 开始在家等电话\(开始监听电话listen\)
- 电话铃响了，接起电话，听到对方的声音\(accept\)
- 等待→响应→等待(read/send/read)。。。。

打电话方(客户端)：

- 首先你得有个电话\(生成socket对象\)
- 输入你想拨打的电话\(connect 远程主机ip+port\)
- 等待对方接听
- say “您好，约饭吗？~”\(send\(\) 发消息。。。\)
- 等待→响应→等待(read/send/read)
- 挂电话(close)

网络编程的基本概念

- Java网络编程



java.net
Java编程在Application层

可以选择TCP和UDP两中
传输协议

这两层Java就管不着了

网络编程的基本概念

- TCP协议

TCP is a connection-based protocol that provides a reliable flow of data between two computers.
TCP是Tranfer Control Protocol的简称，是一种面向连接的保证可靠传输的协议。

例：HTTP，FTP，Telnet

- UDP协议

UDP is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP.
UDP是User Datagram Protocol的简称，是一种无连接的协议，每个数据报都是一个独立的信息，包括完整的源地址或目的地址，它在网络上以任何可能的路径传往目的地，因此能否到达目的地，到达目的地的时间以及内容的正确性都是不能被保证的。

例：Ping

两种协议各有利弊，总的说来TCP/IP更可靠，UDP协议效率更高

网络编程- TCP vs UDP

tcp基于链接通信

- 基于链接，则需要listen（backlog），指定连接池的大小
- 基于链接，必须先运行的服务端，然后客户端发起链接请求
- 如果一端断开了链接，那另外一端的链接也跟着完蛋，recv将不会阻塞，收到的是空(解决方法是：服务端通信循环内加异常处理，捕捉到异常后就break掉通讯循环)

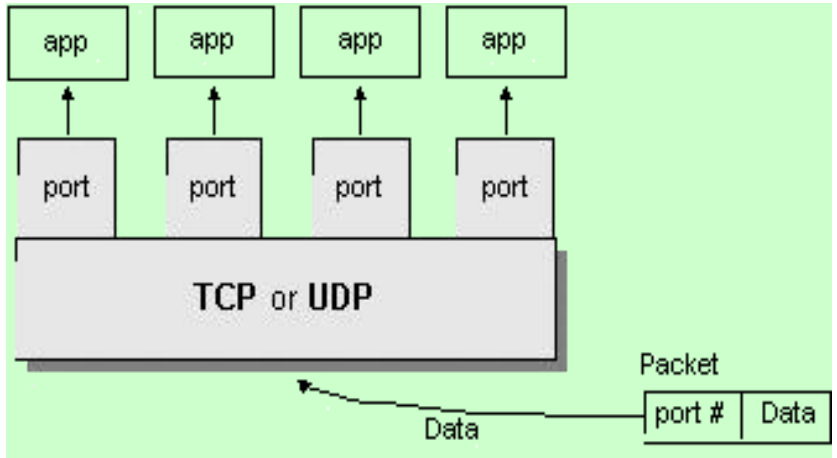
udp无链接

- 无链接，因而无需listen（backlog），更加没有什么连接池之说了
- 无链接，不用管是否有一个正在运行的服务端，客户端可一个劲的发消息，只不过数据丢失
- recvfrom收的数据小于sendinto发送的数据时，在mac和linux系统上数据直接丢失，在windows系统上发送的比接收的大直接报错
- 只有sendinto发送数据，没有recvfrom收数据，数据丢失

网络编程的基本概念

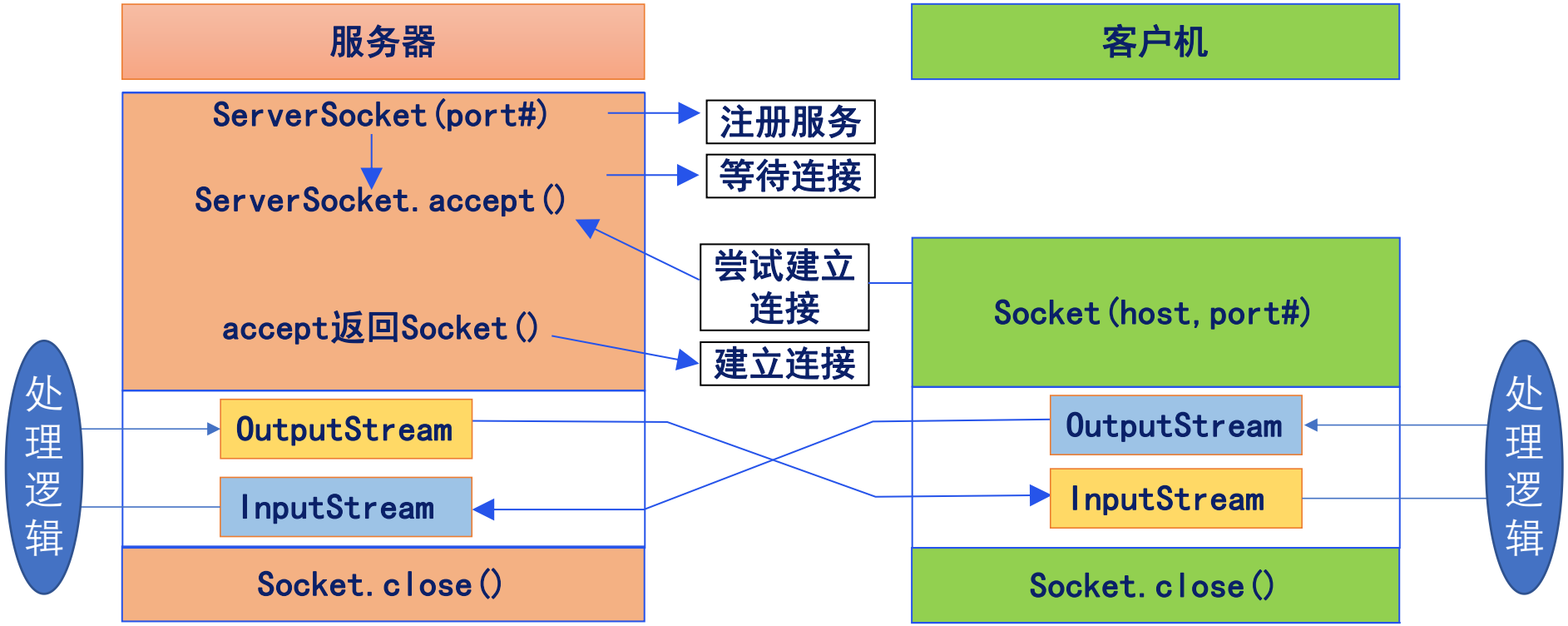
- IP地址
  - 网络中的硬件资源标识
  - IP v4　　例 166.111.60.1
  - IP v6　　例 2402:f000:3:9801:a435:4b12:7cf0:d4a4
- 端口号（port）
  - 标记机器的逻辑通信信道的正整数，不是物理实体
  - 一个16位的整数表达，其范围为0～65535，其中0～1023为系统所保留
- Socket（套接字）
  - 网络上的两个程序通过一个双向的通讯连接实现数据的交换，这个双向链路的一端称为一个Socket
  - 一个Socket由一个IP地址和一个端口号唯一确定

- 一个IP与一台电脑对应
- 一个端口和一个应用对应



基于Socket的网络编程

- 建立连接的过程
  - 服务器端生成一个ServerSocket实例对象，随时监听客户端的连接请求
  - 客户端生成一个Socket实例对象，并发出连接请求
  - 服务器端通过accept()接收到客户端的请求后，开辟一个接口与之进行连接，并生成所需的I/O数据流。
  - 通信都是通过一对InputStream和OutputStream进行的。通信结束后，两端分别关闭对应的Socket接口。



基于Socket的网络编程

在Java编程中，基于TCP协议实现网络通信的类有两个：在客户端的Socket类和在服务器端的ServerSocket类。SeverSocket类的功能是建立一个Sever，并通过accept()方法随时监听客户端的连接请求。

ServerSocket server = new ServerSocket(80); //服务器端在80端口监听，注意端口必须没被其他程序占用

Socket client = new Socket("127.0.0.1.", 80); //客户端连接本机(127.0.0.1)的80端口

由于网络存在不可靠的情况，创建Socket必须有try catch来处理异常。

```
try{
        Socket socket=new Socket("127.0.0.1",80);
        //127.0.0.1是TCP/IP协议中默认的本机地址
}catch(IOException e){
        System.out.println("Error:"+e);
}
```

课堂练习（3）

服务端

```
ServerSocket server = new ServerSocket(1000);
while (true) {
    System.out.println("waiting....");
    Socket sock = server.accept();
    try
    {
        System.out.println("已经链接。。。");
        BufferedReader br =
                new BufferedReader(new InputStreamReader(sock.getInputStream()));
        BufferedWriter bw =
                new BufferedWriter(new OutputStreamWriter(sock.getOutputStream()));
        while (true) {
            String info = br.readLine();
            System.out.println("recv: " + info);
            if (info == "byte") {
                break;
            }
            info = info.toUpperCase();
            System.out.println("send: " + info);
            bw.write(info + "\n");
            bw.flush();
        }
        br.close();
        bw.close();
        sock.close();
        System.out.println("关闭链接。。。");
    }
    catch(Exception e) {

    }
}
```

建立socket连接

发送数据

返回数据

客户端

```
Scanner scan = new Scanner(System.in);
Socket sock = new Socket("127.0.0.1", 1000);
BufferedReader br =
        new BufferedReader(new InputStreamReader(sock.getInputStream()));
BufferedWriter bw =
        new BufferedWriter(new OutputStreamWriter(sock.getOutputStream()));
while (true) {
    String str = scan.nextLine();
    if (str.isEmpty()) {
        continue;
    }
    System.out.println("send: " + str);
    bw.write(str + "\n");
    bw.flush();
    if (str.equals("bye")) {
        break;
    }
    str = br.readLine();
    System.out.println("recv: " + str);
}
br.close();
sock.close();
```

多客户端如何处理?

清華大學 Tsinghua University

Java 多线程编程

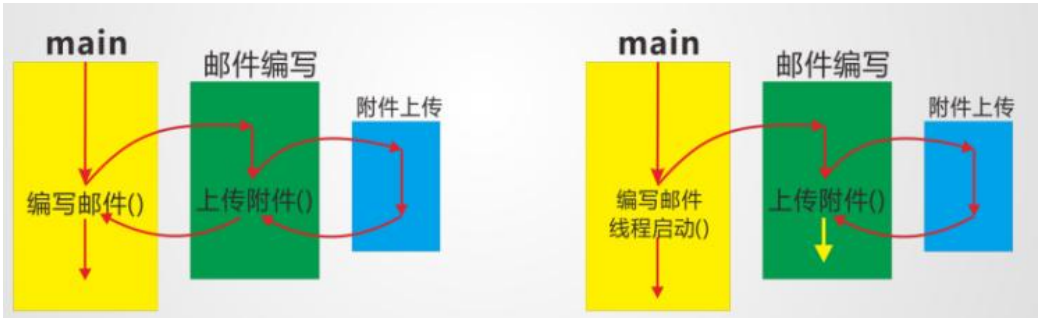
进程与线程

◆ 进程Process

- 一个进程就是一个执行中的程序 （重量级）
- 每一个进程都有自己独立的一块内存空间、一组系统资源
- 不同进程的内部数据和状态都是完全独立

◆ 线程Thread

- 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），能访问共享内存变量，线程切换的开销小。
- 一个进程可以有多个线程。
- 多核服务器，多个线程同时运行。单核CPU，每个时间片只有一个线程在运行，线程快速切换，有同时执行的错觉。



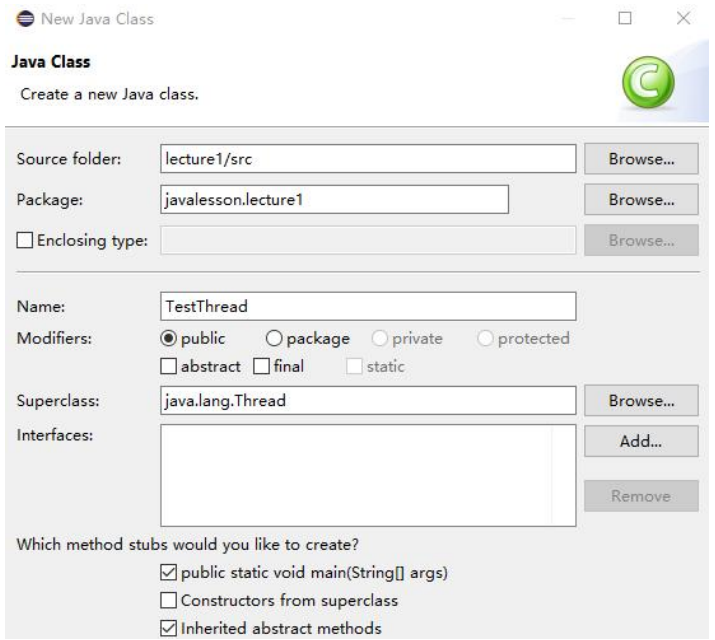
Java是第一个支持内置线程操作的主流编程语言

创建线程的方式1

◆ 继承Thread类，重载run（）函数

```
class MyThread extends Thread{  // 继承Thread类，作为线程的实现类
    private String name ;       // 表示线程的名称
    public MyThread(String name){
        this.name = name ;      // 通过构造方法配置name属性
    }
    public void run(){  // 覆写run()方法，作为线程 的操作主体
        for(int i=0;i<10;i++){
            System.out.println(name + "运行，i = " + i) ;
        }
    }
};

public class TestThread{
    public static void main(String args[]){
        MyThread mt1 = new MyThread("线程A ") ; // 实例化对象
        MyThread mt2 = new MyThread("线程B ") ; // 实例化对象
        mt1.start() ;   // 调用线程主体
        mt2.start() ;   // 调用线程主体
    }
};
```



创建线程的方式2

◆ 实现 Runnable 接口

```
class MyThread2 implements Runnable{ // 实现Runnable接口，作为线程的实现类
    private String name ;        // 表示线程的名称
    public MyThread2(String name){
        this.name = name ;
    }
    public void run(){  // 覆写run()方法，作为线程的操作主体
        for(int i=0;i<10;i++){
            System.out.println(name + "运行，i = " + i) ;
        }
    }
};
public class TestThread2 {
  public static void main(String [] args)
  {
      MyThread2 mt1 = new MyThread2("线程A ") ;     // 实例化对象
      MyThread2 mt2 = new MyThread2("线程B ") ;     // 实例化对象
      new Thread(mt1).start();
      new Thread(mt2).start();
  }
}
```

创建线程的方式3

◆ 匿名内部类

```
public class Demo {
    public static void main(String[] args) {
        // 继承Thread类实现多线程
        new Thread() {
            public void run() {
                for (int i = 0; i < 100; i++) {
                    System.out.println(Thread.currentThread().getName() + "--" + i);
                }
            }
        }.start();
        // 实现Runnable接口实现多线程
        new Thread(new Runnable() {
            public void run() {
                for (int i = 0; i < 100; i++) {
                    System.out.println(Thread.currentThread().getName() + "--" + i);
                }
            }
        }) {
        }.start();
    }
}
```

传递变量方便，简化代码

课堂练习（4）

◆ 使用多线程方式，查找1000以内的水仙花数。

举个例子 **1^3 + 5^3+ 3^3 = 153**

注意线程之间的负载均衡

提高要求：

1）从命令行接受参数 （同时启动的线程数）

2）找到并保存100000000以内所有的自幂数， 并排序输出

例子：**1^3 + 5^3+ 3^3 = 153**

1^4+6^4+3^4+4^4=1634

5^6+4^6+8^6+8^6+3^6+4^6= 548834

3）计算多线程的加速比和效率， 加速比= 单线程计算时间/ 多线程计算时间 效率=加速比/线程数

4）多次试验，确定最佳的并发线程数

提示： System.currentTimeMillis() 可以返回当前时间， 两次做差就是计算时间， 单位是ms

课堂练习（5）

- 重写socket 通信， 支持多客户端访问。