1. TCP与UDP的区别

TCP（Transmission Control Protocol，传输控制协议）是基于连接的协议，也就是说，在正式收发数据前，必须和对方建立可靠的连接。一个TCP连接必须要经过三次“对话”才能建立起来，其中的过程非常复杂，我们这里只做简单、形象的介绍，你只要做到能够理解这个过程即可。我们来看看这三次对话的简单过程：主机A向主机B发出连接请求数据包：“我想给你发数据，可以吗？”，这是第一次对话；主机B向主机A发送同意连接和要求同步（同步就是两台主机一个在发送，一个在接收，协调工作）的数据包：“可以，你什么时候发？”，这是第二次对话；主机A再发出一个数据包确认主机B的要求同步：“我现在就发，你接着吧！”，这是第三次对话。三次“对话”的目的是使数据包的发送和接收同步，经过三次“对话”之后，主机A才向主机B正式发送数据。   
  
UDP（User Data Protocol，用户数据报协议）是与TCP相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去！

小结TCP与UDP的区别：  
1.基于连接与无连接；  
2.对系统资源的要求（TCP较多，UDP少）；  
3.UDP程序结构较简单；  
4.流模式与数据报模式 ；  
5.TCP保证数据正确性，UDP可能丢包，TCP保证数据顺序，UDP不保证。

TCP连接三次握手、断开四次挥手的过程：三次握手：客户端发起请求（序列号）-服务器回应ack并给出自己的序列号-客户端回应ack。 断开四次挥手：一端发起断开（FIN），另外一端回应ack；这样一端的写就关闭了；另外一端发起断开请求（FIN），本端回应ack。这样双方都关闭了连接。因为一端关闭了写，但是另外一端可能还有数据要发送，所以两端可能不同时关掉连接，就出现了4次挥手的过程。

 ping命令是用来查看网络上另一个主机系统的网络连接是否正常的一个工具。ping命令的工作原理是：向网络上的另一个主机系统发送ICMP报文，如果指定系统得到了报文，它将把报文一模一样地传回给发送者，这有点象潜水艇声纳系统中使用的发声装置

进程和线程的区别：进程是分配资源的基本单位（CPU、内存等），有独立的地址空间。线程是进程中的一个实体，共享进程的地址空间，是轻量级的进程，是CPU调度和分配的基本单位，但是各个线程拥有自己的栈空间。 使用多线程编程的好处：（1）单线程遇到阻塞，会卡死，影响交互；（2）发挥多核CPU的计算能力；（3）简化程序结构，使程序便于维护；（4）与进程相比，线程的创建和切换开销更小

nginx的负载均衡策略：轮询、最少连接优先、ip地址哈希（同一客户端的web请求会到同一个服务器上，可以解决session的问题–一般解决是通过数据库持久化session，然后用分布式的数据库）、基于权重的负载均衡。

Java语言中，当创建一个对象后，Java虚拟机就会为其分配一个指向对象本身的指针，这个指针就是“this”。关键字this与对象密切相 关。

1. 使用this调用本类中的成员变量（属性）
2. 使用this调用构造方法，调用时要放在构造方法的首行。
3. this调用本类中的其他方法；

当在子类的构造函数需要调用父类的构造函数时，实用super（），调用，当然super（）,里面可以加形参（根据具体的父类的构造函数是否需要形参），这条语句必须写在子类构造函数的第一句

super不是引用类型，super中存储不是内存地址

\* super指向的不是父类对象

\* super代表当前子类对象中父类型特征

\* 什么时候使用

\* 1.子类父类都有某个数据 例如：子类父类都有name属性

\* 在子类中访问父类中的name属性

\* super用在什么地方

\* 1.用在成员方法

\* 2.用在构造方法

3.this super都不能用在静态方法中

super构造 语法 super(实参) 作用： 通过子类调用父类 的构造方法

\* 无参数构造方法 super()

\* 语规：构造方法第一行如果没有this(..),也没显示去调用

\* super(..)和this()只能放在构造方法的第一行

\* 所以super()和this()不能共存

\* super(..)调用父类的构造方法，但并不会创建父类型对象

\* 在java语言中，只要创建对象都会调用Object的无参数构造方法

/\*final修饰的类无法被继承

\* final修饰的方法无法被覆盖，但可以被继承

\* final修饰局部变量一旦赋值就无法重新赋值

final不能用于修饰构造方法。

\* final修饰成员变量，必须“显示的”初始化或者可以通过构造方法

\* final修饰成员变量一般和static连用

\* java规范要求所有常量“大写”

static表示“全局”或者“静态”的意思，用来修饰成员变量和成员方法，也可以形成静态static代码块，但是Java语言中没有全局变量的概念。

static修饰的成员变量和成员方法习惯上称为静态变量和静态方法，可以直接通过类名来访问，访问语法为：  
类名.静态方法名(参数列表...)   
类名.静态变量名

        用static修饰的代码块表示静态代码块，当Java虚拟机（JVM）加载类时，就会执行该代码块（用处非常大，呵呵）。

static final用来修饰成员变量和成员方法，可简单理解为“全局常量”！  
        对于变量，表示一旦给值就不可修改，并且通过类名可以访问。  
        对于方法，表示不可覆盖，并且可以通过类名直接访问。

**装箱**：把基本类型用它们相应的引用类型包装起来，使其具有对象的性质。int包装成Integer、float包装成Float

**拆箱**：和装箱相反，将引用类型的对象简化成值类型的数据

Integer a = 100;                  这是自动装箱  (编译器调用的是static Integer valueOf(int i))  
int     b = new Integer(100); 这是自动拆箱

//1.自动装箱和自动拆箱是程序编译阶段的一个概念，和程序运行无关

//2.自动装箱和自动拆箱主要目的是方便程序员编码

关于Object中的toString方法

\* SUN在Object类中设计toString的目的：返回java对象的字符串表示

\* 在现实中，Object中toString方法已经不够用了，所以Object的toString的方法实现不满意

\* Object中toString方法要被重写

\* SUN是这样实现toString

\* public String toString(){

\* return getClass().getName()+"@"+Integer.toHexString(hashCode());

\* Object toString方法返回 ：类名 @ java对象的内存地址经过哈希算法得出的int类型

\* 值转换成十六进制，结果可以等同看作java对象在堆中内存地址

\*/

java中八种基本数据类型对应的包装类型

\* 基本数据类型 包装类型

\*

\* byte java.lang.Byte;

\* short java.lang.Short

\* int java.lang.Integer

\* long java.lang.Long

\*

\* float java.lang.Float;

\* double java.lang.Double;

\*

\* boolean java.lang.Boolean;

\*

\* char java.lang.Character;

String 字符串常量

1.StringBuffer和StringBuilder是什么

\* 是一个字符串缓冲区

\* 2.工作原理

\* 预先在内存申请一块空间，以容纳字符序列

\* 如果预留的空间不够用，则进行自动扩容，以容纳更多的字符序列

3.StringBuffer,StringBuilder和String 最大区别

String是不可变的字符序列，存储字符串常量池中

StringBuffer底层是一个char数组，但是该char数组是可变的

4.StringBuffer和StringBuider的默认初始化都是16

5.如何优化优化StringBuffer和StringBuider呢

最后试在创建StringBuffer之前，就预测StringBuffer的存储字符数量

然后再创建StringBuffer的时候采用指定初始化容量的方法创建StringBuffer

为减少底层数组的拷贝，提高效率

6.StringBuffer和StringBuider的区别？

StringBuffer是线程安全的（可以在多线程的环境下使用不会出现问题）

StringBuider是非线性安全的，（在多线程环境下使用可能出现问题）

类型的主要性能区别其实在于 String 是不可变的对象, 因此在每次对 String 类型进行改变的时候其实都等同于生成了一个新的 String 对象，然后将指针指向新的 String 对象，所以经常改变内容的字符串最好不要用 String ，

**对于三者使用的总结**： 1.如果要操作少量的数据用 = String

　　　　　　　　　　              2.单线程操作字符串缓冲区 下操作大量数据 = StringBuilder

　　　　　　　　　　              3.多线程操作字符串缓冲区 下操作大量数据 = StringBuffer

（1） 方法重载是让类以统一的方式处理不同类型数据的一种手段。多个同名函数同时存在，具有不同的参数个数/类型。

重载Overloading是一个类中多态性的一种表现。  
    （2） [Java](http://java.chinaitlab.com/" \t "http://www.cnblogs.com/bluestorm/archive/2012/03/01/_blank)的方法重载，就是在类中可以创建多个方法，它们具有相同的名字，但具有不同的参数和不同的定义。

调用方法时通过传递给它们的不同参数个数和参数类型来决定具体使用哪个方法, 这就是多态性。  
    （3） 重载的时候，方法名要一样，但是参数类型和个数不一样，返回值类型可以相同也可以不相同。无法以返回型别作为重载函数的区分标准。

重写（Overriding）  
    （1） 父类与子类之间的多态性，对父类的函数进行重新定义。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。在[Java](http://java.chinaitlab.com/" \t "http://www.cnblogs.com/bluestorm/archive/2012/03/01/_blank)中，子类可继承父类中的方法，而不需要重新编写相同的方法。

但有时子类并不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写。

方法重写又称方法覆盖。  
    （2）若子类中的方法与父类中的某一方法具有相同的方法名、返回类型和参数表，则新方法将覆盖原有的方法。

如需父类中原有的方法，可使用super关键字，该关键字引用了当前类的父类。  
    （3）子类函数的访问修饰权限不能少于父类的；

面向对象编程有三大特性：封装、继承、多态。

封装隐藏了类的内部实现机制，可以在不影响使用的情况下改变类的内部结构，同时也保护了数据。对外界而已它的内部细节是隐藏的，暴露给外界的只是它的访问方法。

继承是为了重用父类代码。两个类若存在IS-A的关系就可以使用继承。，同时继承也为实现多态做了铺垫。那么什么是多态呢？多态的实现机制又是什么？请看我一一为你揭开：

所谓多态就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量倒底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。因为在程序运行时才确定具体的类，这样，不用修改源程序代码，就可以让引用变量绑定到各种不同的类实现上，从而导致该引用调用的具体方法随之改变，即不修改程序代码就可以改变程序运行时所绑定的具体代码，让程序可以选择多个运行状态，这就是多态性。

多态存在的三个必要条件  
一、要有继承；  
二、要有重写；  
三、父类引用指向子类对象。

Java中多态的实现方式：接口实现，继承父类进行方法重写，同一个类中进行方法重载。

向上转型：子类引用的对象转换为父类类型称为向上转型。通俗地说就是是将子类对象转为父类对象。此处父类对象可以是接口

向下转型：父类引用的对象转换为子类类型称为向下转型

向上转型时，父类指向子类引用对象会遗失除与父类对象共有的其他方法，也就是在转型过程中，子类的新有的方法都会遗失掉，在编译时，系统会提供找不到方法的错误

Java1.5增加了新特性：**可变参数：适用于参数个数不确定，类型确定的情况，java把可变参数当做数组处理。**注意：可变参数必须位于最后一项。

可变参数的特点：

（1）、只能出现在参数列表的最后；

（2）、...位于变量类型和变量名之间，前后有无空格都可以；

（3）、调用可变参数的方法时，编译器为该可变参数隐含创建一个数组，在方法体中一数组的形式访问可变参数。

public class Varable {  
 public static void main(String [] args){  
  System.out.println(add(2,3));  
  System.out.println(add(2,3,5));  
 }  
 public static int add(int x,int ...args){  
  int sum=x;  
  for(int i=0;i<args.length;i++){  
   sum+=args[i];  
  }  
  return sum;  
 }  
}

//获取系统当前时间

Date nowTime=new Date();

System.out.println(nowTime);//Thu Nov 19 17:13:11 CST 2015

首先讲一下进程和线程的区别：

　　进程：每个进程都有独立的代码和数据空间（进程上下文），进程间的切换会有较大的开销，一个进程包含1--n个线程。

　　线程：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)，线程切换开销小。

　　线程和进程一样分为五个阶段：创建、就绪、运行、阻塞、终止。

　　多进程是指[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/gf771115/article/details/_blank)能同时运行多个任务（程序）。

　　多线程是指在同一程序中有多个顺序流在执行。

在java中要想实现多线程，有两种手段，一种是继续Thread类，另外一种是实现Runable接口。

注意：start()方法的调用后并不是立即执行多线程代码，而是使得该线程变为可运行态（Runnable），什么时候运行是由操作系统决定的。

从程序运行的结果可以发现，多线程程序是乱序执行。因此，只有乱序执行的代码才有必要设计为多线程。

Thread.sleep()方法调用目的是不让当前线程独自霸占该进程所获取的CPU资源，以留出一定时间给其他线程执行的机会。

实际上所有的多线程代码执行顺序都是不确定的，每次执行的结果都是随机的。

Thread.sleep(毫秒);

\* sleep方法是静态方法

\* 该方法的作用：阻塞当前线程，腾出CPU,让给其他线程。

\* Thread.yield();

\* 1,该方法是一个静态方法，

\* 2，作用：给同一个优先级的线程让位，但是让位时间不固定。

\* 3，和sleep方法相同，就是yield时间不固定。

\*

\* Thread.join();

\* 该方法是等待该线程终止。

\* 合并线程，

实现Runnable接口比继承Thread类所具有的优势：

1）：适合多个相同的程序代码的线程去处理同一个资源

2）：可以避免java中的单继承的限制

3）：增加程序的健壮性，代码可以被多个线程共享，代码和数据独立

2. 什么条件下使用线程同步？

\* 第一：必须是多线程环境

\* 第二：多线程环境共享同一个数据

\* 第三：共享的数据涉及到修改操作

abstract 如何定义抽象类

\* 1.class 前加abstract

\* 2.抽象类无法实例化 无法创建对象

\* 3.虽然无法实例化 有构造方法是给子类创建对象用的

\*4. 抽象类可以定义抽象方法

\* 语法：在方法修饰符列表中加abstract,抽象方法以“；”结束 不能带有“{}”

\* 例如：public abstract void m1();

\* 5.抽象类中不一定有抽象方法 ，但抽象方法必须在抽象类中

\* 6.一个非抽象类继承抽象类必须将抽象类方法覆盖，实现，重写

单例模式类型：

\* 1 饿汉式：在类加载阶段就创建对象了

\* 2懒汉式：用到对象才会创建

单例模式有以下特点：  
　　1、单例类只能有一个实例。  
　　2、单例类必须自己创建自己的唯一实例。  
　　3、单例类必须给所有其他对象提供这一实例。

单例模式确保某个类只有一个实例，而且自行实例化并向整个系统提供这个实例。

接口的作用：

\* 1.使项目分层，所有层都面向接口开发 ，开发效率提高

\* 2.接口使代码和代码之间的耦合度降低，就像内存条和主板关系 可插拔 随意切换

\* 3.接口和抽象类都可用的情况，优先选择接口，因为接口可以多实现，多继承，

\* 一个类除了可以实现接口，还可以继承（保留了类继承）

\*

\* List集合存储元素特点

\* 1.有序（List集合中存储有下标）：存进去是这样的顺序，取出来还是这样的顺序。

\* 2.可重复

\*

\* 深入List集合

\* ArrayList集合底层是数组，数组是有下标的。

\* 所以ArrayList集合有很多自己的特性

\*

\* ArrayList集合底层默认初始值容量是10，扩大后的容量是原容量的1.5倍

\* Vector集合底层默认初始化容量是10，扩大后的容量是原容量的2倍

\*

\* 如何优化ArrayList和Vector？

\* 尽量减少扩容操作，因为扩容需要数组拷贝，数组拷贝很耗内存

\* 一般推荐在创建集合的时候指定初始化容量

\*/

final用于声明属性，方法和类，分别表示属性不可交变，方法不可覆盖，类不可继承。

finally是异常处理语句结构的一部分，表示总是执行。

finalize是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，供垃圾收集时的其他资源回收，例如关闭文件等。

接口可以继承接口。抽象类可以实现(implements)接口，抽象类可以继承具体类。抽象类中可以有静态的main方法。

局部变量必须初始化

1.异常是什么？

\* 第一，异常模拟的是现实世界中“不正常”事件

\* 第二，java中采用“类”去模拟异常

\* 第三，类是可以创建对象的

\* 例如： NullPointerExceptin c= 0x1234;

\* c是引用类型，c中保存的内存地址指向堆中的对象“对象”

\* 这个对象一定是NullPointerExceptin类型

\* 这个对象就表示真实存在的异常事件

\* NullPointerExceptin是一类异常

\*

\* ”抢劫“就是一类异常

\* ”张三被抢劫“就是一个异常事件

\* 2.异常机制的作用

\* java语言为我们提供一个完善的异常处理机制：

\* 作用是：程序发生异常事件后，为我们输出详细的信息。

\* 程序通过这个信息，可以对程序进行一些处理，使程序更加健壮

\*/

\* 处理异常的第二种方式：捕捉 try...catch...

\* 语法：

\* try{

\* 可能出现的异常;

\* }catch(异常类型1 变量){

\* 处理异常的代码;

\* }catch(异常类型2 变量){

\* 处理异常的代码;

\* }...

\*

\* 1.catch语句块可以写多个

\* 2.但是从上到下catch必须从小类型异常到大类型异常进行捕捉

\* 3.try...catch..中最多执行1个catch语句，执行结束之后try...catch..就结束了

\*

\* 关于getMessage和ptintStackTrace方法应用

\*

会在return中间执行！

try 中的 return 语句调用的函数先于 finally 中调用的函数执行，也就是说 return 语句先执行，finally 语句后执行，但 return 并不是让函数马上返回，而是 return 语句执行后，将把返回结果放置进函数栈中，此时函数并不是马上返回，它要执行 finally 语句后才真正开始返回！但此时会出现两种情况：

①、如果finally中也有return，则会直接返回并终止程序，函数栈中的return不会被完成！；

②、如果finally中没有return，则在执行完finally中的代码之后，会将函数栈中的try中的return的内容返回并终止程序；

catch同try;

InputStream 是字节输入流的所有类的超类,一般我们使用它的子类,如FileInputStream等.

OutputStream是字节输出流的所有类的超类,一般我们使用它的子类,如FileOutputStream等.

2.InputStreamReader  OutputStreamWriter

处理字符流的抽象类

InputStreamReader 是字节流通向字符流的桥梁,它将字节流转换为字符流.

OutputStreamWriter是字符流通向字节流的桥梁，它将字符流转换为字节流.

3.BufferedReader BufferedWriter

BufferedReader 由Reader类扩展而来，提供通用的缓冲方式文本读取，readLine读取一个文本行，

从字符输入流中读取文本，缓冲各个字符，从而提供字符、数组和行的高效读取。

BufferedWriter  由Writer 类扩展而来，提供通用的缓冲方式文本写入， newLine使用平台自己的行分隔符，

将文本写入字符输出流，缓冲各个字符，从而提供单个字符、数组和字符串的高效写入。

InputStream能从來源处读取一個一個byte,  
所以它是最低级的，

InputStreamReader  
 InputStreamReader封裝了InputStream在里头,  
 它以较高级的方式,一次读取一个一个字符,

BufferedReader  
BufferedReader则是比InputStreamReader更高级,  
它封裝了StreamReader类,  
一次读取取一行的字符

Java接口中声明的变量**默认都是final static的，方法默认为抽象**

在java 中，声明一个数组时，不能直接限定数组长度，只有在创建实例化对象时，才能对给定数组长度.。

如下，1，2,3可以通过编译，4，5不行。而String是Object的子类，所以上述BCF均可定义一个存放50个String类型对象的数组。

1. String a[]=new String[50];

2. String b[];

3. char c[];

4. String d[50];

5. char e[50];

重载是在同一个类中，有多个方法名相同，参数列表不同(参数个数不同，参数类型不同),与方法的返回值无关，与权限修饰符无关，参数顺序不同也算重载

1)String---------》char / char[ ]

String str = "ab";

char str1 = str.charAt(0); //得到a

char str2 = str.charAt(1); //得到b

char[ ] str3 = str.toCharArray();

System.out.println (Arrays.toString(str3));

(2)char / char[]-----------》String

char m = 'a';

String str = String.valueOf(m);

String str2 = m + " ";

char[] m = {'a','b'};

String str3 = String.copyValueOf(m);

String str4 = new String(m);

(3)int -----------》 String

int i;

String s = String.valueOf(i);

String s = Integer.toString(i);

String s = "" + i;

(4)String ----------》 int

int i = Integer.parseInt([String]);

i = Integer.parseInt([String],[int radix]);

int i = Integer.valueOf(my\_str).intValue();

**Java中的byte，short，char进行计算时都会提升为int类型。**

****初始化过程：****

****1.**** ****初始化父类中的静态成员变量和静态代码块**** ****；****

****2.**** ****初始化子类中的静态成员变量和静态代码块**** ****；****

****3.初始化父类的普通成员变量和代码块，再执行父类的构造方法；****

****4.初始化子类的普通成员变量和代码块，再执行子类的构造方法；****

****（1）初始化父类的普通成员变量和代码块，执行  Y y=new**** ****Y();**** ****输出Y****

****（2）再执行父类的构造方法；输出X****

****（3）**** ****初始化子类的普通成员变量和代码块，执行  Y y=new****  ****Y();**** ****输出Y****

****（4）再执行子类的构造方法；输出Z****

****所以输出YXYZ****

****jar****        将许多文件组合成一个jar文件

****javac****    编译

****javadoc**** 它从程序源代码中抽取类、方法、成员等注释形成一个和源代码配套的API帮助文档。

****javah**** 把java代码声明的JNI方法转化成C\C++头文件。 JNI可参考java核心技术卷二第12章

**栈：存放局部变量**

**堆：存放所有new出来的东西，数组，对象。jvm只有一个堆区(heap)被所有线程共享，堆中不存放基本类型和对象引用，只存放对象本身**

方法区:  
1.又叫静态区，跟堆一样，被所有的线程共享。方法区包含所有的class和static变量。  
2.方法区中包含的都是在整个程序中永远唯一的元素，如class，static变量。

静态属性和静态方法可以被继承，但是没有被重写(overwrite)而是被隐藏。

如果类没有构造方法，JVM会生成一个默认构造方法，如果定义了任意类型的构造方法，编译器都不会自动生成构造方法。

在根类Object中包含一下方法：

1. clone();
2. equals();
3. finalize();
4. getClass();[align=left][/align]
5. notify(),notifyAll();
6. hashCode();
7. toString();
8. wait();

类:1.java.lang.Object  
2.java.lang.String  
3.java.lang.System  
4.java.io.file  
5.java.io.FileInputStream  
包:1.java.lang包  
2.java.io包  
3.java.swt包  
4.java.util包  
5.java.sql包  
接口:1.java.util.List<E>  
2.java.util.Map<E>  
3.java.util.Iterator<E>  
4.java.sql.CallableStatement  
5.java.lang.Comparable<T>

.**[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "http://blog.csdn.net/wangloveall/article/details/7992448/_blank)**序列化与反序列化

**Java序列化**是指把Java对象转换为字节序列的过程；而**Java反序列化**是指把字节序列恢复为Java对象的过程。

 2.为什么需要序列化与反序列化

 我们知道，当两个进程进行远程通信时，可以相互发送各种类型的数据，包括文本、图片、音频、视频等， 而这些数据都会以二进制序列的形式在网络上传送。那么当两个Java进程进行通信时，能否实现进程间的对象传送呢？答案是可以的。如何做到呢？这就需要Java序列化与反序列化了。换句话说，一方面，发送方需要把这个Java对象转换为字节序列，然后在网络上传送；另一方面，接收方需要从字节序列中恢复出Java对象。

 当我们明晰了为什么需要Java序列化和反序列化后，我们很自然地会想Java序列化的好处。其好处一是实现了数据的持久化，通过序列化可以把数据永久地保存到硬盘上（通常存放在文件里），二是，利用序列化实现远程通信，即在网络上传送对象的字节序列。

字节流适合读取视频，声音，图片等二进制文件。字符流适合读取纯文本文件.

22

33