Java基础篇

Java 语言的优点

面向对象,平台无关,内存管理,安全性,多线程,Java

1.基本数据类型

- byte/8
- char/16
- short/16
- int/32
- float/32
- long/64
- double/64
- boolean/~

2.Java 和 C++的区别

- 多重继承(java接口多重,类不支持,C++支持)
- 自动内存管理
- 预处理功能
- goto语句(java不支持)
- 引用与指针。在Java中不可能直接操作对象本身，所有的对象都由一个引用指向，必须通过这个引用才能访问对象本身，包括获取成员变量的值，改变对象的成员变量，调用对象的方法等。而在C++中存在引用，对象和指针三个东西，这三个东西都可以访问对象。其实，Java中的引用和C++中的指针在概念上是相似的，他们都是存放的对象在内存中的地址值，只是在Java中，引用丧失了部分灵活性，比如Java中的引用不能像C++中的指针那样进行加减运算。

3.值传递和引用传递

变量被值传递，意味着传递了变量的一个副本。因此，就算是改变了变量副本，也不会影响源对象的值。
对象被引用传递，意味着传递的并不是实际的对象，而是对象的引用。因此，外部对引用对象所做的改

变会反映到所有的对象上。 java本质上还是值传递，如方法调用的时候传入一个对象引用进去，在方法栈中会构建一个副本和该引用变量值相同指向同一个地址。如果改变引用的值不会对改变传入的引用的值。

4.静态变量和实例变量的区别

在语法定义上的区别：静态变量前要加 static 关键字，而实例变量前则不加。
在程序运行时的区别：实例变量属于某个对象的属性，必须创建了实例对象，其中的实例变量才会被分配空间，才能使用这个实例变量。静态变量不属于某个实例对象，而是属于类，所以也称为类变量，只要程序加载了类的字节码，不用创建任何实例对象，静态变量就会被分配空间，静态变量就可以被使用了。总之，实例变量必须创建对象后才可以通过这个对象来使用，静态变量则可以直接使用类名来引用。

5.JDK 包

JDK 常用的 package:
java.lang：这个是系统的基础类，比如 String 等都是这里面的，这个 package 是唯一一个可以不用 import 就可以使用的 Package
java.io: 这里面是所有输入输出有关的类，比如文件操作等
java.net: 这里面是与网络有关的类，比如 URL,URLConnection 等。
java.util: 这个是系统辅助类，特别是集合类 Collection,List,Map 等。
java.sql: 这个是数据库操作的类，Connection, Statememt，ResultSet 等

6.JDK, JRE 和 JVM 的区别

JDK,JRE和JVM 是 Java 编程语言的核心概念。尽管它们看起来差不多，作为程序员我们也不怎么关心这些概念，但是它们是不同的针对特定目的的产品。这是一道常见的 Java 面试题，而本文则会一一解释这些概念并给出它们之间的区别。

> 1）Java 开发工具包 (JDK)
> Java 开发工具包是 Java 环境的核心组件，并提供编译、调试和运行一个 Java 程序所需的所有工具，可执行文件和二进制文件。包括java基础jar包、虚拟机、javac等可执行文件等。JDK 是一个平台特定的软件，有针对 Windows，Mac 和 Unix 系统的不同的安装包。可以说 JDK 是 JRE 的超集，它包含了 JRE 的 Java 编译器，调试器和核心类。

> 2）Java 虚拟机(JVM)
> JVM 是 Java 编程语言的核心。当我们运行一个程序时，JVM 负责将字节码转换为特定机器代码。JVM 也是平台特定的，并提供核心的 Java 方法，例如内存管理、垃圾回收和安全机制等。JVM 是可定制化的，我们可以通过 Java 选项(java options)定制它，比如配置 JVM 内存的上下界。JVM 之所以被称为虚拟的是因为它提供了一个不依赖于底层操作系统和机器硬件的接口。这种独立于硬件和操作系统的特性正是 Java 程序可以一次编写多处执行的原因。

> 3）Java 运行时环境(JRE)
> JRE 是 JVM 的实施实现，它提供了运行 Java 程序的平台。JRE 包含了 JVM、Java 二进制文件和其它成功执行程序的类文件。JRE 不包含任何像 Java 编译器、调试器之类的开发工具。如果你只是想要执行 Java 程序，你只需安装 JRE 即可，没有安装 JDK 的必要。

JDK, JRE 和 JVM 的区别:

JDK 是用于开发的而 JRE 是用于运行 Java 程序的。
JDK 和 JRE 都包含了 JVM，从而使得我们可以运行 Java 程序。
JVM 是 Java 编程语言的核心并且具有平台独立性。

7.是否可以在 static 环境中访问非 static 变量

static 变量在 Java 中是属于类的，它在所有的实例中的值是一样的。当类被Java虚拟机载入的时候，会对 static 变量进行初始化。如果你的代码尝试不用实例来访问非 static 的变量，编译器会报错，因为这些变量还没有被创建出来，还没有跟任何实例关联上。

8.Java 中的 final关键字用法

(1)修饰类：表示该类不能被继承；
(2)修饰方法：表示方法不能被覆盖；
(3)修饰变量：表示变量只能一次赋值以后值不能被修改（常量）。

9.assert

assertion(断言)在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。一般来说，assertion 用于保证程序最基本、关键的正确性。assertion 检查通常在开发和测试时开启。为了提高性能，在软件发布后， assertion 检查通常是关闭的。在实现中，断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为 true；如果表达式计算为 false，那么系统会报告一个 AssertionError。

10.final, finally, finalize 的区别?

- **final**:修饰符（关键字）有三种用法：如果一个类被声明为final，意味着它不能再派生出新的子类，即不能被继承，因此它和 abstract 是反义词。将变量声明为 final，可以保证它们在使用中不被改变，被声明为 final 的变量必须在声明时给定初值，而在以后的引用中只能读取不可修改。被声明为 final 的方法也同样只能使用，不能在子类中被重写。
- **finally**:通常放在 try…catch 的后面构造总是执行代码块，这就意味着程序无论正常执行还是发生异常，这里的代码只要 JVM 不关闭都能执行，可以将释放外部资源的代码写在 finally 块中。
- **finalize**：Object 类中定义的方法，Java 中允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在销毁对象时调用的，通过重写finalize() 方法可以整理系统资源或者执行其他清理工作。

11.& 和 &&

& 和 && 都可以用作逻辑与的运算符，表示逻辑与（and），当运算符两边的表达式的结果都为 true 时，整个运算结果才为 true，否则，只要有一方为 false，则结果为 false。
&& 还具有短路的功能，即如果第一个表达式为 false，则不再计算第二个表达式.& 还可以用作位运算符，当 & 操作符两边的表达式不是 boolean 类型时，& 表示按位与操作.

12.char 型变量

char 类型可以存储一个中文汉字，因为 Java 中使用的编码是 Unicode（不选择任何特定的编码，直接使用字符在字符集中的编号，这是统一的唯一方法），一个 char 类型占 2 个字节（16bit），所以放一个中文是没问题的。

补充：使用 Unicode 意味着字符在 JVM 内部和外部有不同的表现形式，在 JVM 内部都是 Unicode，当这个字符被从 JVM 内部转移到外部时（例如存入文件系统中），需要进行编码转换。所以 Java 中有字节流和字符流，以及在字符流和字节流之间进行转换的转换流，如 InputStreamReader 和 OutputStreamReader.

13.String 和StringBuilder、StringBuffer 的区别

答：Java 平台提供了两种类型的字符串：**String 和StringBuffer / StringBuilder**，它们可以储存和操作字符串。其中 String 是只读字符串，也就意味着 String 引用的字符串内容是不能被改变的。

而 StringBuffer 和 StringBuilder 类表示的字符串对象可以直接进行修改。StringBuilder 是 JDK 1.5 中引入的，它和 StringBuffer 的方法完全相同，区别在于它是在单线程环境下使用的，因为它的所有方面都没有被 synchronized 修饰，因此它的效率也比 StringBuffer 略高。

有一个面试题问：有没有哪种情况用 + 做字符串连接比调用 StringBuffer / StringBuilder 对象的 append 方法性能更好?
如果连接后得到的字符串在静态存储区中是早已存在的，那么用+做字符串连接是优于 StringBuffer / StringBuilder 的 append 方法的。

14.错误和异常的区别(Error vs Exception)

java.lang.Error: Throwable 的子类，用于标记严重错误,表示系统级的错误和程序不必处理的异常。合理的应用程序不应该去 try/catch 这种错误。是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况；

java.lang.Exception: Throwable 的子类，表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。并且鼓励用户程序去 catch 它。

15.try{} 里的 return 语句

Java 允许在 finally 中改变返回值的做法是不好的，因为如果存在 finally 代码块，try 中的 return 语句不会立马返回调用者，而是记录下返回值待 finally 代码块执行完毕之后再向调用者返回其值，然后如果在 finally 中修改了返回值，这会对程序造成很大的困扰

16.throws、throw、try、catch、finally

一般情况下是用 try 来执行一段程序，如果出现异常，系统会抛出（throw）一个异常，这时候你可以通过它的类型来捕捉（catch）它，或最后（finally）由缺省处理器来处理；try 用来指定一块预防所有“异常”的程序；catch 子句紧跟在 try 块后面，用来指定你想要捕捉的“异常”的类型；throw 语句用来明确地抛出一个“异常”；throws用来标明一个成员函数可能抛出的各种“异常”；finally 为确保一段代码不管发生什么“异常”都被执行一段代码；

17.throw和throws有什么区别

throw关键字用来在程序中明确的抛出异常，相反，throws语句用来表明方法不能处理的异常。每一个方法都必须要指定哪些异常不能处理，所以方法的调用者才能够确保处理可能发生的异常，多个异常是用逗号分隔的。

18.equals与==的区别

== 是一个运算符。
Equals则是string对象的方法，可以.（点）出来。 我们比较无非就是这两种 1、基本数据类型比较 2、引用对象比较

1、**基本数据类型比较** ==比较两个值是否相等。相等为true 否则为false； equals不能直接用于基本类型的比较。需要将基本类型转换为包装器进行比较。

2、**引用对象比较** ==和equals都是比较栈内存中的地址是否相等 。相等为true 否则为false；

string是一个特殊的引用类型。对于两个字符串的比较，不管是 == 和 equals 这两者比较的都是字符串是否相同

19.String、StringBuffer与StringBuilder的区别

String是字符串常量，是不可变类。如果要操作少量的数据用
StringBuffer是字符串变量，是线程安全的。多线程操作字符串缓冲区 下操作大量数据
StringBuilder是字符串变量，非线程安全。单线程操作字符串缓冲区 下操作大量数据
速度：StringBuilder > StringBuffer > String

20.try catch finally，try里有return，finally还执行么?

1)不管有木有出现异常，finally块中代码都会执行
2)当try和catch中有return时，finally仍然会执行
3)finally是在return后面的表达式运算后执行的（此时并没有返回运算后的值，而是先把要返回的值保存起来，不管finally中的代码怎么样，返回的值都不会改变，任然是之前保存的值），所以函数返回值是在finally执行前确定的
4)finally中最好不要包含return，否则程序会提前退出，返回值不是try或catch中保存的返回值

21.static

1.静态变量

- 静态变量：又称为类变量，也就是说这个变量属于类的，类所有的实例都共享静态变量，可以直接通过类名来访问它。静态变量在内存中只存在一份。
- 实例变量：每创建一个实例就会产生一个实例变量，它与该实例同生共死。

2.静态方法

- 静态方法在类加载的时候就存在了，它不依赖于任何实例。所以静态方法必须有实现，也就是说它不能是抽象方法。
- 只能访问所属类的静态字段和静态方法，方法中不能有 this 和 super 关键字，因此这两个关键字与具体对象关联。

3.静态语句块

- 静态语句块在类初始化时运行一次。

4.静态内部类

- 非静态内部类依赖于外部类的实例，也就是说需要先创建外部类实例，才能用这个实例去创建非静态内部类。而静态内部类不需要。

22.浅拷贝和深拷贝

- **浅拷贝**：拷贝对象和原始对象的引用类型引用同一个对象。
- **深拷贝**：拷贝对象和原始对象的引用类型引用不同对象。

Java面对对象

1.面向对象软件开发的优点

(1) 代码开发模块化，更易维护和修改。
(2) 代码复用。
(3) 增强代码的可靠性和灵活性。
(4) 增加代码的可理解性。 面向对象编程有很多重要的特性，比如：封装，继承，多态和抽象

2.Java面向对象的三个特征与含义

一、继承：
概念：继承是从已有的类中派生出新的类，新的类能吸收已有类的数据属性和行为，并能扩展新的能力.
好处：提高代码的复用，缩短开发周期。

二、多态：
概念：多态（Polymorphism）按字面的意思就是“多种状态，即同一个实体同时具有多种形式"。一般表现形式是程序在运行的过程中，同一种类型在不同的条件下表现不同的结果。多态也称为动态绑定，一般是在运行时刻才能确定方法的具体执行对象，这个过程也称为动态委派。
好处：
1、将接口和实现分开，改善代码的组织结构和可读性，还能创建可拓展的程序。
2、消除类型之间的耦合关系。允许将多个类型视为同一个类型。
3、一个多态方法的调用允许有多种表现形式

三、封装：
概念：就是把对象的属性和行为（或服务）结合为一个独立的整体，并尽可能隐藏对象的内部实现细节。
好处：
1、隐藏信息，实现细节。让客户端程序员无法触及他们不应该触及的部分。
2、允许可设计者可以改变类内部的工作方式而不用担心会影响到客户端程序员。

3.重载(Overload)和重写(Override)的区别

- **重载**:存在于同一个类中，指一个方法与已经存在的方法名称上相同，但是参数类型、个数、顺序至少有一个不同。应该注意的是，返回值不同，其它都相同不算是重载。
- **重写**:
  存在于继承体系中，指子类实现了一个与父类在方法声明上完全相同的一个方法。为了满足里式替换原则，重写有以下三个限制：
  - 子类方法的访问权限必须大于等于父类方法；
  - 子类方法的返回类型必须是父类方法返回类型或为其子类型。
  - 子类方法抛出的异常类型必须是父类抛出异常类型或为其子类型。
  - 使用 @Override 注解，可以让编译器帮忙检查是否满足上面的三个限制条件。

4.接口和抽象类的区别

Java 提供和支持创建抽象类和接口。它们的实现有共同点，不同点在于：
1、接口中所有的方法隐含的都是抽象的。而抽象类则可以同时包含抽象和非抽象的方法。
2、类可以实现很多个接口，但是只能继承一个抽象类

3、类如果要实现一个接口，它必须要实现接口声明的所有方法。但是，类可以不实现抽象类声明的所有方法，当然，在这种情况下，类也必须得声明成是抽象的。
4、抽象类可以在不提供接口方法实现的情况下实现接口。
5、Java 接口中声明的变量默认都是 final 的。抽象类可以包含非 final 的变量。
6、Java 接口中的成员函数默认是 public 的。抽象类的成员函数可以是 private，protected 或者是 public 。
7、接口是绝对抽象的，不可以被实例化。抽象类也不可以被实例化，但是，如果它包含 main 方法的话是可以被调用的

Java集合

1.List、Map、Set 三个接口存储元素时各有什么特点

1）List是有序的Collection，使用此接口能够精确的控制每个元素插入的位置。用户能够使用索引（元素在List中的位置，类似于数组下标）来访问List中的元素，这类似于Java的数组。和下面要提到的Set不同，List允许有相同的元素。实现List接口的常用类LinkedList，ArrayList，Vector和Stack。

2）Set 是一种不包含重复的元素的 Collection，即任意的两个元素 e1 和 e2 都有e1.equals(e2)=false，Set 最多有一个 null 元素。

3）Map 接口： 请注意，Map 没有继承 Collection 接口，Map 提供 key 到 value 的映射．底层使用散列函数。

2.Collection 和 Collections的区别

Collection是集合类的上级接口，子接口主要有Set 和List、Map。
Collections是针对集合类的一个帮助类，提供了操作集合的工具方法：一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

JVM知识

1.内存溢出和内存泄漏

内存溢出：通俗理解就是内存不够，程序所需要的内存远远超出了你虚拟机分配的内存大小，就叫内存溢出

内存泄露：内存泄漏也称作“存储渗漏”，用动态存储分配函数动态开辟的空间，在使用完毕后未释放，结果导致一直占据该内存单元。直到程序结束。（其实说白了就是该内存空间使用完毕之后未回收）即所谓内存泄漏