Java 基础

1.JDK 和 JRE 有什么区别?

JRE：Java Runtime Environment JDK：它是Java开发运行环境，在程序员的电脑上当然要安装JDK；
JDK：Java Development Kit，Java Runtime Environment它是Java运行环境，如果你不需要开发只需要运行Java程序，那么你可以安装JRE。
例如程序员开发出的程序最终卖给了用户，用户不用开发，只需要运行程序，所以用户在电脑上安装JRE即可。
- JDK包含了JRE。
- JRE中包含虚拟机JVM

2.== 和 equals 的区别是什么?

== 比较的是变量(栈)内存中存放的对象的(堆)内存地址，用来判断两个对象的地址是否相同，即是否是指相同一个对象。比较的是真正意义上的指针操作。equals用来比较的是两个对象的内容是否相等，由于所有的类都是继承自java.lang.Object类的，所以适用于所有对象，如果没有对该方法进行覆盖的话，调用的仍然是Object类中的方法，而Object中的equals方法返回的却是==的判断。
String s="abcd"是一种非常特殊的形式,和new 有本质的区别。它是java中唯一不需要new 就可以产生对象的途径。以String s="abcd";形式赋值在java中叫直接量,它是在常量池中而不是象new一样放在压缩堆中。 这种形式的字符串，在JVM内部发生字符串拘留，即当声明这样的一个字符串后，JVM会在常量池中先查找有有没有一个值为"abcd"的对象,如果有,就会把它赋给当前引用.即原来那个引用和现在这个引用指点向了同一对象， 如果没有,则在常量池中新创建一个"abcd",下一次如果有String s1 = "abcd";又会将s1指向"abcd"这个对象,即以这形式声明的字符串,只要值相等,任何多个引用都指向同一对象.

3.两个对象的 hashCode()相同，则 equals()也一定为 true，对吗?

不对

4.final 在 java 中有什么作用?

final作为Java中的关键字可以用于三个地方。用于修饰类、类属性和类方法。
特征：凡是引用final关键字的地方皆不可修改!
(1)修饰类：表示该类不能被继承；
(2)修饰方法：表示方法不能被重写；
(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）。
相信大家都具备基本的常识: 被final修饰的变量是不能够被改变的. 但是这里的"不能够被改变"对于不同的数据类型是有不同的含义的.
当final修饰的是一个基本数据类型数据时， 这个数据的值在初始化后将不能被改变； 当final修饰的是一个引用类型数据时， 也就是修饰一个对象时， 引用在初始化后将永远指向一个内存地址， 不可修改. 但是该内存地址中保存的对象信息， 是可以进行修改的.

5.java 中的 Math.round(-1.5) 等于多少?

答：-1。

Math.round()四舍五入的原理，小数：
大于0.5，舍去小数，绝对值+1；
小于0.5，仅舍去小数；
等于0.5，取原数字+0.5

Math.floor() 求一个最接近它的整数，它的值小于或等于这个浮点数。（正数去掉小数，负数去小数+1）

6.String 属于基础的数据类型吗?

所以String类型不属于基础数据类型,但是为什么会有这么一问呢?

分析:

当我们在给一个String类型的变量赋值的时候我们是可以不使用 “new” 关键字的, 例如: String name = “小明”; 这样就ok了
这种写法和基本数据类型的使用是很像的,如果基础不扎实,整天的业务代码,还真有可能被唬住.
再有就是,java中的设计,String类型的值是存储在常量池中的.String类型是我们在编程中经常使用的数据类型.因此java的设计者将String类型做了一定的特殊处理.

7.java 中操作字符串都有哪些类？它们之间有什么区别?

String、StringBuffer、StringBuilder

相同（StringBuffer、StringBuilder）：
都是字符串的缓冲区、可变的字符序列；具有相同的构造和方法。

区别（String、StringBuffer、StringBuilder）：
内存
String 是不可变的对象，每次操作都会生成新的 String 对象，然后将指针指向新的 String 对象，StringBuffer、StringBuilder 可以在原有对象的基础上进行操作，所以在经常改变字符串内容的情况下最好不要使用 String。

出现版本：
StringBuffer是 Jdk 1.1
StringBuilder是 Jdk 1.5

线程安全：
StringBuffer线程安全，同步锁（synchronized），多线程仍可以保证数据安全
StringBuilder线程不安全，多线程无法保证数据安全

效率：
StringBuilder > StringBuffer > String

总结:
不频繁增改字符，就用String;否则用StringBuffer或StringBuilder

String的常用方法
Stringbuffer的构造（三构造）；常用方法（添加，删除，替换和反转）

8.String str="i"与 String str=new String(“i”)一样吗?

equit相同，==不同
String str="i"; 因为String 是final类型的，所以“i”应该是在常量池。

而new String("i");则是新建对象放到堆内存中。

9.如何将字符串反转?

1. 利用 StringBuffer 或 StringBuilder 的 reverse 成员方法:
2. 利用 String 的 toCharArray 方法先将字符串转化为 char 类型数组，然后将各个字符进行重新拼接:
3. 利用 String 的 CharAt 方法取出字符串中的各个字符:

10.String 类的常用方法都有那些?

(1).indexOf()：返回指定字符的索引。
(2).charAt()：返回指定索引处的字符。
(3).replace()：字符串替换。
(4).trim()：去除字符串两端空白。
(5).split()：分割字符串，返回一个分割后的字符串数组。
(6).getBytes()：返回字符串的 byte 类型数组。
(7).length()：返回字符串长度。
(8).toLowerCase()：将字符串转成小写字母。
(9).toUpperCase()：将字符串转成大写字符。
(10).substring()：截取字符串。
(11).equals()：字符串比较。

11.抽象类必须要有抽象方法吗?

答：不需要，

抽象类不一定有抽象方法；但是包含一个抽象方法的类一定是抽象类。（有抽象方法就是抽象类，是抽象类可以没有抽象方法）

解释：

抽象方法：
java中的抽象方法就是以abstract修饰的方法，这种方法只声明返回的数据类型、方法名称和所需的参数，没有方法体，也就是说抽象方法只需要声明而不需要实现。

抽象方法与抽象类：
当一个方法为抽象方法时，意味着这个方法必须被子类的方法所重写，否则其子类的该方法仍然是abstract的，而这个子类也必须是抽象的，即声明为abstract。abstract抽象类不能用new实例化对象，abstract方法只允许声明不能实现。如果一个类中含有abstract方法，那么这个类必须用abstract来修饰，当然abstract类也可以没有abstract方法。 一个抽象类里面没有一个抽象方法可用来禁止产生这种类的对象。

Java中的抽象类：
abstract class 在 Java 语言中表示的是一种继承关系，一个类只能使用一次继承关系。但是，一个类却可以实现多个interface。

在abstract class 中可以有自己的数据成员，也可以有非abstarct的成员方法，而在interface中，只能够有静态的不能被修改的数据成员（也就是必须是static final的，不过在 interface中一般不定义数据成员），所有的成员方法都是abstract的。

12.普通类和抽象类有哪些区别?

抽象类不能被实例化
抽象类可以有抽象方法，抽象方法只需申明，无需实现
含有抽象方法的类必须申明为抽象类
抽象的子类必须实现抽象类中所有抽象方法，否则这个子类也是抽象类
抽象方法不能被声明为静态
抽象方法不能用private修饰
抽象方法不能用final修饰

13.抽象类能使用 final 修饰吗?

“final修饰的类不能被继承，没有子类。如果类中有抽象的方法也是没有意义的。abstract类为抽象类。即该类只关心子类具有的功能，而不是功能的具体实现。如果 用final修饰方法，那么该方法则不能再被重写。final 是不能修饰abstract所修饰的方法的。”

14.接口和抽象类有什么区别?

抽象类是什么：

抽象类不能创建实例，它只能作为父类被继承。抽象类是从多个具体类中抽象出来的父类，它具有更高层次的抽象。从多个具有相同特征的类中抽象出一个抽象类，以这个抽象类作为其子类的模板，从而避免了子类的随意性。

(1) 抽象方法只作声明，而不包含实现，可以看成是没有实现体的虚方法
(2) 抽象类不能被实例化
(3) 抽象类可以但不是必须有抽象属性和抽象方法，但是一旦有了抽象方法，就一定要把这个类声明为抽象类
(4) 具体派生类必须覆盖基类的抽象方法
(5) 抽象派生类可以覆盖基类的抽象方法，也可以不覆盖。如果不覆盖，则其具体派生类必须覆盖它们

接口是什么：
(1) 接口不能被实例化
(2) 接口只能包含方法声明
(3) 接口的成员包括方法、属性、索引器、事件
(4) 接口中不能包含常量、字段(域)、构造函数、析构函数、静态成员

接口和抽象类的区别：
（1）抽象类可以有构造方法，接口中不能有构造方法。
（2）抽象类中可以有普通成员变量，接口中没有普通成员变量
（3）抽象类中可以包含静态方法，接口中不能包含静态方法
（4） 一个类可以实现多个接口，但只能继承一个抽象类。
（5）接口可以被多重实现，抽象类只能被单一继承
（6）如果抽象类实现接口，则可以把接口中方法映射到抽象类中作为抽象方法而不必实现，而在抽象类的子类中实现接口中方法

接口和抽象类的相同点：
(1) 都可以被继承
(2) 都不能被实例化
(3) 都可以包含方法声明
(4) 派生类必须实现未实现的方法

15.java 中 IO 流分为几种?

InputStream/Reader: 所有的输入流的基类，前者是字节输入流，后者是字符输入流。
OutputStream/Writer: 所有输出流的基类，前者是字节输出流，后者是字符输出流。

16.BIO、NIO、AIO 有什么区别?

BIO (Blocking I/O): 同步阻塞I/O模式，数据的读取写入必须阻塞在一个线程内等待其完成。在活动连接数不是特别高（小于单机1000）的情况下，这种模型是比较不错的，可以让每一个连接专注于自己的 I/O 并且编程模型简单，也不用过多考虑系统的过载、限流等问题。线程池本身就是一个天然的漏斗，可以缓冲一些系统处理不了的连接或请求。但是，当面对十万甚至百万级连接的时候，传统的 BIO 模型是无能为力的。因此，我们需要一种更高效的 I/O 处理模型来应对更高的并发量。
NIO (New I/O): NIO是一种同步非阻塞的I/O模型，在Java 1.4 中引入了NIO框架，对应 java.nio 包，提供了 Channel , Selector，Buffer等抽象。NIO中的N可以理解为Non-blocking，不单纯是New。它支持面向缓冲的，基于通道的I/O操作方法。 NIO提供了与传统BIO模型中的 Socket 和 ServerSocket 相对应的 SocketChannel 和 ServerSocketChannel 两种不同的套接字通道实现,两种通道都支持阻塞和非阻塞两种模式。阻塞模式使用就像传统中的支持一样，比较简单，但是性能和可靠性都不好；非阻塞模式正好与之相反。对于低负载、低并发的应用程序，可以使用同步阻塞I/O来提升开发速率和更好的维护性；对于高负载、高并发的（网络）应用，应使用 NIO 的非阻塞模式来开发
AIO (Asynchronous I/O): AIO 也就是 NIO 2。在 Java 7 中引入了 NIO 的改进版 NIO 2,它是异步非阻塞的IO模型。异步 IO 是基于事件和回调机制实现的，也就是应用操作之后会直接返回，不会堵塞在那里，当后台处理完成，操作系统会通知相应的线程进行后续的操作。AIO 是异步IO的缩写，虽然 NIO 在网络操作中，提供了非阻塞的方法，但是 NIO 的 IO 行为还是同步的。对于 NIO 来说，我们的业务线程是在 IO 操作准备好时，得到通知，接着就由这个线程自行进行 IO 操作，IO操作本身是同步的。查阅网上相关资料，我发现就目前来说 AIO 的应用还不是很广泛，Netty 之前也尝试使用过 AIO，不过又放弃了。

17.Files的常用方法都有哪些?

Files. exists()：检测文件路径是否存在。
Files. createFile()：创建文件。
Files. createDirectory()：创建文件夹。
Files. delete()：删除一个文件或目录。
Files. copy()：复制文件。
Files. move()：移动文件。
Files. size()：查看文件个数。
Files. read()：读取文件。
Files. write()：写入文件。

jdk1.5之后的三大版本

Java SE（J2SE，Java 2 Platform Standard Edition，标准版）
Java SE 以前称为 J2SE。它允许开发和部署在桌面、服务器、嵌入式环境和实时环境中使用的 Java 应用程序。Java SE 包含了支持 Java Web 服务开发的类，并为Java EE和Java ME提供基础。
Java EE（J2EE，Java 2 Platform Enterprise Edition，企业版）
Java EE 以前称为 J2EE。企业版本帮助开发和部署可移植、健壮、可伸缩且安全的服务器端Java 应用程序。Java EE 是在 Java SE 的基础上构建的，它提供 Web 服务、组件模型、管理和通信 API，可以用来实现企业级的面向服务体系结构（service-oriented architecture，SOA）和 Web2.0应用程序。
2018年2月，Eclipse 宣布正式将 JavaEE 更名为 JakartaEE
Java ME（J2ME，Java 2 Platform Micro Edition，微型版）
Java ME 以前称为 J2ME。Java ME 为在移动设备和嵌入式设备（比如手机、PDA、电视机顶盒和打印机）上运行的应用程序提供一个健壮且灵活的环境。Java ME 包括灵活的用户界面、健壮的安全模型、许多内置的网络协议以及对可以动态下载的连网和离线应用程序的丰富支持。基于 Java ME 规范的应用程序只需编写一次，就可以用于许多设备，而且可以利用每个设备的本机功能。

JVM、JRE和JDK的关系

JVM
Java Virtual Machine是Java虚拟机，Java程序需要运行在虚拟机上，不同的平台有自己的虚拟机，因此Java语言可以实现跨平台。
JRE
Java Runtime Environment包括Java虚拟机和Java程序所需的核心类库等。核心类库主要是java.lang包：包含了运行Java程序必不可少的系统类，如基本数据类型、基本数学函数、字符串处理、线程、异常处理类等，系统缺省加载这个包
如果想要运行一个开发好的Java程序，计算机中只需要安装JRE即可。
JDK
Java Development Kit是提供给Java开发人员使用的，其中包含了Java的开发工具，也包括了JRE。所以安装了JDK，就无需再单独安装JRE了。其中的开发工具：编译工具(javac.exe)，打包工具(jar.exe)等

Java语言有哪些特点

- 简单易学（Java语言的语法与C语言和C++语言很接近）
- 面向对象（封装，继承，多态）
- 平台无关性（Java虚拟机实现平台无关性）
- 支持网络编程并且很方便（Java语言诞生本身就是为简化网络编程设计的）
- 支持多线程（多线程机制使应用程序在同一时间并行执行多项任）
- 健壮性（Java语言的强类型机制、异常处理、垃圾的自动收集等）
- 安全性

Java和C++的区别

- 都是面向对象的语言，都支持封装、继承和多态
- Java不提供指针来直接访问内存，程序内存更加安全
- Java的类是单继承的，C++支持多重继承；虽然Java的类不可以多继承，但是接口可以多继承。
- Java有自动内存管理机制，不需要程序员手动释放无用内存

访问修饰符 public,private,protected,以及不写（默认）时的区别

定义：Java中，可以使用访问修饰符来保护对类、变量、方法和构造方法的访问。Java 支持 4 种不同的访问权限。
分类
- private : 在同一类内可见。使用对象：变量、方法。 注意：不能修饰类（外部类）
- default (即缺省，什么也不写，不使用任何关键字）: 在同一包内可见，不使用任何修饰符。使用对象：类、接口、变量、方法。
- protected : 对同一包内的类和所有子类可见。使用对象：变量、方法。 注意：不能修饰类（外部类）。
- public : 对所有类可见。使用对象：类、接口、变量、方法

访问修饰符图

修饰符	当前类	同 包	子 类	其他包
private	√	×	×	×
default	√	√	×	×
protected	√	√	√	×
public	√	√	√	√

this关键字的用法

this是自身的一个对象，代表对象本身，可以理解为：指向对象本身的一个指针。
this的用法在java中大体可以分为3种：
1.普通的直接引用，this相当于是指向当前对象本身。
2.形参与成员名字重名，用this来区分：

```
public Person(String name, int age) {
    this.name = name;
    this.age = age;
}
```

3.引用本类的构造函数

```
class Person{
    private String name;
    private int age;
    public Person() {
    }

    public Person(String name) {
        this.name = name;
    }
    public Person(String name, int age) {
        this(name);
        this.age = age;
    }
}
```

super关键字的用法

super可以理解为是指向自己超（父）类对象的一个指针，而这个超类指的是离自己最近的一个父类。

super也有三种用法：

1.普通的直接引用
与this类似，super相当于是指向当前对象的父类的引用，这样就可以用super.xxx来引用父类的成员。
2.子类中的成员变量或方法与父类中的成员变量或方法同名时，用super进行区分

```
class Person{
    protected String name;
    public Person(String name) {
        this.name = name;
    }

}
class Student extends Person{
    private String name;
    public Student(String name, String name1) {
        super(name);
        this.name = name1;
    }
    public void getInfo(){
        System.out.println(this.name);      //Child
        System.out.println(super.name);     //Father
    }

}
public class Test {
    public static void main(String[] args) {
       Student s1 = new Student("Father","Child");
       s1.getInfo();
```

```
    }
}
```
3.引用父类构造函数

super（参数）：调用父类中的某一个构造函数（应该为构造函数中的第一条语句）。
this（参数）：调用本类中另一种形式的构造函数（应该为构造函数中的第一条语句）。

static存在的主要意义

static的主要意义是在于创建独立于具体对象的域变量或者方法。以致于即使没有创建对象，也能使用属性和调用方法!

static关键字还有一个比较关键的作用就是 用来形成静态代码块以优化程序性能。static块可以置于类中的任何地方，类中可以有多个static块。在类初次被加载的时候，会按照static块的顺序来执行每个static块，并且只会执行一次。

为什么说static块可以用来优化程序性能，是因为它的特性:只会在类加载的时候执行一次。因此，很多时候会将一些只需要进行一次的初始化操作都放在static代码块中进行。

面向对象和面向过程的区别

面向过程：
优点：性能比面向对象高，因为类调用时需要实例化，开销比较大，比较消耗资源;比如单片机、嵌入式开发、Linux/Unix等一般采用面向过程开发，性能是最重要的因素。
缺点：没有面向对象易维护、易复用、易扩展

面向对象：
优点：易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护
缺点：性能比面向过程低

面向过程是具体化的，流程化的，解决一个问题，你需要一步一步的分析，一步一步的实现。
面向对象是模型化的，你只需抽象出一个类，这是一个封闭的盒子，在这里你拥有数据也拥有解决问题的方法。需要什么功能直接使用就可以了，不必去一步一步的实现，至于这个功能是如何实现的，管我们什么事？我们会用就可以了。
面向对象的底层其实还是面向过程，把面向过程抽象成类，然后封装，方便我们使用的就是面向对象了。

其中Java 面向对象编程三大特性：封装 继承 多态

封装：隐藏对象的属性和实现细节，仅对外提供公共访问方式，将变化隔离，便于使用，提高复用性和安全性。
继承：继承是使用已存在的类的定义作为基础建立新类的技术，新类的定义可以增加新的数据或新的功能，也可以用父类的功能，但不能选择性地继承父类。通过使用继承可以提高代码复用性。继承是多态的前提。
关于继承如下 3 点请记住：
- 子类拥有父类非 private 的属性和方法。
- 子类可以拥有自己属性和方法，即子类可以对父类进行扩展。
- 子类可以用自己的方式实现父类的方法。
多态性：父类或接口定义的引用变量可以指向子类或具体实现类的实例对象。提高了程序的拓展性。在Java中有两种形式可以实现多态：继承（多个子类对同一方法的重写）和接口（实现接口并覆盖接口中同一方法）。
- 方法重载（overload）实现的是编译时的多态性（也称为前绑定），而方法重写（override）实现的是运行时的多态性（也称为后绑定）。
一个引用变量到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。运行时的多态是面向对象最精髓的东西，要实现多态需要做两件事：
- 方法重写（子类继承父类并重写父类中已有的或抽象的方法）；
- 对象造型（用父类型引用子类型对象，这样同样的引用调用同样的方法就会根据子类对象的不同而表现出不同的行为）。

什么是多态机制？Java语言是如何实现多态的？

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。

多态分为编译时多态和运行时多态。其中编辑时多态是静态的，主要是指方法的重载，它是根据参数列表的不同来区分不同的函数，通过编辑之后会变成两个不同的函数，在运行时谈不上多态。而运行时多态是动态的，它是通过动态绑定来实现的，也就是我们所说的多态性。

多态的实现
Java实现多态有三个必要条件：继承、重写、向上转型。
继承：在多态中必须存在有继承关系的子类和父类。
重写：子类对父类中某些方法进行重新定义，在调用这些方法时就会调用子类的方法。
向上转型：在多态中需要将子类的引用赋给父类对象，只有这样该引用才能够具备技能调用父类的方法和子类的方法。
只有满足了上述三个条件，我们才能够在同一个继承结构中使用统一的逻辑实现代码处理不同的对象，从而达到执行不同的行为。

对于Java而言，它多态的实现机制遵循一个原则：当超类对象引用变量引用子类对象时，被引用对象的类型而不是引用变量的类型决定了调用谁的成员方法，但是这个被调用的方法必须是在超类中定义过的，也就是说被子类覆盖的方法。

抽象类和接口的对比

抽象类是用来捕捉子类的通用特性的。接口是抽象方法的集合。
从设计层面来说，抽象类是对类的抽象，是一种模板设计，接口是行为的抽象，是一种行为的规范。

相同点
接口和抽象类都不能实例化
都位于继承的顶端，用于被其他实现或继承
都包含抽象方法，其子类都必须覆写这些抽象方法
不同点

参数	抽象类	接口
声明	抽象类使用abstract关键字声明	接口使用interface关键字声明
实现	子类使用extends关键字来继承抽象类。如果子类不是抽象类的话，它需要提供抽象类中所有声明的方法的实现	子类使用implements关键字来实现接口。它需要提供接口中所有声明的方法的实现
构造器	抽象类可以有构造器	接口不能有构造器
访问修饰符	抽象类中的方法可以是任意访问修饰符	接口方法默认修饰符是public。并且不允许定义为 private 或者 protected
多继承	一个类最多只能继承一个抽象类	一个类可以实现多个接口
字段声明	抽象类的字段声明可以是任意的	接口的字段默认都是 static 和 final 的

普通类和抽象类有哪些区别?

普通类不能包含抽象方法，抽象类可以包含抽象方法。
抽象类不能直接实例化，普通类可以直接实例化。

抽象类能使用 final 修饰吗?

不能，定义抽象类就是让其他类继承的，如果定义为 `final` 该类就不能被继承，这样彼此就会产生矛盾，所以 `final` 不能修饰抽象类

自动装箱与拆箱

装箱：将基本类型用它们对应的引用类型包装起来；
拆箱：将包装类型转换为基本数据类型；

int 和 Integer 有什么区别

`Java` 是一个近乎纯洁的面向对象编程语言，但是为了编程的方便还是引入了基本数据类型，但是为了能够将这些基本数据类型当成对象操作，`Java` 为每一个基本数据类型都引入了对应的包装类型（`wrapper class`），`int` 的包装类就是 `Integer`，从 `Java 5` 开始引入了自动装箱/拆箱机制，使得二者可以相互转换。
`Java` 为每个原始类型提供了包装类型：
原始类型：`boolean`，`char`，`byte`，`short`，`int`，`long`，`float`，`double`
包装类型：`Boolean`，`Character`，`Byte`，`Short`，`Integer`，`Long`，`Float`，`Double`

Integer a= 127 与 Integer b = 127相等吗
对于对象引用类型：==比较的是对象的内存地址。
对于基本数据类型：==比较的是值。

如果整型字面量的值在-128到127之间，那么自动装箱时不会new新的Integer对象，而是直接引用常量池中的Integer对象，超过范围 a1==b1的结果是false

```
public static void main(String[] args) {
    Integer a = new Integer(3);
    Integer b = 3;  // 将3自动装箱成Integer类型
    int c = 3;
    System.out.println(a == b); // false 两个引用没有引用同一对象
    System.out.println(a == c); // true a自动拆箱成int类型再和c比较
    System.out.println(b == c); // true

    Integer a1 = 128;
    Integer b1 = 128;
    System.out.println(a1 == b1); // false

    Integer a2 = 127;
    Integer b2 = 127;
    System.out.println(a2 == b2); // true
}
```

String和StringBuffer、StringBuilder的区别是什么？String为什么是不可变的

可变性
String类中使用字符数组保存字符串，private final char value[]，所以string对象是不可变的。StringBuilder与StringBuffer都继承自AbstractStringBuilder类，在AbstractStringBuilder中也是使用字符数组保存字符串，char[] value，这两种对象都是可变的。

线程安全性
String中的对象是不可变的，也就可以理解为常量，线程安全。AbstractStringBuilder是StringBuilder与StringBuffer的公共父类，定义了一些字符串的基本操作，如expandCapacity、append、insert、indexOf等公共方法。StringBuffer对方法加了同步锁或者对调用的方法加了同步锁，所以是线程安全的。StringBuilder并没有对方法进行加同步锁，所以是非线程安全的。

性能
每次对String 类型进行改变的时候，都会生成一个新的String对象，然后将指针指向新的String 对象。StringBuffer每次都会对StringBuffer对象本身进行操作，而不是生成新的对象并改变对象引用。相同情况下使用StirngBuilder 相比使用StringBuffer 仅能获得10%~15% 左右的性能提升，但却要冒多线程不安全的风险。

对于三者使用的总结
- 如果要操作少量的数据用 = String
- 单线程操作字符串缓冲区 下操作大量数据 = StringBuilder
- 多线程操作字符串缓冲区 下操作大量数据 = StringBuffer

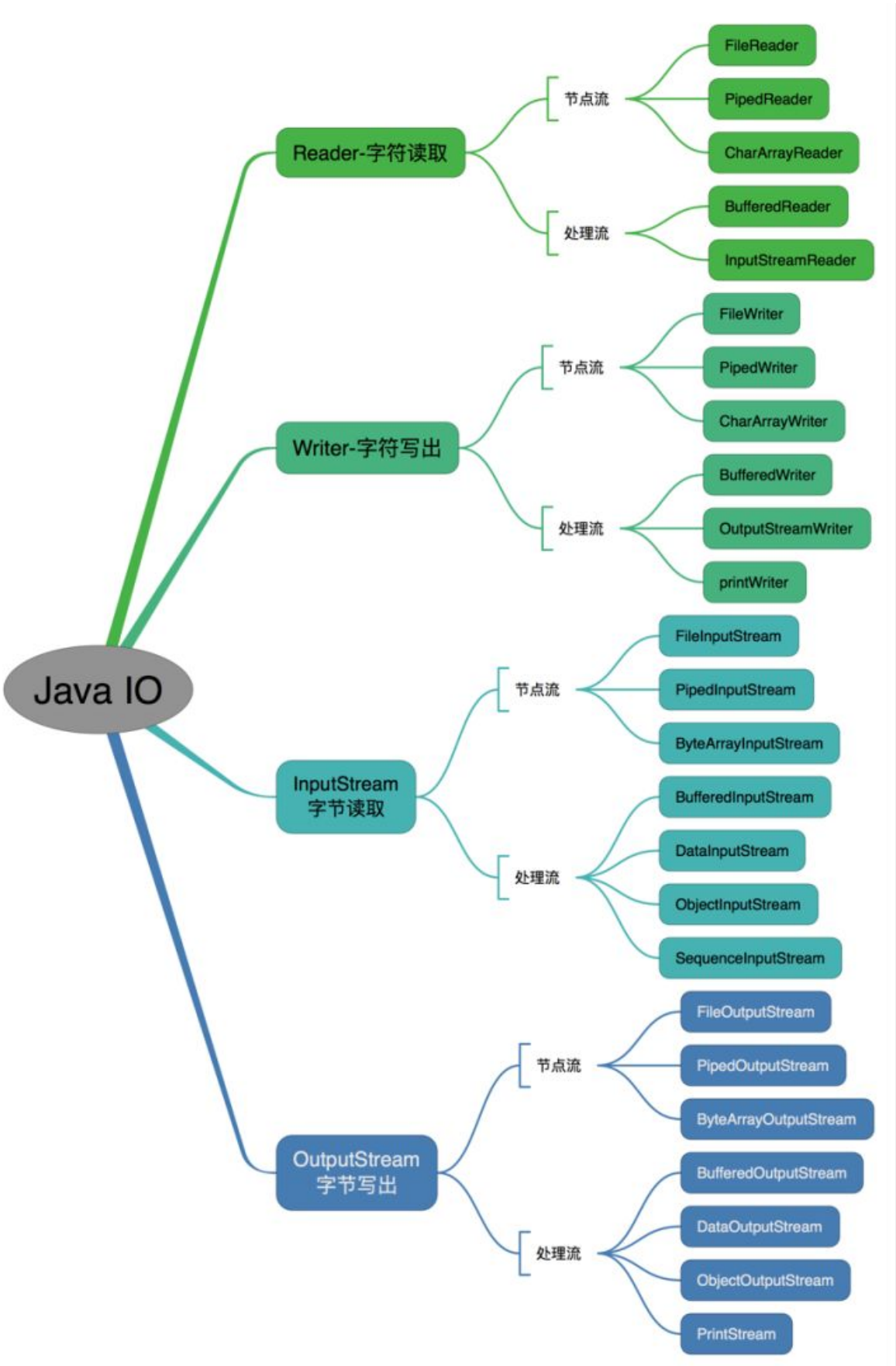
java 中 IO 流分为几种?

- 按照流的流向分，可以分为输入流和输出流；
- 按照操作单元划分，可以划分为字节流和字符流；
- 按照流的角色划分为节点流和处理流。
  Java Io流共涉及40多个类，这些类看上去很杂乱，但实际上很有规则，而且彼此之间存在非常紧

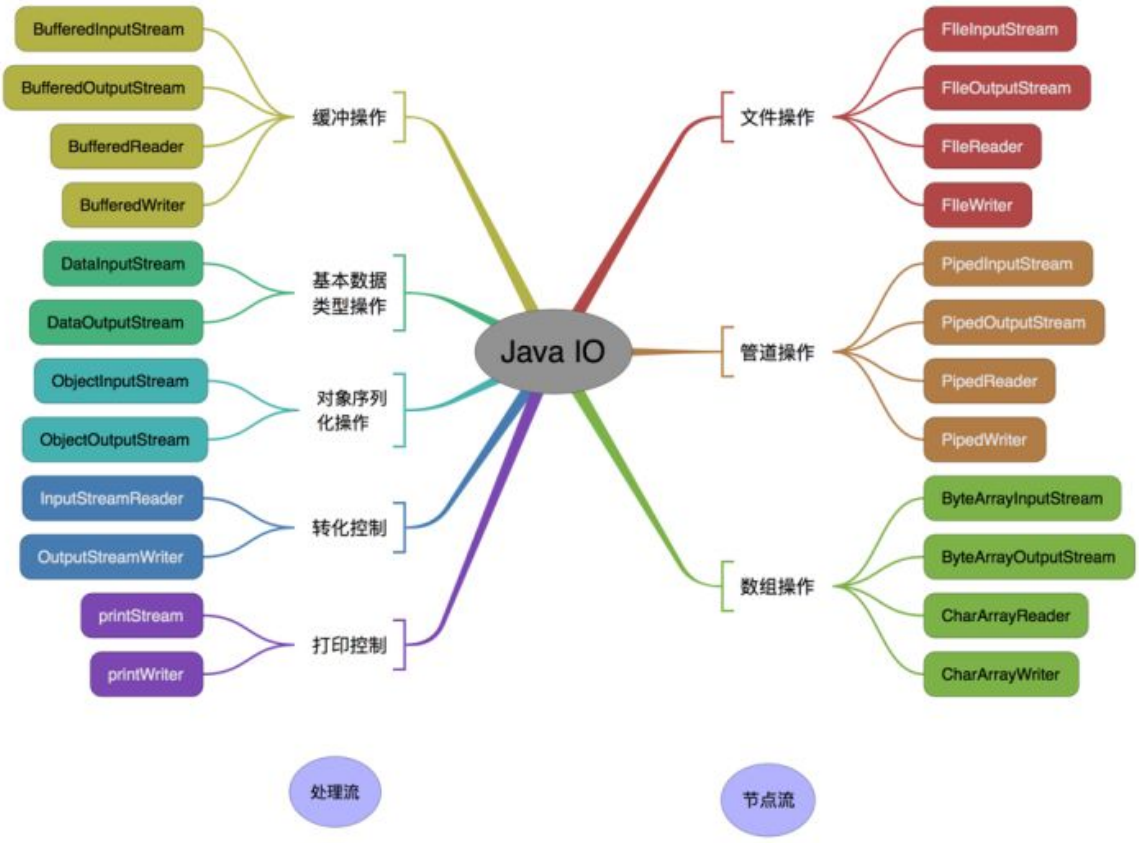
密的联系， Java I0流的40多个类都是从如下4个抽象类基类中派生出来的。

- InputStream/Reader: 所有的输入流的基类，前者是字节输入流，后者是字符输入流。
- OutputStream/Writer: 所有输出流的基类，前者是字节输出流，后者是字符输出流。

按操作方式分类结构图：



按操作对象分类结构图：



重写与重载

构造器（constructor）是否可被重写（override）

构造器不能被继承，因此不能被重写，但可以被重载。

重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分?

方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。
重载：发生在同一个类中，方法名相同参数列表不同（参数类型不同、个数不同、顺序不同），与方法返回值和访问修饰符无关，即重载的方法不能根据返回类型进行区分
重写：发生在父子类中，方法名、参数列表必须相同，返回值小于等于父类，抛出的异常小于等于父类，访问修饰符大于等于父类（里氏代换原则）；如果父类方法访问修饰符为`private`则子类中就不是重写。