专题3： Java 面试题（史上最全、定期更新）

本文版本说明：V1

> 此文的格式，由markdown 通过程序转成而来，由于很多表格，没有来的及调整，出现一个格式问题，尼恩在此给大家道歉啦。
>
> 由于社群很多小伙伴，在面试，不断的交流最新的面试难题，所以，《Java面试红宝书》， 后面会不断升级，迭代。
>
> 本专题，作为 《Java面试红宝书》的第10个专题， 《Java面试红宝书》一共30个面试专题。

《Java面试红宝书》升级的规划为：

后续基本上，**每一个月，都会发布一次**，最新版本，可以扫描扫架构师尼恩微信， 发送 “领取电子书” 获取。

尼恩的微信二维码在哪里呢？ 具体可以百度搜索 **疯狂创客圈 总目录**

面试问题交流说明：

如果遇到分布式事务的面试难题，或者其他的面试难题，都可以来 疯狂创客圈社群交流，

加入交流群，加尼恩微信即可，

尼恩的微信二维码在哪里呢？ 具体可以百度搜索 **疯狂创客圈 总目录**

Java概述

何为编程

编程就是让计算机为解决某个问题而使用某种程序设计语言编写程序代码，并最终得到结果的过程。

为了使计算机能够理解人的意图，人类就必须要将需解决的问题的思路、方法、和手段通过计算机能够理解的形式告诉计算机，使得计算机能够根据人的指令一步一步去工作，完成某种特定的任务。这种人和计算机之间交流的过程就是编程。

什么是Java

Java是一门面向对象编程语言，不仅吸收了C++语言的各种优点，还摒弃了C++里难以理解的多继承、指针等概念，因此Java语言具有功能强大和简单易用两个特征。Java语言作为静态面向对象编程语言的代表，极好地实现了面向对象理论，允许程序员以优雅的思维方式进行复杂的编程 。

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。

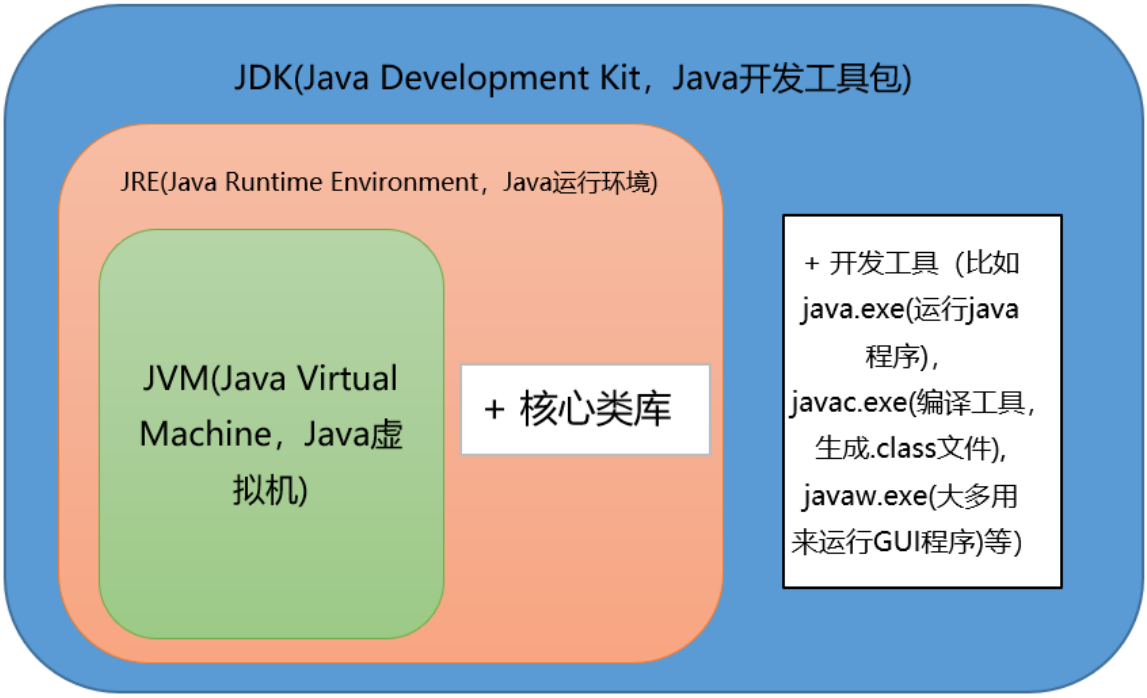
JVM、JRE和JDK的关系

- 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

JVM&JRE&JDK关系图



什么是跨平台性？原理是什么

所谓跨平台性，是指java语言编写的程序，一次编译后，可以在多个系统平台上运行。

实现原理：Java程序是通过java虚拟机在系统平台上运行的，只要该系统可以安装相应的java虚拟机，该系统就可以运行java程序。

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

什么是字节码？采用字节码的最大好处是什么

字节码：Java源代码经过虚拟机编译器编译后产生的文件（即扩展为.class的文件），它不面向任何特定的处理器，只面向虚拟机。

采用字节码的好处：

Java语言通过字节码的方式，在一定程度上解决了传统解释型语言执行效率低的问题，同时又保留了解释型语言可移植的特点。所以Java程序运行时比较高效，而且，由于字节码并不专对一种特定的机器，因此，Java程序无须重新编译便可在多种不同的计算机上运行。

先看下java中的编译器和解释器：

Java中引入了虚拟机的概念，即在机器和编译程序之间加入了一层抽象的虚拟机器。这台虚拟的机器在任何平台上都提供给编译程序一个的共同的接口。编译程序只需要面向虚拟机，生成虚拟机能够理解的代码，然后由解释器来将虚拟机代码转换为特定系统的机器码执行。在Java中，这种供虚拟机理解的代码叫做字节码（即扩展为.class的文件），它不面向任何特定的处理器，只面向虚拟机。每一种平台的解释器是不同的，但是实现的虚拟机是相同的。Java源程序经过编译器编译后变成字节码，字节码由虚拟机解释执行，虚拟机将每一条要执行的字节码送给解释器，解释器将其翻译成特定机器上的机器码，然后在特定的机器上运行，这就是上面提到的Java的特点的编译与解释并存的解释。

```
Java源代码---->编译器---->jvm可执行的Java字节码(即虚拟指令)---->jvm---->jvm中解释器----->机器可执行的二进制机器码---->程序运行。
```

什么是Java程序的主类？应用程序和小程序的主类有何不同？

一个程序中可以有多个类，但只能有一个类是主类。在Java应用程序中，这个主类是指包含main()方法的类。而在Java小程序中，这个主类是一个继承自系统类JApplet或Applet的子类。应用程序的主类不一定要求是public类，但小程序的主类要求必须是public类。主类是Java程序执行的入口点。

Java应用程序与小程序之间有那些差别？

简单说应用程序是从主线程启动(也就是main()方法)。applet小程序没有main方法，主要是嵌在浏览器页面上运行(调用init()线程或者run()来启动)，嵌入浏览器这点跟flash的小游戏类似。

Java和C++的区别

我知道很多人没学过C++，但是面试官就是没事喜欢拿咱们Java和C++比呀！没办法！！！就算没学过C++，也要记下来！

- 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

Oracle JDK 和 OpenJDK 的对比

1. Oracle JDK版本将每三年发布一次，而OpenJDK版本每三个月发布一次；
2. OpenJDK 是一个参考模型并且是完全开源的，而Oracle JDK是OpenJDK的一个实现，并不是完全开源的；
3. Oracle JDK 比 OpenJDK 更稳定。OpenJDK和Oracle JDK的代码几乎相同，但Oracle JDK有更多的类和一些错误修复。因此，如果您想开发企业/商业软件，我建议您选择Oracle JDK，因为它经过了彻底的测试和稳定。某些情况下，有些人提到在使用OpenJDK 可能会遇到了许多应用程序崩溃的问题，但是，只需切换到Oracle JDK就可以解决问题；
4. 在响应性和JVM性能方面，Oracle JDK与OpenJDK相比提供了更好的性能；
5. Oracle JDK不会为即将发布的版本提供长期支持，用户每次都必须通过更新到最新版本获得支持来获取最新版本；
6. Oracle JDK根据二进制代码许可协议获得许可，而OpenJDK根据GPL v2许可获得许可。

面向对象

什么是面向对象?

面向对象程序设计是以建立模型体现出来的抽象思维过程和面向对象的方法。我们可以将某个事物抽象出来，赋予它自己的特征，并且可以针对这个事物进行相应的操作，以及规定与其他对象之间的关系。可以降低代码的耦合度，使程序更加灵活。

多态的好处

允许不同类对象对同一消息做出响应，即同一消息可以根据发送对象的不同而采用多种不同的行为方式(发送消息就是函数调用)。即父类型的引用指向子类型的对象。主要有以下优点：

可替换性：多态对已存在代码具有可替换性

可扩充性：增加新的子类不影响已经存在的类结构

更加灵活

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。

抽象：抽象是将一类对象的共同特征总结出来构造类的过程，包括数据抽象和行为抽象两方面。抽象只关注对象有哪些属性和行为，并不关注这些行为的细节是什么。

面向对象三大特性?

其中Java 面向对象编程三大特性：封装 继承 多态

封装

封装把一个对象的属性私有化，同时提供一些可以被外界访问的属性的方法，如果属性不想被外界访问，我们大可不必提供方法给外界访问。但是如果一个类没有提供给外界访问的方法，那么这个类也没有什么意义了。

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。通过使用继承我们能够非常方便地复用以前的代码。

关于继承如下 3 点请记住：

1. 子类拥有父类非 private 的属性和方法。
2. 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。
3. 子类可以用自己的方式实现父类的方法。（以后介绍）。

多态

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。

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）。

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。

面向对象五大基本原则是什么?

- 单一职责原则SRP(Single Responsibility Principle)
  类的功能要单一，不能包罗万象，跟杂货铺似的。
- 开放封闭原则OCP(Open - Close Principle)
  一个模块对于拓展是开放的，对于修改是封闭的，想要增加功能热烈欢迎，想要修改，哼，一万个不乐意。
- 里式替换原则LSP(the Liskov Substitution Principle LSP)
  子类可以替换父类出现在父类能够出现的任何地方。比如你能代表你爸去你姥姥家干活。哈哈~~
- 依赖倒置原则DIP(the Dependency Inversion Principle DIP)
  高层次的模块不应该依赖于低层次的模块，他们都应该依赖于抽象。抽象不应该依赖于具体实现，具体实现应该依赖于抽象。就是你出国要说你是中国人，而不能说你是哪个村子的。比如说中国人是抽象的，下面有具体的xx省，xx市，xx县。你要依赖的抽象是中国人，而不是你是xx村的。
- 接口分离原则ISP(the Interface Segregation Principle ISP)
  设计时采用多个与特定客户类有关的接口比采用一个通用的接口要好。就比如一个手机拥有打电话，看视频，玩游戏等功能，把这几个功能拆分成不同的接口，比在一个接口里要好的多。

什么是值传递和引用传递?

值传递，是对基本型变量而言的，传递的是该变量的一个副本，改变副本不影响原变量。

引用传递，一般是对于对象型变量而言的，传递的是该对象地址的一个副本，并不是原对象本身。

一般认为，Java 内的传递都是值传递，Java 中实例对象的传递是引用传递。

代码中如何实现多态

实现多态主要有以下三种方式：

1. 接口实现
2. 继承父类重写方法
3. 同一类中进行方法重载

接口的意义

规范，扩展，回调。

抽象类的意义

为其他子类提供一个公共的类型

封装子类中重复定义的内容

定义抽象方法,子类虽然有不同的实现，但是定义时一致的

接口和抽象类的区别

比较	抽象类	接口
默认方法	抽象类可以有默认的方法实现	java 8之前,接口中不存在方法的实现.
实现方式	子类使用extends关键字来继承抽象类.如果子类不是抽象类,子类需要提供抽象类中所声明方法的实现	子类使用implements来实现接口,需要提供接口中所有声明的实现
构造器	抽象类中可以有构造器,	接口中不能
访问修饰符	抽象方法可以有public,protected和default等修饰	接口默认是public,不能使用其他修饰符
多继承	一个子类只能存在一个父类	一个子类可以存在多个接口
访问新方法	想抽象类中添加新方法,可以提供默认的实现,因此可以不修改子类现有的代码	如果往接口中添加新方法,则子类中需要实现该方法.

父类的静态方法能否被子类重写

不能。重写只适用于实例方法,不能用于静态方法，而子类当中含有和父类相同签名的静态方法，我们一般称之为隐藏。

什么是不可变对象

不可变对象指对象一旦被创建，状态就不能再改变。任何修改都会创建一个新的对象，如 String、Integer及其它包装类。

静态变量和实例变量的区别?

静态变量存储在方法区，属于类所有。实例变量存储在堆当中，其引用存在当前线程栈。

能否创建一个包含可变对象的不可变对象?

当然可以创建一个包含可变对象的不可变对象的，你只需要谨慎一点，不要共享可变对象的引用就可以了，如果需要变化时，就返回原对象的一个拷贝。最常见的例子就是对象中包含一个日期对象的引用。

Overload和Override的区别。Overloaded的方法是否可以改变返回值的类型

答： 方法的重写Overriding和重载Overloading是Java多态性的不同表现。重写Overriding是父类与子类之间多态性的一种表现，重载Overloading是一个类中多态性的一种表现。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)。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。Overloaded的方法是可以改变返回值的类型

基础语法

数据类型

Java有哪些数据类型

定义：Java语言是强类型语言，对于每一种数据都定义了明确的具体的数据类型，在内存中分配了不同大小的内存空间。

分类

- 基本数据类型
  - 数值型
    - 整数类型(byte,short,int,long)
    - 浮点类型(float,double)
  - 字符型(char)
  - 布尔型(boolean)
- 引用数据类型
  - 类(class)
  - 接口(interface)
  - 数组([])

Java基本数据类型图

类型	类型名称	关键字	占用内存	取值范围	作为成员变量的默认值
整形	字节型	byte	1 字节	-128(-2^7) ~ 127(2^7-1)	0
	短整型	short	2 字节	-32,768(-2^15) ~ 32,767(2^15-1)	0
	整型	int	4 字节	-2,147,483,648(-2^31) ~ 2,147,483,647(2^31-1)	0
	长整型	long	8 字节	-9,223,372,036,854,775,808(-2^63) ~ 9,223,372,036,854,775,807(2^63-1)	0L
浮点型	单精度浮点型	float	4 字节	-3.403E38 ~ 3.403E38	0.0F
	双精度浮点型	double	8 字节	-1.798E308 ~ 1.798E308	0.0D
字符型	字符型	char	2 字节	表示一个字符，如('a','A','家')	'\u0000'
布尔型	布尔型	boolean	1 字节	只有两个值，true 或 false	false

switch 是否能作用在 byte 上，是否能作用在 long 上，是否能作用在 String 上

在 Java 5 以前，switch(expr)中，expr 只能是 byte、short、char、int。从 Java5 开始，Java 中引入了枚举类型，expr 也可以是 enum 类型，从 Java 7 开始，expr 还可以是字符串（String），但是长整型（long）在目前所有的版本中都是不可以的。

用最有效率的方法计算 2 乘以 8

2 << 3（左移 3 位相当于乘以 2 的 3 次方，右移 3 位相当于除以 2 的 3 次方）。

Math.round(11.5) 等于多少？Math.round(-11.5)等于多少

Math.round(11.5)的返回值是 12，Math.round(-11.5)的返回值是-11。四舍五入的原理是在参数上加 0.5 然后进行下取整。

float f=3.4;是否正确

不正确。3.4 是双精度数，将双精度型（double）赋值给浮点型（float）属于下转型（down-casting，也称为窄化）会造成精度损失，因此需要强制类型转换float f =(float)3.4; 或者写成 float f =3.4F;。

short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗

对于 short s1 = 1; s1 = s1 + 1;由于 1 是 int 类型，因此 s1+1 运算结果也是 int型，需要强制转换类型才能赋值给 short 型。

而 short s1 = 1; s1 += 1;可以正确编译，因为 s1+= 1;相当于 s1 = (short(s1 + 1);其中有隐含的强制类型转换。

编码

Java语言采用何种编码方案？有何特点?

Java语言采用Unicode编码标准，Unicode（标准码），它为每个字符制订了一个唯一的数值，因此在任何的语言，平台，程序都可以放心的使用。

注释

什么Java注释

定义：用于解释说明程序的文字

分类

- 单行注释
  格式： // 注释文字
- 多行注释
  格式： /* 注释文字 */
- 文档注释
  格式：/** 注释文字 */

作用

在程序中，尤其是复杂的程序中，适当地加入注释可以增加程序的可读性，有利于程序的修改、调试和交流。注释的内容在程序编译的时候会被忽视，不会产生目标代码，注释的部分不会对程序的执行结果产生任何影响。

注意事项：多行和文档注释都不能嵌套使用。

访问修饰符

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	√	√	√	√

运算符

&和&&的区别

&运算符有两种用法：(1)按位与；(2)逻辑与。

&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是true 整个表达式的值才是 true。&&之所以称为短路运算，是因为如果&&左边的表达式的值是 false，右边的表达式会被直接短路掉，不会进行运算。

注意：逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

3*0.1==0.3返回值是什么

false，因为有些浮点数不能完全精确的表示出来。

a=a+b与a+=b有什么区别吗?

+=操作符会进行隐式自动类型转换，此处a+=b隐式的将加操作的结果类型强制转换为持有结果的类型，而a=a+b则不会自动进行类型转换。如：

byte a = 127;

byte b = 127;

b = a + b; // error : cannot convert from int to byte

b += a; // ok

（译者注：这个地方应该表述的有误，其实无论 a+b 的值为多少，编译器都会报错，因为 a+b 操作会将 a、b 提升为 int 类型，所以将 int 类型赋值给 byte 就会编译出错）

short s1= 1; s1 = s1 + 1; 该段代码是否有错,有的话怎么改?

有错误，short类型在进行运算时会自动提升为int类型，也就是说s1+1的运算结果是int类型。

int 和 Integer 有什么区别

答： Java 提供两种不同的类型：引用类型和原始类型（或内置类型）。Int是java的原始数据类型，Integer是java为int提供的封装类。Java为每个原始类型提供了封装类。引用类型和原始类型具有不同的特征和用法，它们包括：大小和速度问题，这种类型以哪种类型的数据结构存储，当引用类型和原始类型用作某个类的实例数据时所指定的缺省值。对象引用实例变量的缺省值为 null，而原始类型实例变量的缺省值与它们的类型有关

& 和 &&的区别

&运算符有两种用法：(1)按位与；(2)逻辑与。&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是true整个表达式的值才是true。&&之所以称为短路运算是因为，如果&&左边的表达式的值是false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是null而且不是空字符串，应当写为：username != null &&!username.equals(“”)，二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的equals比较，否则会产生NullPointerException异常。注意：逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

如何将byte转为String

可以使用 String 接收 byte[] 参数的构造器来进行转换，需要注意的点是要使用的正确的编码，否则会使用平台默认编码，这个编码可能跟原来的编码相同，也可能不同。

可以将int强转为byte类型么?会产生什么问题?

我们可以做强制转换，但是Java中int是32位的而byte是8 位的，所以,如果强制转化int类型的高24位将会被丢弃，byte 类型的范围是从-128到128

关键字

Java 有没有 goto

goto 是 Java 中的保留字，在目前版本的 Java 中没有使用。

final 有什么用?

用于修饰类、属性和方法；

- 被final修饰的类不可以被继承
- 被final修饰的方法不可以被重写
- 被final修饰的变量不可以被改变，被final修饰不可变的是变量的引用，而不是引用指向的内容，引用指向的内容是可以改变的

final有哪些用法

final也是很多面试喜欢问的地方，能回答下以下三点就不错了：

1.被final修饰的类不可以被继承

2.被final修饰的方法不可以被重写

3.被final修饰的变量不可以被改变。如果修饰引用，那么表示引用不可变，引用指向的内容可变。

4.被final修饰的方法，JVM会尝试将其内联，以提高运行效率

5.被final修饰的常量，在编译阶段会存入常量池中。

回答出编译器对final域要遵守的两个重排序规则更好：

1.在构造函数内对一个final域的写入，与随后把这个被构造对象的引用赋值给一个引用变量,这两个操作之间不能重排序。

2.初次读一个包含final域的对象的引用，与随后初次读这个final域,这两个操作之间不能重排序。

final finally finalize区别

- final可以修饰类、变量、方法，修饰类表示该类不能被继承、修饰方法表示该方法不能被重写、修饰变量表
  示该变量是一个常量不能被重新赋值。
- finally一般作用在try-catch代码块中，在处理异常的时候，通常我们将一定要执行的代码方法finally代码块
  中，表示不管是否出现异常，该代码块都会执行，一般用来存放一些关闭资源的代码。
- finalize是一个方法，属于Object类的一个方法，而Object类是所有类的父类，该方法一般由垃圾回收器来调
  用，当我们调用System.gc() 方法的时候，由垃圾回收器调用finalize()，回收垃圾，一个对象是否可回收的
  最后判断。

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

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）。

this与super的区别

- super: 它引用当前对象的直接父类中的成员（用来访问直接父类中被隐藏的父类中成员数据或函数，基类与派生类中有相同成员定义时如：super.变量名 super.成员函数据名（实参）

- this：它代表当前对象名（在程序中易产生二义性之处，应使用this来指明当前对象；如果函数的形参与类中的成员数据同名，这时需用this来指明成员变量名）
- super()和this()类似,区别是，super()在子类中调用父类的构造方法，this()在本类内调用本类的其它构造方法。
- super()和this()均需放在构造方法内第一行。
- 尽管可以用this调用一个构造器，但却不能调用两个。
- this和super不能同时出现在一个构造函数里面，因为this必然会调用其它的构造函数，其它的构造函数必然也会有super语句的存在，所以在同一个构造函数里面有相同的语句，就失去了语句的意义，编译器也不会通过。
- this()和super()都指的是对象，所以，均不可以在static环境中使用。包括：static变量,static方法，static语句块。
- 从本质上讲，this是一个指向本对象的指针, 然而super是一个Java关键字。

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。

static的独特之处

1、被static修饰的变量或者方法是独立于该类的任何对象，也就是说，这些变量和方法**不属于任何一个实例对象，而是被类的实例对象所共享。**

> 怎么理解 “被类的实例对象所共享” 这句话呢？就是说，一个类的静态成员，它是属于大伙的【大伙指的是这个类的多个对象实例，我们都知道一个类可以创建多个实例！】，所有的类对象共享的，不像成员变量是自个的【自个指的是这个类的单个实例对象】...我觉得我已经讲的很通俗了，你明白了咩?

2、在该类被第一次加载的时候，就会去加载被static修饰的部分，而且只在类第一次使用时加载并进行初始化，注意这是第一次用就要初始化，后面根据需要是可以再次赋值的。

3、static变量值在类加载的时候分配空间，以后创建类对象的时候不会重新分配。赋值的话，是可以任意赋值的！

4、被static修饰的变量或者方法是优先于对象存在的，也就是说当一个类加载完毕之后，即便没有创建对象，也可以去访问。

static应用场景

因为static是被类的实例对象所共享，因此如果**某个成员变量是被所有对象所共享的，那么这个成员变量就应该定义为静态变量。**

因此比较常见的static应用场景有：

> 1、修饰成员变量 2、修饰成员方法 3、静态代码块 4、修饰类【只能修饰内部类也就是静态内部类】 5、静态导包

static注意事项

1、静态只能访问静态。 2、非静态既可以访问非静态的，也可以访问静态的。

流程控制语句

break ,continue ,return 的区别及作用

break 跳出总上一层循环，不再执行循环(结束当前的循环体)

continue 跳出本次循环，继续执行下次循环(结束正在执行的循环 进入下一个循环条件)

return 程序返回，不再执行下面的代码(结束当前的方法 直接返回)

在 Java 中，如何跳出当前的多重嵌套循环

在Java中，要想跳出多重循环，可以在外面的循环语句前定义一个标号，然后在里层循环体的代码中使用带有标号的break 语句，即可跳出外层循环。例如：

```
public static void main(String[] args) {
    ok:
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 10; j++) {
            System.out.println("i=" + i + ",j=" + j);
            if (j == 5) {
                break ok;
            }

        }
    }
}
```

类

类与接口

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

备注：Java8中接口中引入默认方法和静态方法，以此来减少抽象类和接口之间的差异。

现在，我们可以为接口提供默认实现的方法了，并且不用强制子类来实现它。

接口和抽象类各有优缺点，在接口和抽象类的选择上，必须遵守这样一个原则：

- 行为模型应该总是通过接口而不是抽象类定义，所以通常是优先选用接口，尽量少用抽象类。
- 选择抽象类的时候通常是如下情况：需要定义子类的行为，又要为子类提供通用的功能。

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

创建一个对象用什么关键字？对象实例与对象引用有何不同?

new关键字，new创建对象实例（对象实例在堆内存中），对象引用指向对象实例（对象引用存放在栈内存中）。一个对象引用可以指向0个或1个对象（一根绳子可以不系气球，也可以系一个气球）;一个对象可以有n个引用指向它（可以用n条绳子系住一个气球）

Java中，什么是构造方法？什么是构造方法重载？什么是复制构造方法?

当新对象被创建的时候，构造方法会被调用。每一个类都有构造方法。在程序员没有给类提供构造方法的情况下，Java编译器会为这个类创建一个默认的构造方法。

Java中构造方法重载和方法重载很相似。可以为一个类创建多个构造方法。每一个构造方法必须有它自己唯一的参数列表。

Java不支持像C++中那样的复制构造方法，这个不同点是因为如果你不自己写构造方法的情况下，Java不会创建默认的复制构造方法。

Java支持多继承么?

Java中类不支持多继承，只支持单继承（即一个类只有一个父类）。 但是java中的接口支持多继承，，即一个子接口可以有多个父接口。（接口的作用是用来扩展对象的功能，一个子接口继承多个父接口，说明子接口扩展了多个功能，当类实现接口时，类就扩展了相应的功能）。

变量与方法

成员变量与局部变量的区别有哪些

变量：在程序执行的过程中，在某个范围内其值可以发生改变的量。从本质上讲，变量其实是内存中的一小块区域

成员变量：方法外部，类内部定义的变量

局部变量：类的方法中的变量。

成员变量和局部变量的区别

作用域

成员变量：针对整个类有效。
局部变量：只在某个范围内有效。(一般指的就是方法,语句体内)

存储位置

成员变量：随着对象的创建而存在，随着对象的消失而消失，存储在堆内存中。
局部变量：在方法被调用，或者语句被执行的时候存在，存储在栈内存中。当方法调用完，或者语句结束后，就自动释放。

生命周期

成员变量：随着对象的创建而存在，随着对象的消失而消失
局部变量：当方法调用完，或者语句结束后，就自动释放。

初始值

成员变量：有默认初始值。

局部变量：没有默认初始值，使用前必须赋值。

使用原则

在使用变量时需要遵循的原则为：就近原则
首先在局部范围找，有就使用；接着在成员位置找。

在Java中定义一个不做事且没有参数的构造方法的作用

Java程序在执行子类的构造方法之前，如果没有用super()来调用父类特定的构造方法，则会调用父类中“没有参数的构造方法”。因此，如果父类中只定义了有参数的构造方法，而在子类的构造方法中又没有用super()来调用父类中特定的构造方法，则编译时将发生错误，因为Java程序在父类中找不到没有参数的构造方法可供执行。解决办法是在父类里加上一个不做事且没有参数的构造方法。

在调用子类构造方法之前会先调用父类没有参数的构造方法，其目的是?

帮助子类做初始化工作。

一个类的构造方法的作用是什么？若一个类没有声明构造方法，改程序能正确执行吗？为什么?

主要作用是完成对类对象的初始化工作。可以执行。因为一个类即使没有声明构造方法也会有默认的不带参数的构造方法。

构造方法有哪些特性?

名字与类名相同；

没有返回值，但不能用void声明构造函数；

生成类的对象时自动执行，无需调用。

静态变量和实例变量区别

静态变量： 静态变量由于不属于任何实例对象，属于类的，所以在内存中只会有一份，在类的加载过程中，JVM只为静态变量分配一次内存空间。

实例变量： 每次创建对象，都会为每个对象分配成员变量内存空间，实例变量是属于实例对象的，在内存中，创建几次对象，就有几份成员变量。

静态变量与普通变量区别

static变量也称作静态变量，静态变量和非静态变量的区别是：静态变量被所有的对象所共享，在内存中只有一个副本，它当且仅当在类初次加载时会被初始化。而非静态变量是对象所拥有的，在创建对象的时候被初始化，存在多个副本，各个对象拥有的副本互不影响。

还有一点就是static成员变量的初始化顺序按照定义的顺序进行初始化。

静态方法和实例方法有何不同?

静态方法和实例方法的区别主要体现在两个方面：

1. 在外部调用静态方法时，可以使用"类名.方法名"的方式，也可以使用"对象名.方法名"的方式。而实例方法只有后面这种方式。也就是说，调用静态方法可以无需创建对象。
2. 静态方法在访问本类的成员时，只允许访问静态成员（即静态成员变量和静态方法），而不允许访问实例成员变量和实例方法；实例方法则无此限制

在一个静态方法内调用一个非静态成员为什么是非法的?

由于静态方法可以不通过对象进行调用，因此在静态方法里，不能调用其他非静态变量，也不可以访问非静态变量成员。

什么是方法的返回值？返回值的作用是什么?

方法的返回值是指我们获取到的某个方法体中的代码执行后产生的结果！（前提是该方法可能产生结果）。返回值的作用:接收出结果，使得它可以用于其他的操作!

Object根类

equals()和hashcode()的联系

hashCode()是Object类的一个方法，返回一个哈希值。如果两个对象根据equal()方法比较相等，那么调用这两个对象中任意一个对象的hashCode()方法必须产生相同的哈希值。

如果两个对象根据eqaul()方法比较不相等，那么产生的哈希值不一定相等(碰撞的情况下还是会相等的。)

a.hashCode()有什么用?

将对象放入到集合中时，首先判断要放入对象的hashcode是否已经在集合中存在，不存在则直接放入集合。如果hashcode相等，然后通过equal()方法判断要放入对象与集合中的任意对象是否相等：如果equal()判断不相等，直接将该元素放入集合中，否则不放入。

有没有可能两个不相等的对象有相同的hashcode

有可能，两个不相等的对象可能会有相同的 hashcode 值，这就是为什么在 hashmap 中会有冲突。如果两个对象相等，必须有相同的hashcode 值，反之不成立。

可以在hashcode中使用随机数字吗?

不行，因为同一对象的 hashcode 值必须是相同的

a==b与a.equals(b)有什么区别

如果a 和b 都是对象，则 a==b 是比较两个对象的引用，只有当 a 和 b 指向的是堆中的同一个对象才会返回 true，而 a.equals(b) 是进行逻辑比较，所以通常需要重写该方法来提供逻辑一致性的比较。例如，String 类重写 equals() 方法，所以可以用于两个不同对象，但是包含的字母相同的比较。

基本类型比较用==，比较的是他们的值。默认下，对象用==比较时，比较的是内存地址，如果需要比较对象内容，需要重写equal方法。

为什么要转换?

如果你在 Java5 下进行过编程的话，你一定不会陌生这一点，你不能直接地向集合(Collection)中放入原始类型值，因为集合只接收对象。

通常这种情况下你的做法是，将这些原始类型的值转换成对象，然后将这些转换的对象放入集合中。使用 Integer、Double、Boolean 等这些类，我们可以将原始类型值转换成对应的对象，但是从某些程度可能使得代码不是那么简洁精炼。

为了让代码简练，Java5 引入了具有在原始类型和对象类型自动转换的装箱和拆箱机制。

但是自动装箱和拆箱并非完美，在使用时需要有一些注意事项，如果没有搞明白自动装箱和拆箱，可能会引起难以察觉的 Bug 。

内部类

什么是内部类?

在Java中，可以将一个类的定义放在另外一个类的定义内部，这就是**内部类**。内部类本身就是类的一个属性，与其他属性定义方式一致。

内部类的分类有哪些

内部类可以分为四种：**成员内部类、局部内部类、匿名内部类和静态内部类。**

静态内部类

定义在类内部的静态类，就是静态内部类。

```
public class Outer {

    private static int radius = 1;

    static class StaticInner {
        public void visit() {
            System.out.println("visit outer static  variable:" + radius);
        }
    }
}
```

静态内部类可以访问外部类所有的静态变量，而不可访问外部类的非静态变量；静态内部类的创建方式，`new 外部类.静态内部类()`，如下：

```
Outer.StaticInner inner = new Outer.StaticInner();
inner.visit();
```

成员内部类

定义在类内部，成员位置上的非静态类，就是成员内部类。

```
public class Outer {

    private static  int radius = 1;
    private int count =2;

     class Inner {
        public void visit() {
            System.out.println("visit outer static  variable:" + radius);
            System.out.println("visit outer   variable:" + count);
        }
    }
}
```

成员内部类可以访问外部类所有的变量和方法，包括静态和非静态，私有和公有。成员内部类依赖于外部类的实例，它的创建方式`外部类实例.new 内部类()`，如下：

```
Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();
inner.visit();
```

局部内部类

定义在方法中的内部类，就是局部内部类。

```
public class Outer {

    private  int out_a = 1;
    private static int STATIC_b = 2;
```

```
    public void testFunctionClass(){
        int inner_c =3;
        class Inner {
            private void fun(){
                System.out.println(out_a);
                System.out.println(STATIC_b);
                System.out.println(inner_c);
            }
        }
        Inner  inner = new Inner();
        inner.fun();
    }
    public static void testStaticFunctionClass(){
        int d =3;
        class Inner {
            private void fun(){
                // System.out.println(out_a); 编译错误，定义在静态方法中的局部类不可以访问外部类的实例变量
                System.out.println(STATIC_b);
                System.out.println(d);
            }
        }
        Inner  inner = new Inner();
        inner.fun();
    }
}
```

定义在实例方法中的局部类可以访问外部类的所有变量和方法，定义在静态方法中的局部类只能访问外部类的静态变量和方法。局部内部类的创建方式，在对应方法内，`new 内部类()`，如下：

```
 public static void testStaticFunctionClass(){
    class Inner {
    }
    Inner  inner = new Inner();
 }
```

匿名内部类

匿名内部类就是没有名字的内部类，日常开发中使用的比较多。

```
public class Outer {

    private void test(final int i) {
        new Service() {
            public void method() {
                for (int j = 0; j < i; j++) {
                    System.out.println("匿名内部类" );
                }
            }
        }.method();
    }
 }
 //匿名内部类必须继承或实现一个已有的接口
 interface Service{
    void method();
```

```
}
```

除了没有名字，匿名内部类还有以下特点：

- 匿名内部类必须继承一个抽象类或者实现一个接口。
- 匿名内部类不能定义任何静态成员和静态方法。
- 当所在的方法的形参需要被匿名内部类使用时，必须声明为 final。
- 匿名内部类不能是抽象的，它必须要实现继承的类或者实现的接口的所有抽象方法。

匿名内部类创建方式：

```
new 类/接口{
  //匿名内部类实现部分
}
```

内部类的优点

我们为什么要使用内部类呢？因为它有以下优点：

- 一个内部类对象可以访问创建它的外部类对象的内容，包括私有数据！
- 内部类不为同一包的其他类所见，具有很好的封装性；
- 内部类有效实现了“多重继承”，优化 java 单继承的缺陷。
- 匿名内部类可以很方便的定义回调。

内部类有哪些应用场景

1. 一些多算法场合
2. 解决一些非面向对象的语句块。
3. 适当使用内部类，使得代码更加灵活和富有扩展性。
4. 当某个类除了它的外部类，不再被其他的类使用时。

局部内部类和匿名内部类访问局部变量的时候，为什么变量必须要加上final?

局部内部类和匿名内部类访问局部变量的时候，为什么变量必须要加上final呢？它内部原理是什么呢？

先看这段代码：

```
public class Outer {

    void outMethod(){
        final int a =10;
        class Inner {
            void innerMethod(){
                System.out.println(a);
            }

        }
    }
}
```

以上例子，为什么要加final呢？是因为**生命周期不一致**， 局部变量直接存储在栈中，当方法执行结束后，非final的局部变量就被销毁。而局部内部类对局部变量的引用依然存在，如果局部内部类要调用局部变量时，就会出错。加了final，可以确保局部内部类使用的变量与外层的局部变量区分开，解决了这个问题。

内部类相关，看程序说出运行结果

```
public class Outer {
    private int age = 12;

    class Inner {
        private int age = 13;
        public void print() {
            int age = 14;
            System.out.println("局部变量：" + age);
            System.out.println("内部类变量：" + this.age);
            System.out.println("外部类变量：" + Outer.this.age);
        }
    }

    public static void main(String[] args) {
        Outer.Inner in = new Outer().new Inner();
        in.print();
    }

}
```

运行结果：

```
局部变量：14
内部类变量：13
外部类变量：12
```

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。

对象相等判断

== 和 equals 的区别是什么

== : 它的作用是判断两个对象的地址是不是相等。即，判断两个对象是不是同一个对象。(基本数据类型 == 比较的是值，引用数据类型 == 比较的是内存地址)

equals() : 它的作用也是判断两个对象是否相等。但它一般有两种使用情况：

情况1：类没有覆盖 equals() 方法。则通过 equals() 比较该类的两个对象时，等价于通过“==”比较这两个对象。

情况2：类覆盖了 equals() 方法。一般，我们都覆盖 equals() 方法来两个对象的内容相等；若它们的内容相等，则返回 true (即，认为这两个对象相等)。

举个例子：

```
public class test1 {
    public static void main(String[] args) {
        String a = new String("ab"); // a 为一个引用
        String b = new String("ab"); // b为另一个引用,对象的内容一样
        String aa = "ab"; // 放在常量池中
        String bb = "ab"; // 从常量池中查找
        if (aa == bb) // true
            System.out.println("aa==bb");
        if (a == b) // false，非同一对象
            System.out.println("a==b");
        if (a.equals(b)) // true
            System.out.println("aEQb");
        if (42 == 42.0) { // true
            System.out.println("true");
        }
    }
}
```

说明：

- String中的equals方法是被重写过的，因为object的equals方法是比较的对象的内存地址，而String的equals方法比较的是对象的值。
- 当创建String类型的对象时，虚拟机会在常量池中查找有没有已经存在的值和要创建的值相同的对象，如果有就把它赋给当前引用。如果没有就在常量池中重新创建一个String对象。

hashCode 与 equals (重要)

HashSet如何检查重复

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

hashCode和equals方法的关系

面试官可能会问你：“你重写过 hashcode 和 equals 么，为什么重写equals时必须重写hashCode方法？”

hashCode()介绍

hashCode() 的作用是获取哈希码，也称为散列码；它实际上是返回一个int整数。这个哈希码的作用是确定该对象在哈希表中的索引位置。hashCode() 定义在JDK的Object.java中，这就意味着Java中的任何类都包含有hashCode()函数。

散列表存储的是键值对(key-value)，它的特点是：能根据“键”快速的检索出对应的“值”。这其中就利用到了散列码！（可以快速找到所需要的对象）

为什么要有 hashCode

我们以“HashSet 如何检查重复”为例子来说明为什么要有 hashCode:

当你把对象加入 HashSet 时，HashSet 会先计算对象的 hashcode 值来判断对象加入的位置，同时也会与其他已经加入的对象的 hashcode 值作比较，如果没有相符的hashcode，HashSet会假设对象没有重复出现。但是如果发现有相同 hashcode 值的对象，这时会调用 equals()方法来检查 hashcode 相等的对象是否真的相同。如果两者相同，HashSet 就不会让其加入操作成功。如果不同的话，就会重新

散列到其他位置。（摘自我的Java启蒙书《Head first java》第二版）。这样我们就大大减少了 equals 的次数，相应就大大提高了执行速度。

hashCode()与equals()的相关规定

如果两个对象相等，则hashcode一定也是相同的

两个对象相等，对两个对象分别调用equals方法都返回true

两个对象有相同的hashcode值，它们也不一定是相等的

因此，equals 方法被覆盖过，则 hashCode 方法也必须被覆盖

hashCode() 的默认行为是对堆上的对象产生独特值。如果没有重写 hashCode()，则该 class 的两个对象无论如何都不会相等（即使这两个对象指向相同的数据）

对象的相等与指向他们的引用相等，两者有什么不同?

对象的相等 比的是内存中存放的内容是否相等而 引用相等 比较的是他们指向的内存地址是否相等。

值传递

当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递

是值传递。Java 语言的方法调用只支持参数的值传递。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的属性可以在被调用过程中被改变，但对对象引用的改变是不会影响到调用者的

为什么 Java 中只有值传递

首先回顾一下在程序设计语言中有关将参数传递给方法（或函数）的一些专业术语。**按值调用(call by value)表示方法接收的是调用者提供的值，而按引用调用（call by reference)表示方法接收的是调用者提供的变量地址。一个方法可以修改传递引用所对应的变量值，而不能修改传递值调用所对应的变量值。** 它用来描述各种程序设计语言（不只是Java)中方法参数传递方式。

Java程序设计语言总是采用按值调用。也就是说，方法得到的是所有参数值的一个拷贝，也就是说，方法不能修改传递给它的任何参数变量的内容。

下面通过 3 个例子来给大家说明

example 1

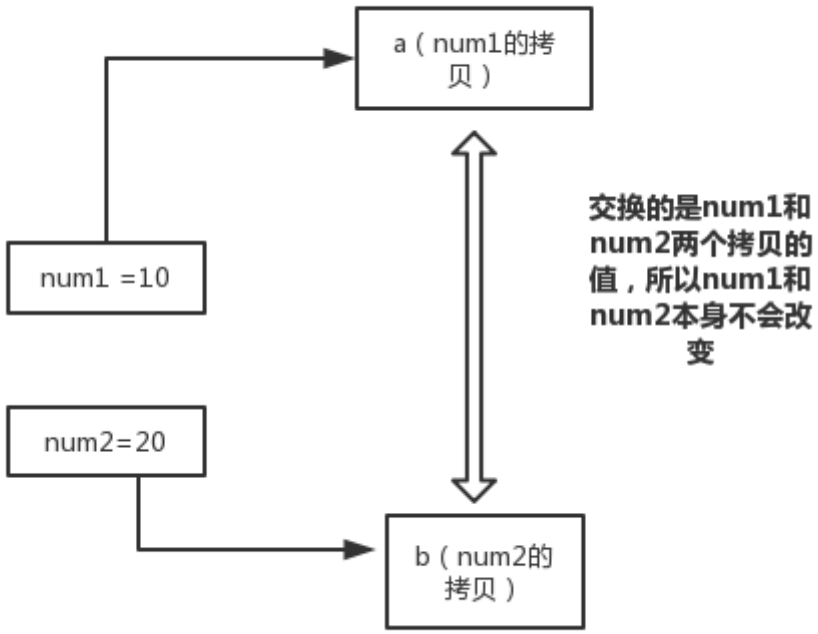
```
public static void main(String[] args) {
    int num1 = 10;
    int num2 = 20;

    swap(num1, num2);

    System.out.println("num1 = " + num1);
    System.out.println("num2 = " + num2);
}

public static void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
```

```
    System.out.println("a = " + a);
    System.out.println("b = " + b);
}
```

结果:

```
a = 20
b = 10
num1 = 10
num2 = 20
```

解析:



在swap方法中，a、b的值进行交换，并不会影响到 num1、num2。因为，a、b中的值，只是从 num1、num2 的复制过来的。也就是说，a、b相当于num1、num2 的副本，副本的内容无论怎么修改，都不会影响到原件本身。

通过上面例子，我们已经知道了一个方法不能修改一个基本数据类型的参数，而对象引用作为参数就不一样，请看 example2.

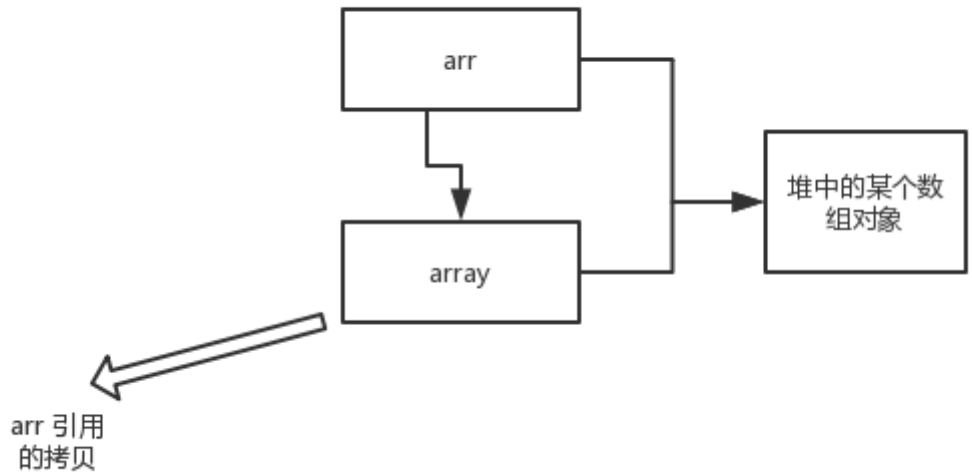
example 2

```
    public static void main(String[] args) {
        int[] arr = { 1, 2, 3, 4, 5 };
        System.out.println(arr[0]);
        change(arr);
        System.out.println(arr[0]);
    }

    public static void change(int[] array) {
        // 将数组的第一个元素变为0
        array[0] = 0;
    }
```

结果:

```
1
0
```

解析:



array 被初始化 arr 的拷贝也就是一个对象的引用，也就是说 array 和 arr 指向的时同一个数组对象。因此，外部对引用对象的改变会反映到所对应的对象上。

通过 example2 我们已经看到，实现一个改变对象参数状态的方法并不是一件难事。理由很简单，方法得到的是对象引用的拷贝，对象引用及其他的拷贝同时引用同一个对象。

很多程序设计语言（特别是，C++和Pascal)提供了两种参数传递的方式：值调用和引用调用。有些程序员（甚至本书的作者）认为Java程序设计语言对对象采用的是引用调用，实际上，这种理解是不对的。由于这种误解具有一定的普遍性，所以下面给出一个反例来详细地阐述一下这个问题。

example 3

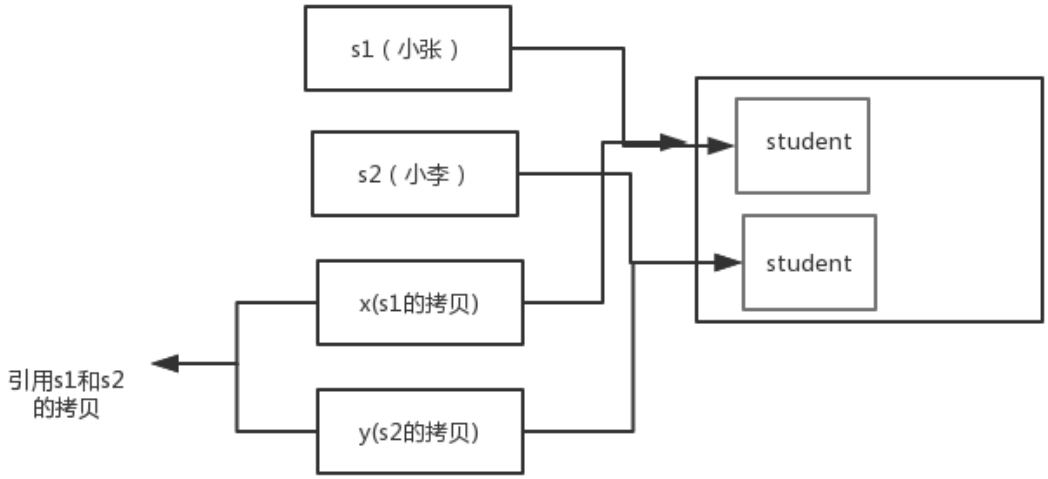
```
public class Test {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Student s1 = new Student("小张");
        Student s2 = new Student("小李");
        Test.swap(s1, s2);
        System.out.println("s1:" + s1.getName());
        System.out.println("s2:" + s2.getName());
    }

    public static void swap(Student x, Student y) {
        Student temp = x;
        x = y;
        y = temp;
        System.out.println("x:" + x.getName());
        System.out.println("y:" + y.getName());
    }
}
```

结果:

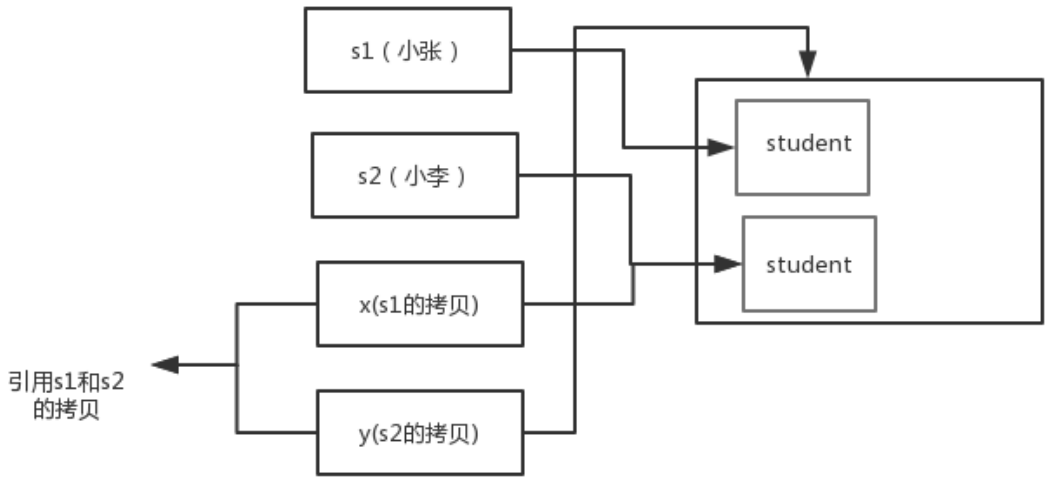
```
x:小李
y:小张
s1:小张
s2:小李
```

解析:

交换之前:



交换之后:



通过上面两张图可以很清晰的看出：**方法并没有改变存储在变量 s1 和 s2 中的对象引用。swap方法的参数x和y被初始化为两个对象引用的拷贝，这个方法交换的是这两个拷贝**

总结

Java程序设计语言对对象采用的不是引用调用，实际上，对象引用是按值传递的。

下面再总结一下Java中方法参数的使用情况:

- 一个方法不能修改一个基本数据类型的参数（即数值型或布尔型》
- 一个方法可以改变一个对象参数的状态。

- 一个方法不能让对象参数引用一个新的对象。

值传递和引用传递有什么区别

值传递：指的是在方法调用时，传递的参数是按值的拷贝传递，传递的是值的拷贝，也就是说传递后就互不相关了。

引用传递：指的是在方法调用时，传递的参数是按引用进行传递，其实传递的引用的地址，也就是变量所对应的内存空间的地址。传递的是值的引用，也就是说传递前和传递后都指向同一个引用（也就是同一个内存空间）。

Java包

JDK 中常用的包有哪些

- java.lang：这个是系统的基础类；
- java.io：这里面是所有输入输出有关的类，比如文件操作等；
- java.nio：为了完善 io 包中的功能，提高 io 包中性能而写的一个新包；
- java.net：这里面是与网络有关的类；
- java.util：这个是系统辅助类，特别是集合类；
- java.sql：这个是数据库操作的类。

import java和javax有什么区别

刚开始的时候 JavaAPI 所必需的包是 java 开头的包，javax 当时只是扩展 API 包来说使用。然而随着时间的推移，javax 逐渐的扩展成为 Java API 的组成部分。但是，将扩展从 javax 包移动到 java 包将是太麻烦了，最终会破坏一堆现有的代码。因此，最终决定 javax 包将成为标准API的一部分。

所以，实际上java和javax没有区别。这都是一个名字。

IO流

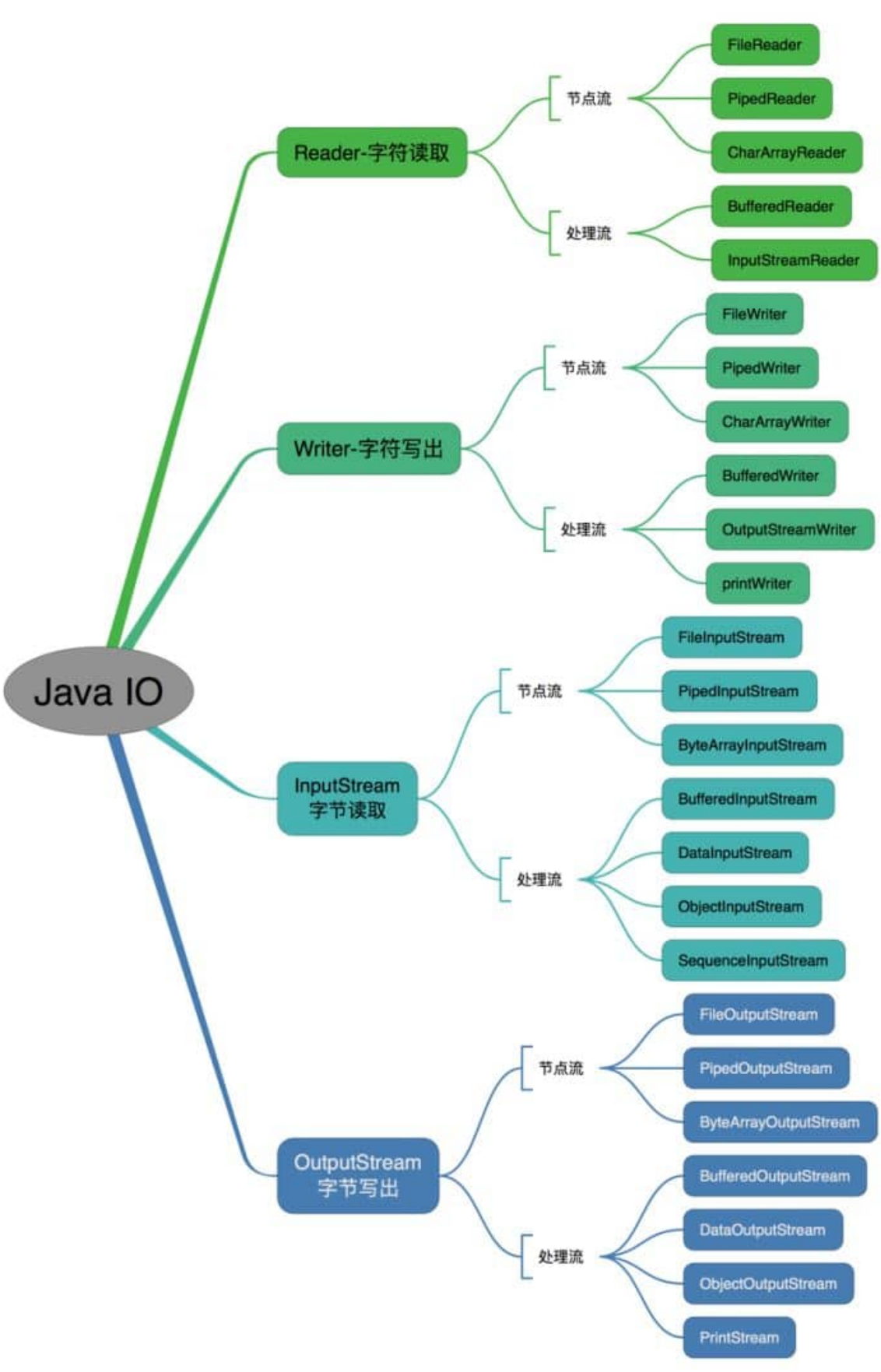
java 中 IO 流分为几种?

- 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。

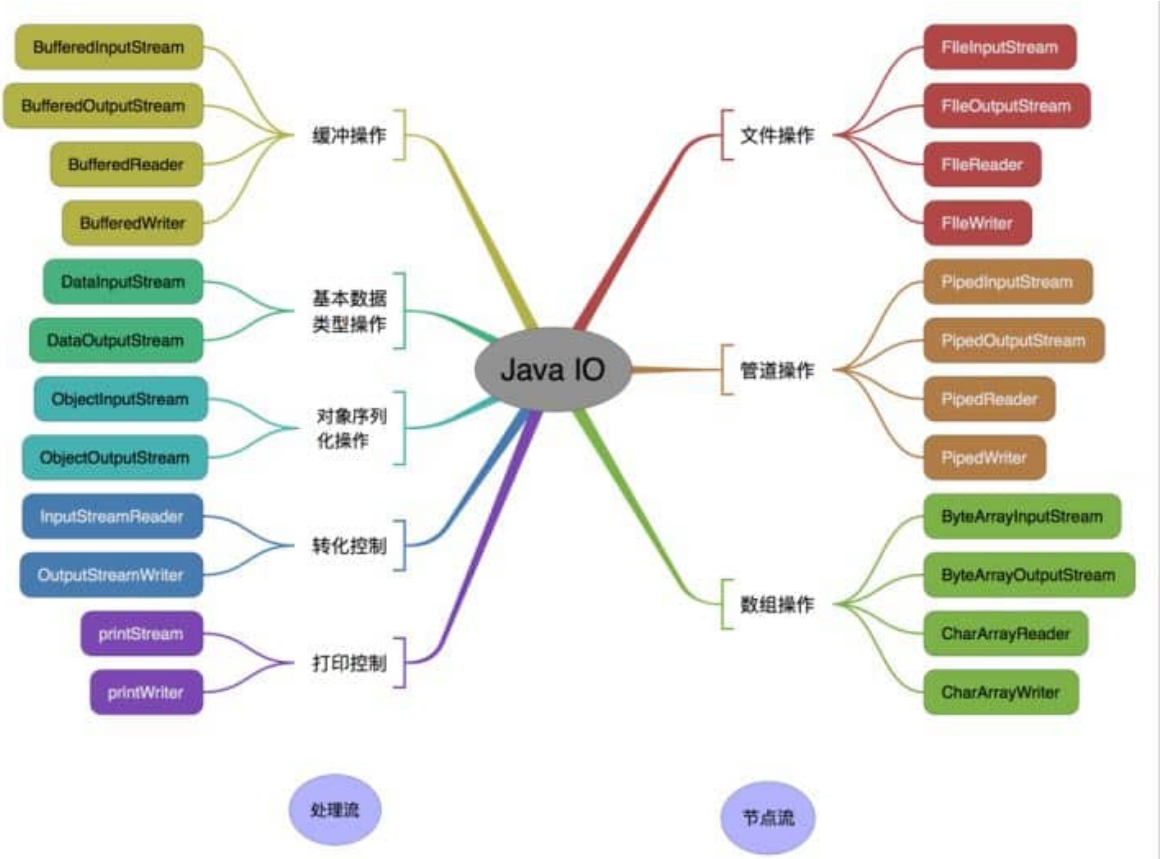
Java Io流共涉及40多个类，这些类看上去很杂乱，但实际上很有规则，而且彼此之间存在非常紧密的联系， Java I0流的40多个类都是从如下4个抽象类基类中派生出来的。

- 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：


Java IO
Reader-字符读取
节点流
FileReader
PipedReader
CharArrayReader
处理流
BufferedReader
InputStreamReader
Writer-字符写出
节点流
FileWriter
PipedWriter
CharArrayWriter
处理流
BufferedWriter
OutputStreamWriter
printWriter
InputStream
字节读取
节点流
FileInputStream
PipedInputStream
ByteArrayInputStream
处理流
BufferedInputStream
DataInputStream
ObjectInputStream
SequenceInputStream
OutputStream
字节写出
节点流
FileOutputStream
PipedOutputStream
ByteArrayOutputStream
处理流
BufferedOutputStream
DataOutputStream
ObjectOutputStream
PrintStream

按操作对象分类结构图：



BIO,NIO,AIO 有什么区别?

简答

- BIO：Block IO 同步阻塞式 IO，就是我们平常使用的传统 IO，它的特点是模式简单使用方便，并发处理能力低。
- NIO：Non IO 同步非阻塞 IO，是传统 IO 的升级，客户端和服务器端通过 Channel（通道）通讯，实现了多路复用。
- AIO：Asynchronous IO 是 NIO 的升级，也叫 NIO2，实现了异步非堵塞 IO ，异步 IO 的操作基于事件和回调机制。

详细回答

- **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。

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。

反射与动态代理

反射

什么是反射机制?

简单说，反射机制值得是程序在运行时能够获取自身的信息。在java中，只要给定类的名字，那么就可以通过反射机制来获得类的所有信息。

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

静态编译和动态编译

- **静态编译：**在编译时确定类型，绑定对象
- **动态编译：**运行时确定类型，绑定对象

反射机制优缺点

- **优点：** 运行期类型的判断，动态加载类，提高代码灵活度。
- **缺点：** 性能瓶颈：反射相当于一系列解释操作，通知 JVM 要做的事情，性能比直接的java代码要慢很多。

反射机制的应用场景有哪些?

反射是框架设计的灵魂。

在我们平时的项目开发过程中，基本上很少会直接使用到反射机制，但这不能说明反射机制没有用，实际上有很多设计、开发都与反射机制有关，例如模块化的开发，通过反射去调用对应的字节码；动态代理设计模式也采用了反射机制，还有我们日常使用的 Spring / Hibernate 等框架也大量使用到了反射机制。

举例：①我们在使用JDBC连接数据库时使用Class.forName()通过反射加载数据库的驱动程序；②Spring框架也用到很多反射机制，最经典的就是xml的配置模式。Spring 通过 XML 配置模式装载 Bean 的过程：1) 将程序内所有 XML 或 Properties 配置文件加载入内存中; 2)Java类里面解析xml或properties里面的内容，得到对应实体类的字节码字符串以及相关的属性信息; 3)使用反射机制，根据这个字符串获得某个类的Class实例; 4)动态配置实例的属性

Java获取反射的三种方法

1.通过new对象实现反射机制 2.通过路径实现反射机制 3.通过类名实现反射机制

```
public class Student {
    private int id;
    String name;
    protected boolean sex;
    public float score;
}
123456
public class Get {
    //获取反射机制三种方式
    public static void main(String[] args) throws ClassNotFoundException {
        //方式一(通过建立对象)
        Student stu = new Student();
        Class classobj1 = stu.getClass();
        System.out.println(classobj1.getName());
        //方式二（所在通过路径-相对路径）
        Class classobj2 = Class.forName("fanshe.Student");
        System.out.println(classobj2.getName());
        //方式三（通过类名）
        Class classobj3 = Student.class;
        System.out.println(classobj3.getName());
    }
}
```

哪里用到反射机制?

jdbc中有一行代码：Class.forName('com.MySQL.jdbc.Driver.class').newInstance();那个时候只知道生成驱动对象实例，后来才知道，这就是反射，现在

很多框架都用到反射机制，hibernate，struts都是用反射机制实现的。

反射机制的优缺点?

静态编译：在编译时确定类型，绑定对象，即通过

动态编译：运行时确定类型，绑定对象。动态编译最大限度的发挥了java的灵活性，体现了多态的应用，有利于降低类之间的耦合性。

一句话，反射机制的优点就是可以实现动态创建对象和编译，体现出很大的灵活性，特别是在J2EE的开发中

它的灵活性就表现的十分明显。比如，一个大型的软件，不可能一次就把把它设计的很完美，当这个程序编

译后，发布了，当发现需要更新某些功能时，我们不可能要用户把以前的卸载，再重新安装新的版本，假如

这样的话，这个软件肯定是没有多少人用的。采用静态的话，需要把整个程序重新编译一次才可以实现功能

的更新，而采用反射机制的话，它就可以不用卸载，只需要在运行时才动态的创建和编译，就可以实现该功

能。

它的缺点是对性能有影响。使用反射基本上是一种解释操作，我们可以告诉JVM，我们希望做什么并且它

满足我们的要求。这类操作总是慢于只直接执行相同的操作。

反射的功能：

在运行时构造一个类的对象。

判断一个类所具有的成员变量和方法。

调用一个对象的方法。

生成动态代理。

反射的应用很多，很多框架都有用到：

反射的用途：

Spring 框架的 IoC 基于反射创建对象和设置依赖属性。

Spring MVC 的请求调用对应方法，也是通过反射。

JDBC 的 Class#forName(String className) 方法，也是使用反射。

反射中，Class.forName 和 ClassLoader 区别

java中class.forName()和classLoader都可用来对类进行加载。

class.forName()前者除了将类的.class文件加载到jvm中之外，还会对类进行解释，执行类中的static块。

而classLoader只干一件事情，就是将.class文件加载到jvm中，不会执行static中的内容,只有在newInstance才会去执行static块。

动态代理

什么是动态代理

代理类在程序运行时创建的代理方式被成为 动态代理。 也就是说，这种情况下，代理类并不是在Java代码中定义的，而是在运行时根据我们在Java代码中的“指示”动态生成的。相比于静态代理， 动态代理的优势在于可以很方便的对代理类的函数进行统一的处理，而不用修改每个代理类的函数。

Java动态代理的两种实现方法

jdk动态代理是由java内部的反射机制来实现的，cglib动态代理底层则是借助asm来实现的。

总的来说，反射机制在生成类的过程中比较高效，而asm在生成类之后的相关执行过程中比较高效（可以通过将asm生成的类进行缓存，这样解决asm生成类过程低效问题）。还有一点必须注意：jdk动态代理的应用前提，必须是目标类基于统一的接口。如果没有上述前提，jdk动态代理不能应用。由此可以看出，jdk动态代理有一定的局限性，cglib这种第三方类库实现的动态代理应用更加广泛，且在效率上更有优势。。

jdk动态代理是jdk原生就支持的一种代理方式，它的实现原理，就是通过让target类和代理类实现同一接口，代理类持有target对象，来达到方法拦截的作用，这样通过接口的方式有两个弊端，一个是必须保证target类有接口，第二个是如果想要对target类的方法进行代理拦截，那么就要保证这些方法都要在接口中声明，实现上略微有点限制。

Cglib是一个优秀的动态代理框架，它的底层使用ASM在内存中动态的生成被代理类的子类，使用CGLIB即使代理类没有实现任何接口也可以实现动态代理功能。CGLIB具有简单易用，它的运行速度要远远快于JDK的Proxy动态代理：

为什么要用动态代理

他可以在不修改别代理对象代码的基础上，通过扩展代理类，进行一些功能的附加与增强

静态代理与动态代理的区别

动态代理使我们免于去重写接口中的方法，而着重于去扩展相应的功能或是方法的增强，与静态代理相比简单了不少，减少了项目中的业务量

动态代理机制

Proxy这个类的作用就是用来动态创建一个代理对象的类。每一个动态代理类都必须要实现InvocationHandler这个接口，并且每个代理类的实例都关联到了一个handler，当我们通过代理对象调用一个方法的时候，这个方法的调用就会被转发为由InvocationHandler这个接口的 invoke 方法来进行调用。

序列化

什么是 Java 序列化?

序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。

可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。

序列化是为了解决在对对象流进行读写操作时所引发的问题。

反序列化的过程，则是和序列化相反的过程。

另外，我们不能将序列化局限在 Java 对象转换成二进制数组，例如说，我们将一个 Java 对象，转换成 JSON 字符串，或者 XML 字符串，这也可以理解为是序列化。

如何实现 Java 序列化?

如下的方式，就是 Java 内置的序列化方案，实际场景下，我们可以自定义序列化的方案，例如说 Google Protobuf 。

将需要被序列化的类，实现 Serializable 接口，该接口没有需要实现的方法，implements Serializable 只是为了标注该对象是可被序列化的。

序列化

然后，使用一个输出流(如：FileOutputStream)来构造一个 ObjectOutputStream(对象流)对象

接着，使用 ObjectOutputStream 对象的 #writeObject(Object obj) 方法，就可以将参数为 obj 的对象写出(即保存其状态)。

反序列化

要恢复的话则用输入流。

Java 序列话中，如果有些字段不想进行序列化怎么办?

对于不想进行序列化的变量，使用 transient 关键字修饰。

当对象被序列化时，阻止实例中那些用此关键字修饰的的变量序列化。

当对象被反序列化时，被 transient 修饰的变量值不会被持久化和恢复。

transient 只能修饰变量，不能修饰类和方法。

常用API

String相关

字符型常量和字符串常量的区别

1. 形式上: 字符常量是单引号引起的一个字符 字符串常量是双引号引起的若干个字符

2. 含义上: 字符常量相当于一个整形值(ASCII值),可以参加表达式运算 字符串常量代表一个地址值(该字符串在内存中存放位置)
3. 占内存大小 字符常量只占一个字节 字符串常量占若干个字节(至少一个字符结束标志)

什么是字符串常量池?

字符串常量池位于堆内存中，专门用来存储字符串常量，可以提高内存的使用率，避免开辟多块空间存储相同的字符串，在创建字符串时 JVM 会首先检查字符串常量池，如果该字符串已经存在池中，则返回它的引用，如果不存在，则实例化一个字符串放到池中，并返回其引用。

String 是最基本的数据类型吗

不是。Java 中的基本数据类型只有 8 个： byte、short、int、long、float、double、char、boolean；除了基本类型（primitive type），剩下的都是引用类型（referencetype），Java 5 以后引入的枚举类型也算是一种比较特殊的引用类型。

这是很基础的东西，但是很多初学者却容易忽视，Java 的 8 种基本数据类型中不包括 String，基本数据类型中用来描述文本数据的是 char，但是它只能表示单个字符，比如 'a','好' 之类的，如果要描述一段文本，就需要用多个 char 类型的变量，也就是一个 char 类型数组，比如"你好" 就是长度为2的数组 char[] chars = {'你','好'};

但是使用数组过于麻烦，所以就有了 String，String 底层就是一个 char 类型的数组，只是使用的时候开发者不需要直接操作底层数组，用更加简便的方式即可完成对字符串的使用。

String有哪些特性

- 不变性：String 是只读字符串，是一个典型的 immutable 对象，对它进行任何操作，其实都是创建一个新的对象，再把引用指向该对象。不变模式的主要作用在于当一个对象需要被多线程共享并频繁访问时，可以保证数据的一致性。
- 常量池优化：String 对象创建之后，会在字符串常量池中进行缓存，如果下次创建同样的对象时，会直接返回缓存的引用。
- final：使用 final 来定义 String 类，表示 String 类不能被继承，提高了系统的安全性。

String为什么是不可变的吗?

简单来说就是String类利用了final修饰的char类型数组存储字符，源码如下图所以：

```
/** The value is used for character storage. */
private final char value[];
```

String真的是不可变的吗?

我觉得如果别人问这个问题的话，回答不可变就可以了。 下面只是给大家看两个有代表性的例子：

1) String不可变但不代表引用不可以变

```
String str = "Hello";
str = str + " World";
System.out.println("str=" + str);
```

结果：

```
str=Hello World
```

解析：

实际上，原来String的内容是不变的，只是str由原来指向"Hello"的内存地址转为指向"Hello World"的内存地址而已，也就是说多开辟了一块内存区域给"Hello World"字符串。

2) 通过反射是可以修改所谓的“不可变”对象

```
// 创建字符串"Hello World"， 并赋给引用s
String s = "Hello World";

System.out.println("s = " + s); // Hello World

// 获取String类中的value字段
Field valueFieldOfString = String.class.getDeclaredField("value");

// 改变value属性的访问权限
valueFieldOfString.setAccessible(true);

// 获取s对象上的value属性的值
char[] value = (char[]) valueFieldOfString.get(s);

// 改变value所引用的数组中的第5个字符
value[5] = '_';

System.out.println("s = " + s); // Hello_World
```

结果：

```
s = Hello World
s = Hello_World
```

解析：

用反射可以访问私有成员， 然后反射出String对象中的value属性， 进而改变通过获得的value引用改变数组的结构。但是一般我们不会这么做，这里只是简单提一下有这个东西。

是否可以继承 String 类

String 类是 final 类，不可以被继承。

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

不一样，因为内存的分配方式不一样。String str="i"的方式，java 虚拟机会将其分配到常量池中；而 String str=new String(“i”) 则会被分到堆内存中。

String s = new String(“xyz”);创建了几个字符串对象

两个对象，一个是静态区的"xyz"，一个是用new创建在堆上的对象。

```
String str1 = "hello"; //str1指向静态区
String str2 = new String("hello");  //str2指向堆上的对象
String str3 = "hello";
String str4 = new String("hello");
System.out.println(str1.equals(str2)); //true
System.out.println(str2.equals(str4)); //true
System.out.println(str1 == str3); //true
System.out.println(str1 == str2); //false
System.out.println(str2 == str4); //false
System.out.println(str2 == "hello"); //false
str2 = str1;
System.out.println(str2 == "hello"); //true
```

如何将字符串反转?

使用 StringBuilder 或者 stringBuffer 的 reverse() 方法。

示例代码：

```
// StringBuffer reverse
StringBuffer stringBuffer = new StringBuffer();
stringBuffer. append("abcdefg");
System. out. println(stringBuffer. reverse()); // gfedcba
// StringBuilder reverse
StringBuilder stringBuilder = new StringBuilder();
stringBuilder. append("abcdefg");
System. out. println(stringBuilder. reverse()); // gfedcba
```

数组有没有 length()方法？String 有没有 length()方法

数组没有 length()方法 ，有 length 的属性。String 有 length()方法。JavaScript中，获得字符串的长度是通过 length 属性得到的，这一点容易和 Java 混淆。

String 类的常用方法都有那些?

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

在使用 HashMap 的时候，用 String 做 key 有什么好处?

HashMap 内部实现是通过 key 的 hashcode 来确定 value 的存储位置，因为字符串是不可变的，所以当创建字符串时，它的 hashcode 被缓存下来，不需要再次计算，所以相比于其他对象更快。

String, StringBuffer和StringBuilder区别

String是字符串常量，final修饰：StringBuffer字符串变量(线程安全)；

StringBuilder 字符串变量(线程不安全)。

String和StringBuffer

String和StringBuffer主要区别是性能：String是不可变对象，每次对String类型进行操作都等同于产生了一个新的String对象，然后指向新的String对象。所以尽量不在对String进行大量的拼接操作，否则会产生很多临时对象，导致GC开始工作，影响系统性能。

StringBuffer是对对象本身操作，而不是产生新的对象，因此在有大量拼接的情况下，我们建议使用StringBuffer。

但是需要注意现在JVM会对String拼接做一定的优化：

String s="This is only "+"simple"+"test"会被虚拟机直接优化成String s="This is only simple test"，此时就不存在拼接过程。

StringBuffer和StringBuilder

StringBuffer是线程安全的可变字符串，其内部实现是可变数组。StringBuilder是jdk 1.5新增的，其功能和StringBuffer类似，但是非线程安全。因此，在没有多线程问题的前提下，使用StringBuilder会取得更好的性能。

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

Date相关

包装类相关

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
}
```

异常处理

error和exception有什么区别

error表示系统级的错误，是java运行环境内部错误或者硬件问题，不能指望程序来处理这样的问题，除了退出运行外别无选择，它是Java虚拟机抛出的。

exception 表示程序需要捕捉、需要处理的异常，是由与程序设计的不完善而出现的问题，程序必须处理的问题

运行时异常和一般异常有何不同

Java提供了两类主要的异常：runtimeException和checkedException

一般异常（checkedException）主要是指IO异常、SQL异常等。对于这种异常，JVM要求我们必须对其进行cathc处理，所以，面对这种异常，不管我们是否愿意，都是要写一大堆的catch块去处理可能出现的异常。

运行时异常（runtimeException）我们一般不处理，当出现这类异常的时候程序会由虚拟机接管。比如，我们从来没有去处理过NullPointerException，而且这个异常还是最常见的异常之一。

出现运行时异常的时候，程序会将异常一直向上抛，一直抛到遇到处理代码，如果没有catch块进行处理，到了最上层，如果是多线程就有Thread.run()抛出，如果不是多线程那么就由main.run()抛出。抛出之后，如果是线程，那么该线程也就终止了，如果是主程序，那么该程序也就终止了。

其实运行时异常的也是继承自Exception，也可以用catch块对其处理，只是我们一般不处理罢了，也就是说，如果不对运行时异常进行catch处理，那么结果不是线程退出就是主程序终止。

如果不想终止，那么我们就必须捕获所有可能出现的运行时异常。如果程序中出现了异常数据，但是它不影响下面的程序执行，那么我们就该在catch块里面将异常数据舍弃，然后记录日志。如果，它影响到了下面的程序运行，那么还是程序退出比较好些。

Java中异常处理机制的原理

Java通过面向对象的方式对异常进行处理，Java把异常按照不同的类型进行分类，并提供了良好的接口。当一个方法出现异常后就会抛出一个异常对象，该对象中包含有异常信息，调用这个对象的方法可以捕获到这个异常并对异常进行处理。Java的异常处理是通过5个关键词来实现的：try catch throw throws 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（throws），我们可以通过它的类型来捕捉它，或最后由缺省处理器来处理它（finally）。

try：用来指定一块预防所有异常的程序

catch：紧跟在try后面，用来捕获异常

throw：用来明确的抛出一个异常

throws：用来标明一个成员函数可能抛出的各种异常

finally：确保一段代码无论发生什么异常都会被执行的一段代码。

你平时在项目中是怎样对异常进行处理的。

（1）尽量避免出现runtimeException 。例如对于可能出现空指针的代码，带使用对象之前一定要判断一下该对象是否为空，必要的时候对runtimeException

也进行try catch处理。

（2）进行try catch处理的时候要在catch代码块中对异常信息进行记录，通过调用异常类的相关方法获取到异常的相关信息，返回到web端，不仅要给用户良好

的用户体验，也要能帮助程序员良好的定位异常出现的位置及原因。例如，以前做的一个项目，程序遇到异常页面会显示一个图片告诉用户哪些操作导致程序出现

了什么异常，同时图片上有一个按钮用来点击展示异常的详细信息给程序员看的。

final、finally、finalize的区别

（1）、final用于声明变量、方法和类的，分别表示变量值不可变，方法不可覆盖，类不可以继承

（2）、finally是异常处理中的一个关键字，表示finally{}里面的代码一定要执行

（3）、finalize是Object类的一个方法，在垃圾回收的时候会调用被回收对象的此方法。

try()里面有一个return语句，那么后面的finally{}里面的code会不会被执行，什么时候执行，是在return前还是return后?

你曾经自定义实现过异常吗？怎么写的?

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。

异常处理的时候，finally代码块的重要性是什么?

无论是否抛出异常，finally代码块总是会被执行。就算是没有catch语句同时又抛出异常的情况下，finally代码块仍然会被执行。最后要说的是，finally代码块主要用来释放资源，比如：I/O缓冲区，数据库连接。

异常的使用的注意地方?

不要将异常处理用于正常的控制流（设计良好的 API 不应该强迫它的调用者为了正常的控制流而使用异常）。

对可以恢复的情况使用受检异常，对编程错误使用运行时异常。

避免不必要的使用受检异常（可以通过一些状态检测手段来避免异常的发生）。

优先使用标准的异常。

每个方法抛出的异常都要有文档。

保持异常的原子性

不要在 catch 中忽略掉捕获到的异常。

请列出 5 个运行时异常?

NullPointerException

IndexOutOfBoundsException

ClassCastException

ArrayStoreException 当你试图将错误类型的对象存储到一个对象数组时抛出的异常

BufferOverflowException 写入的长度超出了允许的长度

硬核推荐：尼恩Java硬核架构班

又名疯狂创客圈社群 VIP

详情：

https://www.cnblogs.com/crazymakercircle/p/9904544.html

架构班（社群 VIP）的起源：

最初的视频，主要是给读者加餐。很多的读者，需要一些高质量的实操、理论视频，所以，我就围绕书，和底层，做了几个实操、理论视频，然后效果还不错，后面就做成迭代模式了。

架构班（社群 VIP）的功能：

提供高质量实操项目整刀真枪的架构指导、快速提升大家的:

- 开发水平
- 设计水平
- 架构水平

弥补业务中 CRUD 开发短板，帮助大家尽早脱离具备 3 高能力，掌握：

- 高性能
- 高并发
- 高可用

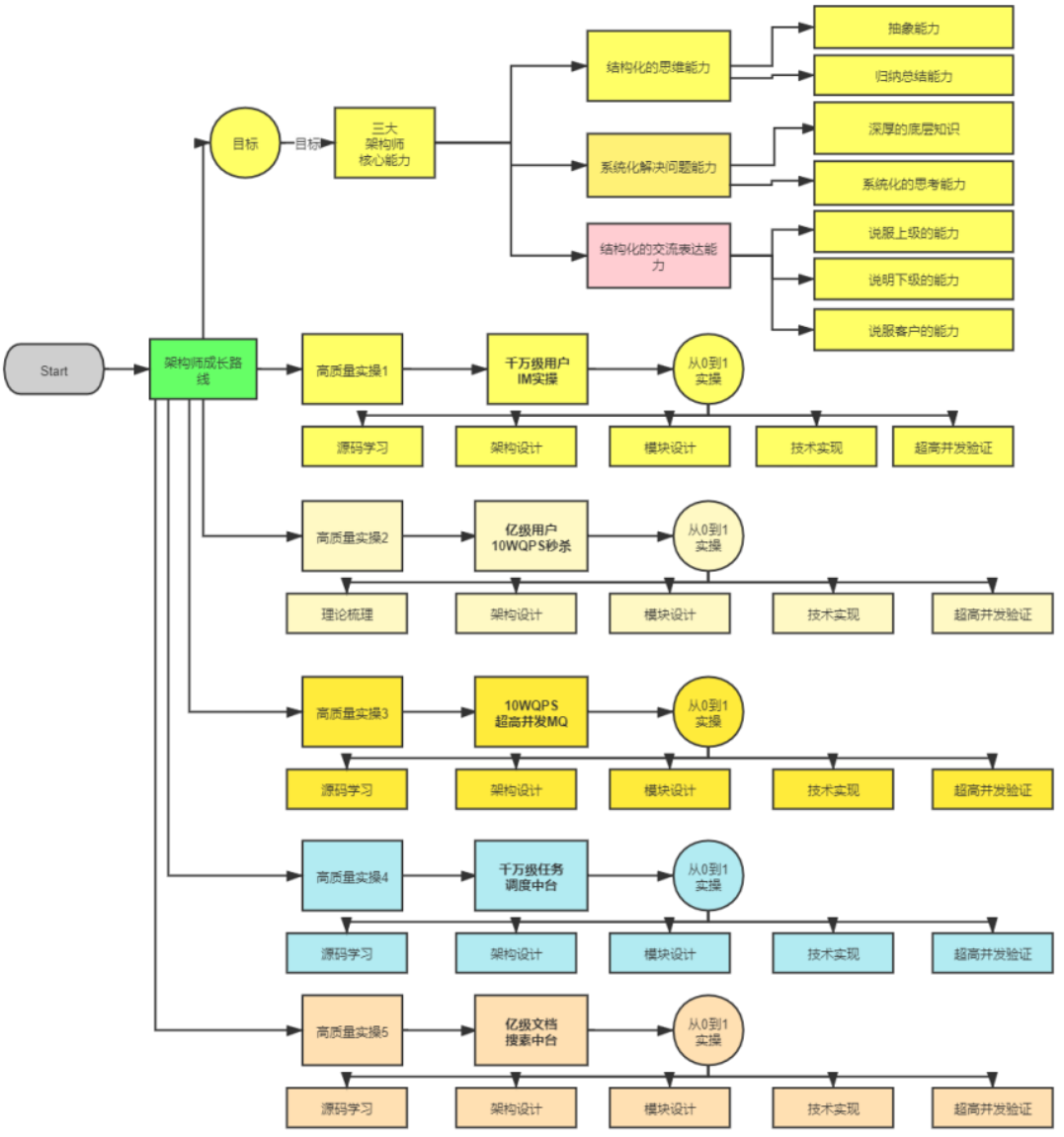
作为一个高质量的架构师成长、人脉社群，把所有的卷王聚焦起来，一起卷：

- 卷高并发实操
- 卷底层原理
- 卷架构理论、架构哲学
- 最终成为顶级架构师，实现人生理想，走向人生巅峰

架构班（社群 VIP）的目的：

- 高质量的实操，大大提升简历的含金量，吸引力，增强面试的召唤率
- 为大家提供九阳真经、葵花宝典，快速提升水平
- 进大厂、拿高薪
- 一路陪伴，提供助学视频和指导，辅导大家成为架构师
- 自学为主，和其他卷王一起，卷高并发实操，卷底层原理、卷大厂面试题，争取狠卷 3 月成高手，狠卷 3 年成为顶级架构师

N 个超高并发实操项目：简历压轴、个顶个精彩



【样章】第 17 章:横扫全网 Rocketmq 视频第 2 部曲: 工业级 rocketmq 高可用(HA)底层原理和实操

工业级 rocketmq 高可用底层原理，包含：消息消费、同步消息、异步消息、单向消息等不同消息的底层原理和源码实现；消息队列非常底层的主从复制、高可用、同步刷盘、异步刷盘等底层原理。

工业级 rocketmq 高可用底层原理和搭建实操，包含：高可用集群的搭建。

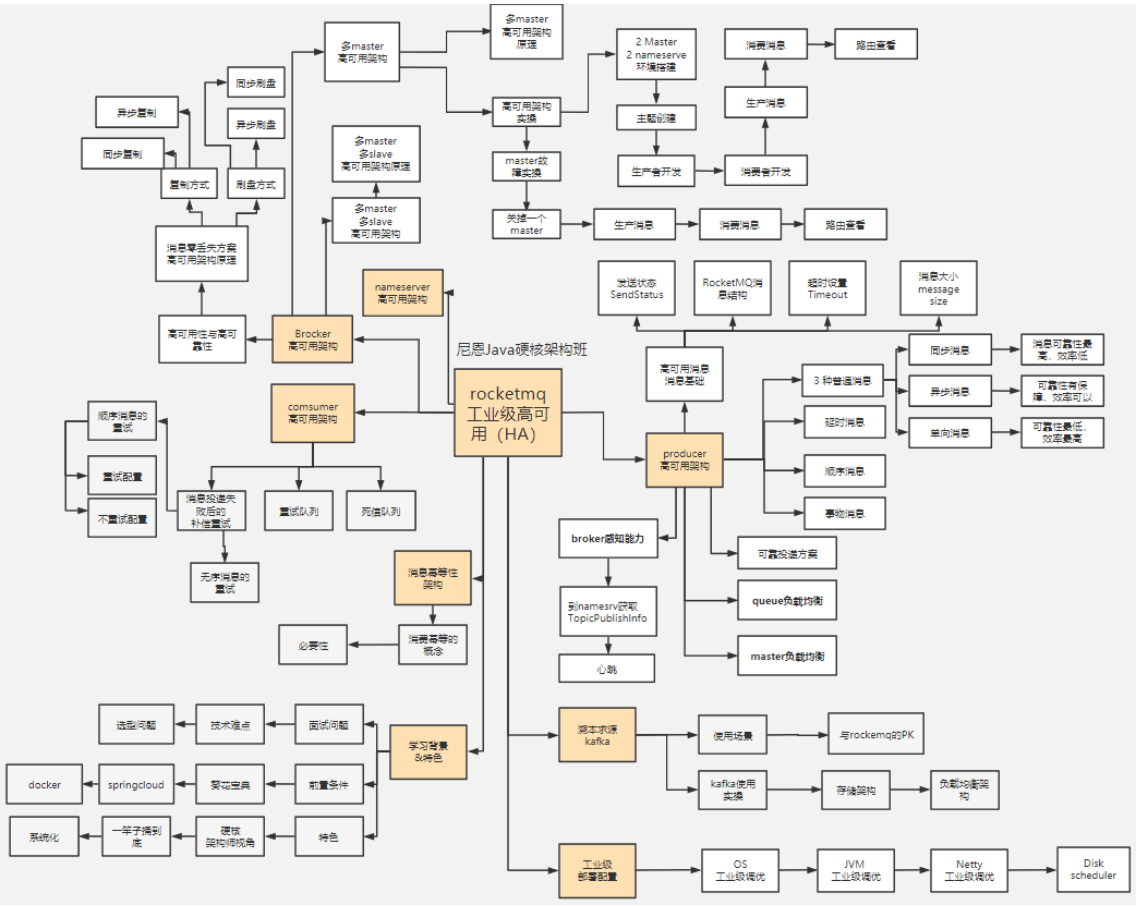
解决以下难题：

1、技术难题：RocketMQ 如何最大限度的保证消息不丢失的呢？RocketMQ 消息如何做到高可靠投递？

2、技术难题：基于消息的分布式事务，核心原理不理解

3、选型难题： kafka or rocketmq ，该娶谁？

下图链接：https://www.processon.com/view/6178e8ae0e3e7416bde9da19



成功案例：2 年翻 3 倍，35 岁卷王成功转型为架构师

详情：http://topcoder.cloud/forum.php?mod=forumdisplay&fid=43&page=1

最新 最后发表 热门 精华

- 成功案例：[1057号卷王] 3年小伙拿到外企offer，薪酬涨了200% — 卷王1号 超级版主 前天 17:41
- 成功案例：[645号卷王] 4年经验卷王逆袭，被毕业后，反涨24W — 卷王1号 超级版主 2022-9-21
- 成功案例：[878号卷王] 小伙8年经验，年薪60W — 卷王1号 超级版主 2022-8-13
- 年薪70W案例：通过尼恩的指导，小伙伴年薪从40W涨到70W — 卷王1号 超级版主 2022-2-11
- 成功案例：[493号卷王] 5年小伙拿满意offer，就业寒冬季逆涨30% — 卷王1号 超级版主 前天 17:43
- 成功案例：[250号卷王] 就业极寒时代，收offer 涨25% — 卷王1号 超级版主 前天 17:38
- 成功案例：[612号卷王] 就业极寒时代，从外包到自研 — 卷王1号 超级版主 前天 17:15
- 成功案例：[913号卷王] 热烈祝贺6年经验卷王，年薪40W — 卷王1号 超级版主 2022-9-21
- 成功案例：[959号卷王] 4年经验卷王，喜获百度、Boss直聘等N个优质offer，最高涨100% — 卷王1号 超级版主 2022-9-21
- 成功案例：[529号卷王] 5年经验卷王喜收2大offer，最高涨5K — 卷王1号 超级版主 2022-9-21
- 成功案例：[811号卷王] 热烈祝贺7年经验卷王，薪酬涨30% — 卷王1号 超级版主 2022-9-21
- 成功案例：[287号卷王] 不惧大寒潮，卷王逆市收4 offer，涨30%，可喜可贺 — 卷王1号 超级版主 2022-5-30
- 成功案例：[1002号卷王] 5月份"被毕业"，改简历后，斩获顶级央企Offer，涨薪7000+ — 卷王1号 超级版主 2022-7-5
- 成功案例：[7号卷王] 热烈祝贺小伙伴涨薪120% — 卷王1号 超级版主 2022-8-13
- 成功案例：[134号卷王] 大三小伙卷1年，斩获顶级央企Offer，成功逆袭 — 卷王1号 超级版主 2022-7-6
- 成功案例：[1008号卷王] 5年经验卷王收42W offer，月涨8000，可喜可贺 — 卷王1号 超级版主 2022-5-30
- 成功案例：[453号卷王] 非全日制 6年卷王喜提3 offer，年薪30W，可喜可贺 — 卷王1号 超级版主 2022-5-21
- 成功案例：[924号卷王] 6年卷王喜提4 offer，最高涨薪9000，可喜可贺 — 卷王1号 超级版主 2022-5-21
- 成功案例：[15号卷王] 4年卷王入职 微软，涨薪50%，可喜可贺 — 卷王1号 超级版主 2022-5-12
- 成功案例：[527号卷王] 4年卷王喜提2 offer，涨薪50%，可喜可贺 — 卷王1号 超级版主 2022-5-13
- 成功案例：[788号卷王] 3年卷王喜提优质Offer，涨薪60% — 卷王1号 超级版主 2022-5-11
- 成功案例：热烈祝贺：非全日制卷王，喜提2个心仪offer，面3家过2家 — 卷王1号 超级版主 2022-4-21

最新 最后发表 热门 精华

- 成功案例：[693号卷王] 二线城市6年卷王喜提4大优质Offer，含央企offer，最高薪酬35W — 卷王1号 超级版主 2022-4-16
- 成功案例：[85号卷王] 双非2本小伙，春招大捷，喜提9个offer，最高薪酬近30万 — 卷王1号 超级版主 2022-4-14
- 成功案例：[741号卷王] 卷王逆袭！6年小伙从很少面试机会到搞定35K*14薪Offer — 卷王1号 超级版主 2022-4-12
- 成功案例：[642号卷王] 热烈祝贺，6年卷王喜提优质国企offer — 卷王1号 超级版主 2022-4-7
- 成功案例：[796号卷王] 热烈祝贺，36岁卷王喜提52万优质offer — 卷王1号 超级版主 2022-3-25
- 成功案例：[15号卷王] 小伙卷1年，涨薪9K+，喜收ebay等多个优质offer — 卷王1号 超级版主 2022-3-24
- 成功案例：[821号卷王] 小伙狠卷3个月，喜提10多个offer — 卷王1号 超级版主 2022-3-21
- 成功案例：[736号卷王] 3年半经验收22k offer，但是小伙志存高远，冲击25k+ — 卷王1号 超级版主 2022-3-20
- 成功案例：热烈祝贺1群小卷王offer拿到手软，甚至拒了阿里offer — 卷王1号 超级版主 2022-3-16
- 简历案例：简历一改，腾讯的邀请就来了！热烈祝贺，小伙收到一大堆面试邀请 — 卷王1号 超级版主 2022-3-10
- 成功案例：祝贺我圈两大超级卷王，一个过了阿里HR面，一个过了阿里2面 — 卷王1号 超级版主 2022-3-10
- 成功案例：小伙伴php转Java，卷1.5年Java，涨薪50%，喜收多个优质offer — 卷王1号 超级版主 2022-3-10
- 成功案例：4年小伙狠卷半年，拿到 移动、京东 两大顶级offer — 尼恩 超级版主 2022-3-5
- 成功案例：[267号卷王] 助力3年经验卷王，拿到蜂巢的17k x 14薪的offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[143号卷王] 二本院校00后卷神，毕业没到一年跳到宇节，年薪45W — 卷王1号 超级版主 2022-2-27
- 成功案例：[494号卷王] 尼恩分布式事务助力卷王拿到 中信银行offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[76号卷王] 2线城市卷王，狠卷1.5年，喜收22K offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[429号卷王] 小伙伴在社群卷5个月，涨8k+ — 卷王1号 超级版主 2022-2-27
- 成功案例：[154号卷王] 双非学校毕业卷王，连拿 京东到家&滴滴 两个大厂Offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[232号卷王] 涨薪10K，继续卷向食物链顶端 — 卷王1号 超级版主 2022-2-27
- 成功案例：狠卷1年技术，喜收 腾讯、阿里、微软三大Offer，最高年薪56W — 卷王1号 超级版主 2022-2-27
- 成功案例：[449号卷王] 应届毕业卷王喜收 滴滴offer，年薪33W — 卷王1号 超级版主 2022-2-27

- 成功案例：[732号卷王] 尼恩助力3年经验卷王收获 京东offer，年薪35W — 卷王1号 超级版主 2022-2-27
- 成功案例：[558号卷王] 2年经验卷王，喜收 网易和阿里子公司两个优质offer — 卷王1号 超级版主 2022-2-27
- 成功案例：[569号卷王] 双非应届生卷王，喜收字节跳动实习offer — 卷王1号 超级版主 2022-2-25
- 成功案例：[420号卷王] 狠卷1年，卷王涨薪80%，涨薪12000元！ — 卷王1号 超级版主 2022-2-25
- 成功案例：[76号卷王] 通过尼恩1年半的指导，专科学历小伙伴从0.8K涨到22K — 卷王1号 超级版主 2022-2-10

- 成功案例：[551号卷王] 小伙伴学完后，成功进入大厂，并且推荐自己的朋友加VIP学习 — 卷王1号 超级版主 2022-2-10
- 成功案例：[214号卷王] 助力2年经验卷王，成功拿到17K月薪 — 卷王1号 超级版主 2022-2-10
- 成功案例：[92号卷王] 课程实操助力社群小伙伴喜收 喜马拉雅Offer — 卷王1号 超级版主 2022-2-10
- 成功案例：社群卷王小伙伴成功过了滴滴三面 获滴滴Offer — 卷王1号 超级版主 2022-2-10
- [612号卷王]滴滴小伙伴，蹲点考察半年，觉得靠谱后加入 疯狂创客圈 — 卷王1号 超级版主 2022-2-10

简历优化后的成功涨薪案例（VIP 含免费简历优化）

修改简历找尼恩（资深简历优化专家）

- 如果面试表达不好，尼恩会提供 简历优化指导
- 如果项目没有亮点，尼恩会提供 项目亮点指导
- 如果面试表达不好，尼恩会提供 面试表达指导

作为 40 岁老架构师，尼恩长期承担技术面试官的角色：

- **从业以来，“阅历”无数，对简历有着点石成金、改头换面、脱胎换骨**的指导能力。
- **尼恩指导过刚刚就业的小白，也指导过 P8 级的老专家，都指导他们上岸。**

如何联系尼恩。尼恩微信，请参考下面的地址：

语雀：https://www.yuque.com/crazymakercircle/gkkw8s/khigna
码云：https://gitee.com/crazymaker/SimpleCrayIM/blob/master/疯狂创客圈总目录.md