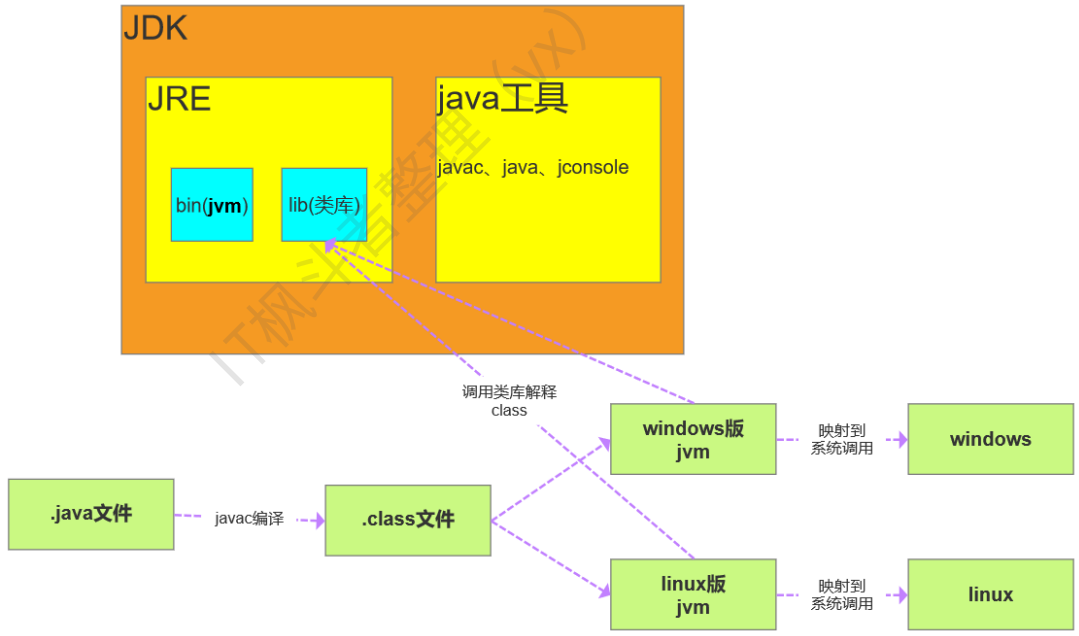
JAVA基础面试题

面向对象

- **什么是面向对象?**
- 对比面向过程，是两种不同的处理问题的角度面向过程更注重事情的每一个步骤及顺序，面向对象更注重事情有哪些参与者（对象）、及各自需要做什么
- **面向对象三大特性**
- **封装**：封装的意义，在于明确标识出允许外部使用的所有成员函数和数据项，内部细节对外部调用透明，外部调用无需修改或者关心内部实现
- **继承**：继承基类的方法，并做出自己的改变和/或扩展，子类共性的方法或者属性直接使用父类的，而不需要自己再定义，只需扩展自己个性化的
- **多态**：基于对象所属类的不同，外部对同一个方法的调用，实际执行的逻辑不同。 继承，方法重写，父类引用指向子类对象

JDK JRE JVM

- JDK： Java Develpment Kit java 开发工具
- JRE： Java Runtime Environment java运行时环境
- JVM： java Virtual Machine java 虚拟机
-



==和equals比较

- ==对比的是栈中的值，基本数据类型是变量值，引用类型是堆中内存对象的地址
- equals：是比较两个值的内容

hashCode与equals

- 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）

final

- 修饰类：表示类不可被继承 。
- 修饰方法：表示方法不可被子类覆盖，但是可以重载 。
- 修饰变量：表示变量一旦被赋值就不可以更改它的值。

String、StringBuffer、StringBuilder

- String是final修饰的，不可变，每次操作都会产生新的String对象
- StringBuffer和StringBuilder都是在原对象上操作
- StringBuffer是线程安全的，StringBuilder线程不安全的
- StringBuffer方法都是synchronized修饰的
- 性能：StringBuilder > StringBuffer > String

重载和重写的区别

- **重载：** 发生在同一个类中，方法名必须相同，参数类型不同、个数不同、顺序不同，方法返回值和访问修饰符可以不同，发生在编译时。
- **重写：** 发生在父子类中，方法名、参数列表必须相同，返回值范围小于等于父类，抛出的异常范围小于等于父类，访问修饰符范围大于等于父类；如果父类方法访问修饰符为private则子类就不能重写该方法。

接口和抽象类的区别

- 抽象类可以存在普通成员函数，而接口中只能存在public abstract 方法。
- 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是public static final类型的。
- 抽象类只能继承一个，接口可以实现多个。
- 接口的设计目的，是对类的行为进行约束（更准确的说是一种"有"约束，因为接口不能规定类不可以有什么行为），也就是提供一种机制，可以强制要求不同的类具有相同的行为。它只约束了行为的有无，但不对如何实现行为进行限制。
- 而抽象类的设计目的，是代码复用。当不同的类具有某些相同的行为(记为行为集合A)，且其中一部分行 为的实现方式一致时A的非真子集，记为B），可以让这些类都派生于一个抽象类。在这个抽象类中实现了B，避免让所有的子类来实现B，这就达到了代码复用的目的。而A减B的部分，留给各个子类自己实现。正是因为A-B在这里没有实现，所以抽象类不允许实例化出来（否则当调用到A-B时，无法执 行）。
- 抽象类是对类本质的抽象，表达的是 is a 的关系，比如： BMW is a Car 。抽象类包含并实现子类的通用特性，将子类存在差异化的特性进行抽象，交由子类去实现。
- 而接口是对行为的抽象，表达的是 like a 的关系。比如： Bird like a Aircraft （像飞行器一样可以飞），但其本质上 is a Bird 。接口的核心是定义行为，即实现类可以做什么，至于实现类主体是谁、是如何实现的，接口并不关心。
- 使用场景：当你关注一个事物的本质的时候，用抽象类；当你关注一个操作的时候，用接口。

什么是字节码？采用字节码的好处是什么？

- **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的机器。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（即扩展名为 .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。这也就是解释了Java的编译与解释并存的特点。
- 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。
- **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。

八种基本数据类型的大小，以及他们的封装类

基本类型	大小（字节）	默认值	封装类
byte	1	(byte)0	Byte
short	2	(short)0	Short
int	4	0	Integer
long	8	0L	Long
float	4	0.0f	Float
double	8	0.0d	Double
boolean	-	false	Boolean
char	2	\u0000(null)	Character

JAVA 四种引用类型

- **强引用**
- 在 Java 中最常见的就是强引用，把一个对象赋给一个引用变量，这个引用变量就是一个强引用。当一个对象被强引用变量引用时，它处于可达状态，它是不可能被垃圾回收机制回收的，即使该对象以后永远都不会被用到 JVM 也不会回收。因此强引用是造成 Java 内存泄漏的主要原因之一。
- **软引用**
- 软引用需要用 SoftReference 类来实现，对于只有软引用的对象来说，当系统内存足够时它不会被回收，当系统内存空间不足时它会被回收。软引用通常用在对内存敏感的程序中。
- **弱引用**
- 弱引用需要用 WeakReference 类来实现，它比软引用的生存期更短，对于只有弱引用的对象来说，只要垃圾回收机制一运行，不管 JVM 的内存空间是否足够，总会回收该对象占用的内存。
- **虚引用**
- 虚引用需要 PhantomReference 类来实现，它不能单独使用，必须和引用队列联合使用。虚引用的主要作用是跟踪对象被垃圾回收的状态。

使用泛型的好处?

- 以集合来举例，使用泛型的好处是我们不必因为添加元素类型的不同而定义不同类型的集合，如整型集合类，浮点型集合类，字符串集合类，我们可以定义一个集合来存放整型、浮点型，字符串型数据，而这并不是最重要的，因为我们只要把底层存储设置了Object即可，添加的数据全部都可向上转型为Object。 更重要的是我们可以通过规则按照自己的想法控制存储的数据类型。

深拷贝和浅拷贝的区别是什么?

- 浅拷贝:被复制对象的所有变量都含有与原来的对象相同的值,而所有的对其他对象的引用仍然指 向原来的对象.换言之,浅拷贝仅仅复制所考虑的对象,而不复制它所引用的对象.
- 深拷贝:被复制对象的所有变量都含有与原来的对象相同的值.而那些引用其他对象的变量将指向 被复制过的新对象.而不再是原有的那些被引用的对象.换言之.深拷贝把要复制的对象所引用的 对象都复制了一遍.

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

- 不管有木有出现异常，finally块中代码都会执行;
- 当try和catch中有return时，finally仍然会执行;
- 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，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;
- 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。

Excption与Error包结构

- Java可抛出(Throwable)的结构分为三种类型：被检查的异常(CheckedException)，运行时异常(RuntimeException)，错误(Error)。

Java IO与NIO的区别

- NIO即New IO，这个库是在JDK1.4中才引入的。NIO和IO有相同的作用和目的，但实现方式不同，NIO主要用到的是块，所以NIO的效率要比IO高很多。在Java API中提供了两套NIO，一套是针对标准输入输出NIO，另一套就是网络编程NIO

java反射的作用于原理

- 反射机制是在运行时，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意个对象，都能够调用它的任意一个方法。在java中，只要给定类的名字，就可以通过反射机制来获得类的所有信息。

反射的实现方式:

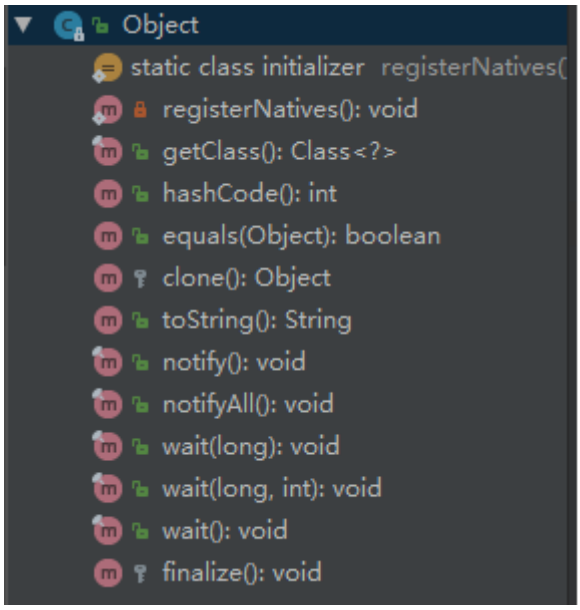
- Class.forName(“类的路径”);
- 类名.class
- 对象名.getClass()
- 基本类型的包装类，可以调用包装类的Type属性来获得该包装类的Class对象

反射机制的优缺点:

- **优点**
- 能够运行时动态获取类的实例，提高灵活性;
- 与动态编译结合
- **缺点:**
- 使用反射
- 性能较低，需要解析字节码，将内存中的对象进行解析。
- 解决方案:

- 通过setAccessible(true) 关闭JDK的安全检查来提升反射速度；
- 多次创建一个类的实例时，有缓存会快很多
- ReflflectASM工具类，通过字节码生成的方式加快反射速度 2）相对不安全，破坏了封装性（因为通过反射可以获得私有方法和属性）

Object 有哪些常用方法？大致说一下每个方法的含义

- 

获取一个类Class对象的方式有哪些

- 通过类对象的 getClass() 方法获取，细心点的都知道，这个 getClass 是 Object 类里面的方法。
- 通过类的静态成员表示，每个类都有隐含的静态成员 class。
- 通过 Class 类的静态方法 forName() 方法获取。