Java【基础】面试题

以下面试题，基于网络整理，和自己编辑。具体参考的文章，会在文末给出所有的链接。

如果胖友有自己的疑问，欢迎在星球提问，我们一起整理吊吊的 Java【基础】面试题的大保健。

而题目的难度，尽量按照从容易到困难的顺序，逐步下去。

考虑到 Java 涉及的知识点非常非常非常之多，所以我们会分成五大篇来分享，分别是：

- Java【基础】
  > 本文。分不到其它类别里的，都会放到本文。
- Java【集合】
- Java【并发】
- Java【网络】
  > 因为 Java 网络编程大多使用 Netty ，所以面试题统一整理到 《精尽 Netty 面试题》 中，避免重复。
- Java【虚拟机】

什么是面向对象？

面向对象是一种思想，世间万物都可以看做一个对象，这里只讨论面向对象编程（OOP），Java 是一个支持并发、基于类和面向对象的计算机编程语言。面向对象软件开发具有以下优点：

- 代码开发模块化，更易维护和修改。
- 代码复用性强。
- 增强代码的可靠性和灵活性。
- 增加代码的可读性。

请说说面向对象的特征？

四点：**封装**、**继承**、**多态**、**抽象**。

> ：这个题目，能说出上述四点就好了。

1）封装

封装，给对象提供了隐藏内部特性和行为的能力。对象提供一些能被其他对象访问的方法来改变它内部的数据。在 Java 当中，有 4 种修饰符：`default`、`public`、`private` 和 `protected` 。每一种修饰符给其他的位于同一个包或者不同包下面对象赋予了不同的访问权限。

下面列出了使用封装的一些好处：

- 通过隐藏对象的属性来保护对象内部的状态。
- 提高了代码的可用性和可维护性，因为对象的行为可以被单独的改变或者是扩展。
- 禁止对象之间的不良交互提高模块化。

2）继承

继承，给对象提供了从基类获取字段和方法的能力。继承提供了代码的重用行，也可以在不修改类的情况下给现存的类添加新特性。

3）多态

多态，是编程语言给不同的底层数据类型做相同的接口展示的一种能力。一个多态类型上的操作，可以应用到其他类型的值上面。

4）抽象

抽象，是把想法从具体的实例中分离出来的步骤，因此，要根据他们的功能而不是实现细节来创建类。

Java 支持创建只暴漏接口而不包含方法实现的抽象的类。这种抽象技术的主要目的是把类的行为和实现细节分离开。

📣 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。

📣 重载和重写的区别?

1）重写 `override`

- 方法名、参数、返回值相同。
- 子类方法不能缩小父类方法的访问权限。
- 子类方法不能抛出比父类方法更多的异常(但子类方法可以不抛出异常)。
- 存在于父类和子类之间。
- 方法被定义为 `final` 不能被重写。

2）重载 `overload`

- 参数类型、个数、顺序至少有一个不相同。
- 不能重载只有返回值不同的方法名。
- 存在于父类和子类、同类中。

📣 对比图

对比图

📣 Java 中，什么是构造方法？什么是构造方法重载？什么是拷贝构造方法?

1）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。

2）构造方法重载

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。

3）拷贝构造方法

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

JDK、JRE、JVM 分别是什么关系?

JDK

JDK 即为 Java 开发工具包，包含编写 Java 程序所必须的编译、运行等开发工具以及 JRE。开发工具如：

- 用于编译 Java 程序的 javac 命令。
- 用于启动 JVM 运行 Java 程序的 Java 命令。
- 用于生成文档的 Javadoc 命令。
- 用于打包的 jar 命令等等。

> 简单说，就是 JDK 包含 JRE 包含 JVM。

JRE

JRE 即为 Java 运行环境，提供了运行 Java 应用程序所必须的软件环境，包含有 Java 虚拟机（JVM）和丰富的系统类库。系统类库即为 Java 提前封装好的功能类，只需拿来直接使用即可，可以大大的提高开发效率。

> 简单说，就是 JRE 包含 JVM。

JVM

JVM 即为 Java 虚拟机，提供了字节码文件(`.class`)的运行环境支持。

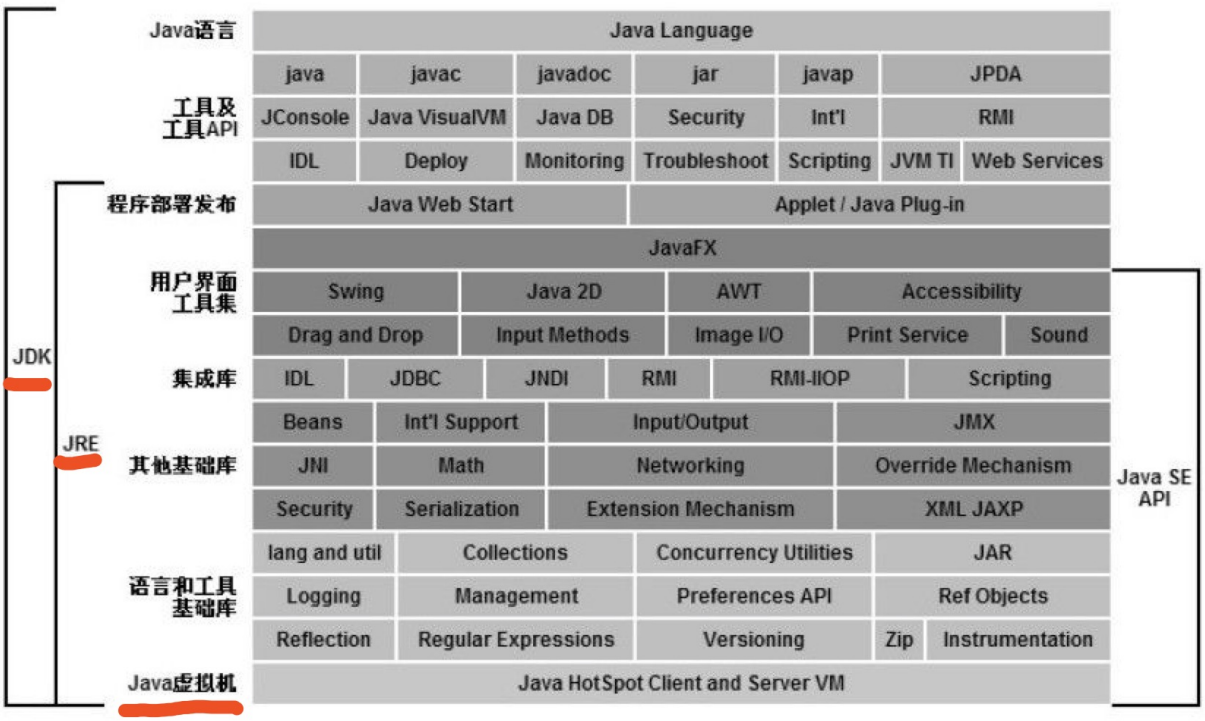


图　1-2　Java技术体系所包含的内容[2]

📣 为什么 Java 被称作是“平台无关的编程语言”?

Java 虚拟机是一个可以执行 Java 字节码的虚拟机进程。

- Java 源文件(`.java`)被编译成能被 Java 虚拟机执行的字节码文件(`.class`)。
- Java 被设计成允许应用程序可以运行在任意的平台，而不需要程序员为每一个平台单独重写或者是重新编译。Java 虚拟机让这个变为可能，因为它知道底层硬件平台的指令长度和其他特性。

📣 JDK 各版本的新特性?

- JDK5 ~ JDK10 ，看 https://www.jianshu.com/p/37b52f1ebd4a 。
- JDK11 ，看 https://www.jianshu.com/p/81b65eded96c 。

对于大多数面试官，肯定不会问你 JDK 各版本的新特性，更多的会问 JDK8 引入了什么重要的特性？一般上，关键的回答是Lambda 表达式和集合之流式操作，然后说说你在项目中怎么使用的。

📣 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。
- 【重要】Java 有自动内存管理机制，不需要程序员手动释放无用内存。

什么是字节码？采用字节码的最大好处是什么?

📣 什么是字节码?

> 这个问题，面试官可以衍生提问，Java 是编译执行的语言，还是解释执行的语言。

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。

> 解释型语言：解释型语言，是在运行的时候将程序翻译成机器语言。解释型语言的程序不需要在运行前编译，在运行程序的时候才翻译，专门的解释器负责在每个语句执行的时候解释程序代码。这样解释型语言每执行一次就要翻译一次，效率比较低。——百度百科
>
> 例如：Python、PHP 。

Java 中的几种基本数据类型是什么？各自占用多少字节?

Java 支持的数据类型包括基本数据类型和引用类型。

基本数据类型如下：

- 整数值型：`byte`、`short`、`int`、`long`
- 字符型：`char`
- 浮点类型：`float`、`double`
- 布尔型：`boolean`
- 整数型：默认 `int` 型，小数默认是 `double` 型。Float 和 Long 类型的必须加后缀。比如：`float f = 100f`。

引用类型声明的变量是指该变量在内存中实际存储的是一个引用地址，实体在堆中。

- 引用类型包括类、接口、数组等。
- 特别注意，String 是引用类型不是基本类型。

📣 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。

📣 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。

📣 char 型变量中能不能存贮一个中文汉字？为什么?

- 在 C 语言中，char 类型占 1 个字节，而汉字占 2 个字节，所以不能存储。
- 在 Java 语言中，char 类型占 2 个字节，而且 Java 默认采用 Unicode 编码，一个 Unicode 码是 16 位，所以一个 Unicode 码占两个字节，Java 中无论汉字还是英文字母，都是用 Unicode 编码来表示的。所以，在 Java 中，char 类型变量可以存储一个中文汉字。

String、StringBuffer、StringBuilder 的区别?

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。

- 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。

  > 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/StringBuilder 类，表示的字符串对象可以直接进行修改。StringBuilder 是 Java 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 要高。

  > 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?

- 操作少量的数据 = String 。

  > 这个也是实际编码较为经常使用的方式。

- 单线程操作字符串缓冲区下操作大量数据 = StringBuilder 。

  > 甚至有时，我们为了避免每个线程重复创建 StringBuilder 对象，会通过 ThreadLocal + StringBuilder 的方式，进行对 StringBuilder 的重用。具体可以参考 《StringBuilder 在高性能场景下的正确用法》 文章。

- 多线程操作字符串缓冲区下操作大量数据 = StringBuffer

  > 实际场景下，我们基本不太会出现，多线程操作同一个 StringBuffer 对象。

📣 String s = new String("xyz") 会创建几个对象?

- 首先，在 String 池内找，找到 `"xyz"` 字符串，不创建 `"xyz"` 对应的 String 对象，否则创建一个对象。
- 然后，遇到 `new` 关键字，在内存上创建 String 对象，并将其返回给 `s` ，又一个对象。

所以，总共是 1 个或者 2 个对象。

具体的，可以看看 《关于String s = new String("xyz"); 创建几个对象的问题》 文章的测试代码。

📣 String 为什么是不可变的?

简单的来说，String 类中使用 `final` 关键字字符数组保存字符串。代码如下：

```
// String.java

private final char[] value;
```

- 所以 String 对象是不可变的。

而 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` ，但是没有用 `final` 关键字修饰。代码如下：

```
// AbstractStringBuilder.java

char[] value;
```

- 所以这两种对象都是可变的。

📣 StringTokenizer 是什么?

StringTokenizer ，是一个用来分割字符串的工具类。

示例代码如下：

```
StringTokenizer st = new StringTokenizer("Hello World");
while (st.hasMoreTokens()) {
    System.out.println(st.nextToken());
}
```

输出如下：

```
Hello
World
```

什么是自动拆装箱?

自动装箱和拆箱，就是基本类型和引用类型之间的转换。

📣 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 。

📣 int 和 Integer 有什么区别?

- `int` 是基本数据类型。
- Integer 是其包装类，注意是一个类。

当然，要注意下 Integer 的缓存策略，可以看看 《理解Java Integer 的缓存策略》 文章。

equals 与 == 的区别?

- 值类型（`int,char,long,boolean` 等）的话
  - 都是用 == 判断相等性。
- 对象引用的话
  - == 判断引用所指的对象是否是同一个。
  - equals 方法，是 Object 的成员函数，有些类会覆盖(`override`) 这个方法，用于判断对象的等价性。

> 例如 String 类，两个引用所指向的 String 都是 `"abc"` ，但可能出现他们实际对应的对象并不是同一个（和 JVM 实现方式有关），因此用 == 判断他们可能不相等，但用 equals 方法判断一定是相等的。

📣 如何在父类中为子类自动完成所有的 hashCode 和 equals 实现？这么做有何优劣?

父类的 equals ，一般情况下是无法满足子类的 equals 的需求。

- 比如所有的对象都继承 Object ，默认使用的是 Object 的 equals 方法，在比较两个对象的时候，是看他们是否指向同一个地址。但是我们的需求是对象的某个属性相同，就相等了，而默认的 equals 方法满足不了当前

的需求，所以我们要重写 equals 方法。
- 如果重写了 equals 方法，就必须重写 hashCode 方法，否则就会降低 Map 等集合的索引速度。

📣 说一说你对 java.lang.Object 对象中 hashCode 和 equals 方法的理解。在什么场景下需要重新实现这两个方法?

这个问题，和上个「如何在父类中为子类自动完成所有的 hashCode 和 equals 实现？这么做有何优劣？」一样的答案。

📣 这样的 a.hashCode() 有什么用，与 a.equals(b) 有什么关系?

> 这个问题，和上述问题，就是换个姿势，差不了太多。

1. equals 方法，用于比较对象的内容是否相等。

   > 当覆盖了 equals 方法时，比较对象是否相等将通过覆盖后的 equals 方法进行比较（判断对象的内容是否相等）。

2. hashCode 方法，大多在集合中用到。

   > 将对象放入到集合中时，首先判断要放入对象的 hashCode 值与集合中的任意一个元素的 hashCode 值是否相等，如果不相等直接将该对象放入集合中。
   >
   > 如果 hashCode 值相等，然后再通过 equals 方法判断要放入对象与集合中的任意一个对象是否相等，如果 equals 判断不相等，直接将该元素放入到集合中，否则不放入。

📣 有没有可能 2 个不相等的对象有相同的 hashCode?

可能会发生，这个被称为**哈希碰撞**。当然，相等的对象，即我们重写了 equals 方法，一定也要重写 hashCode 方法，否则将出现我们在 HashMap 中，相等的对象作为 key ，将找不到对应的 value 。

所以说，equals 和 hashCode 的关系会是：

- equals 不相等，hashCode 可能相等。
- equals 相等，请重写 hashCode 方法，保证 hashCode 相等。

一般来说，hashCode 方法的重写，可以看看《科普：为什么 String hashCode 方法选择数字31作为乘子》方法。

final、finally、finalize 的区别?

1）final

`final` ，是修饰符关键字。

- 如果一个类被声明为 `final` ，意味着它不能再派生出新的子类，不能作为父类被继承。因此一个类不能既被声明为 `abstract` 的，又被声明为 `final` 的。
- 将变量或方法声明为 `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，不能重写。

> 另外，在早期的 Java 实现版本中，会将 `final` 方法转为内嵌调用。但是如果方法过于庞大，可能看不到内嵌调用带来的任何性能提升（现在的 Java 版本已经不需要使用 `final` 方法进行这些优化了）。类中所有的 `private` 方法都隐式地指定为 `final` 。

2）finally

在异常处理时提供 `finally` 块来执行任何清除操作。如果抛出一个异常，那么相匹配的 `catch` 子句就会执行，然后控制就会进入 `finally` 块（如果有的话）。

在以下 4 种特殊情况下，finally块不会被执行：

- 在 `finally` 语句块中发生了异常。
- 在前面的代码中用了 `System.exit()` 退出程序。
- 程序所在的线程死亡。
- 关闭 CPU 。

3）finalize

`finalize` ，是方法名。

Java 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确定这个对象没有被引用时对这个对象调用的。

- 它是在 Object 类中定义的，因此所有的类都继承了它。
- 子类覆盖 `finalize()` 方法，以整理系统资源或者执行其他清理工作。
- `#finalize()` 方法，是在垃圾收集器删除对象之前对这个对象调用的。

一般情况下，我们在业务中不会自己实现这个方法，更多是在一些框架中使用，例如 《Netty Using finalize() to release ByteBufs》 。

📣 String 类能被继承吗，为什么?

不能，因为 String 是 `final` 修饰。

抽象类和接口有什么区别?

从设计层面来说，抽象是对类的抽象，是一种模板设计，接口是行为的抽象，是一种行为的规范。

- 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：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。
- 类可以实现很多个接口，但是只能继承一个抽象类。类可以不实现抽象类和接口声明的所有方法，当然，在这种情况下，类也必须得声明成是抽象的。
- 抽象类可以在不提供接口方法实现的情况下实现接口。
- 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。
- 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` 。
- 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(String[] args)` 方法的话是可以被调用的。

📣 继承和组合的区别在哪?

- 继承：指的是一个类（称为子类、子接口）继承另外的一个类（称为父类、父接口）的功能，并可以增加它自己的新功能的能力，继承是类与类或者接口与接口之间最常见的关系。在 Java 中，此类关系通过关键字 `extends` 明确标识，在设计时一般没有争议性。
- 组合：组合是关联关系的一种特例，他体现的是整体与部分、拥有的关系，即 has-a 的关系，此时整体与部分之间是可分离的，他们可以具有各自的生命周期，部分可以属于多个整体对象，也可以为多个整体对象共享。
    - 比如，计算机与 CPU 、公司与员工的关系等。

- 表现在代码层面，和关联关系是一致的，只能从语义级别来区分。

因为组合能带来比继承更好的灵活性，所以有句话叫做“组合优于继承”。感兴趣的胖友，可以看看 《怎样理解“组合优于继承”以及“OO的反模块化”，在这些方面FP具体来说有什么优势？》 文章。

📣 请详细讲述一下 RandomAccess 接口有什么作用?

RandomAccess 用来当标记的，是一种**标记**接口，接口的非典型用法。意思是，随机访问任意下标元素都比较快。

用处，当要实现某些算法时，会判断当前类是否实现了 RandomAccess 接口，会根据结果选择不同的算法。

讲讲类的实例化顺序?

初始化顺序如下：

- 父类静态变量
- 父类静态代码块
- 子类静态变量、
- 子类静态代码块
- 父类非静态变量（父类实例成员变量）
- 父类构造函数
- 子类非静态变量（子类实例成员变量）
- 子类构造函数

感兴趣的胖友，可以详细看看 《Java 类的实例化顺序》 文章，提供的示例。

什么是内部类?

简单的说，就是在一个类、接口或者方法的内部创建另一个类。这样理解的话，创建内部类的方法就很明确了。当然，详细的可以看看 《Java 内部类总结（吐血之作）》 文章。

📣 内部类的作用是什么?

内部类提供了更好的封装，除了该外围类，其他类都不能访问。

📣 Anonymous Inner Class(匿名内部类)是否可以继承其它类？是否可以实现接口?

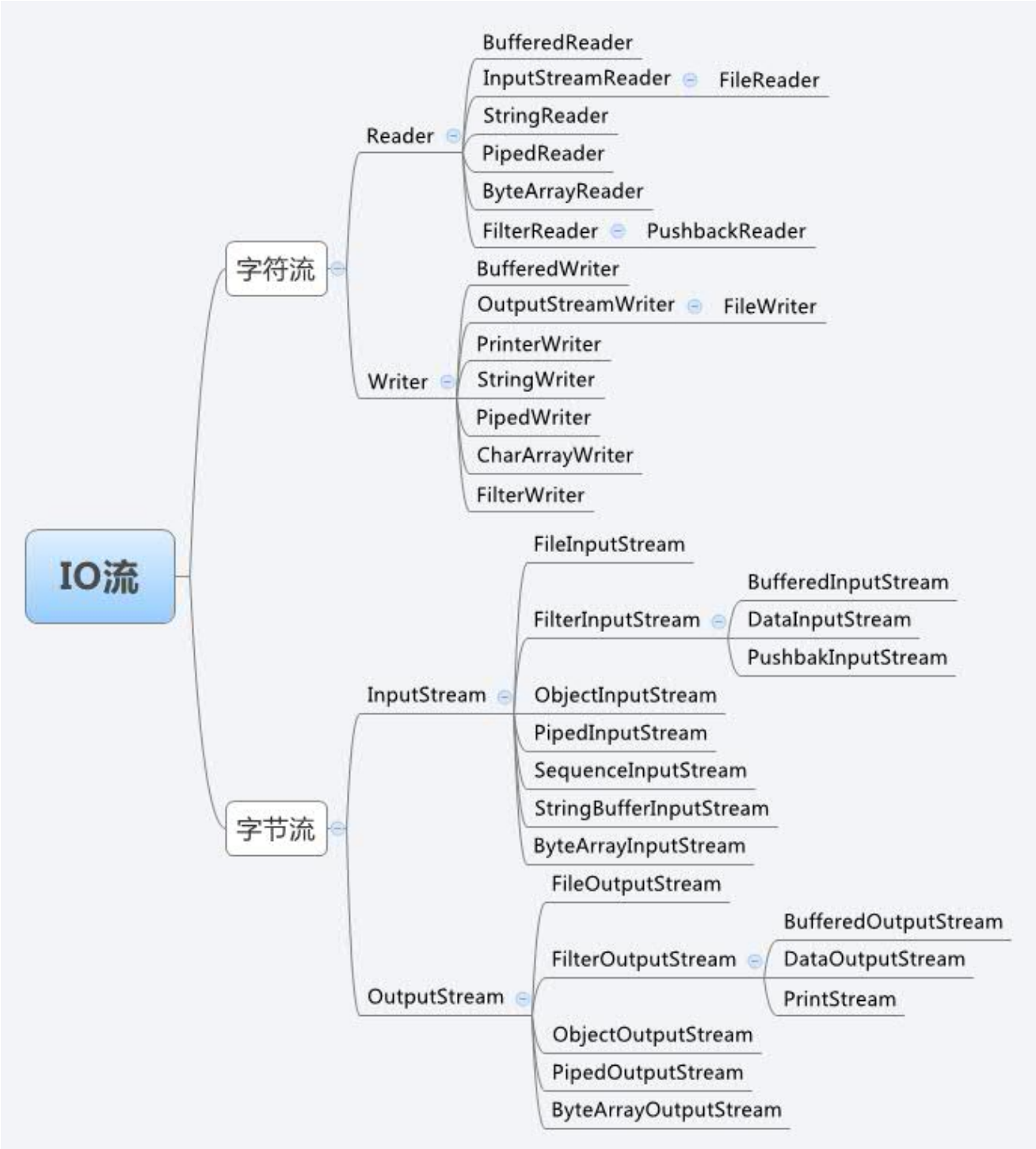
可以继承其他类或实现其他接口，在 Java 集合的流式操作中，我们常常这么干。

📣 内部类可以引用它的包含类（外部类）的成员吗？有没有什么限制?

一个内部类对象可以访问创建它的外部类对象的成员，包括私有成员。

什么是 Java IO ?

Java IO 相关的类，在 `java.io` 包下，具体操作分成面向字节(Byte)和面向字符(Character)两种方式。如下图所示：



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。

如何实现对象克隆？

一般来说，有两种方式：

- 1、实现 Cloneable 接口，并重写 Object 类中的 `#clone()` 方法。可以实现**浅克隆**，也可以实现**深克隆**。
- 2、实现 Serializable 接口，通过对象的序列化和反序列化实现克隆。可以实现真正的**深克隆**。

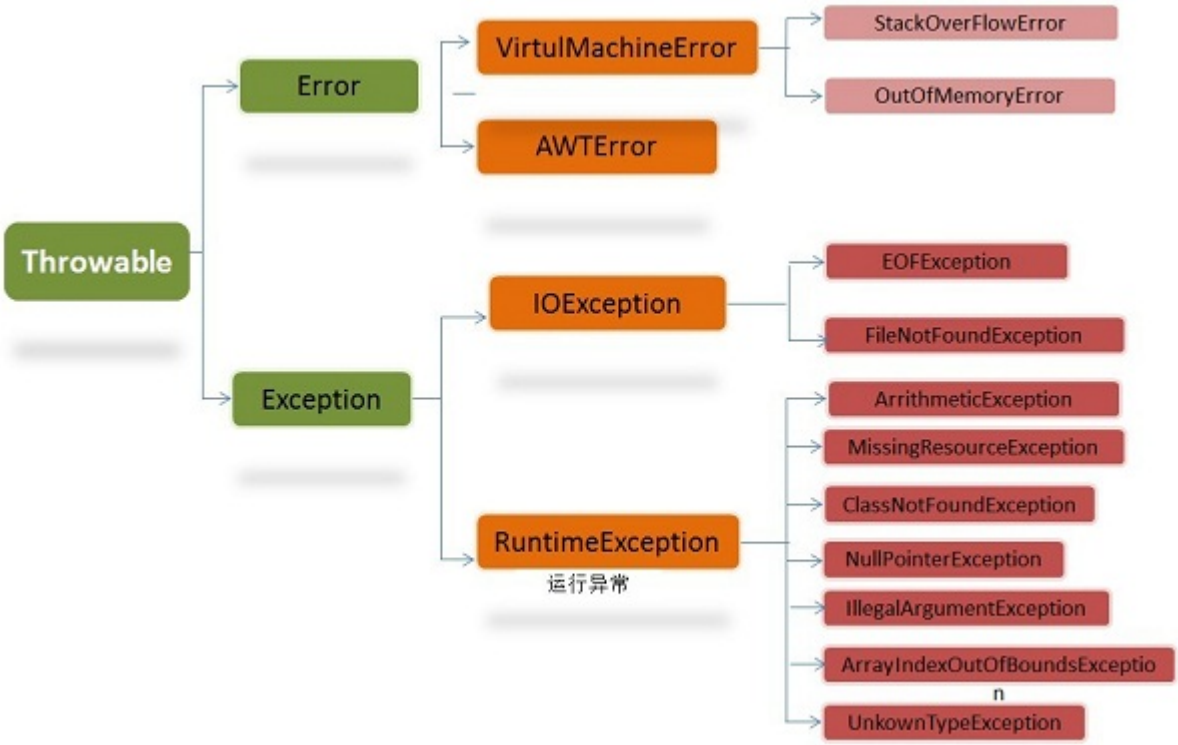
> ：这个问题，也可以变种来问，什么是**浅克隆**和**深克隆**。

具体的代码实现，可以看看 《Java 对象的浅克隆和深克隆》 文章。

实际场景下，我们使用的克隆比较少，更多是对象之间的属性克隆。例如说，将 DO 的属性复制到 DTO 中，又或者将 DTO 的属性复制到 VO 中。此时，我们一般使用 BeanUtils 工具类。具体的使用，看看 《浅谈 BeanUtils 的拷贝，深度克隆》 文章。

error 和 exception 有什么区别？CheckedException 和 RuntimeException 有什么区别？

Java 的异常体系，基于共同的祖先 `java.lang.Throwable` 类。如下图所示：



> 感谢【贾鹤鸣】胖友指出的问题。图有点问题：
>
> 1. 图中的 ArrithmeticException 异常，多了一个 r ，正确拼写是 ArithmeticException 。
> 2. 图中 ClassNotFoundException 异常，父类是 `ReflectiveOperationException => Exception` ，不属于 RunTimeException 。

- Error（错误），表示系统级的错误和程序不必处理的异常，是 Java 运行环境中的内部错误或者硬件问题。
  - 例如：内存资源不足等。
  - 对于这种错误，程序基本无能为力，除了退出运行外别无选择，它是由 Java 虚拟机抛出的。
- Exception（异常），表示需要捕捉或者需要程序进行处理的异常，它处理的是因为程序设计的瑕疵而引起的问题或者在外的输入等引起的一般性问题，是程序必须处理的。Exception 又分为运行时异常，受检查异常。
  - RuntimeException(运行时异常)，表示无法让程序恢复的异常，导致的原因通常是因为执行了错误的操作，建议终止逻辑，因此，编译器不检查这些异常。
  - CheckedException(受检查异常)，是表示程序可以处理的异常，也即表示程序可以修复（由程序自己接受异常并且做出处理），所以称之为受检查异常。

异常的使用的注意地方?

神作《Effective Java》中对异常的使用给出了以下指导原则：

> ：该书，十分推荐去阅读。

- 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。

📣 **Throwable 类常用方法?**

- `#getMessage()` 方法：返回异常发生时的详细信息。
- `#getCause()` 方法：获得导致当前 Throwable 异常的 Throwable 异常。
- `#getStackTrace()` 方法：获得 Throwable 对象封装的异常信息。
    - `#printStackTrace()` 方法：在控制台上打印。

📣 **请列出 5 个运行时异常?**

- NullPointerException
- IndexOutOfBoundsException
- ClassCastException
- ArrayStoreException
- BufferOverflowException

📣 **throw 与 throws 的区别 ?**

- `throw` ，用于在程序中显式地抛出一个异常。
- `throws` ，用于指出在该方法中没有处理的异常。**每个方法必须显式指明哪些异常没有处理，以便该方法的调用者可以预防可能发生的异常。**最后，多个异常用逗号分隔。

📣 **异常处理中 finally 语句块的重要性?**

不管程序是否发生了异常，`finally` 语句块都会被执行，甚至当没有 `catch` 声明但抛出了一个异常时，`finally` 语句块也会被执行。

`finally` 语句块通常用于释放资源，如 I/O 缓冲区，数据库连接等等。

📣 **异常被处理后异常对象会发生什么?**

> ：这个问题有点奇怪，从网上找来的...

异常对象会在下次 GC 执行时被回收。

说说反射的用途及实现?

Java 反射机制主要提供了以下功能：

- 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：

- 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。

不了解 Java 反射的同学，可以看看 《什么是反射、反射可以做些什么》 。

> 对于大多数 Java 萌新，包括在内。不过后来发现，再难的 Java 知识，未来都需要变成我们的基础知识，嘻嘻。

📣 **反射中，Class.forName 和 ClassLoader 区别?**

这两者，都可用来对类进行加载。差别在于：

- `Class#forName(...)` 方法，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。
- 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。

  > `Class#forName(name, initialize, loader)` 方法，带参函数也可控制是否加载 `static` 块，并且只有调用了newInstance 方法采用调用构造函数，创建类的对象。

详细的测试，可以看看 《Java 反射中，Class.forName 和ClassLoader 的区别(代码说话)》 文章。

UnsupportedOperationException 是什么?

UnsupportedOperationException ， 是用于表明操作不支持的异常。

在 JDK 类中已被大量运用，在集合框架 `java.util.Collections.UnmodifiableCollection` 将会在所有 add 和 remove 操作中抛出这个异常。

什么是注解?

直接看 《深入浅出 Java 注解》 。

如果胖友没有自己实现自定义的注解，千万一定马上去尝试写下。酱紫，我们会对注解，有更好且清晰的认识。

什么时候用断言（`assert`）?

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，断言用于保证程序最基本、关键的正确性。断言检查通常在开发和测试时开启。为了保证程序的执行效率，在软件发布后断言检查通常是关闭的。
- 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`；如果表达式的值为 `false` ，那么系统会报告一个AssertionError 错误。断言的使用如下面的代码所示：

  ```
  assert(a > 0); // throws an AssertionError if a <= 0
  ```

  - 断言可以有两种形式：
    - `assert Expression1;` 。
    - `assert Expression1 : Expression2;` 。
    - Expression1 应该总是产生一个布尔值。
    - Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调试信息的字符串消息。
- 要在运行时启用断言，可以在启动 JVM 时使用 `-enableassertions` 或者 `-ea` 标记。要在运行时选择禁用断言，可以在启动 JVM 时使用 `-da` 或者 `-disableassertions` 标记。要在系统类中启用或禁用断言，可使用 `-esa` 或 `-dsa` 标记。还可以在包的基础上启用或者禁用断言。

当然，实际场景下，我们会在 Spring 的源码中看到，它自己封装了 Assert 类，实现更方便的断言功能，并且，在生产环境下也启用。

另外，在单元测试中，也会使用自己封装的断言类，判断执行结果的正确与错误。

Java 对象创建的方式?

1. 使用 `new` 关键字创建对象。
2. 使用 Class 类的 newInstance 方法(反射机制)。
3. 使用 Constructor 类的 newInstance 方法(反射机制)。
4. 使用 clone 方法创建对象。
5. 使用(反)序列化机制创建对象。

彩蛋

一写到 Java 基础，怎么说呢？就觉着概念挺多的，有点无从下手。实际情况下呢，自己面试的时候，很多问题可能都不太会问。那么，就把本文当成一种复习的文章吧，哈哈哈哈。

参考与推荐如下文章：

- 《2018 年最新 Java 面试题及答案整理》
- 《32 道常见的 Java 基础面试题》
- 《Java 总结》
- 《Java 基础知识》
- Java 面试宝典