链滴

Java 回顾复习

作者：ChenforCode

原文链接：https://ld246.com/article/1610890154471

来源网站：链滴

<h3 id="1-Java相关规范">1、Java 相关规范</h3>

 Java 的三个版本：

   JavaSE（标准版），JavaEE（企业版），JavaME（微型版），后两位基本上都没人用了，我现在用的都是第一个

 JDK 和 JRE，Java 和 Javac

   JDK 全名叫做 java development toolkit，也就是 java 开发工具，是 Java 下载下来自带的一工具类集合，还包含了 Java，Javac 等命令工具
   JRE 叫做 Java runtime environment，也就是 java 运行时环境，一般来说 jdk 会包含 jre

   Javac 命令，将.java 文件编译成字节码文件.class，javac hello.java
   Java 命令，运行.class 文件，java hello。注意这里没有 class 后缀

<h3 id="2-简单程序">2、简单程序</h3>

 scanner.next()和 scanner.nextLine()

   next：next 将空格，tab 和回车作为结束符（无效字符），但是只有在接收到一个有效字符之，再收到结束符才会停止
   nextLine：遇见回车符（换行符）就停止，就算只接受了一个回车符也会停止。
   建议在使用 next()，nextInt()之后不使用 nextLine，因为前者输入完毕之后用户键入的回车会接被 Line 接收，导致 Line 失效。

<h3 id="3-选择-分支结构-">3、选择（分支结构）</h3>

 浮点数不能直接相等，而是应该用相减的到的值和 10-7（float）或 10-14（double）来进行较，即误差足够小则看做相等
 产生随机数用 Math.random()获得一个 0.0-1.0 之间的随机 double，不包括 1.0

<h3 id="4-数学函数-字符-字符串">4、数学函数，字符，字符串</h3>

 unicode 编码，用两个字节表示所有字符（表示不完所以出现了补充字符，这里不用管），unicde 以\u0000 到\uFFFF 结束
 ASCII 码，他被包含在 unicode 编码里，表示的是从\u0000 到\u007F 共 128 个字符
 char 型可以转换为任何类型数值，其他数值转换为 char 的时候会丢弃高位，留下一个字节，cha比较就是 ascii 的比较
 String 的相关方法

   求字符串的长度用的是.length()，是类中的方法。求数组的长度用的是.length，他是数组的一属性。

charAt(index)获取 index 位置的字符。
indexof('a')获取 a 字符的 index。
substring(begin, end)，截取字符串，截取结果包含 begin，不包含 end
字符串和数字转换，String.valueOf(1)，Integer.parseInt("123")。

格式化字符输出

System.out.printf("abc is %4.2f, def is %d", abc, def);其中 4 代表域宽，2 代表保留小数位，f 代表浮点数，%b 是 bool，%c 是字符，%e 是科学计数法，%s 是字符串

<h3 id="5-循环">5、循环</h3>

for 循环，(a;b;c)，其中 a 是初始条件，b 是循环继续条件，c 是每次迭代后进行的操作。可以的很复杂。注意这里多个语句用逗号隔开，而不是分号。 <pre><code class="highlight-chroma">//可以有多个条件进行控制，也可以在每次迭代后完成多个操作
for(int i = 10, j = 100; i + j < 1000; i++, sout);
//这就是个死循环，类似于while(true)
for(;;);
for(int i = 0; i < 10; i++)
</code></pre>
循环的考点之一，考循环体的重复次数

while 循环是先判条件，然后执行，再判条件，可能一次都不执行
dowhile 是先执行，然后判条件，再执行，一定会执行一次
for 循环是先判条件，然后执行，然后更新条件，再判断，可能一次都不执行

<h3 id="6-方法">6、方法</h3>

方法定义： <pre><code class="highlight-chroma">//修饰符 返回值 方法名 参数列表
public static int max(int num1, int num2){
}
</code></pre>
参数传递，形参是否会改变实参?

基本数据类型和 String,Integer 等不会影响，可以看成是值传递
数组，类对象，集合（list）之类的会影响，可以看出是引用传递

函数重载的唯一标准：具有不同的参数列表。即返回值不同不属于重载

<h3 id="7-一维数组">7、一维数组</h3>

一维数组的声明，int 默认值为 0，char 数组默认值是'\u0000'，boolean 默认值是 false。数创建完毕之后就不能修改它的大小，除非重新创建一个。但是 ArrayList 底层由数组实现，但是它可进行扩容（随意改变大小），那是它底层重写了一些东西。数组必须创建完毕之后才能赋值 <pre><code class="highlight-chroma">//直接定义一个空数组
int[] intArray = new int[10];

//定义并且初始化，这时不再使用new关键字
int[] intArrayInit = {1, 2, 3, 4, 5};
</code></pre>

foreach 循环 <pre><code class="highlight-chroma">//直接定义一个空数组
int[] intArray = new int[10];
for (int eachElement: intArray) {
  // TODO
}
</code></pre>
数组赋值，直接用 arr1 = arr2，最终的作用是使得 arr1 和 arr2 均指向同一个地址空间，并不真的赋值。想把 arr2 拷贝给 arr1 可以采用的方法是

循环语句逐个拷贝
Arrays.copyOf()
System.arrayCopy()
clone 方法

变长参数列表，可以将同样类型但是个数不定的参数传递给方法，方法会当做数组进行处理.比，确定参数类型是 int，但是不知道有多少个，可以通过下面的方式，会默认将传入的参数封装成一个 array 数组 <pre><code class="highlight-chroma">public static void print(int... array) {
  System.out.println(array.length);
}
</code></pre>
Arrays 工具类

sort()，升序排序
binarySearch()，二分查找
fill()，用指定元素填充数组

命令行参数

java TestMain arg0 arg1 "arg 2"。如果用该命令执行 java 程序，除了 java 命令，要执行的 Java 程序，后边的所有都会被当做命令行参数可以传入主方法，会被一起送入 String[] args 数组中

第一种方法是错的，因为没有对 myList 进行 new 申请空间。但是第二种会自动的帮你做 new 作。

<h3 id="8-二维数组">8、二维数组</h3>

二维数组的定义 <pre><code class="highlight-chroma">//直接创建
int [][]matrix = new int[5][5];
//创建并初始化，创建出来的是一个3行2列的二位数组
int [][]matrix = {{1, 2}, {3, 4}, {5, 6}};
//行长度不同的二维数组
int [][]matrix = {{1}, {1, 2}, {1, 2, 3}};
//可以指定行数，不指定列数，但是不可以没有指定行数。也就是说第一个参数必须指定
int [][]matrix = new int[3][];
</code></pre>
二维数组的长度

二维数组本质上是一个数组，只是每个数组元素都是一个数组。可以用 matrix.length 获取行，matrix[0]获取第一行的列数。
二维数组的每一行的长度都可以不同，如上方定义

三维数组的定义 <pre><code class="highlight-chroma">//[4][3][2]数组
double [][][] score = {
{
  {{1, 2}, {3, 4}, {5, 6}},
  {{7, 8}, {9, 10}, {11, 12}},
  {{13, 14}, {15, 16}, {17, 18}},
  {{19, 20}, {21, 22}, {23, 24}}
}
}
</code></pre>

<h3 id="9-对象和类">9、对象和类</h3>
<pre><code class="highlight-chroma">* 静态变量被类中的所有对象共享，静态方法只能访问静变量，静态方法不能访问类中的实例成员（因为静态方法是和类绑定的，当类完成初始化的时候实例量还没有初始化，因为没有实例出现）
* 实例方法可以调用静态的，但是静态的不能调用实例的
</code></pre>

可见性修饰符

public：类内，本包，子类，外部

protected：类内，本包，子类

default：类内，本包
private：类内

<img src="https://ld246.com/images/img-loading.svg" alt="img" data-src="https://b3logfile.com/file/2021/01/solofetchupload3347694905222860899-741faec1.png?imageView2/2/interlace/1/format/jpg">

类变量作用域

实例变量和静态变量的作用域是整个类，无论变量在哪里声明
局部变量和实例变量同名，则局部变量优先，即谁近用谁

<h3 id="10-面向对象思考">10、面向对象思考</h3>

基本类型和包装类型之间的转换

基本类型-》包装类型称为装箱，相反叫做开箱，其中开箱和装箱一般都会自动进行。
例如 <pre><code class="highlight-chroma">//二者都是等价的 自动装箱
Integer intObj = new Integer(2);
Integer intObj = 2;
//自动开箱，Integer(5)会自动变成int 5类型
int a = 3 + new Integer(5);
</code></pre>

大数类（任意大小）和高精度类（任意精度）

BigInteger，add，subtract，multiple，divide，remainder 算术运算 <pre><code class=highlight-chroma">bigInteger1.add(bigInteger2);
</code></pre>
BigDecimal

String 类

String 对象是不能变的指的是，字符串的内容是不能变的，但是这个对象的指向是可以变的。如下的例子。将 s1 重新赋值之后，"Java"这个字符串本身并没有改变，而是 s1 这个对象由原来的指 Java 变成了指向 HTML。 <pre><code class="highlight-chroma">String s1 = "Java";
s1 = "HTML";
</code></pre>
由于字符串不可变，但是会经常使用，所以 JVM 会把相同的字符串，直接使用一个实例，举下例子：其中 s1 和 s3 是相同的两个实例，他们均指向同一个字符串"i am a string"，这个字符串已被加入到了字符串的常量池中，不管以后有多少个引用使用，都是指向这一个字符串。但是 s2 不同他是用了一个 new 运算符，是直接开辟了一个新的内存写入了一个字符串。假如通过一些字符串操，改变 s1，实际上是让 s1 指向了一个新的字符串空间，原来的空间存的东西没变，因此 s3 的内容不变。 <pre><code class="highlight-chroma">String s1 = "i am a string";
String s2 = new String("i am a string");
String s3 = "i am a string";
</code></pre>
字符串本身是不能变的，但是可以通过这个字符串进行一些操作，得到一些新的字符串，新的符串实际上是被存到了一个新的地址空间中。这就是 String 中的 replace，subString 之类方法的的理，他们得到的字符串不是将源字符串修改，而是得到的一个全新的字符串。
字符串和数组之间的转换

数组转为字符串： <pre><code class="highlight-chroma">char []chars = {'a', 'b', 'c'};
//一下两种方法均可将数组转为字符串
String str1 = new String(chars);
String str2 = String.valueOf(chars);
</code></pre>
数组转为字符串 <pre><code class="highlight-chroma">String str = "abc";
char[] chars = str.toCharArray();
</code></pre>

数字和字符转换成字符串，利用 String.valueOf()可以将很多类型转化成字符串，例如 char，in，double，float，boolean
字符串的地址相同判断用==，内容相同判断用 equal()

StringBuilder 和 StringBuffer

二者的用途基本完全一致，并且与 String 的用法也十分相似
与 String 的区别在于 String 创建出来的字符串是不可变的，但是这二者是可变的。
因此如果字符串将有大量的修改操作，建议使用 StringBuilder 和 Buffer，因为他们是直接修，而不是创建新的字符串

data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in
>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<h3 id="11-继承和多态">11、继承和多态</h3>

  子类继承父类，只能继承除 private 之外的，意思就是子类无法直接访问到父类的 private 变量，除非父类对这个变量写好了 get 和 set。
  Java 中不支持多重继承，即一个子类只能有一个父类
  在子类中，使用 super 代表父类，可以用 super.xxx 调用父类中的构造方法和普通方法。用父类构造方法的方式是，必须放在子类构造方法中的第一句 <pre><code class="highlight-hroma">public ChildrenConstructor() {
  super();
  //super(param);
  //TODO Children class contructor
}
</code></pre>
  即使子类中没有显式的通过 super 来调用，编译器还是会在子类构造函数中默认加上 super 进行父类构造调用。
  子类构造之前必须先构造父类，父类构造又必须先构造父类的父类，这个过程叫做构造方链。
  如下场景会编译出错 <pre><code class="highlight-chroma">class A {
    public A(int a) {

   }
}
class B extends A {

}
</code></pre>

   关键点 1：如果一个类中没有写构造函数，那么会默认调用无参构造函数
   关键点 2：如果一个类中写了构造函数，那么无参构造函数就会自动消失。
   关键点 3：子类调用无参构造函数，也会默认调用父类的无参构造函数。
   所以，类 B 会调用无参，然后会调用父类类 A 的无参构造，但是 A 没有，有参构造函数覆盖无参，无参消失，所以就会编译出错

  方法重写

   子类需要对继承自父类的某个方法进行重写实现。
   子类中的方法必须和父类中的方法完全一致（包括返回值，函数名，参数列表），然后对函数进行重写。
   和重载的区别

     重写是两个方法完全一致，只有函数体实现不一样。重写的意思是对一个已有方法的重新实。
     重载是具有不同的参数列表，重载的意思是使用同名字但是不同签名（参数列表）来定义多方法。
     重载是针对于同一个类通过参数列表不同定义了多个方法，重写是子类对父类方法的重新实现/li>

 静态方法可以被继承，但是不能被重写

假如子类中具有和父类中同样的静态方法，只是实现不同，这种情况不是父类的函数被子类重，而是属于子类重定义，也就是自己定义了一个静态方法。
无法重写的原因可以这样理解：重写就是为了在不同的子类对象中呈现不同的效果实现多态，是静态变量与对象毫无关联。因此静态方法不能重写。

 子类不能改变父类中的同名函数（只能重写实现），假如父类中有一个 void func()，子类就不能有一个 int func()·
 多态

先简单给你说下多态啥：

一般情况 Parent p = new Parent(); p.xxfunc()；这样就很好理解，创建一个父类对象，调用类对象的方法。
第二种情况 Child c = new Child()；c.xxfunc()；这样也好理解对吧，创建一个子类对象，然调用子类对象的方法。
那么第三种情况呢就是。Parent p = new Child()；注意，这样的用法是没错的，但是要注意点就是只能将子类赋值给父类，不能反过来（可以类比强转进行理解，首先子类包含的东西要比父类很多，因此子类比父类大，因此我子类赋值给父类一定能把父类填满，顶多是丢失一些东西，类似丢精度。但是反过来父类赋值给子类，你东西没人家多，子类都填不满，这个类怎么用？）。
接第三点，父类和子类都 xxfunc()方法，用的是上边讲的重写。Parent p = new Child()；这如果 p.xxfunc()；那么这个 xxfunc()调用的是子类中的方法，而不是父类的方法。
如何理解第四点，对于 Parent p = new Child()；我们把 Parent 叫做声明类型，Child 叫做际类型，声明类型调用的函数如果在子类父类都有，那么调用哪个函数取决于实际类型。
那么为什么叫多态？？？？就是说子类可以有很多个，或者说可以继承很多层。这样就可能现下边的情况 Parent p1 = new Child1()； Parent p2 = new Child2()； Parent p3 = new Child3(；这样按照上一点的理解，如果 p1，p2，p3 分别调用 xxfunc()方法是不是就分别调用了三个子类的xfunc()方法。所以就是说同一个父类，调用同一个方法，但是出现了三种不同的结果，这样就是多态多种形态嘛），理解了吧！！！
还有一种情况，就是那假如 Parent p = new Child()；p.xxfunc()；这里按照上述理解应该调子类的 xxfunc()方法了吧，但是假如子类没有实现这个方法呢？？？？那么这时编译器就会通过继承，从子类往上找父类，直到找到最近的一个实现了该方法的父类，然后调用这个父类中的方法。如果后都找到 Parent 了才找到实现，那就没办法，只能调用 Parent 里边的方法了。

理解了什么叫做多态了吧，现在了解一下术语。

声明类型的变量调用某个函数，具体会执行某个函数取决于实际类型，这个过程叫做动态绑。
同一个对象，呈现出多个实现，多个形态，这样的现象称为多态（hhh 这是我自己编的）

动态绑定和重载有啥区别?

动态绑定是帮你找到你具体执行的函数在哪个类里
重载就是在你找到那个类之后，再通过参数确定你要执行类里的哪个函数
二者的层次不同。

 对象转换和 instanceof 运算符

对象的引用可以转换成另外一种对象的引用，这个叫做对象转换

举个例子 Parent p = new Child()就是一种对象转换，这种方式称为隐式转换。
刚才我们讲到一个父类对象是不能赋值给子类对象的（这里说的是不能通过第二点的隐式转换但是如果你非要转，那也可以），这里就用到的就是显式转换：Child c = (Child)new Parent()这样可以，但是没有什么必要，也没有什么意义。一般来说显式转换会用在如下场景：
Object o = new Student();Student stu = (Student)o;就是说你明知一个对象的实际类型是想要的类型，但是声明类型不是，这个时候你可以通过强转把他转回来。
在上述进行强转的时候我们说到就是说你明知一个对象的实际类型是你想要的类型</trong>，那么如何判断某个对象是我们想要的类型呢，这里就用到了 instanceof 运算符。
instanceof 运算符的作用就是判断，一个对象是否是为某个类的实例。当然这个实例包括了自的实例，也可能是子类的实例。如下例子 <pre><code class="highlight-chroma">//Test是父类，是子类
Test test = new B();
//test是子类对象，所以既是父类实例，也是子类实例
System.out.println(test instanceof Test); //true
System.out.println(test instanceof B); //true

//test1是父类对象，所以只是父类实例，不是子类实例
Test test1 = new Test();
System.out.println(test1 instanceof Test); //true
System.out.println(test1 instanceof B); //false
</code></pre>

 Object 类

Object 类是所有类的父类
Object 类中的 equal 方法，默认实现是直接用==判断两个对象，也就是两个对象必须除了内一样，还要是地址一样。
如果我们的需求是如果两个对象里边的内容一致，就认为两个对象相等，那么我们可以在自己类中重写 equal 方法，只判断内容，不判断地址。

 通过添加 final 关键字，对于变量说明改变了不能被修改，对于方法来说，该方法是最终的不能被重写。
 public static final int a = 10;对于上述所有的修饰符，这样理解，int a 是永远不能分开的并且后边除了赋值不能有其他的东西。剩余的三个的顺序是可以互换的
 但是为了提高可读性，一般的写法是 public static final int a = 10;方法亦然。

<h3 id="12-异常处理和文本I-O">12、异常处理和文本 I/O</h3>

 异常处理的基本结构

 运行可能会有异常出现的代码并进行相应的处理。
 主动抛出异常，主动抛出异常的话需要在函数最后加上异常声明，如下点 <pre><code class="highlight-chroma">try {
   // 尝试执行的可能会出现异常的代码
   int a = 1 / 0;
   // 主动抛出异常
   throw new RuntimeException();
} catch (RuntimeException e) {
   // 捕捉到异常后需要执行的代码
   e.printStackTrace();
} finally {

   // 无论有无异常都会执行的代码
}
</code></pre>
 假如在当前函数中，你明知会有异常，但是你不想处理，那么你可以在函数最后，加上去个 throws xxxException，代表将由上层函数，即这个函数的调用者进行异常处理，并且你要在内部该异常抛出。 <pre><code class="highlight-chroma">static private void test3() throws RntimeException{
    throw new RuntimeException();
}
</code></pre>

 异常的种类

广义的异常分为两种，Exception 和 Error，二者都继承自 Throwable
对于 Error，指的是由 Java 虚拟机抛出的系统错误，很少发生骂我们不讨论这个东西
我们经常说的异常大多数是指 Exception，它是由程序和外部环境引起的一些错误，他可以被获和处理
常见的 Exception 主要有

RuntimeException：运行时异常，下边都是其子类（不需要处理，即免检）

NullPointerException 空指针
ArithmeticException 算术异常

除了运行时异常以外，所有的异常都是必检异常，也就是必须进行捕获并处理的异常，例如 IException。

 自定义异常类，只需要继承 Exception 类即可
 File 类

File 是对物理机器上一个文件的抽象，我们可以通过文件路径创建一个 FIle 对象，当然文件路不存在也可以，我们可以通过 isExists 来判断文件是否真实存在。

 文件输入和输出

PrintWriter，使用如下方法创建一个文件并向文件中写入数据 <pre><code class="highlightchroma">PrintWriter output = new PrintWriter(filename);
output.print("写入文件的内容");
output.close();
</code></pre>

 Try-with-resources 语法

就是对于那些文件操作，数据库操作的相关资源，必须在 finally 中进行关闭，这样的操作很麻而创建出来的一种新的语法。如下 <pre><code class="highlight-chroma">try (
  PrintWriter output = new PrintWriter(filename);
) {
  out.print("xxx");
}
</code></pre>
这种语法形式会自动的关闭 output，但是注意，资源必须是一个 AutoCloseable 类型

 使用 Scanner 从文件中读取。如果传入 System.in 代表从系统控制台读取 <pre><c

de class="highlight-chroma">Scanner input = new Scanner(new File(filename));
input.next();
input.nextInt();
</code></pre>

<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></scr
pt>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342"
data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in
>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<h3 id="13-抽象类和接口">13、抽象类和接口</h3>

  抽象类的特点

   不能用于创建对象，这个类只能是用来让子类继承。抽象类中只声明没有实现的方法成为抽象
法，是为了让子类继承并实现。如果子类继承了，将所有抽象方法实现完毕，那么这个类是个正常的
，可以创建实例。但是只要有一个没有实现，那么这个子类也是抽象类，不能创建实例。
   抽象方法不能包含在非抽象类中，换句话说，如果一个类中有抽象方法，那么这个类必须是抽
类。
   抽象方法是静态的
   父类是具体的，子类可以是抽象的，例如 Object 是具体的，但是其子类中会存在抽象类（这
情况很特殊，因为所有的类都是 Object 的子类，包括抽象类）
   抽象类不能创建实例，指的是不能创建一个抽象类实例，但是如果子类不是抽象类了，当然可
通过多态特性，将子类实例赋值给抽象类父类。

  抽象方法的访问权限（书里没写，大概了解下吧）

   1.8 之前，抽象方法的默认访问权限是 protected
   1.8 之后，抽象方法的默认权限为 default

  接口的特点

   接口和抽象类本身就很相似
   接口只包含常量和抽象方法
   接口中的所有数据域都是 public static final，并且所有方法都是 public abstract，但是接口
许这些修饰符忽略。也就是说，如下定义等价 <pre><code class="highlight-chroma">//以下两
定义是等价的e
public interface T {
  public static final int k = 1;
  public abstract void p();
}
public interface T {
  int k = 1;
  void p();
}
</code></pre>

  comparable 接口的比较方法，里边有个 compareTo 方法，传入 o，理解如下：obj.com
areTo(o);如果 obj 比 o 小返回-1，如果 obj 和 o 相等返回 0，大返回 1
  comparable 排序方法。重点理解这样的写法是升序要是降序。如何理解：
 <img sr

="https://ld246.com/images/img-loading.svg" alt="image-20210115194728047" data-src="https://b3logfile.com/file/2021/01/solofetchupload7644132335050310145-ee20349e.png?imageView2/2/interlace/1/format/jpg">

   二者返回 1，才会进行交换操作。
   图中 getArea()是前者，o 是后者，返回 1 的情况，也就是前者大于后者，此时交换
   交换完后者（小的）会跑到前边，所以自然而然前边都是小的，所以是降序排列。
   也就是说前者 > 后者返回 1 就是降序，前者大于后者返回-1 就是升序。

  理解了以上的写法，那么可以直接记忆，不用每次都推算。

   如果写成 return s1 > s2 ? 1 : -1。代表前者大于后者就交换，所以大的换到后边，会升序</i>
   如果写成 return s1 > s2 ? -1 : 1。代表前者小于后者才交换，小的会换到后边，降序。

  或者可以写的更简单

   return s1 - s2。只有 s1 > s2 才会返回正数才会交换，升序
   return s2 - s1，降序。

  Cloneable 接口

   两个对象在克隆的时候有两种克隆方式，浅复制和深复制
   浅复制，对简单数据对象直接赋值，对对象类型也是直接赋值，所以是两个对象同时指向一个存空间
   深复制，会对对象类型申请新空间，然后对里边的数据进行复制

  java 类不允许多重继承，即不允许有多个父类，但是允许实现多个接口
  java 允许接口继承多个接口
  接口和抽象类的简单理解区别：

   抽象类具有一种强烈的父子关系，例如水果和苹果，水果里的很多具体方法都不能实现，例如的方法，没有指定到具体的水果是不知道怎么吃的，所以只能在苹果中实现吃的方法。
   接口表示某个类具有某个属性，例如房子类和门接口，只是说门是房子具有的一个属性，但是不存在严格的父子关系，因为除了房子，例如车子也可以有门。

<h3 id="17-二进制IO">17、二进制 IO</h3>

  首先明确：计算及本身并不区分二进制文件和文本文件，所有的文件存储都是以二进制的式存储的。文本 I/O 是在二进制 I/O 的基础上建立起来的。前边讲到的 printwriter 就是一种文本 IO
  二进制 IO 的分类：
 <img src="https://ld246.com/images/img-loading.svg" alt="image-20210115201908145" data-src="https://b3logfile.com/file/2021/01/solofetchupload924472808068916996-0ba2da73.png?imageView2/2/interlace/1/format/jpg">

    FileInputStream 和 FileOutputStream

     构造方法需传入文件名或者文件对象
     可以使用二者创建的对象使用 input.read()从文件中读取，或者使用 output.write()向文件写
 <pre><code class="highlight-chroma">//建立输入对象，从文件中读取
FileInputStream input = new FileInputStream("a.txt");
//建立输出对象，向文件写入

FileOutputStream output = new FileOutputStream("a.txt");
output.write("balabala");
//读取的时候读取到了-1代表读取结束
while ((value = input.read()) != -1) {
  //TODO balabala
}
</code></pre>

    FilterInputStream 和 FilterOutputStream

     某种目的过滤字节的数据流。从来没见有人用过

    DataInputStream 和 DataOutputStream

     适合处理基本数值类型或者字符串，从来没见有人用过。
     它的构造函数需要传入 FileInputStream 和 FileOutputStream

    BufferInputStream 和 BufferOutputStream

     其他几个的原理都是，读什么就从磁盘中拿。
     这个的原理是设置了一个缓冲区，会一次性从磁盘中读取某些数据到缓冲区，然后任何时候要该数据的时候会从缓冲区里拿。减少了与磁盘的读写次数。
     他的创建同样需要传入 FileInputStream 和 FileOutputStream<img src="https://ld246.co/images/img-loading.svg" alt="image-20210115203252574" data-src="https://b3logfile.com/ile/2021/01/solofetchupload594757466961161149-16c4ef29.png?imageView2/2/interlace/1/ormat/jpg">

    BufferReader 和 BufferWriter

     是最快的
     创建方式需要传入 FileReader 和 FileWriter。

  比较常见的文件操作三件套：一层套一层，反正只要知道 bufferReader 是最快的就完事了/p> <pre><code class="highlight-chroma">FileInputStream fis=new FileInputStream(sourceFle);
InputStreamReader fr=new InputStreamReader(fis,"utf-8");
BufferedReader br=new BufferedReader(fr);
</code></pre>
  序列化和反序列化

   简单理解：序列化就是将一个 Java 对象变成字节序列的过程，反序列化就是将字节序列恢复为 Java 对象的过程。
   不是所有的对象都能够写到输出流，只有是可序列化的才能够写入
   可序列化的对象都是 Serializable 接口的实例。
   实现这个接口就能够启动 Java 的序列化机制，自动完成对象和数组的存储过程。

  随机访问文件 RandomAccessFile

   以上所有的流都是顺序流，只能够顺序访问文件
   RandomAccessFile 的关键函数就是 seek，可以在设置文件指针的位置，可以随机读取

