Java 葵花宝典 v1.0

对于这个系列里的问题，每个学 Java 的人都应该搞懂。当然，如果只是学 Java 玩玩就无所谓了。如果你认为自己已经超越初学者了，却不很懂这些问

题，请将你自己重归初学者行列。内容均来自于 CSDN 的经典老贴。

问题一：我声明了什么！

```
String s = "Hello world!";
```

许多人都做过这样的事情，但是，我们到底声明了什么？回答通常是：一个 String，内容是“Hello world!”。这样模糊的回答通常是概念不清的根

源。如果要准确的回答，一半的人大概会回答错误。

这个语句声明的是一个指向对象的引用，名为“s”，可以指向类型为 String 的任何对象，目前指向"Hello world!"这个 String 类型的对象。这就是

真正发生的事情。我们并没有声明一个 String 对象，我们只是声明了一个只能指向 String 对象的引用变量。所以，如果在刚才那句语句后面，如果再

运行一句：

```
String string = s;
```

我们是声明了另外一个只能指向 String 对象的引用，名为 string，并没有第二个对象产生，string 还是指向原来那个对象，也就是，和 s 指向同一个

对象。

问题二："=="和 equals 方法究竟有什么区别？

==操作符专门用来比较变量的值是否相等。比较好理解的一点是：

int a=10;

int b=10;

则 a==b 将是 true。

但不好理解的地方是：

String a=new String("foo");

String b=new String("foo");

则 a==b 将返回 false。

根据前一帖说过，对象变量其实是一个引用，它们的值是指向对象所在的内存地址，而不是对象本身。a 和 b 都使用了 new 操作符，意味着将在内存中

产生两个内容为"foo"的字符串，既然是“两个”，它们自然位于不同的内存地址。a 和 b 的值其实是两个不同的内存地址的值，所以使用"=="操作符

，结果会是 false。诚然，a 和 b 所指的对象，它们的内容都是"foo"，应该是“相等”，但是==操作符并不涉及到对象内容的比较。

对象内容的比较，正是 equals 方法做的事。

看一下 Object 对象的 equals 方法是如何实现的：

boolean equals(Object o){

return this==o;

}

Object 对象默认使用了==操作符。所以如果你自创的类没有覆盖 equals 方法，

那你的类使用 equals 和使用==会得到同样的结果。同样也可以看出，

Object 的 equals 方法没有达到 equals 方法应该达到的目标：比较两个对象内容是否相等。因为答案应该由类的创建者决定，所以 Object 把这个任务留

给了类的创建者。

看一下一个极端的类：

```
Class Monster{
private String content;
…
boolean equals(Object another){ return true;}

}
```

我覆盖了 equals 方法。这个实现会导致无论 Monster 实例内容如何，它们之间的比较永远返回 true。

所以当你是用 equals 方法判断对象的内容是否相等，请不要想当然。因为可能你认为相等，而这个类的作者不这样认为，而类的 equals 方法的实现是

由他掌握的。如果你需要使用 equals 方法，或者使用任何基于散列码的集合（HashSet,HashMap,HashTable），请察看一下 java doc 以确认这个类的

equals 逻辑是如何实现的。

问题三：String 到底变了没有？

没有。因为 String 被设计成不可变(immutable)类，所以它的所有对象都是不可变对象。请看下列代码：

String s = "Hello";

s = s + " world!";

s 所指向的对象是否改变了呢？从本系列第一篇的结论很容易导出这个结论。我们来看看发生了什么事情。在这段代码中，s 原先指向一个 String 对象

，内容是"Hello"，然后我们对 s 进行了+操作，那么 s 所指向的那个对象是否发生了改变呢？答案是没有。这时，s 不指向原来那个对象了，而指向了

另一个 String 对象，内容为"Hello world!"，原来那个对象还存在于内存之中，只是 s 这个引用变量不再指向它了。

通过上面的说明，我们很容易导出另一个结论，如果经常对字符串进行各种各样的修改，或者说，不可预见的修改，那么使用 String 来代表字符串的

话会引起很大的内存开销。因为 String 对象建立之后不能再改变，所以对于每一个不同的字符串，都需要一个 String 对象来表示。这时，应该考虑使

用 StringBuffer 类，它允许修改，而不是每个不同的字符串都要生成一个新的对象。并且，这两种类的对象转换十分容易。

同时，我们还可以知道，如果要使用内容相同的字符串，不必每次都 new 一个 String。例如我们要在构造器中对一个名叫 s 的 String 引用变量进行初始

化，把它设置为初始值，应当这样做：

```
public class Demo {
private String s;
…
public Demo {
s = "Initial Value";
}
…
```

}

而非

s = new String("Initial Value");

后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，因为 String 对象不可改变，所以对于内容相同的字符串，只要一个

String 对象来表示就可以了。也就说，多次调用上面的构造器创建多个对象，他们的 String 类型属性s 都指向同一个对象。

上面的结论还基于这样一个事实：对于字符串常量，如果内容相同，Java 认为它们代表同一个 String 对象。而用关键字 new 调用构造器，总是会创建

一个新的对象，无论内容是否相同。

至于为什么要把 String 类设计成不可变类，是它的用途决定的。其实不只 String，很多 Java 标准类库中的类都是不可变的。在开发一个系统的时候，

我们有时候也需要设计不可变类，来传递一组相关的值，这也是面向对象思想的体现。不可变类有一些优点，比如因为它的对象是只读的，所以多线

程并发访问也不会有任何问题。当然也有一些缺点，比如每个不同的状态都要一个对象来代表，可能会造成性能上的问题。所以 Java 标准类库还提供

了一个可变版本，即 StringBuffer。

问题四：final 关键字到底修饰了什么？

final 使得被修饰的变量"不变"，但是由于对象型变量的本质是“引用”，使得“不变”也有了两种含义：引用本身的不变，和引用指向的对象不变

。

引用本身的不变：

```
final StringBuffer a=new StringBuffer("immutable");
final StringBuffer b=new StringBuffer("not immutable");
a=b;//编译期错误
```

引用指向的对象不变：

```
final StringBuffer a=new StringBuffer("immutable");
a.append(" broken!"); //编译通过
```

可见，final 只对引用的“值”(也即它所指向的那个对象的内存地址)有效，它迫使引用只能指向初始指向的那个对象，改变它的指向会导致编译期

错误。至于它所指向的对象的变化，final 是不负责的。这很类似==操作符：==操作符只负责引用的“值”相等，至于这个地址所指向的对象内容是

否相等，==操作符是不管的。

理解 final 问题有很重要的含义。许多程序漏洞都基于此--- final 只能保证引用永远指向固定对象，不能保证那个对象的状态不变。在多线程的操作

中,一个对象会被多个线程共享或修改，一个线程对对象无意识的修改可能会导致另一个使用此对象的线程崩溃。一个错误的解决方法就是在此对象

新建的时候把它声明为 final，意图使得它“永远不变”。其实那是徒劳的。

问题五：到底要怎么样初始化！

本问题讨论变量的初始化，所以先来看一下 Java 中有哪些种类的变量。

1. 类的属性，或者叫值域

2. 方法里的局部变量

3. 方法的参数

对于第一种变量，Java 虚拟机会自动进行初始化。如果给出了初始值，则初始化为该初始值。如果没有给出，则把它初始化为该类型变量的默认初始

值。

int 类型变量默认初始值为 0

float 类型变量默认初始值为 0.0f

double 类型变量默认初始值为 0.0

boolean 类型变量默认初始值为 false

char 类型变量默认初始值为 0(ASCII 码)

long 类型变量默认初始值为 0

所有对象引用类型变量默认初始值为 null，即不指向任何对象。注意数组本身也是对象，所以没有初始化的数组引用在自动初始化后其值也是 null。

对于两种不同的类属性，static 属性与 instance 属性，初始化的时机是不同的。instance 属性在创建实例的时候初始化，static 属性在类加载，也就

是第一次用到这个类的时候初始化，对于后来的实例的创建，不再次进行初始化。这个问题会在以后的系列中进行详细讨论。

对于第二种变量，必须明确地进行初始化。如果再没有初始化之前就试图使用它，编译器会抗议。如果初始化的语句在 try 块中或 if 块中，也必须要

让它在第一次使用前一定能够得到赋值。也就是说，把初始化语句放在只有 if 块的条件判断语句中编译器也会抗议，因为执行的时候可能不符合 if 后

面的判断条件，如此一来初始化语句就不会被执行了，这就违反了局部变量使用前必须初始化的规定。但如果在 else 块中也有初始化语句，就可以通

过编译，因为无论如何，总有至少一条初始化语句会被执行，不会发生使用前未被初始化的事情。对于 try-catch 也是一样，如果只有在 try 块里才有

初始化语句，编译部通过。如果在 catch 或 finally 里也有，则可以通过编译。总之，要保证局部变量在使用之前一定被初始化了。所以，一个好的做

法是在声明他们的时候就初始化他们，如果不知道要出事化成什么值好，就用上面的默认值吧！

其实第三种变量和第二种本质上是一样的，都是方法中的局部变量。只不过作为参数，肯定是被初始化过的，传入的值就是初始值，所以不需要初始

化。

问题六：instanceof 是什么东东？

instanceof 是 Java 的一个二元操作符，和==，>，<是同一类东东。由于它是由字母组成的，所以也是 Java 的保留关键字。它的作用是测试它左边的对

象是否是它右边的类的实例，返回 boolean 类型的数据。举个例子：

```
String s = "I AM an Object!";
boolean isObject = s instanceof Object;
```

我们声明了一个 String 对象引用，指向一个 String 对象，然后用 instancof 来测试它所指向的对象是否是 Object 类的一个实例，显然，这是真的，所

以返回 true，也就是 isObject 的值为 True。

instanceof 有一些用处。比如我们写了一个处理账单的系统，其中有这样三个类：

```
public class Bill {//省略细节}
public class PhoneBill extends Bill {//省略细节}
public class GasBill extends Bill {//省略细节}
```

在处理程序里有一个方法，接受一个 Bill 类型的对象，计算金额。假设两种账单计算方法不同，而传入的 Bill 对象可能是两种中的任何一种，所以要

用 instanceof 来判断：

```
public double calculate(Bill bill) {
if (bill instanceof PhoneBill) {
//计算　　账单
}
if (bill instanceof GasBill) {
//计算燃气账单
}
…
}
```

这样就可以用一个方法处理两种子类。

然而，这种做法通常被认为是没有好好利用面向对象中的多态性。其实上面的功能要求用方法重载完全可以实现，这是面向对象变成应有的做法，避

免回到结构化编程模式。只要提供两个名字和返回值都相同，接受参数类型不同的方法就可以了：

```
public double calculate(PhoneBill bill) {
//计算　　账单
}

public double calculate(GasBill bill) {
//计算燃气账单
}
```

所以，使用 instanceof 在绝大多数情况下并不是推荐的做法，应当好好利用多态。

java 方向及学习方法

java 分成 J2ME（移动应用开发），J2SE（桌面应用开发），J2EE(Web 企业级应用)，所以 java 并不是单机版的，只是面向对象语言。建议如果学习

java 体系的话可以这样去学习：

*第一阶段：Java 基础，包括 java 语法，面向对象特征，常见 API，集合框架；

*第二阶段：java 界面编程，包括 AWT，事件机制，SWING，这个部分也可以跳过，用的时候再看都能来及；

*第三阶段：java API：输入输出，多线程，网络编程，反射注解等，java 的精华部分；

*第四阶段：数据库 SQL 基础，包括增删改查操作以及多表查询；

*第五阶段：JDBC 编程：包括 JDBC 原理，JDBC 连接库，JDBC API，虽然现在 Hibernate 比 JDBC 要方便许多，但是 JDBC 技术仍然在使用，JDBC 思想尤为重

要；

*第六阶段：JDBC 深入理解高级特性：包括数据库连接池，存储过程，触发器，CRM 思想；

*第七阶段：HTML 语言学习，包括 HTML 标签，表单标签以及 CSS，这是 Web 应用开发的基础；

*第八阶段：JavaScript 脚本语言，包括 javaScript 语法和对象，就这两个方面的内容；

*第九阶段：DOM 编程，包括 DOM 原理，常用的 DOM 元素以及比较重要的 DOM 编程思想；

*第十阶段：Servlet 开发，从此开始踏入 java 开发的重要一步，包括 XML，Tomcat 服务器的安装使用操作，HTTP 协议简单理解，Servlet API 等，这个

是 java web 开发的基础。

*第十一阶段：JSP 开发：JSP 语法和标签，自定义标签，EL,JSTL 库了解以及 MVC 三层架构的设计模式理念；

*第十二阶段：AJAX 开发：AJAX 原理，请求响应处理，AJAX 开发库；

*第十三阶段：轻量级框架，三大框架之一 Struts 框架的学习，自此踏入 java web 开发的精华部分，包括 Struts 体系架构，各种组件，标签库和扩展

性的学习；

*第十四阶段：Hibernate 框架学习，三大框架之一，包括检索映射技术，多表查询技术，缓存技术以及性能方面的优化；

*第十五阶段：Spring 框架的学习，三大框架之一，包括了IOC,AOP,DataSource，事务，SSH 集成以及 JPA 集成；

*最后呢，还有些 java 的技术，包括 EJB3.0 等，可以选择学习，与三大轻量级框架相比，EJB 就是当之无愧的重量级了。

面向对象编程遇到的问题汇总

java 自学 day2

对象：java 是面向对象的编程，对象就相当于我们生活中的一个具体的事物，比如一个手机，它拥有尺寸，颜色，cpu，内存等属性，还拥有上网，

打　　等功能。在 java 中，对象也拥有数据和方法。

类：类就是一组拥有相同属性和相同方法的对象的集合。

成员变量和局部变量

成员变量：在类的方法外面，能被类中所有的方法使用，有默认初始值，用来定义对象有什么属性

局部变量：在类的方法里面，只能被本方法使用，无默认初始值，需要赋值，用来临时保存数据

构造方法

构造方法是定义在 java 类中的一个用来初始化对象的方法，使用 new+构造方法创建一个新的对象。

构造方法名称与类名相同并且没有返回值。

构造方法可以给成员变量赋值。

还可以用初始化块为成员变量赋值，形式如下：

{ a=1　}

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：https://d.book118.com/335200001134011041