## 常量池梳理及相关练习

详情见CSDN

1. 常量池的分类及含义
2. 静态常量池

所谓静态常量池，即\*.class文件中的常量池（Constant pool部分），占用class文件绝大部分空间，这种常量池主要用于存放两大类常量：字面量(Literal)和符号引用量(Symbolic References)，

* 字面量包括：文本字符串、被声明为final的常量值，基本数据类型，通常表现为等号的右值

如：Integer a = 2;

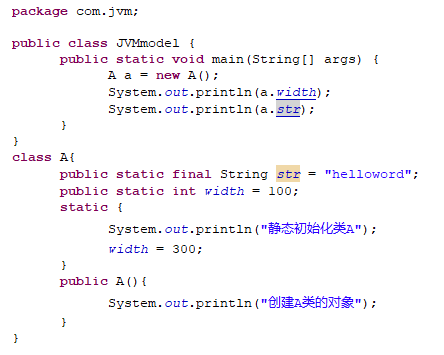
2就是字面量

* 符号引用量包括：类和结构的完全限定名、字段名称和描述符、方法名称和描述符

如：Integer a = 2;

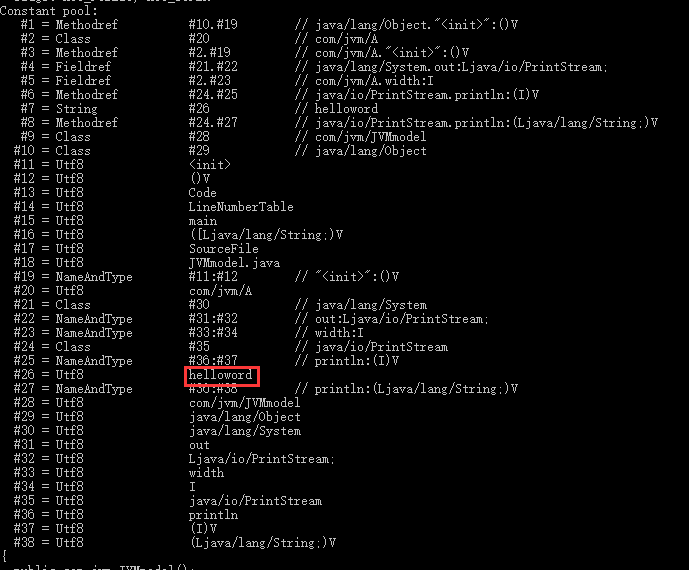
Integer 和 a就是符号引用

下面以简单类进行讲解



|  |  |  |
| --- | --- | --- |
| 字面量（其实就是一些变量的具体值） | 文本字符串 | Helloword（因为在A类中是用final限定的，所以编译阶段直接进入JVMmodel的常量池） |
| 被声明为final的常量值 |
| 基本数据类型 | 无 |
| 符号引用量（其实就是名字：类名，方法名，变量名） | 类和结构的完全限定名 | 包名+类名 如上图中import之后  JVMmodel、A、a、System |
| 字段名称 | Width、str |
| 方法名称 | Out、println |
| 描述符 | Public、static、 |

注意：Helloword因为是用final限定的，所以在编译阶段就会进入到JVMmodel的常量池中



同理，A的常量池为

|  |  |  |  |
| --- | --- | --- | --- |
| 字面量（其实就是一些变量的具体值） | 文本字符串 | helloword | 静态初始化类A、创建A类的对象 |
| 被声明为final的常量值 |  |
| 基本数据类型 | 100、300 | |
| 符号引用量（其实就是名字：类名，方法名，变量名） | 类和结构的完全限定名 | 包名+类名 如上图中import之后  A、System | |
| 字段名称 | Width、str | |
| 方法名称 | Out、println | |
| 描述符 | Public、static、final | |



Class文件在加载到JVM中后，会变成运行常量池和字符串常量池

1. 运行时常量池和字符串常量池
2. 概念

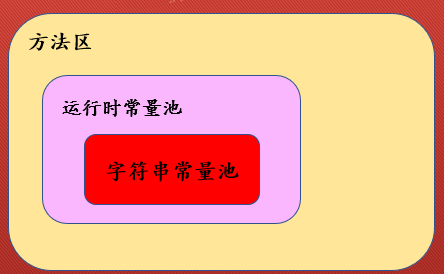
运行时常量池（Runtime Constant Pool） 是方法区的一部分。 Class文件常量池里的内容将在类加载后存放到方法区的运行时常量池中，并在类加载的解析阶段，将符号引用转换为直接引用，指向真正的内存地址

字符串常量池里的内容是在类加载之后，代码执行在堆中生成字符串对象实例，然后将该字符串对象实例的引用值存到string pool中，被所有的类共享。

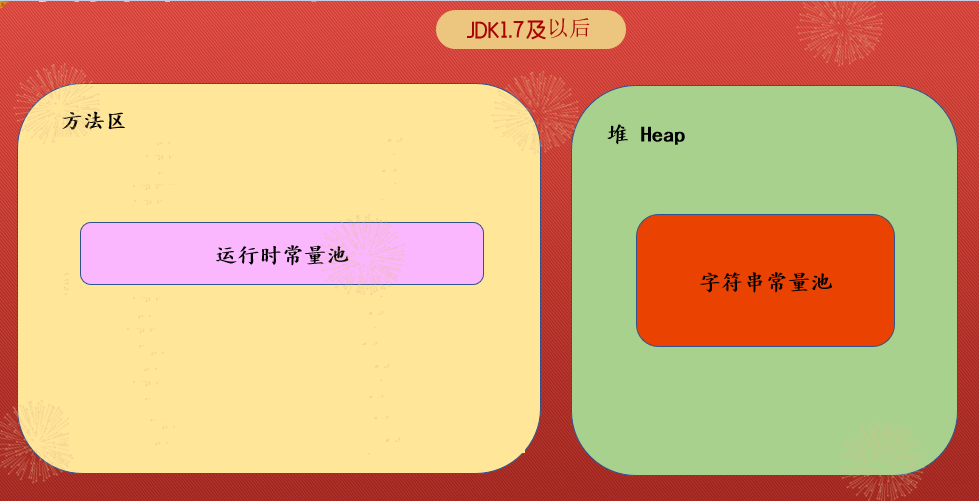
需要注意的是运行时常量池是静态常量池加载到JVM后形成的，但是字符串常量池是代码运行过程中产生的字符串示例对象；

1. 位置

在JDK 6或更早之前常量池都是分配在永久代中

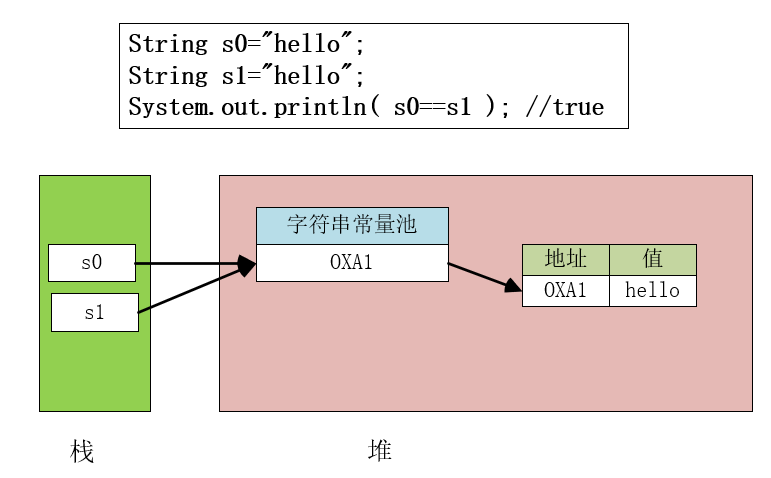


JDK7及以前使用永久代代替方法区，JDK8使用元空间来代替方法区，不管怎样JDK7及以后运行时常量池存储在方法区，字符串常量池存储在堆中



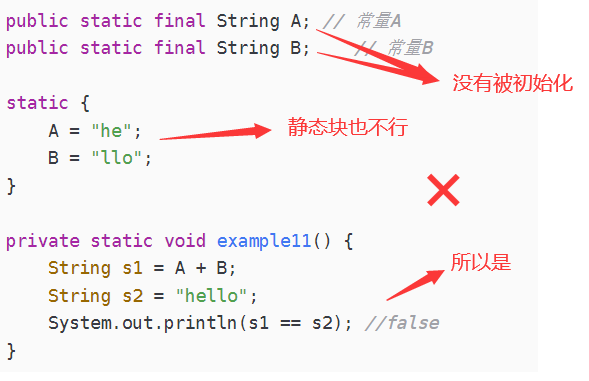
1. 字符串常量区详解
2. 直接赋值方式

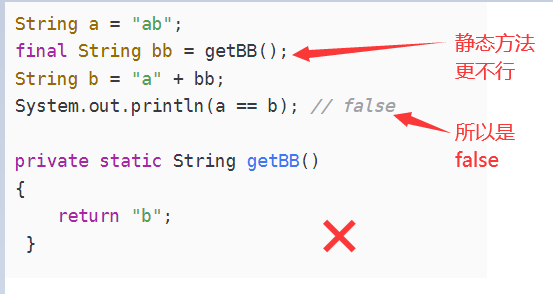
* 使用字面量赋值，只会直接分配到字符串常量池中。如果字符串常量池中存在则直接返回，如果不存在则先创建，然后返回。
* 这里需要注意字符串常量池中存储的是引用值而不是实例对象，具体的实例对象存储在堆中。

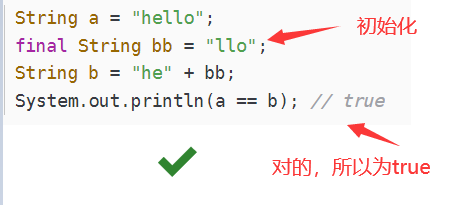


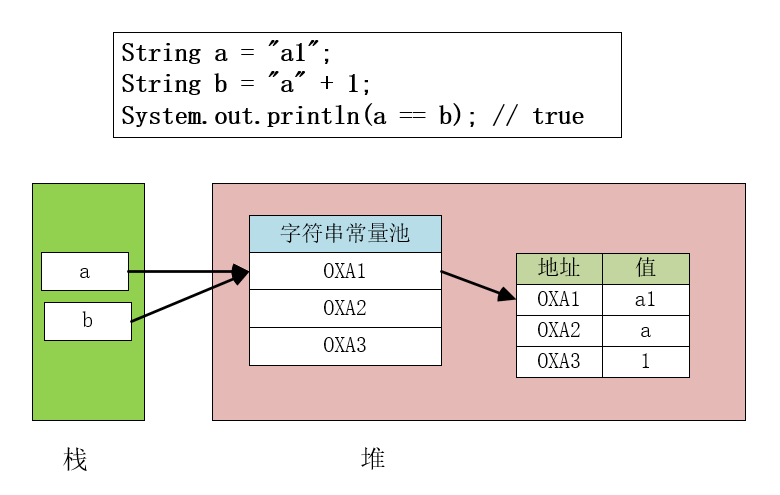
* 如果是字面量或被声明为final的符号引用之间相加，会被认为是字符串常量池中的字符串相互拼接
* **注意：这种情况下，final修饰的符号引用量必须被字面量初始化！！！，否则不适用**

**如：**



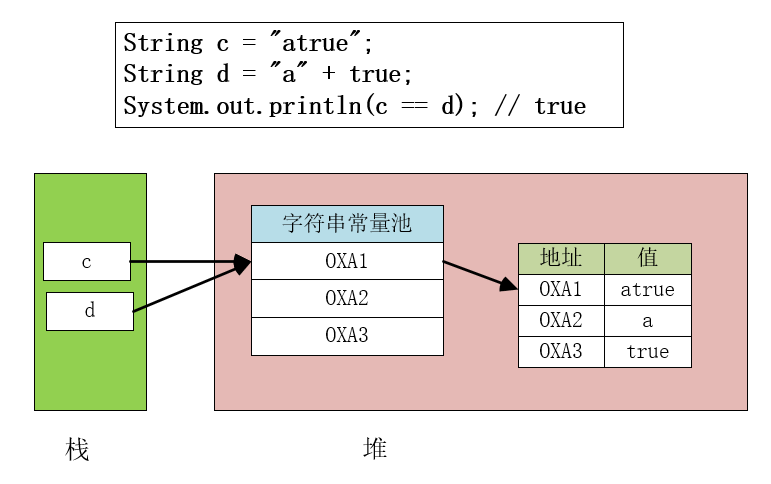






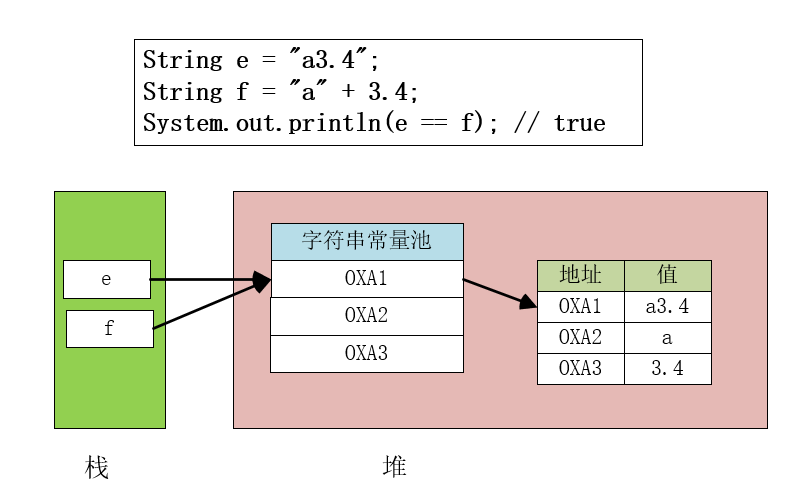
说明：

* 首先，字面量为：a1,a,1
* 在编译阶段，”a” + 1时，会自动的将1转化为字符串常量，放置在字符串常量池中
* 所以会检测字符串常量池中是否存在字面量：a1,a,1，没有则添加
* 然后会将栈中的a指向字符串常量池中的a1
* 然后会将字符串常量池中的a和1进行拼接，为a1，并检测字符串常量池中是否有a1，没有则添加
* 然后会将栈中的b指向字符串常量池中的a1

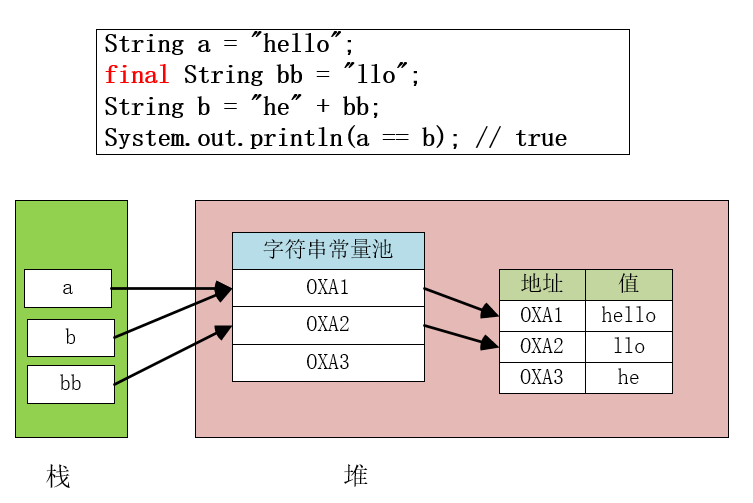


说明：

* 同理，字面量是：atrue、a、true
* 在编译阶段，”a” + true,会自动的将true转化为字符串常量，放置在字符串常量池中
* 所以会检测字符串常量池中是否存在字面量：atrue、a、true，没有则添加
* 然后会将栈中的c指向字符串常量池中的atrue
* 然后会将字符串常量池中的a和true进行拼接，为atrue，并检测字符串常量池中是否有atrue，没有则添加
* 然后会将栈中的c指向字符串常量池中的atrue



说明： 以上同理

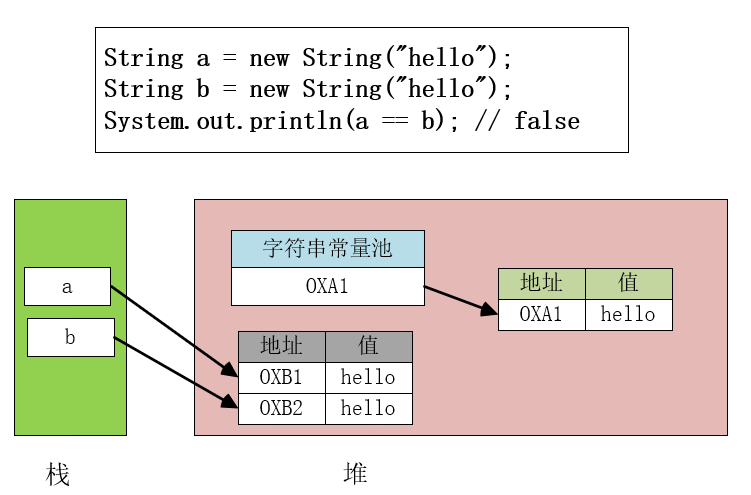


说明：

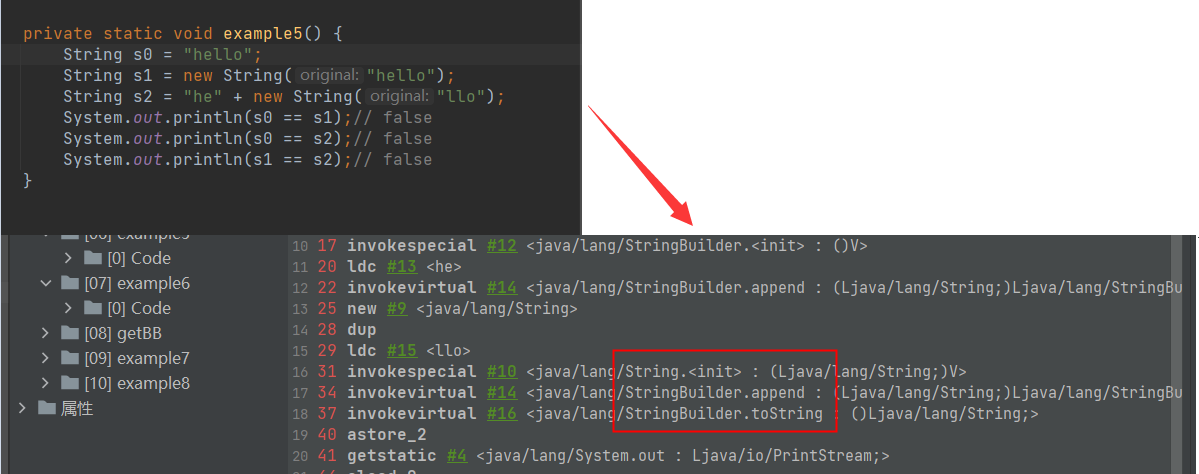
* 与被final修饰的符号引用量相加，可以认为是直接和字面量相加，所以具体过程如下：
* String a = “hello” : 先检测字符串常量池中是否有”hello”这个字符串，没有则添加
* 然后会将栈中的a指向字符串常量池中的hello
* final String bb = "llo"：先检测字符串常量值中是否有”llo”这个字符串，没有则添加
* 然后会将栈中的bb指向字符串常量池中的llo
* String b = "he" + bb：如果符号引用量bb不是被final修饰，就需要按照下面的讲解，但是被final修饰，就可以直接认为是和字面量”llo”相加,为hello
* 然后会将栈中的b指向字符串常量池中的hello

1. new String(“xxxx”)创建方式

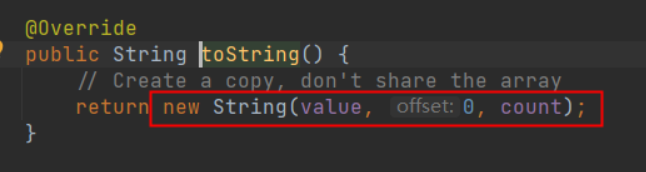
* 首先会在字符串常量池中，检测new String(“xxx”)中的字面量是否存在，没有则创建
* 然后会在堆中开辟一块空间，创建String对象，并将地址返回给栈



* 这里需要特别说一下，涉及非final声明的符号引用量,以及new String(“xxx”)的相加，不管它两其中一个或者任意一个，只要位于 ”+”号的左边或者右边，底层实际使用的是StringBuilder.append进行拼接，然后再使用StringBuilder.toString返回，我们可以使用工具来进行观察

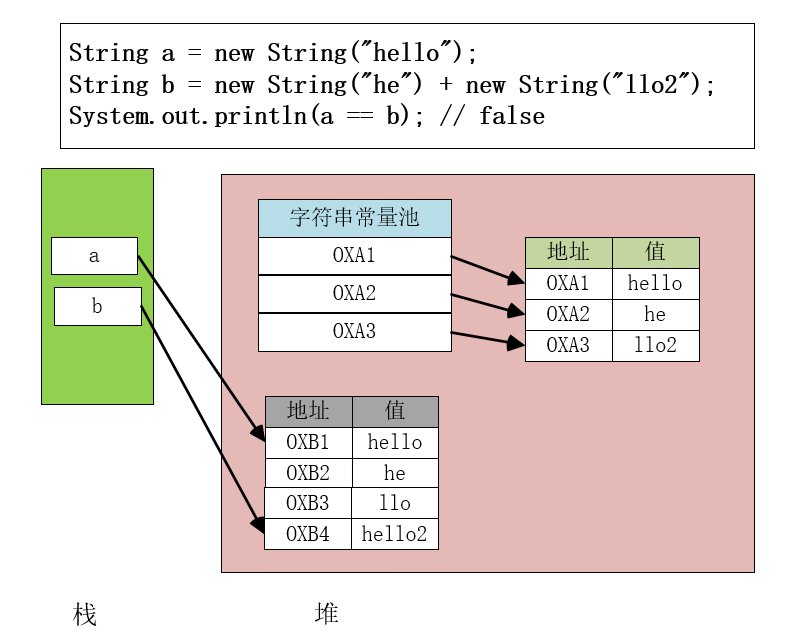


* 再来看下StringBuilder类的toString源码，底层还是一个new String操作，注意这个new String()是不产生字面量的。



* 所以，涉及非final声明的符号引用量或者new String()的相加就等于new String()的效果

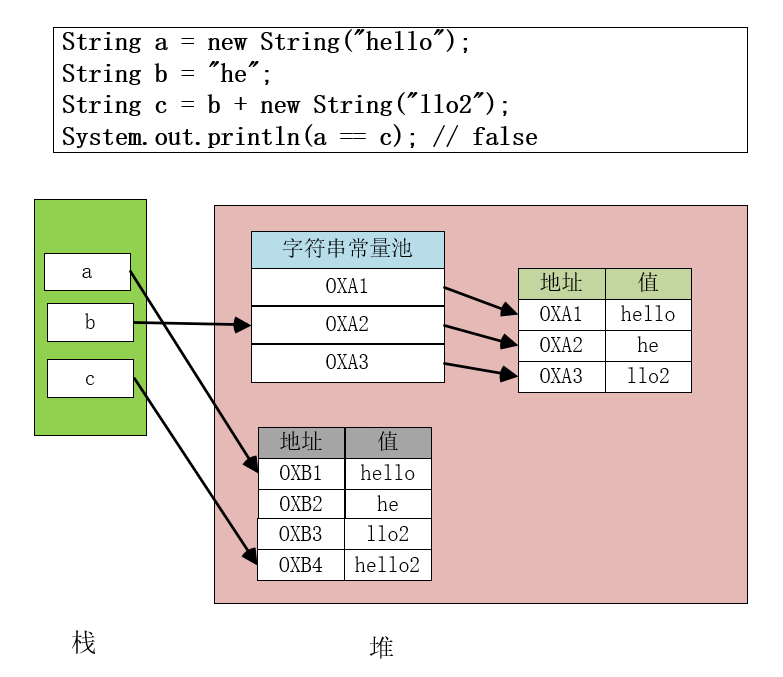
1. new String(“xxx”)和new String(“yyy”)相加



说明：

* String a = new String("hello"):
* 字面量：hello,首先会检测字符串常量池中是否有hello，没有则添加到字符串常量池中
* 然后在堆中创建对象new String(“hello”),并将地址返回到栈中
* String b = new String("he") + new String("llo2");
* 字面量：he 和llo2,首先会检测字符串常量池中是否有he和llo2，没有则添加到字符串常量池中
* 然后会在堆中创建new String(“he”)对象和new String(“llo2”)对象
* 然后就是new String("he") + new String("llo2"),根据上述说明，是使用new String()拼接，但是不会产生拼接后的字面量hello2，直接在堆中创建new String(“hello2”),将引用返回到栈中；

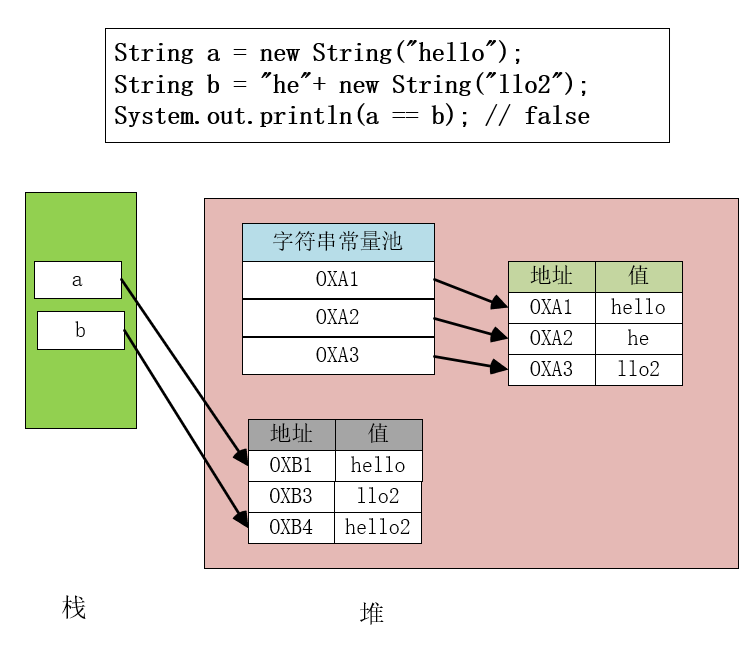
1. new String(“xxx”)和符号引用量相加



说明：

* String a = new String("hello");
* 字面量：hello,首先会检测字符串常量池中是否有hello，没有则添加到字符串常量池中
* 然后在堆中创建对象new String(“hello”),并将地址返回到栈中
* String b = "he";
* 字面量：he,首先会检测字符串常量池中是否有he，没有则添加到字符串常量池中
* 由于采用的是直接赋值，所以会直接将字符串常量池中的地址返回到栈中
* String c = b + new String("llo2");
* 字面量：llo2,首先会检测字符串常量池中是否有llo2，没有则添加到字符串常量池中
* 然后在堆中创建对象new String(“llo2”)
* 然后就是 b + new String("llo2")，根据上述说明，是使用new String()拼接，但是不会产生拼接后的字面量hello2，直接在堆中创建new String(“hello2”),将引用返回到栈中；

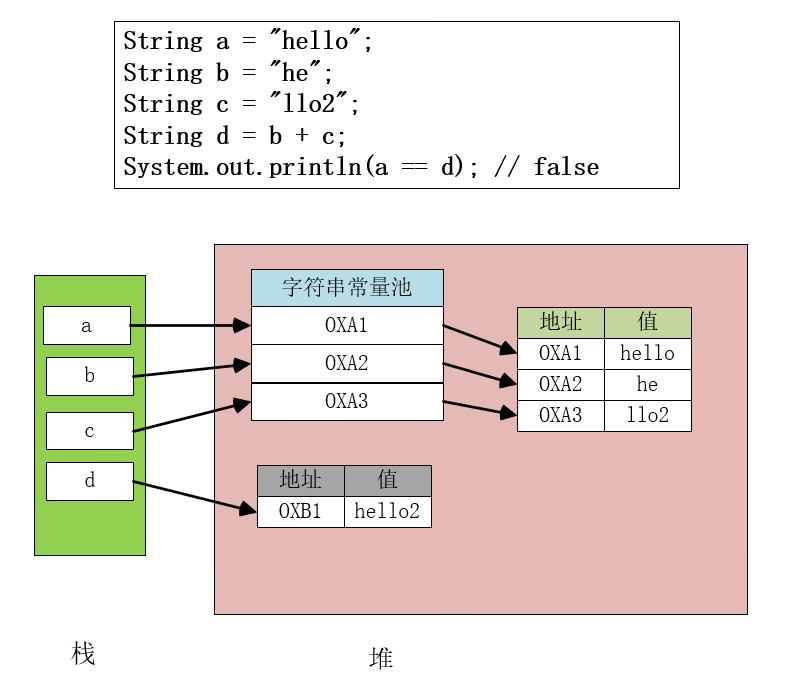
1. new String(“xxx”)和字面量相加



说明：

* String a = new String("hello");
* 字面量：hello,首先会检测字符串常量池中是否有hello，没有则添加到字符串常量池中
* 然后在堆中创建对象new String(“hello”),并将地址返回到栈中
* String b = "he"+ new String("llo2");
* 字面量：he和llo2,首先会检测字符串常量池中是否有he和llo2，没有则添加到字符串常量池中
* 然后在堆中创建对象new String(“llo2”)
* 然后就是"he"+ new String("llo2")，根据上述说明，是使用new String()拼接，但是不会产生拼接后的字面量hello2，直接在堆中创建new String(“hello2”),将引用返回到栈中

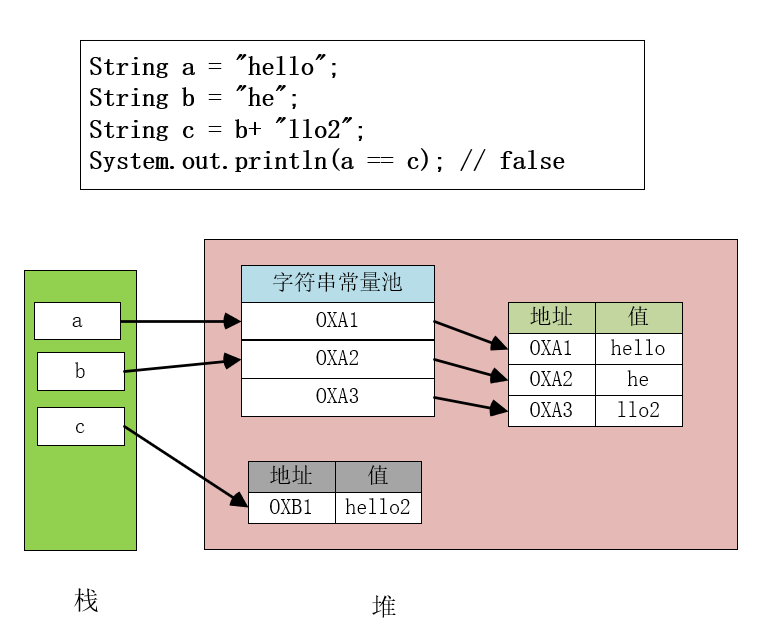
1. 符号引用量和符号引用量相加



说明：

* String a = "hello";String b = "he";String c = "llo2";
* 字面量：hello,he,llo2,首先会检测字符串常量池中是否有hello,he,llo2，没有则添加到字符串常量池中
* 由于是直接赋值方式，会将字符串常量池中的地址直接返回到栈中
* String d = b + c;
* 根据上述说明，是使用new String()拼接，但是不会产生拼接后的字面量hello2，直接在堆中创建new String(“hello2”),将引用返回到栈中

1. 符号引用量和字面量相加



说明：

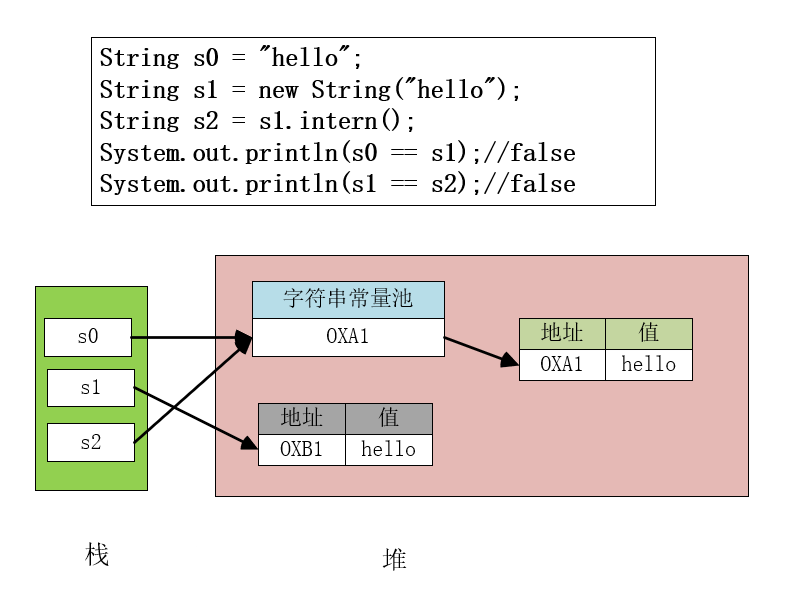
* String a = "hello";String b = "he";
* 字面量：hello,he,首先会检测字符串常量池中是否有hello,he，没有则添加到字符串常量池中
* String c = b+ "llo2";
* 字面量：llo2,首先会检测字符串常量池中是否有llo2，没有则添加到字符串常量池中
* 然后b+ "llo2"：根据上述说明，是使用new String()拼接，但是不会产生拼接后的字面量hello2，直接在堆中创建new String(“hello2”),将引用返回到栈中

1. 字面量和字面量相加

见上

1. Intern方法创建

intern方法，返回String对象所对应的字面量在常量池中的引用，如果常量池中没有，则添加到常量池中，并返回常量池中的引用；



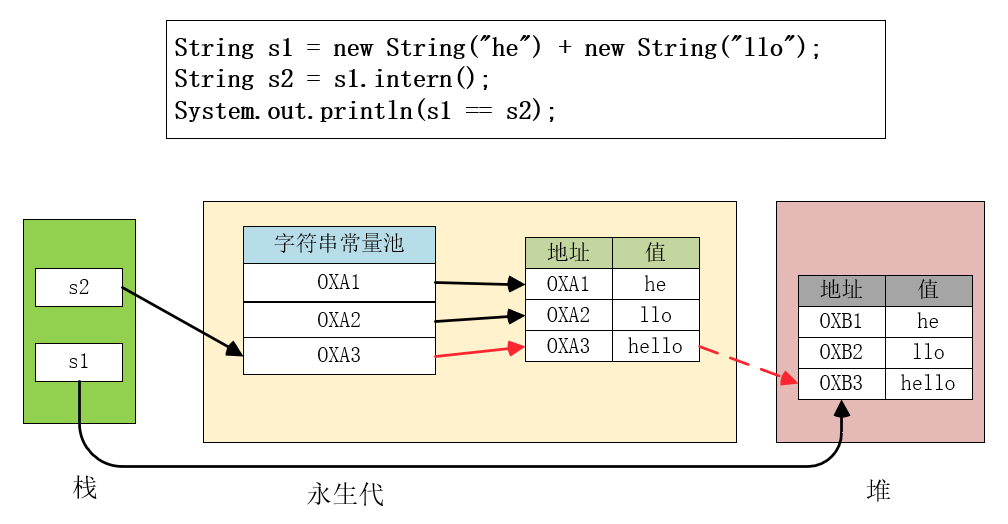
说明：

* String s0 = "hello";
* 字面量：hello,首先会检测字符串常量池中是否有hello，没有则添加到字符串常量池中
* String s1 = new String("hello");
* 字面量：hello,首先会检测字符串常量池中是否有hello，没有则添加到字符串常量池中
* 然后在堆中创建对象new String(“hello”),并将地址返回到栈中
* String s2 = s1.intern();
* 根据上面规则，首先会检测字符串常量池中是否有对应的字面量：”hello”，返回在常量池中的引用

注意，存在一种情况，堆对象的intern，但是堆对象所对应的字面量，并不存在于字符串常量池中，就需要分两种情况来考虑

* JDK1.6及之前：

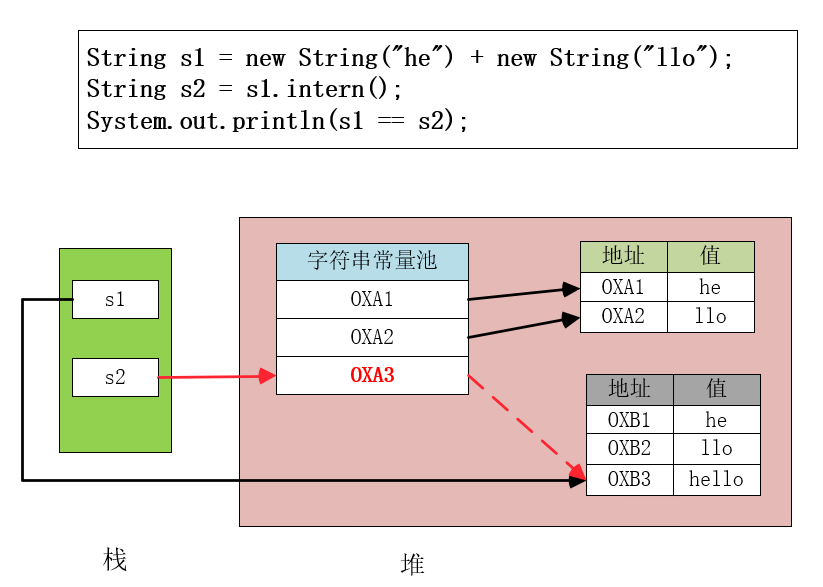
在JDK1.6及之前，常量池是存在于永生代中的，所以在执行堆对象的Intern()方法后，会检测字符串常量池是否含有堆对象所对应的字符串常量，没有则会在永生代中创建，并将引用放到字符串常量池中，返回到栈中



说明：

* String s1 = new String("he") + new String("llo");
* 字面量：he 和llo,首先会检测字符串常量池中是否有he和llo，没有则添加到字符串常量池中
* 然后会在堆中创建new String(“he”)对象和new String(“llo”)对象
* 然后就是new String("he") + new String("llo"),根据上述说明，是使用new String()拼接，但是不会产生拼接后的字面量hello，直接在堆中创建new String(“hello”),将引用返回到栈中；
* String s2 = s1.intern();
* 首先会检测s1对应地字面量”hello”,是否存在于永生代中的字符串常量池中，没有则在永生代中创建，并将引用放到字符串常量池中，返回到栈中
* JDK1.7及以后

在JDK1.7以后，由于取消了永生代概念，并且将常量池放入到了堆中，所以在执行堆对象的Intern()方法后，会检测字符串常量池是否含有堆对象所对应的字符串常量，没有就会将堆对象的地址放入到字符串常量池中，并将字符串常量池中的引用，返回到栈。



说明：

* String s1 = new String("he") + new String("llo");
* 字面量：he 和llo,首先会检测字符串常量池中是否有he和llo，没有则添加到字符串常量池中
* 然后会在堆中创建new String(“he”)对象和new String(“llo”)对象
* 然后就是new String("he") + new String("llo"),根据上述说明，是使用new String()拼接，但是不会产生拼接后的字面量hello，直接在堆中创建new String(“hello”),将引用返回到栈中；
* String s2 = s1.intern();
* 会检测字符串常量池是否含有堆对象所对应的字符串常量，没有就会将堆对象的地址放入到字符串常量池中，并将字符串常量池中的引用，返回到栈。

1. 基本数据类型的包装类的池技术

在8种基本类型种，除float和double，其他六种基本类型对应的包装类都应用了池技术（Byte，Short，Integer，Long，Character，Boolean）

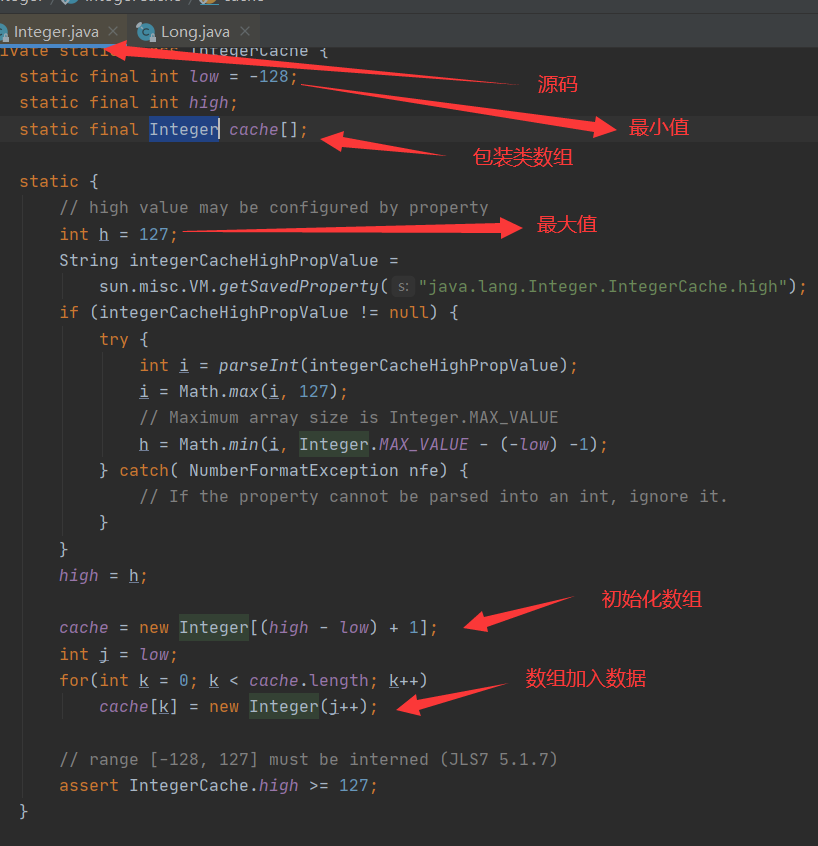
池技术大致分为三个技术要点：常量池，自动装箱以及自动拆箱

* 常量池

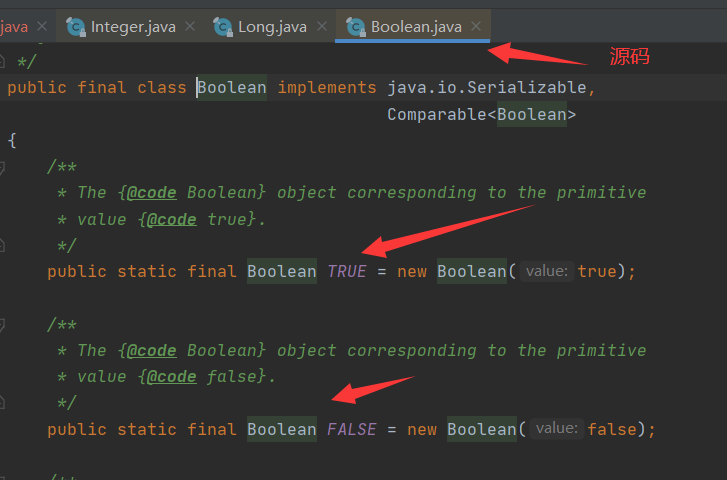
在包装类进行JVM加载的时候，会用静态块/静态变量初始化一个包装类数组，数组内部的数据是该包装类被常用到的数据，我们称之为常量池或者包装类缓存。

如：Integer/Long类型，数组范围为：-128~127，Boolean类型为true/false

源码截图如下：

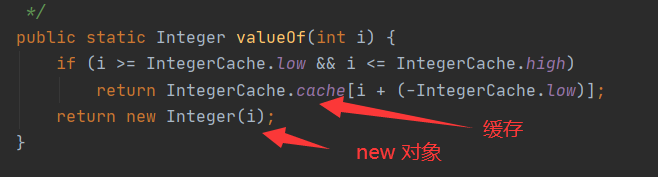


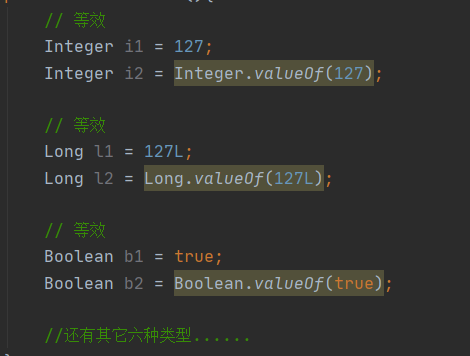




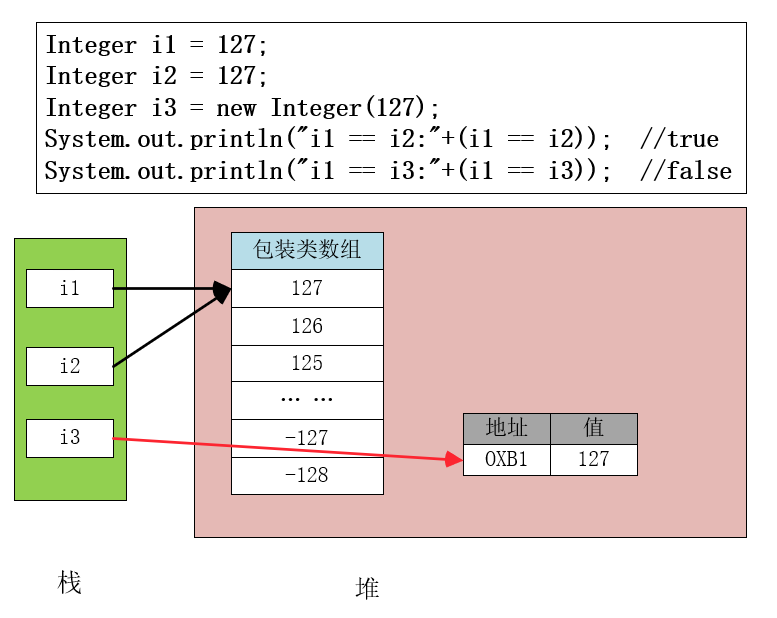
* 自动装箱

自动装箱往往发生在用字面量给包装类进行赋值时，底层实际调用的是valueOf()方法，如果该字面量位于包装类的缓存数组的范围中，则返回该数组中对应的对象；如果该字面量不位于包装类的缓存数组的范围中，则返回堆中的new 对象





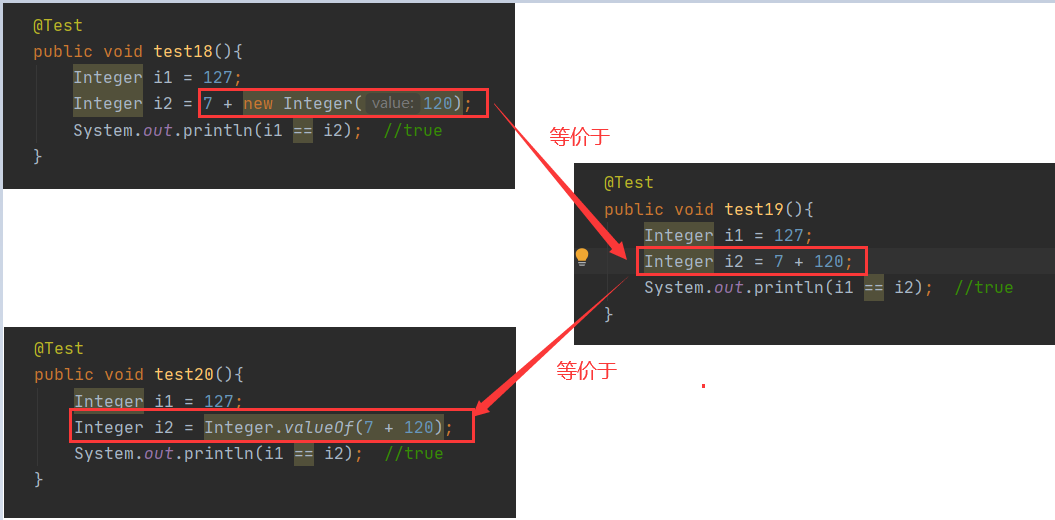
当然在使用new 包装类的方式进行赋值，即使数据范围位于包装类数组缓存的范围，也会返回堆中新创建的对象，而不是数组中的对象。



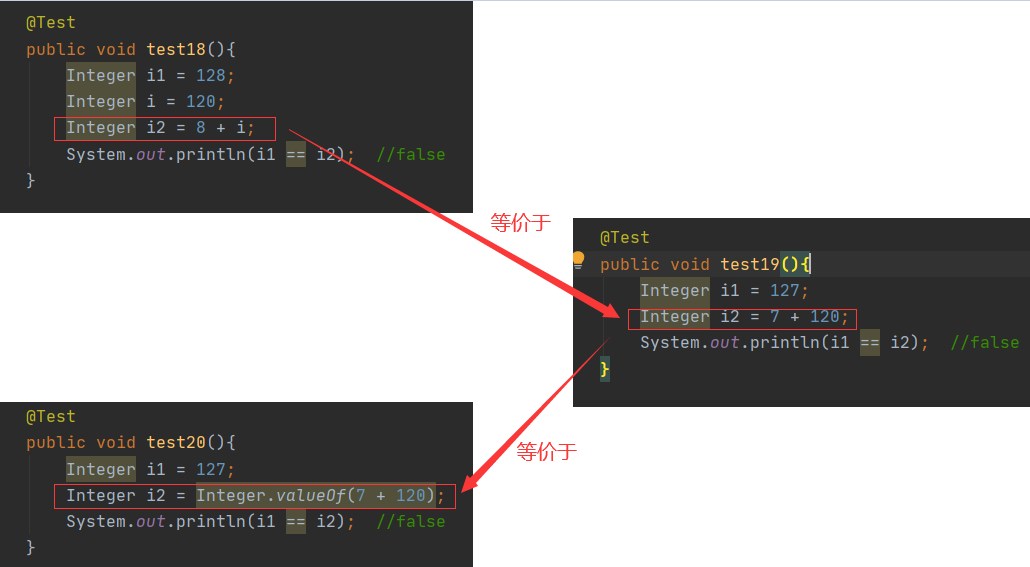
* 自动拆箱

当包装类型参与数学数学运算（加、减、乘、除、与、或、非等）（包装类型之间、包装类型和基本类型），包装类型会调用intValue()方法自动转为基本类型，然后参与运算，包装类型自动转为基本类型的过程叫自动拆箱；

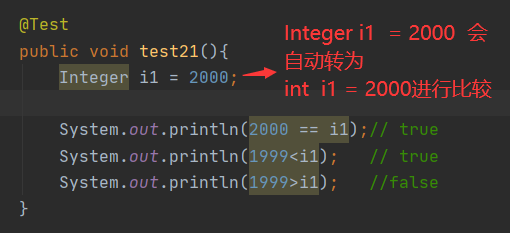
在计算完后，又会调用valueOf()方法，进行自动装箱(**见上面的自动装箱理论**)



下面是因为超过了自动装箱的范围（Integer -128~127），所以报false



当然还有一种自动拆箱发生在基本数据类型和包装类的比较（等于、小于、大于等），包装类会自动转为基本数据类型进行比较



包装类和包装类的比较，等于(==)不会发生自动拆箱,大于(>)小于(<)会发生自动拆箱,即包装类和包装类进行等于比较的时候，不会转为基本数据类型，但是进行大于小于比较的时候会转为基本数据类型进行比较

