JAVA方法传值（从虚拟机的角度思考问题）

### 赋值的流程

1. **基本类型的赋值计算**

基本类型有int，short，boolean, byte，char, long, float, double

以int 赋值为例：

Int num = 10;

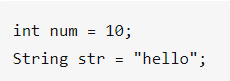
1. **结构体赋值计算**

结构体(我个人对复杂结构的称谓，非官方的称谓)的赋值一般是建立对应的variable标签，在内存中分配variable所需要的内存，然后用箭头把标签和内存对应的地址连接起来

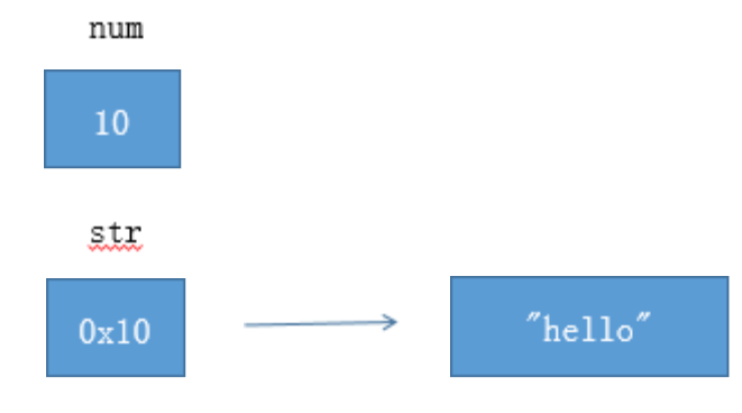
常见的结构体主要有array，string，我们自己定义的class类

以string为例

String str = “hello”;

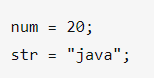


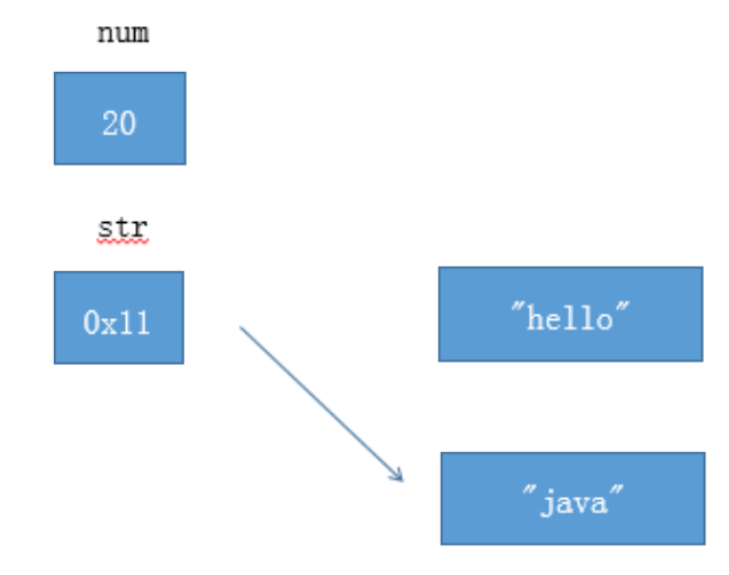
**在java中大概的情形:**



注：这里的0x10代表的是java运行的时候给string这个variable分配的地址

假设我们在这两行后，直接重新去用”=“去赋值一个新的



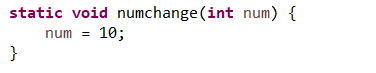


注：JAVA的虚拟机存在自动的GC功能（garbage collection），会把没有箭头指向的variable给删除掉，所以经过这轮赋值后，”hello”就从内存中被移除了

这里对于基本类型，int就是简单地改一下值，但是对于string这种复杂的引用体（可以把string这种东西看成是char array的变形哦），我们赋值后，str的地址改变了，它指向了另一个东西，注意，这里的箭头非常重要，箭头表示的是指向）

1. **method赋值计算**

定义一个function，来改变int类型这个variable的值看看有什么样的结果：



接下来在main里面去call这个function，parameter就是之前的那个num，然后我们把num给print出来，**可是num的结果是20，并没有发生变化**

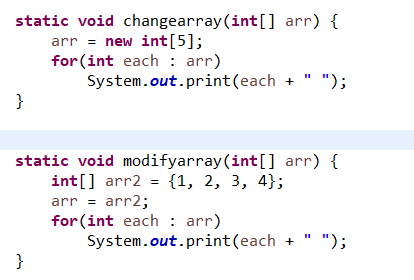
**类似的，我们可以把同样的方法运用到array上：**

先定义一个 int array:

int arr= new int[10];

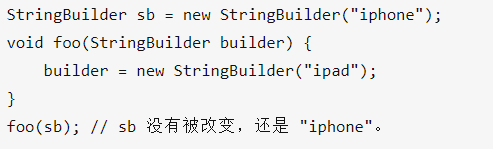
那么这里应该是一个有10位的int类型array

然后定义两个方法，都是把刚刚定义的10index的array给传输进去

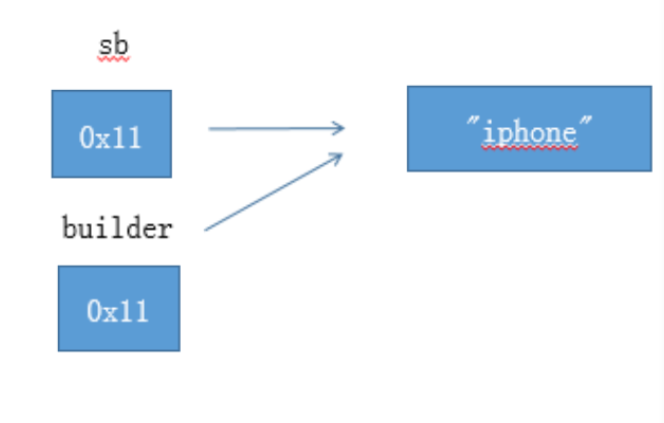


然而结果是，在执行到function里面的这个array的for each loop产生的结果是改变后的，但是等到我们去call function后，我们去在main里面再去print出来array，它的值并没有改变

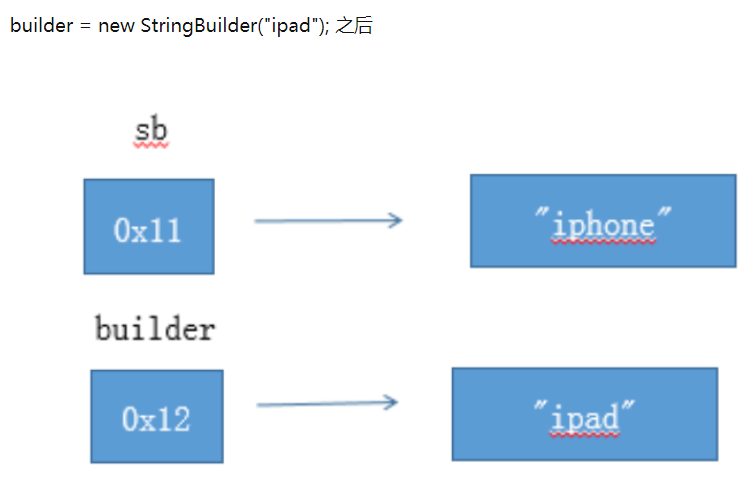
**另一个关于string的example:**



赋值的流程图：

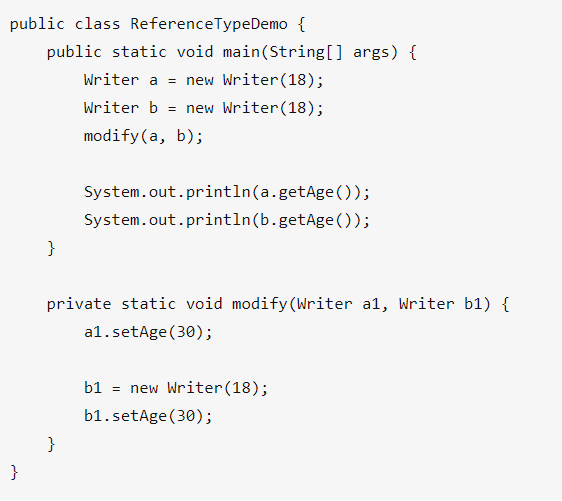


call method的时候，java会建立一个原来variable的副本，然后两个variable会都指向”iphone“这个string，注意看这两个地址目前还是一样的，指向同一个东西

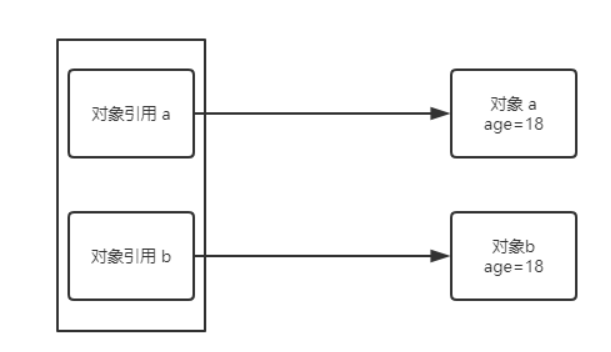


在执行method的时候，出现了这样的情况，什么意思，我去initialize一个新的stringbuilder = ”ipad“给builder，也就是我们的副本，然后，”=“的作用就出现了，等于号，就是在这里把”ipad“这个string的地址付给了builder，所以现在两个variable指向的位置不一样了，而且，原来的variable的值并没有发生改变，所以在我执行完这个method后，假设我们再想print之前的sb的值，它还是”iphone“，也就是说，假设你在function去用等于号给副本variable去赋值，是无法改变原来的main 里面定义的variable的值的

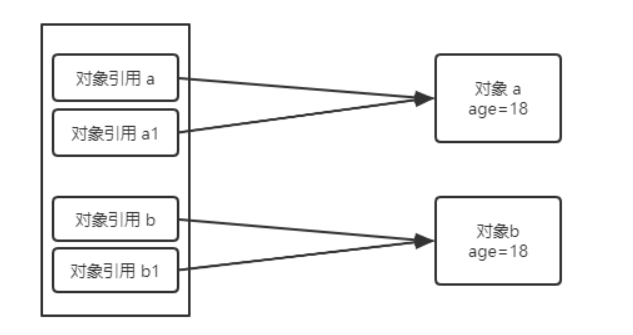
关于class的example:



writer是我定义的一个class，我现在建立了2个instance variable，一个是writer a，一个是writer b，然后哦我们还定义了modify这个方法用来改writer的值，而且parameter的type就是writer, 刚开始，a和b的年龄都是18，我们有下图：



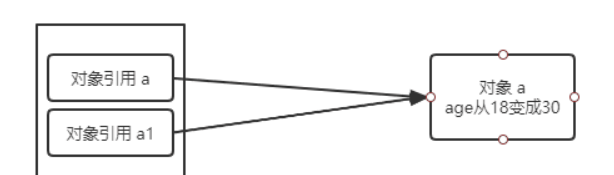
在我们开始call method，modify之后，一开始call这个method，系统就开始给两个variable，a和b分别建立了一个副本，在这里我们称呼为a1和b1



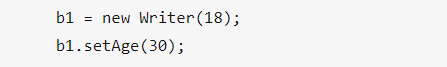
在系统读取了如下代码后，把a1的age进行了改变



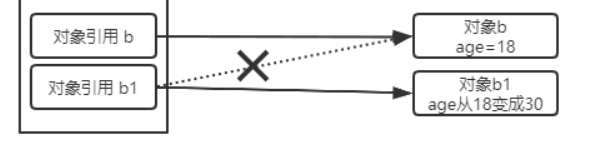
注意这里我们没有用刚刚我说的等号哦，也就是说，这里a和a1仍然会指向之前的那个age，setAge就会随着图上的箭头，沿着a1找到a1的age，把它悄悄地从18改成30，但是因为a和a1是共享age的，所以a的age也被改了，那么我们就得到了下图：



而对于b和b1，系统执行了这个步骤：



先进行了赋值运算，再setAge，等号意味着，标签的箭头指向发生了改变，所以b1，就发生了这样的情况：

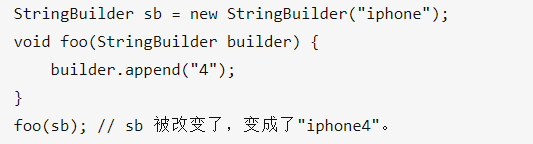


本身两个都是箭头指向18，然后我们给了b1一个等号，把一个新的地址赋值给了b1，然后b1的箭头就变成下面那个了，再然后，setAge，我们沿着箭头从b1找到b1的新age，然后把18改成了30，但是这里b的age仍然是18！！！，只是副本的值发生了改变

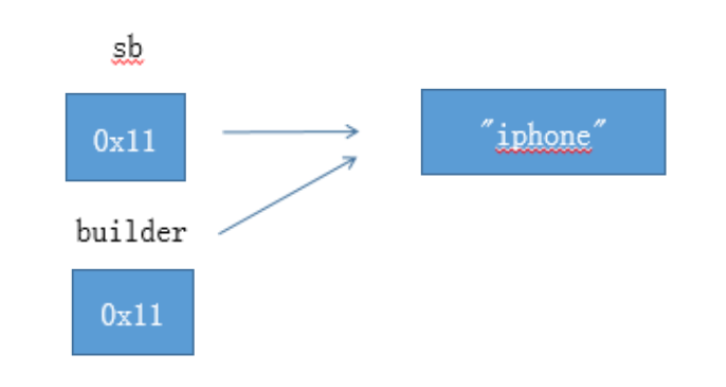
所以我们回去在main里面再分别把a和b的年龄print out出来，结果：a.age = 30，b.age = 18

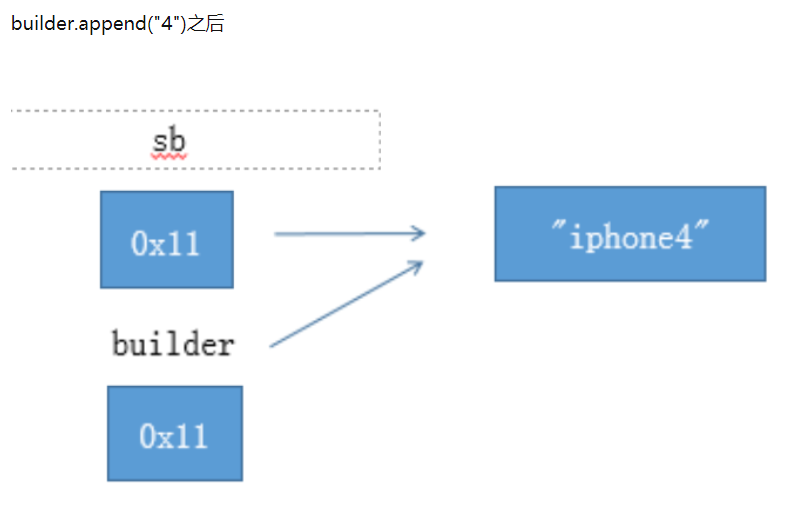
### 总结：

1. method传值相当于给原来的variable建立了一个副本，刚开始副本和原来的variable指向同一个内容
2. 如果在method中，对副本variable进行了赋值运算，那么不会改变原来的main中的variable，赋值运算相当于让副本variable的箭头指向了其他内容，所以这个时候对于副本进行修改无法改变原来的variable的箭头和对象的内容
3. 只有在副本和varibale指向同一内容的时候我们沿着箭头去改变箭头指向的对象，我们才能真正改变原来的variable，比如说java自带的Arrays.sort()函数，或者如下例子：



假设你对string进行append操作，那么会沿着箭头去把那个对象后面增加内容，所以副本和原先的variable会同时改变





对于class variable，如果你用modifier来重新set一个variable的某个attribute，就可以同时让副本和原先的variable同时改变

### 测试代码

public class Variabletest {

static int num = 20;

static String str = "Hello";

static int[] array;

public static void main(String args[]) {

num = 30;

str = "World";

array = new int[10];

numchange(num);

strchange(str);

System.out.println(num);

System.out.println(str);

changearray(array);

System.out.println();

for(int each : array)

System.out.print(each + " ");

System.out.println();

modifyarray(array);

System.out.println();

for(int each : array)

System.out.print(each + " ");

changeindex(array);

System.out.println();

for(int each : array)

System.out.print(each + " ");

System.out.println();

print();

}

static void print() {

System.out.print(num);

}

static void numchange(int num) {

num = 10;

}

static void strchange(String str) {

str = "You are Wrong!";

}

static void changearray(int[] arr) {

arr = new int[5];

for(int each : arr)

System.out.print(each + " ");

}

static void modifyarray(int[] arr) {

int[] arr2 = {1, 2, 3, 4};

arr = arr2;

for(int each : arr)

System.out.print(each + " ");

}

static void changeindex(int[] arr) {

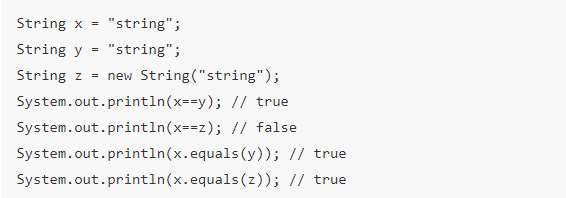
arr[0] = 10;

}

}

### 补充：week3 office hour内容（equals和“=”）

问题引入：



1. **“==”的比较**

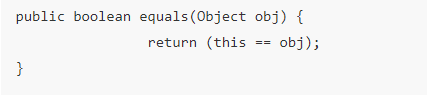
对于基本类型，int，short等，比较的是数值

对于结构体，array，string，class等，比较的是java虚拟机内存的地址，也就是在java里沿着我们stack往queue找对应的箭头看看是不是地址是不是一样。对于上面代码，因为 x 和 y 指向的是同一个引用，所以 == 也是 true，而 new String()方法则重写开辟了内存空间，所以 == 结果为 false，而 equals 比较的一直是值，所以结果都为 true。

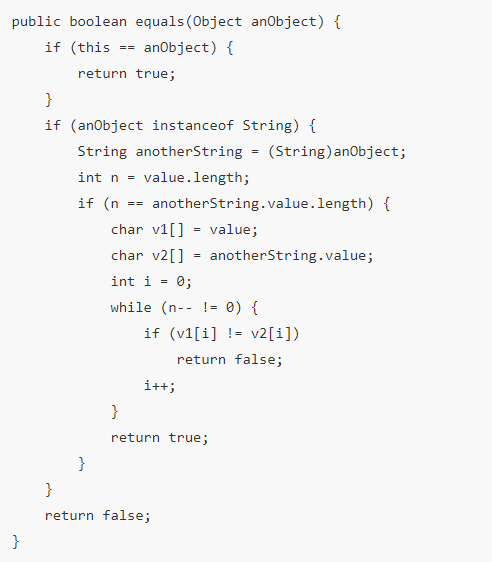
1. **Equals()函数**

equals 本质上就是 ==，但是 String 和 Integer 等很多java内部自带的package重写了 equals 方法，把它变成了值比较。

Equals函数的源代码（出自object class）

****

在很多的java的class里面，java其实对equals进行了一次重写，这里以string来进行介绍：

****

**总结：**

== 对于基本类型来说是值比较，对于引用类型来说是比较的是引用；而 equals 默认情况下是引用比较，只是很多类重写了 equals 方法，比如 String、Integer 等把它变成了值比较，所以一般情况下 equals 比较的是值是否相等