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;
```

2. **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”;
```

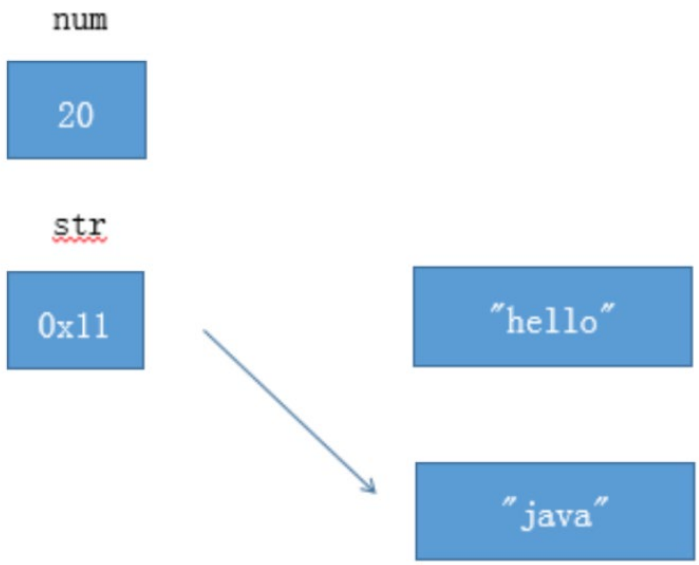
```
int num = 10;
String str = "hello";
```

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

```
num = 20;
str = "java";
```



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）

3. 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:

```
static void numchange(int num) {
    num = 10;
}
```

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

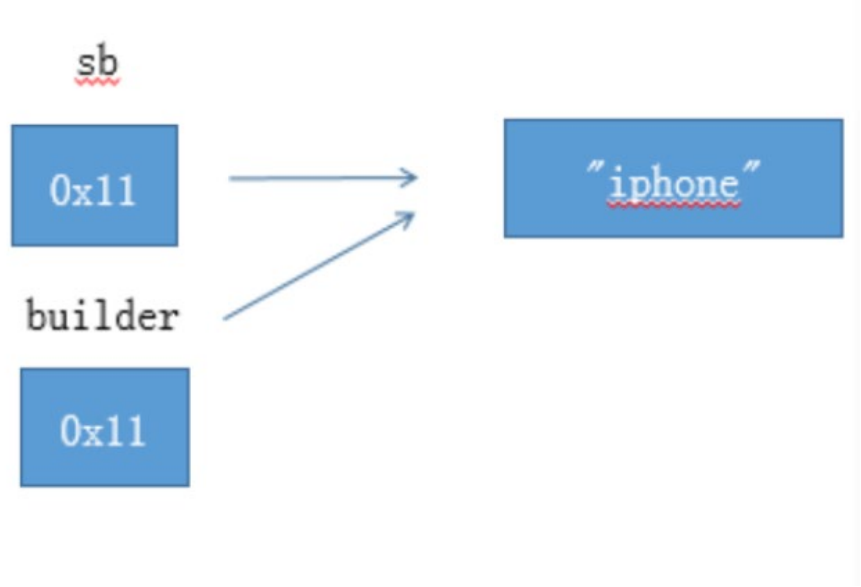
```java
static void changearray(int[] arr) {
    arr = new int[5];
    for(int each : arr)
        System.out.print(each + " ");
}

static void modifyarray(int[] arr) {
    int[] arr2 = {1, 2, 3, 4};
    arr = arr2;
    for(int each : arr)
        System.out.print(each + " ");
}
```

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:

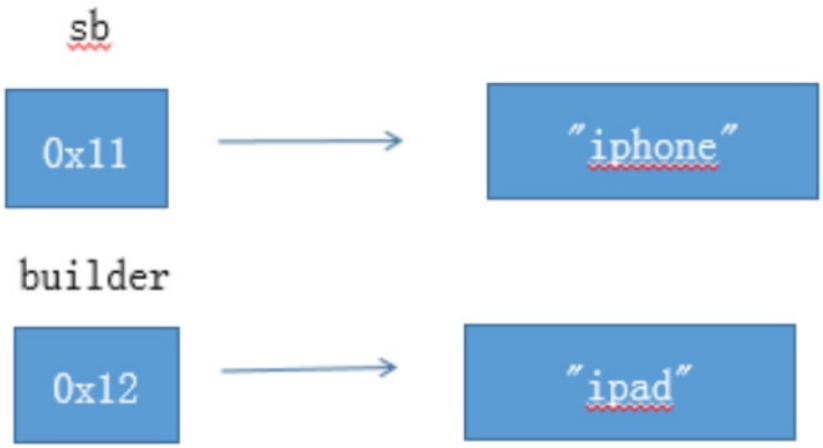
```java
StringBuilder sb = new StringBuilder("iphone");
void foo(StringBuilder builder) {
    builder = new StringBuilder("ipad");
}
foo(sb); // sb 没有被改变，还是 "iphone"。
```

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

builder = new StringBuilder("ipad"); 之后

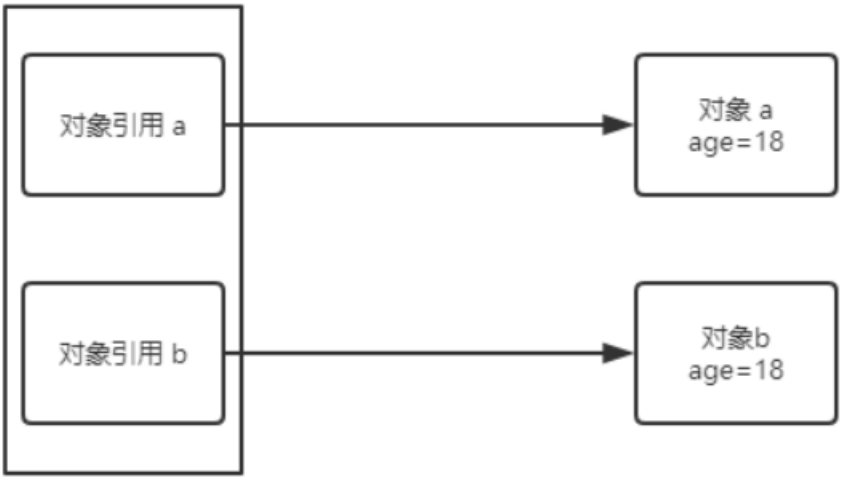


在执行 method 的时候，出现了这样的情况，什么意思，我去 initialize 一个新的 stringbuilder = "ipad"给 builder，也就是我们的副本，然后，"="的作用就出现了，等于号，就是在这里把"ipad"这个 string 的地址付给了 builder，所以现在两个 variable 指向的位置不一样了，而且，原来的 variable 的值并没有发生改变，所以在我执行完这个 method 后，假设我们再想 print 之前的 sb 的值，它还是"iphone"，也就是说，假设你在 function 去用等于号给副本 variable 去赋值，是无法改变原来的 main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

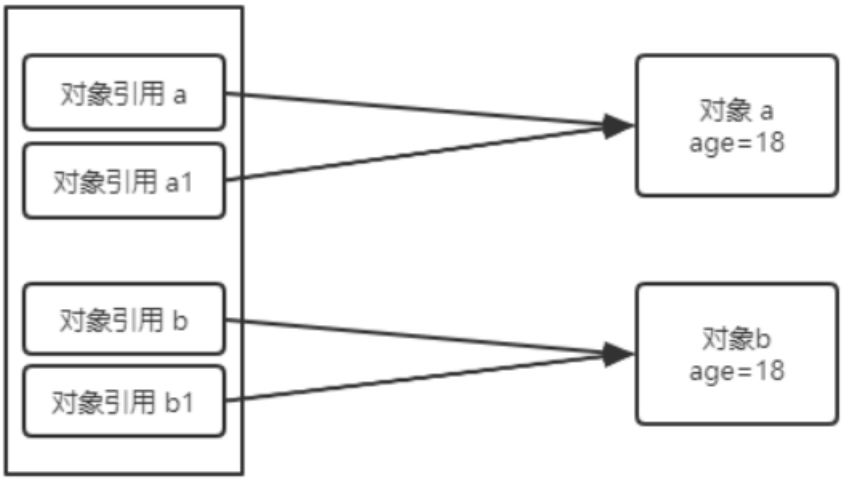
关于 class 的 example:

```
public class ReferenceTypeDemo {
    public static void main(String[] args) {
        Writer a = new Writer(18);
        Writer b = new Writer(18);
        modify(a, b);

        System.out.println(a.getAge());
        System.out.println(b.getAge());
    }

    private static void modify(Writer a1, Writer b1) {
        a1.setAge(30);

        b1 = new Writer(18);
        b1.setAge(30);
    }
}
```

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：



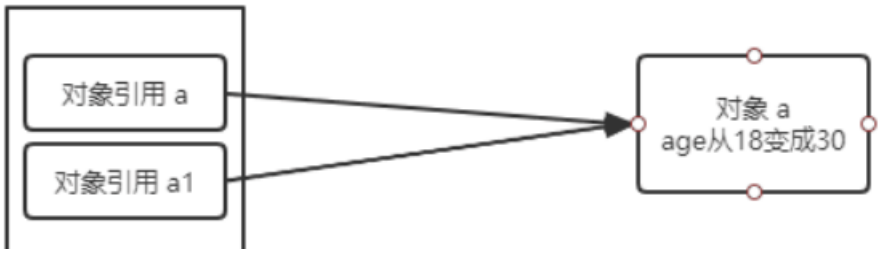
在我们开始 call method, modify 之后, 一开始 call 这个 method, 系统就开始给两个 variable, a 和 b 分别建立了一个副本，在这里我们称呼为 a1 和 b1



在系统读取了如下代码后，把 a1 的 age 进行了改变

```
a1.setAge(30);
```

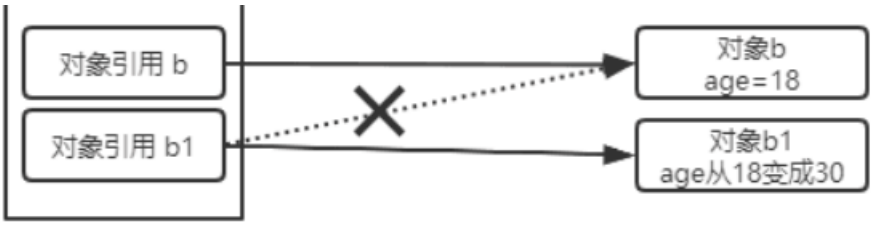
注意这里我们没有用刚刚我说的等号哦，也就是说，这里 a 和 a1 仍然会指向之前的那个 age，setAge 就会随着图上的箭头，沿着 a1 找到 a1 的 age，把它悄悄地从 18 改成 30，但是因为 a 和 a1 是共享 age 的，所以 a 的 age 也被改了，那么我们就得到了下图：



而对于 b 和 b1，系统执行了这个步骤：

```
b1 = new Writer(18);
b1.setAge(30);
```

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：

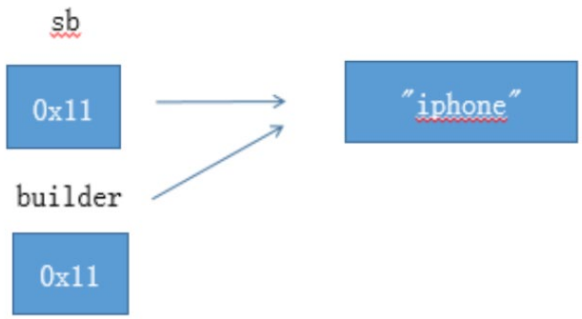
1. method 传值相当于给原来的 variable 建立了一个副本，刚开始副本和原来的 variable 指向同一个内容

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

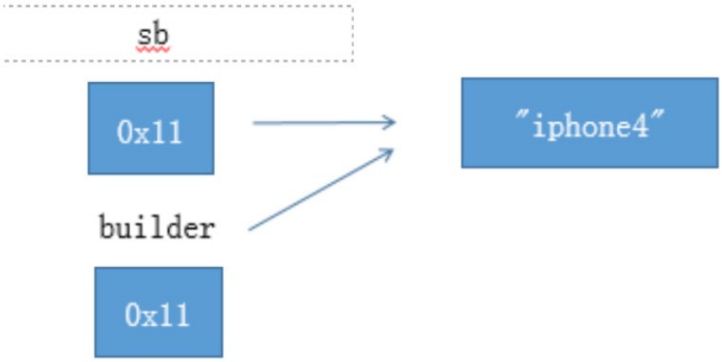
3. 只有在副本和 varibale 指向同一内容的时候我们沿着箭头去改变箭头指向的对象，我们才能真正改变原来的 variable，比如说 java 自带的 Arrays.sort()函数，或者如下例子：

```
StringBuilder sb = new StringBuilder("iphone");
void foo(StringBuilder builder) {
    builder.append("4");
}
foo(sb); // sb 被改变了，变成了"iphone4"。
```

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



builder.append("4")之后



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：

```
String x = "string";
String y = "string";
String z = new String("string");
System.out.println(x==y); // true
System.out.println(x==z); // false
System.out.println(x.equals(y)); // true
System.out.println(x.equals(z)); // true
```

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。

2. **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）

```
public boolean equals(Object obj) {
                return (this == obj);
}
```

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：

```
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    if (anObject instanceof String) {
        String anotherString = (String)anObject;
        int n = value.length;
        if (n == anotherString.value.length) {
            char v1[] = value;
            char v2[] = anotherString.value;
            int i = 0;
            while (n-- != 0) {
                if (v1[i] != v2[i])
                    return false;
                i++;
            }
            return true;
        }
    }
    return false;
}
```

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