Java传参除基本类型外，参数都是引用传递。

但有一个例外，就是实参为Null时，其实，他依然是一个值传递。

也就是说，传参为Null时，不管函数体内用这个参数做了什么，跳出函数体后，该参数依然为null。

或者说，引用传参退化成了值传递。

举例：

public class Test {

public static void handleContext(Context context) {

if(context == null) {

context = new Context();

}

context.addNum();

}

public static void main(String[] args) {

Context context = null;

handleContext(context);

System.out.println(context.getNum());

}

static class Context {

private int num;

public int getNum() {

return num;

}

public void addNum() {

this.num ++;

}

}

}

执行报空指针错误。

基础类型变量：Java的8中基础数据类型：byte(8位)、short(16位)、int(32位)、long(64位)、float(32位)、double(64位)、char(16位)、boolean(8位)，基础类型的数据存储在栈中，即是栈中分配内存空间存储所包含的值，其值就代表数据本身，值类型的数据具有较快的存取速度。

引用类型变量：除了基础类数据外，其余都是引用类型，包括类、数组等。引用类型数据的具体对象存放在堆中，而栈中存放的是该对象的内存地址。当引用类型没有赋值时，其引用为null，表示不指向任何对象。

Context context = new Context();

该操作可以分为两个部分：

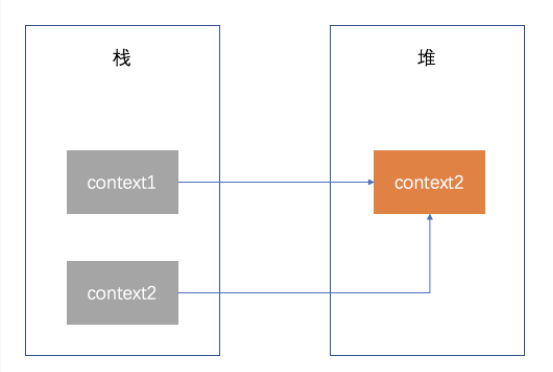
Context context;

context = new Context();

第一步：创建Context引用，在栈中开辟一块空间用于存储该引用；

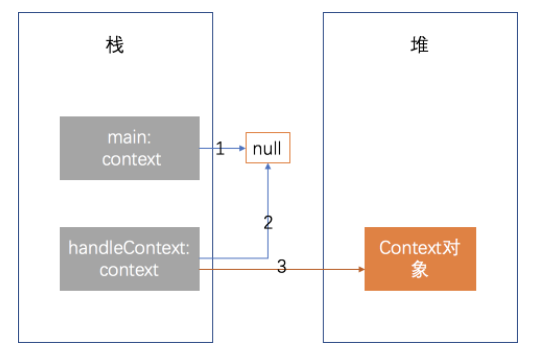
第二步：创建Context对象，在堆中开辟一块空间存储该对象，并将该对象的存储地址赋值给栈中的引用。

将一个引用赋值给另一个引用时，其实是将该引用指向的地址赋值给另个应用，让两个引用指向同一个对象，示意图如下。



其实无论是值传递还是引用传递，其本质都是讲栈中存储的数据复制一份，传递给栈中的另一个变量，对于基本数据类型，是将数据本身复制一份传递；对于引用类型，是将引用的地址复制一份传递。因此，上述问题可以描述为如下示意图。当handleContext方法执行返回后，main方法中的context引用仍然为null。

注：为方便理解，将null特殊处理。当引用为null时候表示不指向任何对象



值得注意的是包装类型(Intger;Long;Short;Double;Float;Char;Boolean;Byte，以及String(char[]的包装类型))，虽然是引用类型数据，但其效果等同于传值调用，示例如下：

public class Test {

public static void change(String str) {

str = "xyz";

}

public static void main(String[] args) {

String str = "abc";

change(str);

System.out.println(str);

}

}

---

abc

其原因是String类型是不可变(immutable)的。String类型及其成员变量均是final的，这意味着String的value字符数组不能指向其它地址，同时value字符数组的值也不可能通过继承String后修改。在change方法中，参数String引用str初始化时指向对象abc，执行方法后指向了方法区的另一个字符串常量（xyz），而main方法中的String引用str仍然指向方法区的字符串常量（abc）。

public final class String implements java.io.Serializable, Comparable<String>, CharSequence {

...

private final char[ ] value;

...

}