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：

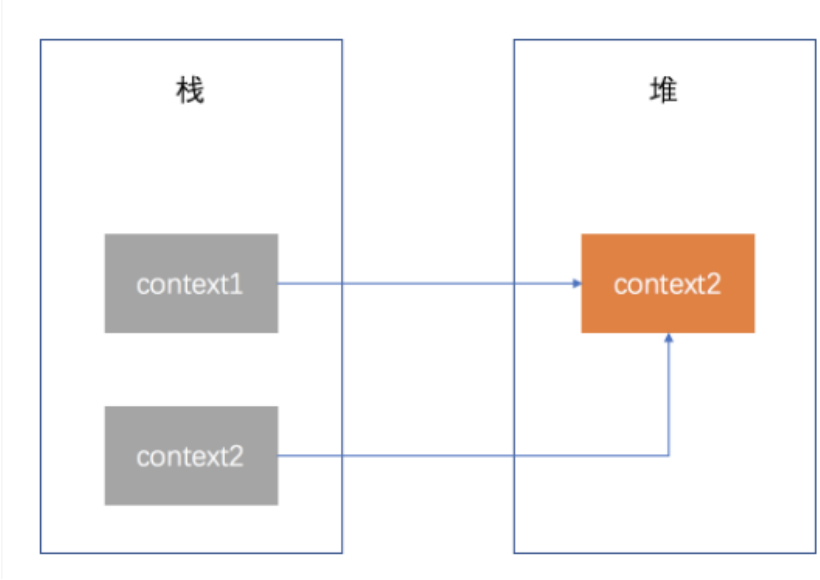
Context context;

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；

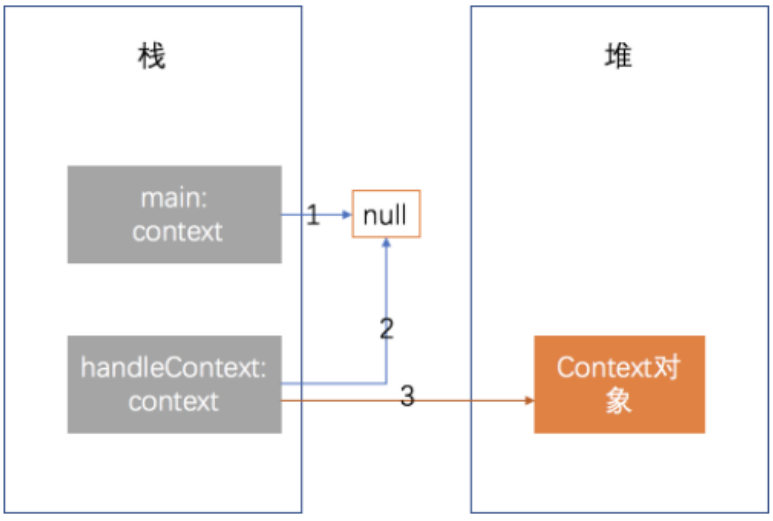
第二步：创建 Context 对象，在堆中开辟一块空间存储该对象，并将该对象的存储地址赋值给栈中的引用。

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;
...
}
```