变量在内存中的分布

如何在内存中区分类和对象?

- 类是静态的概念，位于代码区
- 对象是new出来的，位于堆内存，类的每一个变量在不同的对象中都有不同的值（除了静态变量），而方法只有一份，执行的时候才占用内存
- 静态变量、字符串常量位于数据区
- 局部变量位于栈内存

java中进行函数调用中传递参数时，遵循传递的原则：
基本类型数据传递的是该值本身，引用类型传递的是对象的引用，并不是对象本身。

实例：

package 面向对象编程;

import org.junit.Test;

public class day02_one {
class Birthday{
private int day;
private int month;
private int year;

```
    public Birthday(int day,int month,int year){
        this.day = day;
        this.month = month;
        this.year = year;
    }

    public int getDay() {
        return day;
    }

    public void setDay(int day) {
        this.day = day;
    }

    public int getMonth() {
        return month;
    }

    public void setMonth(int month) {
        this.month = month;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    @Override
    public String toString() {
```

```
        return "Birthday{" +
                "day=" + day +
                ", month=" + month +
                ", year=" + year +
                '}';
    }
}

public void change1(int i){
    i = 123;
}
public void change2(Birthday birthday){
    System.out.println(birthday);
    birthday = new Birthday(8,11,1999);
    System.out.println(birthday);
}
public void change3(Birthday birthday){
    birthday.setDay(22);
    birthday.setMonth(9);
    birthday.setYear(1999);
}

@Test
public void test(){
    Birthday b1 = new Birthday(0,0,0);
    Birthday b2 = new Birthday(1,1,1);

    change2(b1);
    System.out.println(b1);

    change3(b2);
    System.out.println(b2);
}
```

}

结果:

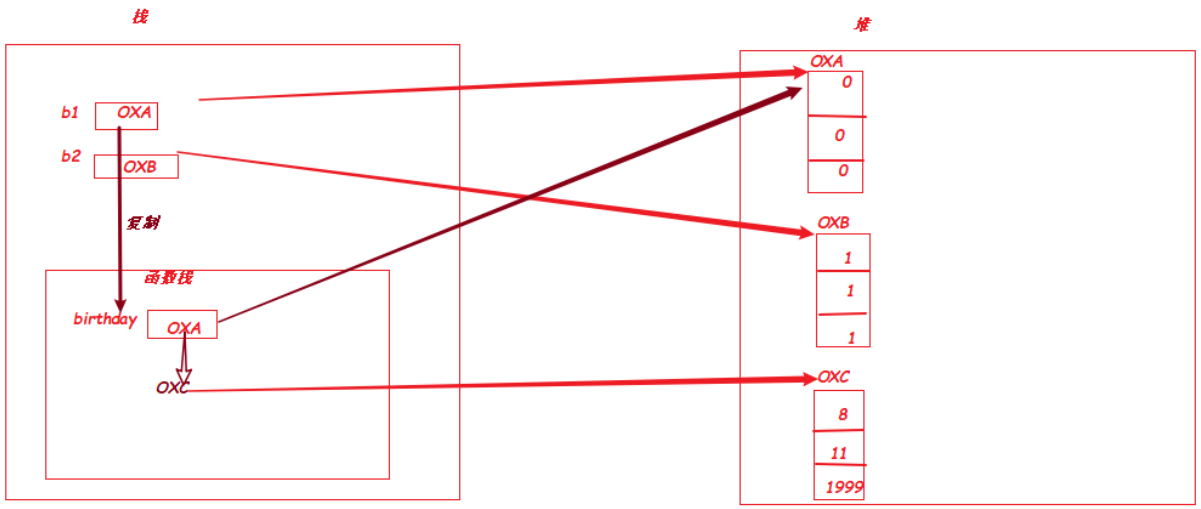
Birthday{day=0, month=0, year=0}

Birthday{day=8, month=11, year=1999}

Birthday{day=0, month=0, year=0}

Birthday{day=22, month=9, year=1999}

内存分析:



方法的重载

方法的重载是指在一个类中可以定义有相同的名字，但参数不同的多个方法。调用时会根据不同的参数表选择对应的方法。只要函数名相同，参数个数或者参数类型不同即可构成重载。
返回值不相同并不能构成重载，因为在有的函数中，返回值并不太重要从而并不会接收返回值。

内存分析——参数传递

java变量的值：方法栈里面的内容

- 基本类型变量
- 引用类型变量：指向堆（存放实际内容）的引用

Pass By Value陷阱

- 原变量的值不允许被传入方法的执行过程改变
- 引用类型变量的值指向堆的内容运行被传入方法的执行过程改变

实例一（无法交换）：

```
package 面向对象编程;

class A{
public int value;
A(int k){
value = k;
}
```

}

public class day02_two {
public static void swap(A a,A b){
A t = a;
a = b;
b = t;
System.out.println(a.value);
System.out.println(b.value);
}
public static void main(String[] arugs){
A a = new A(1);
A b = new A(4);

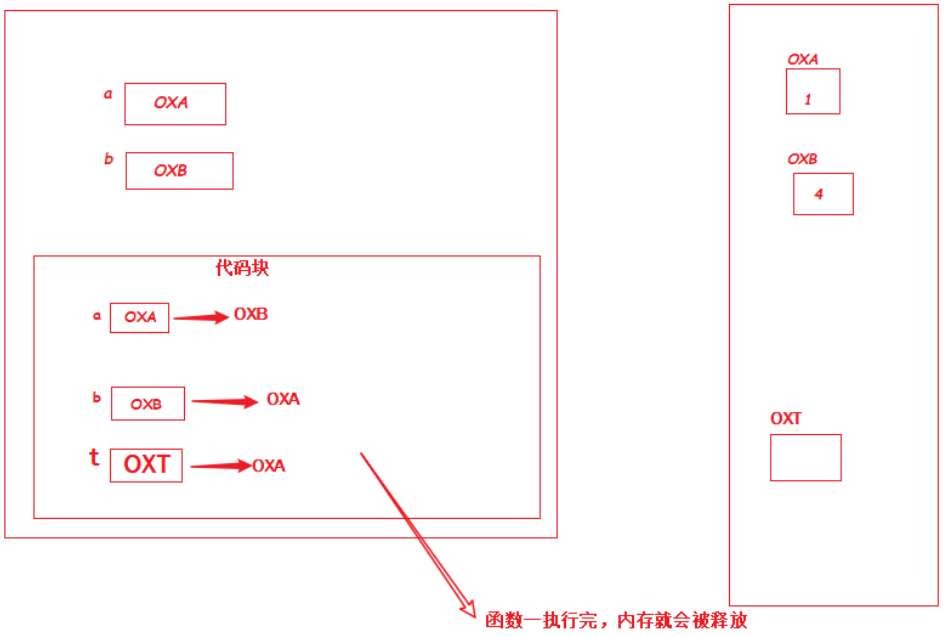
```
    swap(a,b);

    System.out.println(a.value);
    System.out.println(b.value);
}
```

}

结果:
4
1
1
4

内存分析:



实例二（可以交换）：

package 面向对象编程;

class A{
public int value;
A(int k){
value = k;
}
}

public class day02_two {
public static void swap(A a,A b){
int t = a.value;
a.value = b.value;
b.value = t;
System.out.println(a.value);
System.out.println(b.value);
}
public static void main(String[] arugs){
A a = new A(1);
A b = new A(4);

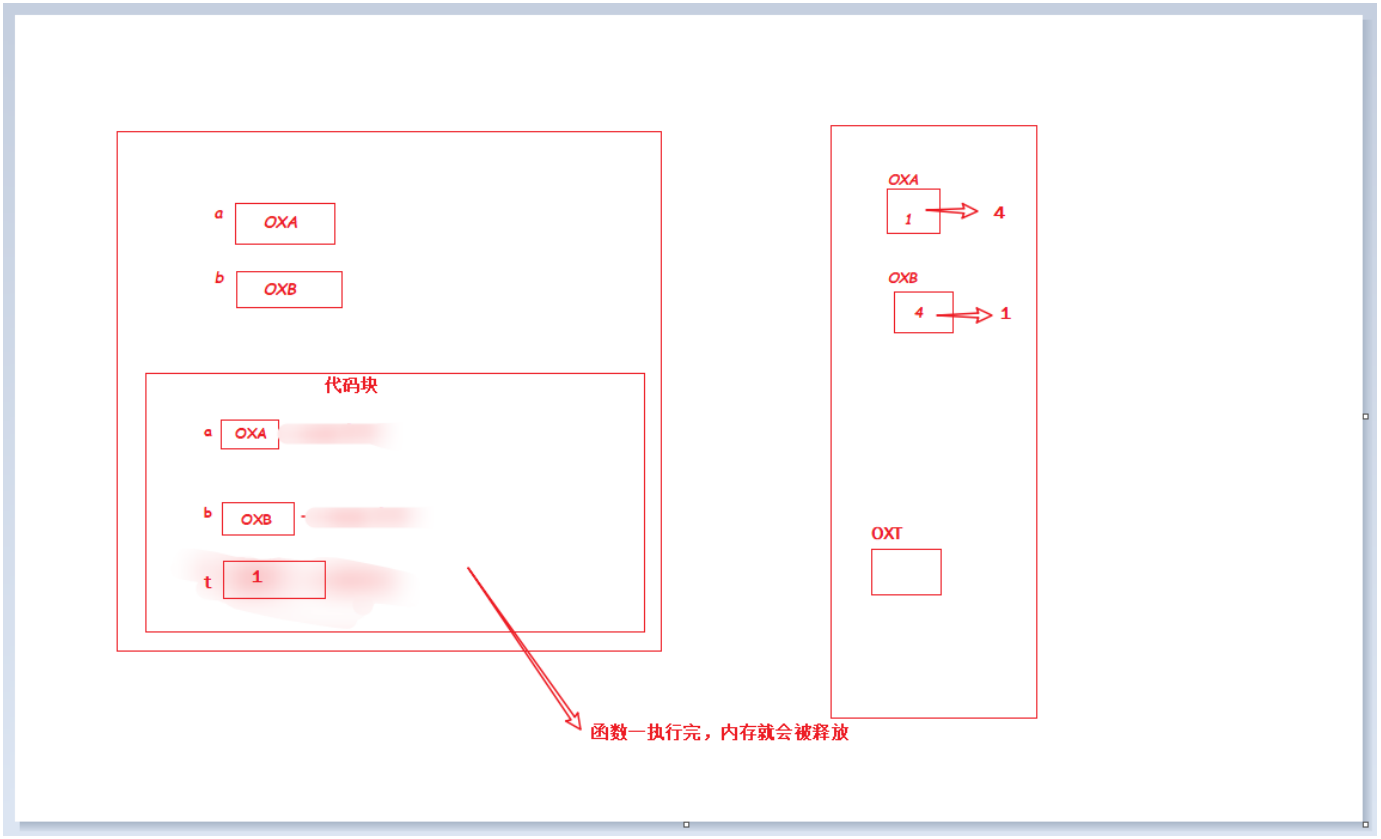
```
    swap(a,b);

    System.out.println(a.value);
    System.out.println(b.value);
}
```

}

结果:
4
1
4
1

内存分析:


a
OXA
b
OXB
代码块
a
OXA
b
OXB
t
1
OXA
1
4
OXB
4
1
OXT
函数一执行完，内存就会被释放