类和对象的基础

对象主要是由**属性**和**行为**构成的

类就是把一个东西的具体特征给他提取出来总结成一个类的数据类型，，一个类下面可以具体成一个对象，比如猫类，可以具体成一个猫，以下代码就是定义了一个猫类并把猫对象化的一个代码

```
1
2
3  public class object01 {
4      //main
5      public static void main(String[] args){
6          //创建猫
7          Cat cat1 = new Cat();
8          //给创建猫的属性赋值
9          cat1.name = "小白";
10         cat1.age = 2;
11         cat1.color = "白色";
12         cat1.weight = 10.3;
13
14         Cat cat2 = new Cat();
15         cat2.name = "小黑";
16         cat2.age = 3;
17         cat2.color = "黑色";
18         cat2.weight = 11.3;
19
20         //访问对象的属性
21         System.out.println("第一只猫的信息:"+cat1.name+" "+cat1.age+"
   "+cat1.color+" "+cat1.weight);
22         System.out.println("第二只猫的信息:"+cat2.name+" "+cat2.age+"
   "+cat2.color+" "+cat2.weight);
23     }
24
25 }
26
27
28 //有点像C的结构体
29 class Cat {
30     String name;
31     int age;
32     String color;
33     double weight;
34 }
```

属性/成员变量

⚠ 属性的注意事项和细节

- 属性的定义和变量的定义的语法引用 `访问修饰符 属性类型 变量名`
  - 访问修饰符就是控制属性的访问范围的有四种访问修饰符: `public 、protected 、默认、private`
- 属性/成员变量是类的一部分，它可以是基本类型也可以是引用对象
- 属性如果不赋值，它会有默认值，和数组一致
- 可以在类中添加代码快，代码块会在new的时候就指向一次

```

public class Person {
    String name;
    int age;
    String sex;

    {
        System.out.println("我是代码块");    //代码块中的内容会在对象创建时仅执行一次
    }

    Person(String name, int age, String sex){
        System.out.println("我被构造了");
        this.name = name;
        this.age = age;
        this.sex = sex;
    }
}
```

⚠ 构建函数

我们前面创建对象，都是直接使用 `new` 关键字就能直接搞定了，但是我们发现，对象在创建之后，各种属性都是默认值，那么能否实现在对象创建时就为其指定名字、年龄、性别呢？要在对象创建时进行处理，我们可以使用构造方法（构造器）来完成。

实际上每个类都会有一个默认的构建方法，可以通过查看 `.class` 文件来查看，

```

package com.company;

public class Main {
    public Main() { //这个Main()就是主函数的构建方法，
    }

    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

而且它不需要填写返回值，并且名字和类名相同，默认情况是每一个类都有，但是我们也可以手动进行修改，在你修改完成后，你每次new一个新的类，就都会是这个默认值，就好像每个手机的出厂设置一样；

```

package com.company;

public class Cat {
    String name;
    String sex;
    int age;
    Cat(){
        name = "小花";
        sex = "女";
        age = 3;
    }
}

```

也可以使用全参，可以使得在new的时候就可以指定成员属性的值

```
public class Cat {
    String name;
    String sex;
    int age;

    Cat(String name,String sex,int age){
        this.name = name;
        this.sex = sex;
        this.age = age;
    }
}



在new时直接指定

 Cat cat = new Cat("小花","女",3);
```

当然也可以两个都写上

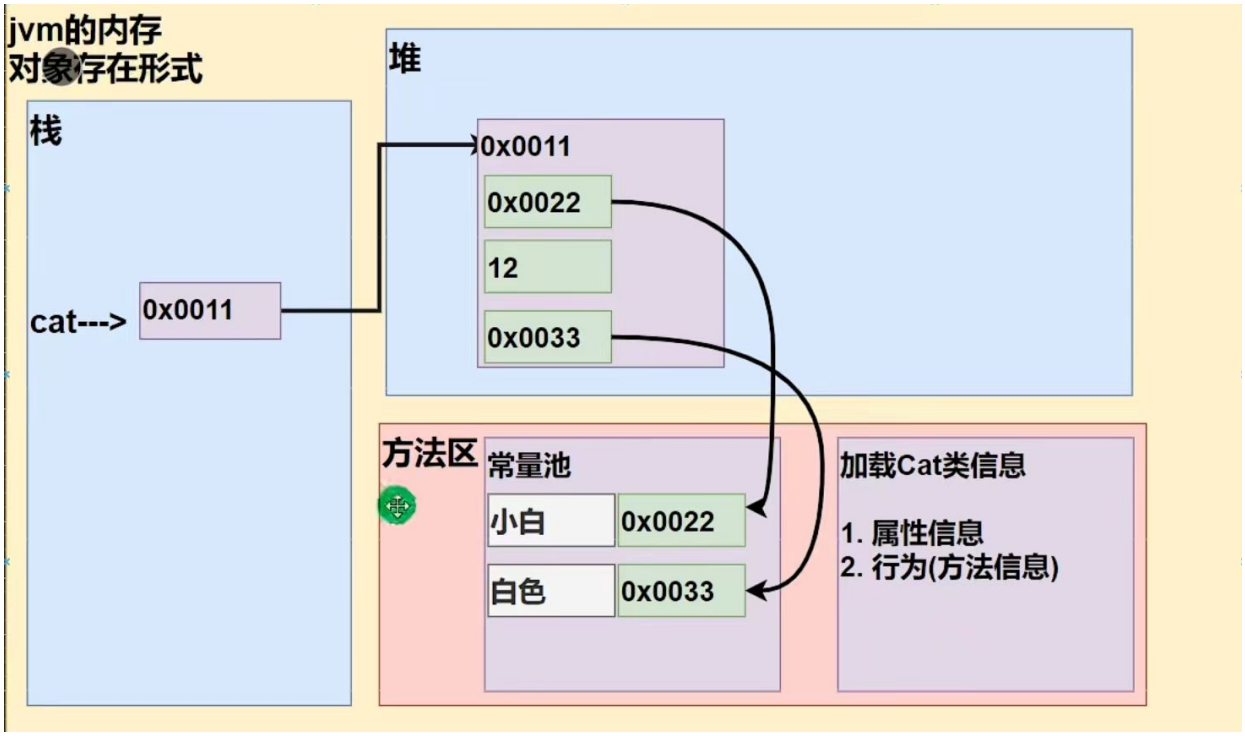
类和对象的区别

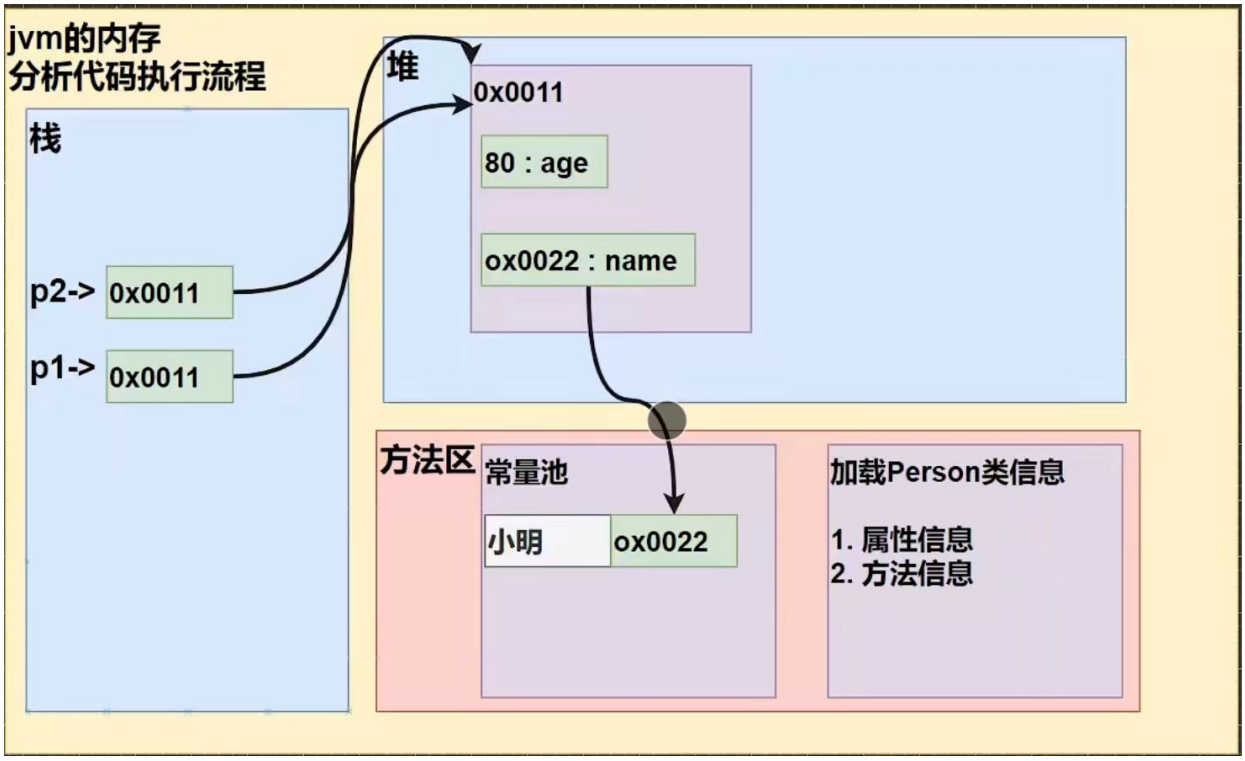
1. 类是抽象的，概念的，代表一类事物,比如人类,猫类, 即它是数据类型.
2. 对象是具体的，实际的，代表一个具体事物, 即 是实例
3. 类是对象的模板，对象是类的一个个体，对应一个实例

类和对象的内存分配机制

`java`内存结构:

1. 栈：存放基本数据类型的或者说是局部变量
2. 堆：存放对象
3. 方法区：存放常量，类加载信息






```java


public class object02 {
    //main
    public static void main(String[] args){
        //创建猫
        Cat cat1 = new Cat();
        cat1.name = "小白";
        cat1.age = 2;
        cat1.color = "白色";
```

```
        cat1.weight = 10.3;

        Cat cat2 = cat1; //这里将cat2指向cat1指向的地址了，也就说cat1和cat2都是指向同一个类的地址

        System.out.println(cat2.age);
    }

}


//有点像C的结构体
class Cat {
    String name;
    int age;
    String color;
    double weight;
}
```

创建类的过程

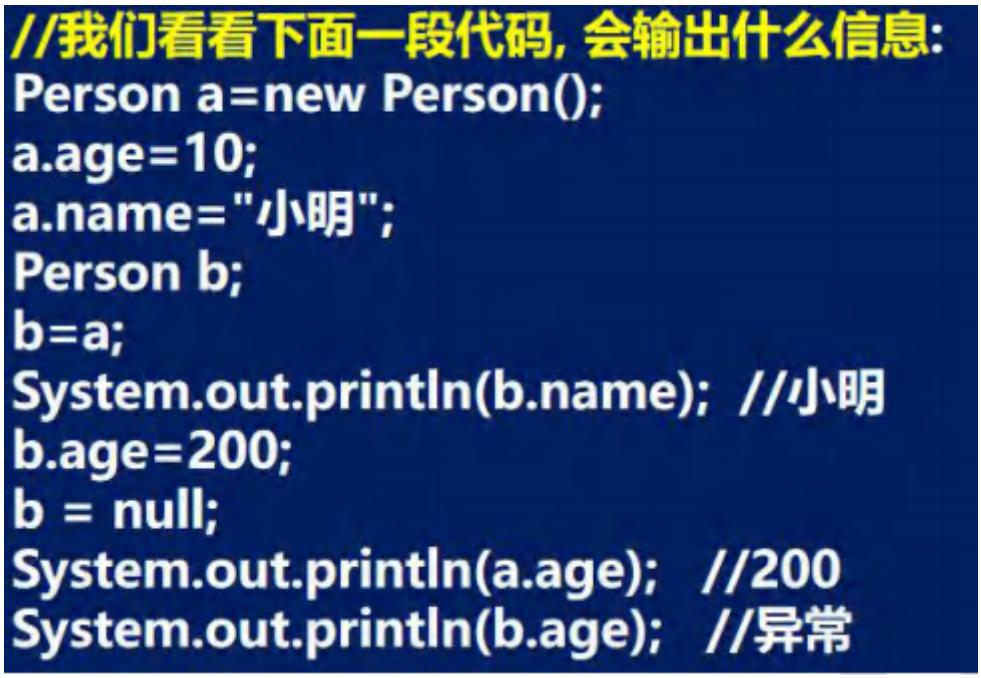
`Cat cat1= new Cat()`

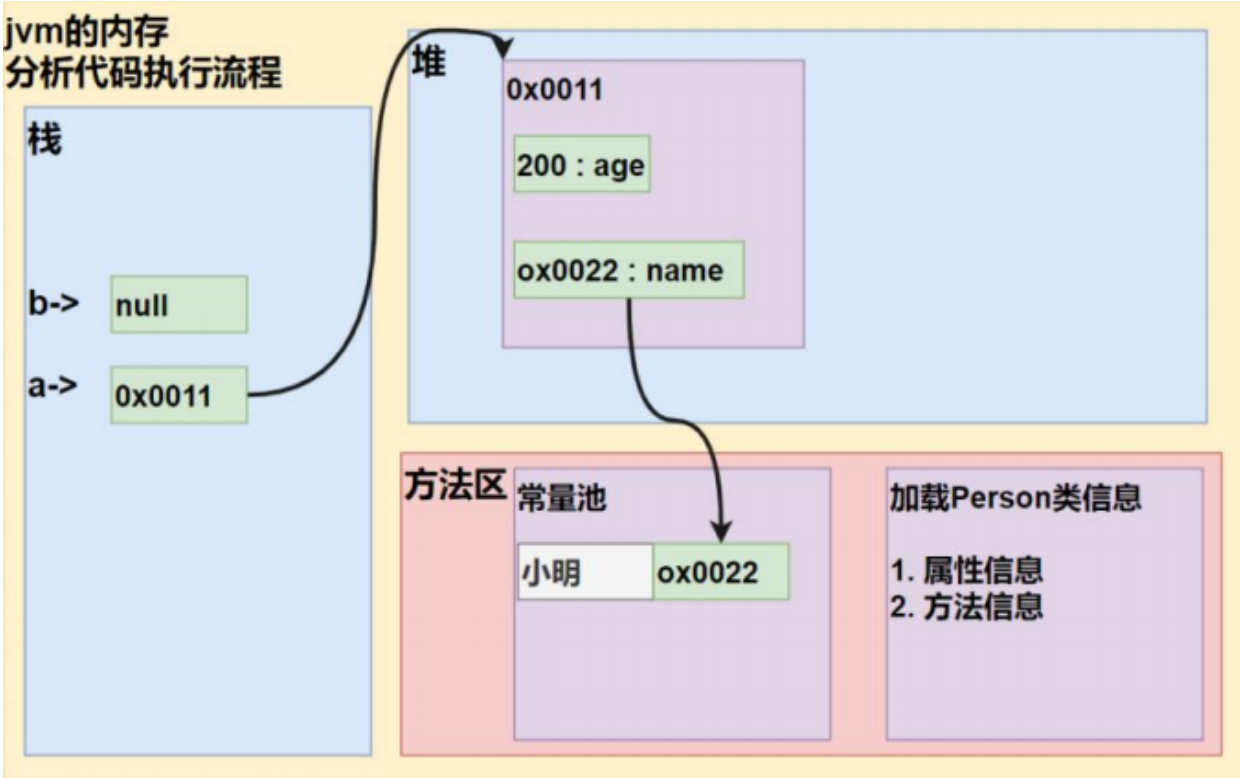
`cat1.age=3;`

类也可以在创建的时候这样 `Cat cat1 = null;` 说明现在cat没有指向任何类

1. 加载Cat类信息(属性信息和方法信息)，只会加载一次
2. 在堆中分配空间，进行默认初始化，就是系统赋值初始值
3. 把堆中的地址赋给创建的对象cat1
4. 进行指定赋值初始化

练习





成员方法

方法定义

```
public 返回数据类型 方法名(接收参数列表/形参列表){ //void表示没有返回值，也就不用写return
    语句
    return 返回值;
}
```

⚠ 方法使用细节

1. 和类一样有修饰符，有四种 `public 、protected 、默认、private`
2. 一个方法最多只有一个返回值，可以是基本类型也可以是引用类型
3. 如果方法定义了返回数据的类型，则方法中的最后一个执行语句必须为 `return` 语句，而且 `return` 后面返回的值也需要和定义的返回值类型一样
4. 如果返回值类型为 `void` 就表示没有返回值，语句内部可以不用写 `return` 或者只写 `return ;`
5. 一般使用小驼峰，也就是小谢字母开头

⚠ 形参列表细节

1. 一个方法可以没有参数也可以有多个参数，中间使用 `,` 号隔开
2. 参数类型也可以为任意类型
3. 调用方法时，一定对应着参数列表传入相同类型或兼容的参数
4. 方法定义时的参数称为形式参数，**简称形参**，调用时时实际的参数，简称实参
5. **形参如果是基本数据类型，在方法中任何改变时不会影响到实参的**
6. **形参如果是引用类型如数组，在方法中改变会影响到实参**，引用类型传递的不是值，而是地址

7. 实参和形参的类型要一致或兼容、个数、顺序必须一致

结合6,7点有一些练习题

```
//引用类型
public class Method03 {
    //main
    public static void main(String[] args){
        A a = new A();
        int[] arr = {1,2,3};
        a.test001(arr);
        System.out.println("\nmain 方法内的arr");

        for (int i = 0; i<arr.length; i++){
            System.out.print(arr[i] + "\t");
        }

    }
}

class A {
    public void test001(int[] arr) {
        arr[0]=200;
        System.out.println("test001 方法内的arr");
        for (int i = 0; i<arr.length; i++){
            System.out.print(arr[i] + "\t");
        }
    }
}

输出:
test001 方法内的arr
200     2       3
main 方法内的arr
200     2       3
```

```
public class Method03 {
    //main
    public static void main(String[] args){
        A a = new A();
        int num = 100;
        a.test001(num);
        System.out.println("\nmain 方法内的arr: "+ num);

    }
}

class A {
    public void test001(int num) {
        num = 200;
        System.out.println("test001 方法内的num: " + num);
```

```
    }
}
```

```

public class Method_Exam03 {
    //main
    public static void main(String[] args){
        A a = new A();
        B b = new B();
        C c = new C();
        D d = new D();
        b.name = "小明";
        b.age = 18;
        a.test001(b);
        System.out.println("经过A类test001方法的值: "+b.age); //b.age 的值还是18不会变

        c.test001(b.age);
        System.out.println("经过C类test001方法的值: "+b.age); //b.age 的值还是18不会

        d.test001(b);
        System.out.println("经过C类test001方法的值: "+b.age); //b.age 的值就会变了



    }
}

class A {
    public void test001(B b) {
        b=null; //这里修改是指向的地方，并不是直接修改值
    }
}

class B {
    String name;
    int age;
}

class C {
    public void test001(int age) {
        age=19;
    }
}

class D {
    public void test001(B b) {
        b.age=19; //修改指向过去的值
    }
}
```

⚠ 方法内部细节

- 方法内部不能嵌套方法来使用

⚠ 方法调用细节

1. 同一个类中的方法可以互相直接调用

```

public class Method03 {
    //main
    public static void main(String[] args){
        A a = new A();
        a.printOK();
    }
}

class A {
    //同一个内中方法可以直接互相调用
    public void print(String n){
        System.out.print(n);
    }

    public void printOK(){
        print("OK"); //直接调用
    }

}
```

2. 跨类中的方法需要创建对象后在调用

方法入门代码01

```
//方法入门
import java.util.Scanner;

public class Method01 {
    //main
    public static void main(String[] args){
        //方法使用
        Person p1 = new Person();
        p1.name = "小明";
        p1.age = 23;
        p1.speak(); //调用方法1
        p1.cal01(); //调用方法2
        System.out.print("输入n的数值: ");
        Scanner num = new Scanner(System.in);
        int n = num.nextInt();
        p1.cal02(n); //调用方法3

        int p1Sum = p1.getSum(5,6);
        System.out.println("两个数相加的总和为" + p1Sum);
```

```
    }
}

class Person{
    String name;
    int age;
    //方法1.输出一段文字
    // public表示方法公开
    // void表示没有返回值
    // speak(),speak表示方法名，()表示要传入的参数
    // {}方法体,可以写要执行的代码
    public void speak(){
        System.out.println("我叫"+name+"今年"+age+"我是一个好人");
    }

    //方法2.添加 cal01 成员方法,可以计算从 1+..+1000 的结果
    public void cal01() {
        int sum=0;
        for(int i = 1; i<=1000; i++){
            sum += i;
        }
        System.out.println("从1加到1000的结果为: "+sum);
    }

    //方法2.添加 cal02 成员方法,接收一个值n,可以计算从 1+..+n 的结果
    public void cal02(int n) {
        int sum=0;
        for(int i = 1; i<=n; i++){
            sum += i;
        }
        System.out.println("从1加到"+n+"的结果为: "+sum);
    }

    //方法4.添加 getSum 成员方法,可以计算两个数的和
    public int getSum(int i,int j){
        int sum = i + j;
        return sum;
    }
}
```

⚠ 行参和成员方法重名

有时候行参可能与成员方法重名，**如果重名变量会使用最近的一个参数来调用**

方法入门代码02

遍历数组

```

public class Method02 {
    //main
    public static void main(String[] args){
```

```
        int[][] map={{0,0,1},{1,1,1},{1,1,3}};
        MyTools tool_1 = new MyTools();
        tool_1.Traverse(map);
    }

}


class MyTools{
    //遍历列表
    public void Traverse(int[][] map) {
        for (int i = 0; i< map.length; i++){
            for (int j = 0; j< map[i].length; j++){
                System.out.print(map[i][j]+"\t");
            }
            System.out.println();
        }
    }
}
```

方法递归

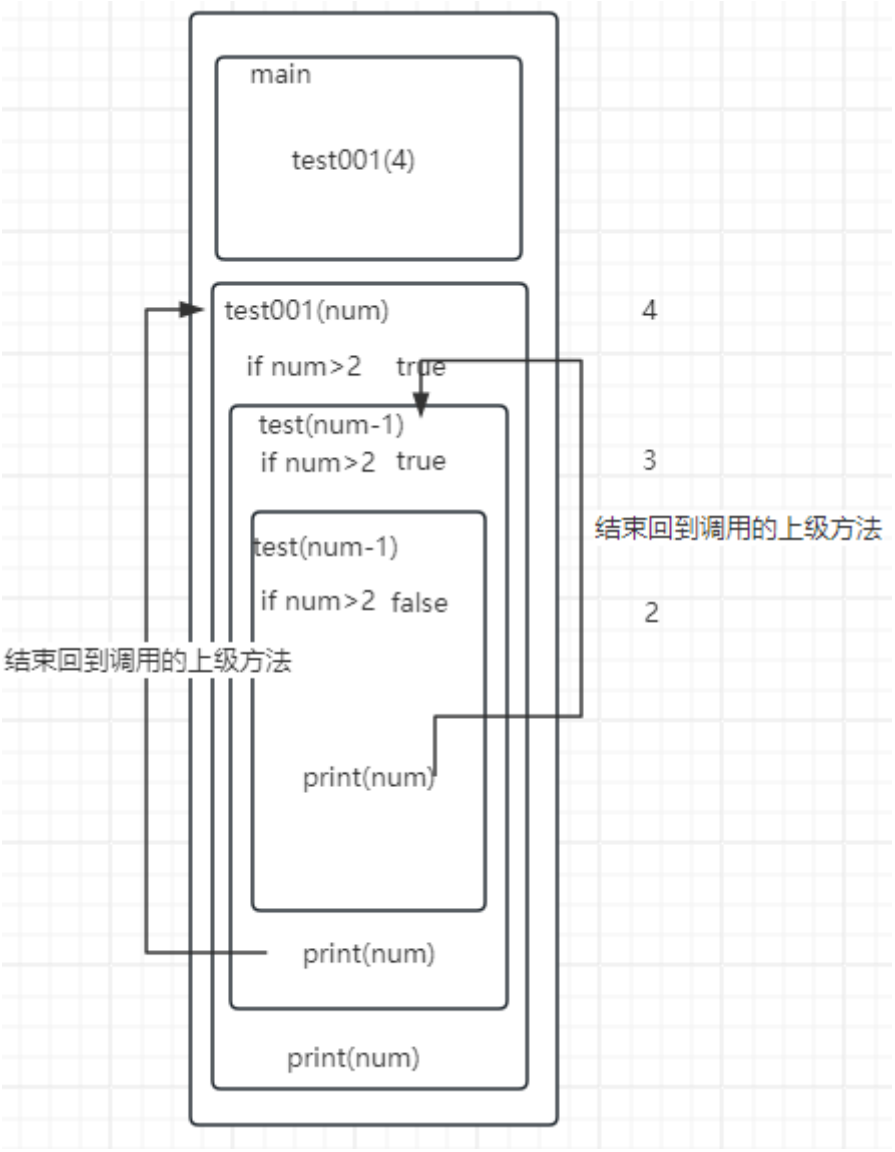
递归调用就是自己调用自己

递归入门代码01: 打印问题

```
//递归调用

public class Method03 {
    //main
    public static void main(String[] args){
        A a = new A();
        a.test001(4);
    }
}

class A {
    public void test001(int num) {
        if(num > 2){
            test001(num - 1);
        }
        System.out.print(num + "\t");
    }
}
```



递归入门代码02: 阶乘案列

```
//递归调用

public class Method03 {
    //main
    public static void main(String[] args){
        A a = new A();
        a.test001(4);
        System.out.print(a.factorial(4));
    }
}

class A {
    public void test001(int num) {
        if(num > 2){
            test001(num - 1);
        }
        System.out.print(num + "\t");
    }
```

```
    public int factorial(int n){
        if(n==1){
            return 1;
        }else{
            return factorial(n-1)*n;
        }
    }
}
```

⚠ **递归的重点**

- 执行一个方法，就会创建一个新的独立空间，哪怕是执行一个相同的方法
- 方法内部的变量是独立的，不会互相影响
- 如果方法中使用的是引用类型，就会共享改引用类型的数据
- 递归必须向退出递归的条件逼近，否者就是无限递归，出现 `StackOverflowrErro`
- 当一个方法执行完毕，或者遇到return，就会返回，遵守设调用就会将数值返回给谁

练习01：判断奇数偶数

```java
import java.util.Scanner;


public class MethodTest01 {
    //main
    public static void main(String[] args){
        Scanner numScanner = new Scanner(System.in);
        AA a = new AA();
        System.out.print("输入一个数判断奇偶：  ");
        int num  = numScanner.nextInt();
        Boolean judge = a.Judge(num);
        System.out.print(judge);
    }
}

class AA {
    public Boolean Judge(int n ){
        if(n % 2 == 0){
            return true;
        }else {
            return false;
        }
    }

}
```

静态变量和静态方法

上面提到的成员属性，是在创建对象后，属于对象的，每个对象的成员属性都是不会相互冲突的，而静态变量就是属于类的。在一个类中如果定义了一个 `static` 变量，那么每个创建的类对象，都会指向这个变量，也就是说，当其中一个对象修改了这个 `static` 变量后，其他类再去读取的时候，就是是这个修改后的值

```
package com.company;

public class Cat {
    String name;
    String sex;
    int age;
    static  String info; //定义了一个静态方法

    Cat(String name,String sex,int age){
        this.name = name;
        this.sex = sex;
        this.age = age;
    }
    Cat(){

    }
}
```

调用测试

```
package com.company;

public class Main {

    public static void main(String[] args) {
    // write your code here
        Cat cat1 = new Cat();
        Cat cat2 = new Cat();
        cat2.info="easy-going"; //这里编译器可能会报警告
//      Cat.info="Easy-going" //是因为通常都是直接用类名来修改而不是对象
        System.out.println(cat1.info);
    }
}

```

同样的也可以将方法也修改成静态方法，但是静态方法不能调用和使用成员属性，因为他属于类，而不是某个具体的对象，也无法只用 `this` 关键字，以为 `this` 就是代表当前的对象本身

```
    static void show(){
        System.out.println("我是一个静态方法！我不能调用成员属性！因为我属于类，不属于对象！");
    }
```

但是静态方法是可以使用静态变量的，因为它们都属于类

```
    static void show(){
        System.out.println("我是一个静态方法！我不能调用成员属性！因为我属于类，不属于对象！");
        System.out.println("我可以调用 "+info+" 因为我们都属于类");
    }
```

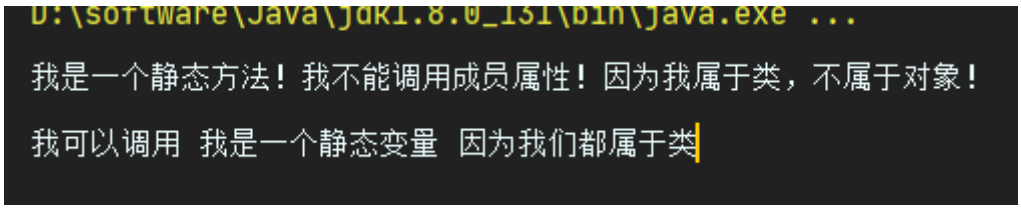
```
package com.company;

public class Main {

    public static void main(String[] args) {
    // write your code here
        Cat.info = "我是一个静态变量";
        Cat.show(); //静态变量同样可以使用类名直接调用
    }
}

```

执行结果



静态变量在上面时候进行初始化呢?

我们在一开始介绍了，我们实际上是将 `.class` 文件丢给JVM去执行的，而每一个 `.class` 文件其实就是我们编写的一个类，我们在Java中使用一个类之前，JVM并不会在一开始就去加载它，而是在需要时才会去加载（优化）一般遇到以下情况时才会会加载类：

- 访问类的静态变量，或者为静态变量赋值
- new 创建类的实例（隐式加载）
- 调用类的静态方法
- 子类初始化时
- 其他的情况会在讲到反射时介绍

所有被标记为静态的内容，会在类刚加载的时候就分配，而不是在对象创建的时候分配，所以说静态内容一定会在第一个对象初始化之前完成加载。

可以通过一下代码来测试

```
public class Person {
    String name = test();  //这里我们用test方法的返回值作为变量的初始值，便于观察
    int age;
    String sex;

    {
        System.out.println("我是普通代码块");
    }

    Person(){
        System.out.println("我是构造方法");
    }

    String test(){
        System.out.println("我是成员变量初始化");
        return "小明";
    }

    static String info = init();   //这里我们用init静态方法的返回值作为变量的初始值，便于观察

    static {
        System.out.println("我是静态代码块");
    }

    static String init(){
        System.out.println("我是静态变量初始化");
        return "test";
    }
}
```

练习

练习02: 克隆对象

```
import java.util.Scanner;


public class MethodTest02 {
    //main
    public static void main(String[] args){
        Person a = new Person();
        a.name = "小明";
        a.age = 18;
        Person b = CopyTools.copyPerson(a);
        System.out.println("复制的人名:"+b.name+",年龄为:"+b.age);
        System.out.print(b==a); //判断是否为同一个对象
    }
}
```

```

class Person{
    String name;
    int age;
}

class CopyTools {
    public static Person copyPerson(Person a){
        Person b = new Person();
        b.age = a.age;
        b.name = a.name;
        return b;
    }
}



输出:
复制的人名:小明,年龄为:18
false
```

练习03: 递归斐波那契数

```
//求出斐波那契数1,1,2,3,5,8,13...


//递归调用

public class MethodRecursionTest01 {
    //main
    public static void main(String[] args){
        Test01 A = new Test01();
        System.out.print(A.Fibonacci(8));

    }
}

class  Test01{
    public int Fibonacci(int n){
        int num=0;
        if(n==1 || n==2){
            return 1;
        }else if(n>2){
            num +=  Fibonacci(n-1)+Fibonacci(n-2);
        }
        return num;
    }
}


//也可以使用数组的方式来解决

```

```java

public class Main {
    public static void main(String[] args)  {
        System.out.println(Fibonacci(6));
    }
    public static int Fibonacci(int n){
        //定义一个数组，用于存储计算的斐波那契数
        int[] arr = new int[n+1];
        arr[0]=1; //前两个都是为1的
        arr[1]=1;
        for (int i = 2;i<=n;i++){
            arr[i] = arr[i-1] + arr [i-2];
        }
        System.out.println( Arrays.toString(arr));
        return  arr[n];
    }
}


```

练习04:猴子吃桃

```java
/*
猴子吃桃问题：有一堆桃子，猴子第一天吃了其中的一半，并再多吃了一个！
以后每天猴子都吃其中的一半，然后再多吃一个。当到第 10 天时，
想再吃时（即还没吃），发现只有 1 个桃子了。问题：最初共多少个桃子？
思路分析 逆推
1. day = 10 时 有 1 个桃子
2. day = 9 时 有 (day10 + 1) * 2 = 4
3. day = 8 时 有 (day9 + 1) * 2 = 10
4. 规律就是 前一天的桃子 = (后一天的桃子 + 1) *2//就是我们的能力
5. 递归
*/


public class MethodRecursionTest02 {
    //main
    public static void main(String[] args){

    }
}

class  Test02{
    public int peach(int n){
        if(n==10){
            return 1;
        }else{
            return(peach(n+1)+1)*2
        }
    }
}

```

```
31
```