面向对象编程-类与对象（上）

> ## 本节目标
>
> 1. 面向对象编程简介
> 2. 类与对象的定义与使用
> 3. private实现封装处理&构造方法（匿名对象）
> 4. this关键字
> 5. static关键字

1. 面向对象编程简介

面向过程编程缺少了可重用性设计

1.1 面向对象三大特征

- **封装性**：所谓封装，也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。简而言之就是，内部操作对外部而言不可见(保护性)
- **继承性**：继承是指这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。
- **多态性**：所谓多态就是指一个类实例的相同方法在不同情形有不同表现形式。多态机制使具有不同内部结构的对象可以共享相同的外部接口。（利用多态可以得到良好的设计）

1.2 面向对象名词扩展

- OOA：面向对象分析
- OOD：面向对象设计
- OOP：面向对象编程

面向对象最大的特征：可以对现实生活进行抽象。

2. 类与对象的定义与使用

2.1 基本概念

所谓的类就是指共性的概念，而对象指的是一个具体的、可以使用的事物。

首先产生类（类是生产对象的蓝图），而后才可以产生对象。对象的所有行为，一定在类中进行了完整的定义。

类中的组成：

- 属性（变量，描述每个对象的具体特点）
- 方法（操作的行为）

2.2 定义与使用

定义一个类的语法如下：

```
class 类名称 {
    属性1;
    属性2;
    属性n...;

    方法1 () {}
    方法2 () {}
    方法n () {}...
}
```

如上便是一个类的完整定义，此时的方法不再由主类直接调用，而需要由对象调用。

范例：Person类的定义

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

    public Person(String name,int age){
        this.name = name ;
        this.age = age ;
    }

    public String getPersonInfo(){
        return "姓名："+this.name+",年龄："+this.age;
    }
}
```

有了类（蓝图），我们就可以定义（生产）对象了。

生产对象的语法很简单，如下：

```
类名称 对象名称 = new 类名称();
```

以Person类为例 我们可以如下产生一个Person类的实例（对象）：

```
Person p1 = new Person();
Person p2 = new Person("Steven",25);
```

范例：通过对象调用实例变量与实例方法

```
Person p = new Person("Steven",25);
System.out.println(p.name);
System.out.println(p.getPersonInfo());
```

只要出现了关键字new，就开辟了内存

Java中，所谓的性能调优，调整的就是内存问题

2.3 对象内存分析

我们可以简单的将Java中的内存区域分为**栈内存**和**堆内存**两块区域（实际Java内存区域的划分远比这个复杂，后续讲到JVM原理和性能调优会详细讲述）

- 栈内存（虚拟机局部变量表）：存放的是局部变量（包含编译期可知的各种**基本数据类型**、对象引用-即堆内存的地址，可以简单的理解为对象的名称），Java栈是与线程对应起来的，每当创建一个线程，JVM就会为这个线程创建一个对应的Java栈。
- 堆内存：保存的是真正的数据，即对象的属性信息。

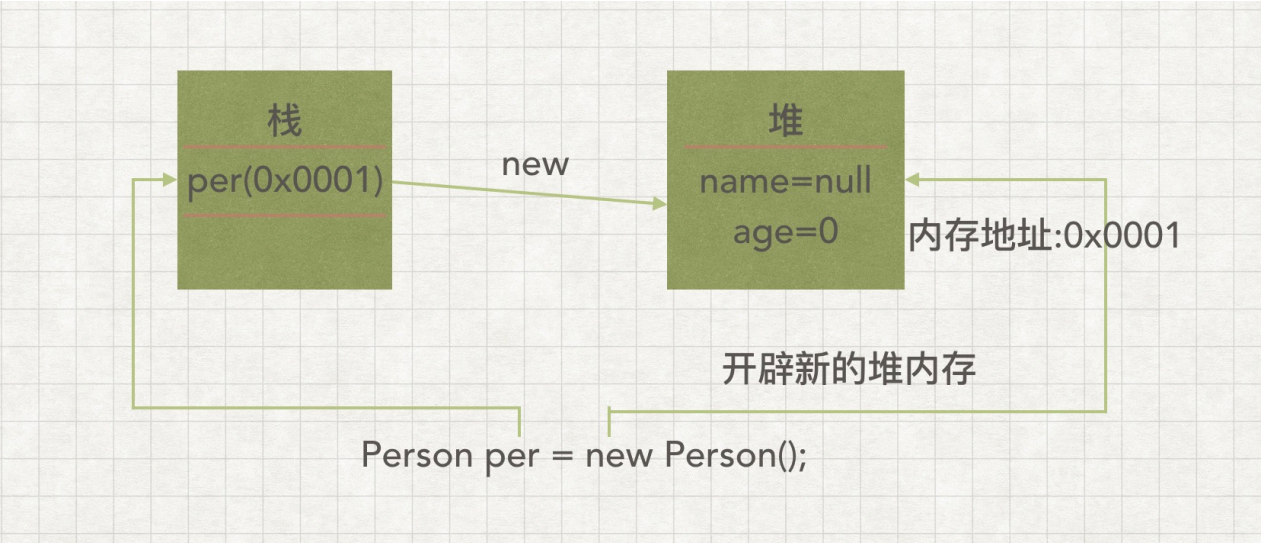
下面我们通过代码和内存分析图来给大家讲述这两部分内存

```
class Person{
    String name;
    int age;
}
public class Test{
    public static void main(String[] args) {
        Person per = new Person();
        per.name = "张三" ;
        per.age = 18 ;
}
```

main方法中的第一行代码：

```
Person per = new Person();
```

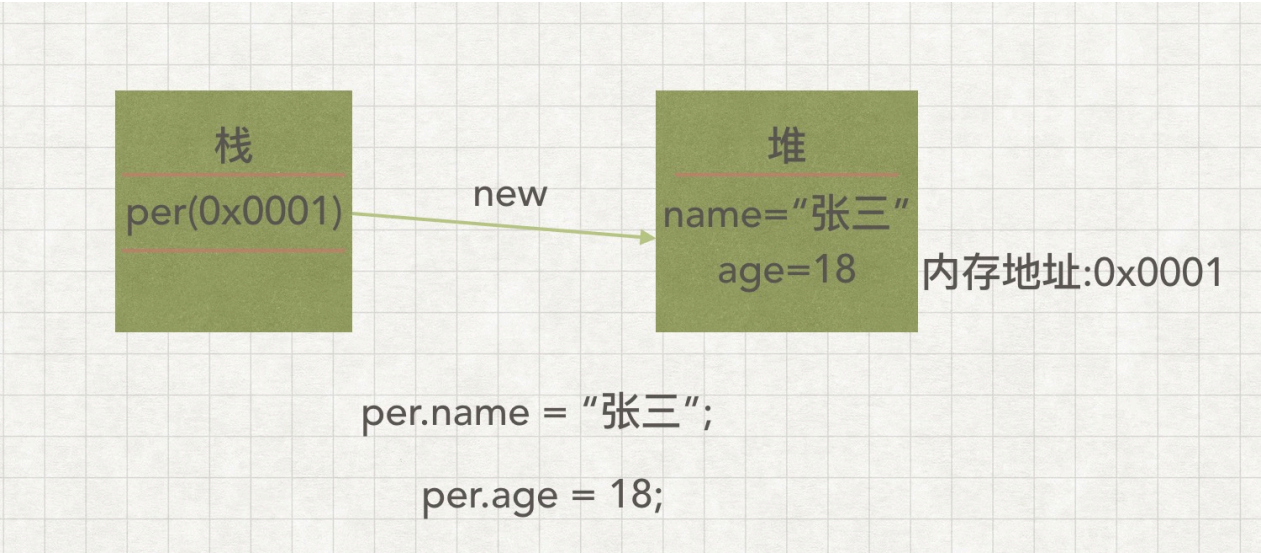
上面已经讲过了，只要出现了关键词new，表明在堆上分配了内存并且产生了Person类的对象per引用这部分内存。内存图如下：



接下来的两句代码:

```
per.name = “张三” ;
per.age = 18 ;
```

通过per引用设置堆中属性值，内存图如下：



对象（引用数据类型）必须在实例化后调用，否则会产生`NullPointerException`（运行时错误），编译时不会出错。

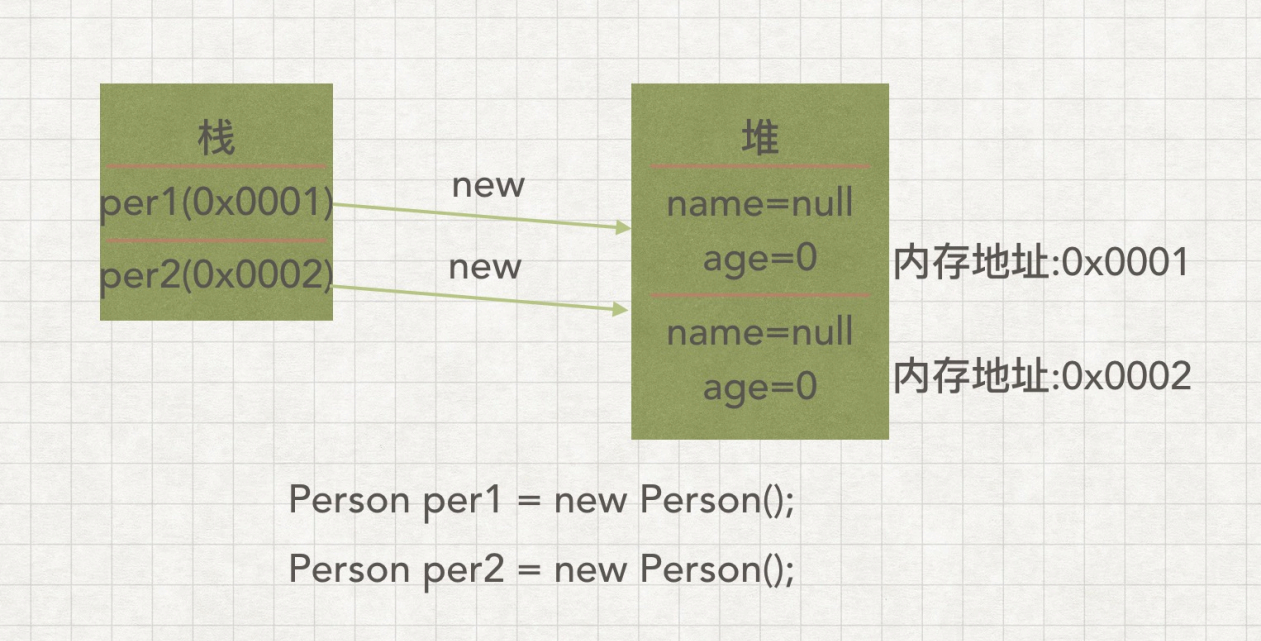
`NullPointerException`在各位今后的开发生涯中会一直存在，只有引用类型（数组、类、接口）才会产生此类异常。以后出现此类异常，就根据出错位置查看引用类型变量是否初始化。

2.4 引用传递分析

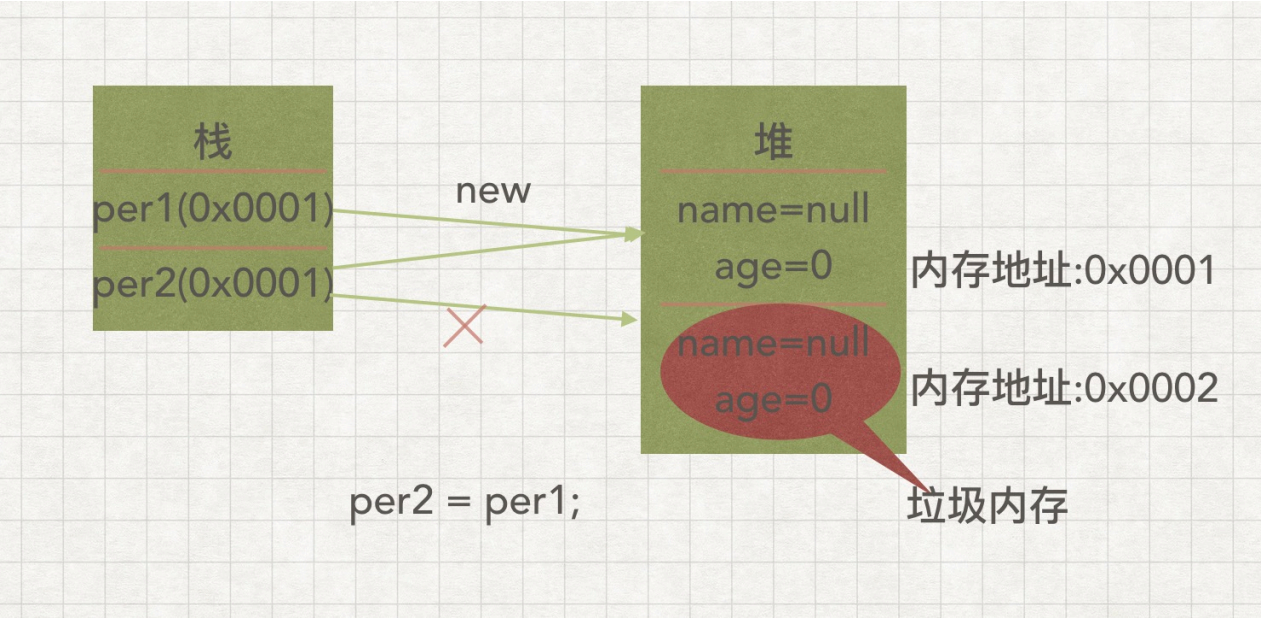
引用传递的本质：一块堆内存可以被多个栈内存所指向

```
Person per1 = new Person();
Person per2 = new Person();
per2 = per1 ;
```

前两句代码的内存图如下：



那么当`per2 = per1`;执行后，内存会怎样呢?



垃圾空间：没有任何栈内存指向的堆内存空间。

所有的垃圾空间会不定期GC，GC会影响性能，所以开发之中一定要控制好对象的产生数量（无用的对象尽量少产生）。

3.封装和构造方法

3.1 private实现封装

封装是面向对象里最复杂的概念，使用private关键字实现的封装处理只是封装的第一步，若想完全掌握封装，需要后续将继承和多态学完。

范例：无封装程序

```
class Person{
        String name;
        int age;
        public void getPersonInfo(){
                System.out.println("姓名："+name+",年龄:"+age);
        }
}
public class Test{
        public static void main(String[] args) {
                Person person = new Person();
                person.name = "张三" ;
                person.age = -200 ;
                person.getPersonInfo();
        }
}
```

此时，要回避此类问题，让内部操作对外部不可见（对象不能直接操作属性），可以使用private进行封装。

范例：使用private封装属性

```
private String name;
private int age;
```

此时使用了private对属性进行了封装，要访问私有属性，按照Java的设计原则必须提供以下两种方法：

- setter方法：主要用于进行属性内容的设置与修改
- getter方法：主要用于属性内容的取得

范例：扩展Person类的内容

```
class Person{
        private String name;
        private int age;

        public void setName(String n){
                name = n ;
        }
        public String getName(){
                return name;
        }
        public void setAge(int i){
                if (i>0&&i<=200) {
                        age = i ;
                }else {
                        age = 0 ;
                }
        }
```

```
        public int getAge(){
                return age;
        }

        public void getPersonInfo(){
                System.out.println("姓名: "+name+",年龄:"+age);
        }
}
public class Test{
        public static void main(String[] args) {
                Person person = new Person();
                person.setName("张三");
                person.setAge(-200);
                person.getPersonInfo();
        }
}
```

通过以上代码我们可以发现，private实现封装的最大特征在于：只允许本类访问，而不允许外部类访问。

3.2 类的设计原则

- 编写类时，类中的所有属性必须使用private封装。
- 属性若要被外部访问，必须定义setter、getter方法。

3.2 构造方法

同学们还记得之前讲过的如何产生对象么?

```
①类名称 ②对象名称 = ③new ④类名称();
```

针对以上定义我们做如下分析:

- 1.任何对象都应该有其对应的类，类是对象的蓝图
- 2.是一个唯一的标记，引用一块堆内存
- 3.表示开辟新的堆内存空间
- 4.构造方法

通过以上分析可以得知，所谓的构造方法就是使用关键字new实例化新对象时来进行调用的操作方法。对于构造方法的定义，也需要遵循以下原则:

- 1.方法名称必须与类名称相同
- 2.构造方法没有返回值类型声明
- 3.每一个类中一定至少存在一个构造方法（没有明确定义，则系统自动生成一个无参构造）

> 问题：构造方法无返回值，为什么没有void声明?

回答该问题前我们看看现在类中的组成：属性、构造方法、普通方法。

- 1.属性是在对象开辟堆内存时开辟的空间
- 2.构造方法是在使用new后调用的

- 3.普通方法是在空间开辟了、构造方法执行之后可以多次调用的

```
public void Person(){} //命名不标准的普通方法

public Person(){} //无参构造方法
```

因此，编译器是根据程序结构来区分普通方法与构造方法的，所以在构造方法前没有返回值类型声明

若类中定义了构造方法，则默认的无参构造将不再生成

范例：使用构造方法设置对象属性

```
class Person{
        private String name;
        private int age;

        public Person(String n,int i){
                name = n ;
                setAge(i);
        }

        public void setName(String n){
                name = n ;
        }
        public String getName(){
                return name;
        }
        public void setAge(int i){
                if (i>0&&i<=200) {
                        age = i ;
                }else {
                        age = 0 ;
                }
        }
        public int getAge(){
                return age;
        }

        public void getPersonInfo(){
                System.out.println("姓名："+name+",年龄:"+age);
        }
}
public class Test{
        public static void main(String[] args) {
                Person person = new Person("张三",-200);
                person.getPersonInfo();
        }
}
```

构造方法的调用和对象内存分配几乎是同步完成的，因此我们可以利用构造方法来为类中的属性进行初始化操作（可以避免多次setter调用）

3.3 构造方法重载

范例：构造参数重载

```
public Person(){
        System.out.println("===无参构造===");
}
public Person(String n){
        name = n ;
        System.out.println("===有参构造===");
}
```

> 建议：若干构造方法，请按照参数个数升序或降序

在进行类定义时：①定义属性->②定义构造方法->③定义普通方法

3.4 匿名对象

```
new Person("张三",20).getPersonInfo();
```

由于匿名对象不会有任何的栈空间所指向，所以使用一次后就成为垃圾空间。

4.this关键字

this关键字主要有以下三个方面用途：

> 1. this调用本类属性
> 2. this调用本类方法
> 3. this表示当前对象

4.1 this调用本类属性

来看以下代码：

```
class Person{
        private String name;
        private int age;

        public Person(String name,int age){
        this.name = name ;
                this.age = age ;
        }
        public String getPersonInfo(){
                return "姓名：" + name + ",年龄："+age;
```

```
        }
}
public class Test{
        public static void main(String[] args) {
                Person per = new Person("张三",20);
                System.out.println(per.getPersonInfo());
        }
}
```

通过以上代码我们发现，当参数与类中属性同名时，类中属性无法被正确赋值。此时我们加上this关键字便可以正确给对象属性赋值。

```
this.name = name ;
this.age = age ;
```

只要在类中方法访问类中属性，一定要加this关键字

4.2 this调用本类方法

this调用本类方法有以下两种情况：

> 1.调用普通方法：this.方法名称(参数) 2.调用构造方法：this(参数)

范例：this调用普通方法

```
class Person{
        private String name;
        private int age;

        public Person(String name,int age){
                this.name = name ;
                this.age = age ;
                this.print();//调用普通方法
        }
        public String getPersonInfo(){
                return "姓名：" + name + ",年龄："+age;
        }
        public void print(){
                System.out.println("*****************");
        }
}
public class Test{
        public static void main(String[] args) {
                Person per = new Person("张三",20);
                System.out.println(per.getPersonInfo());
        }
}
```

虽然调用本类普通方法不需要加this也可以正常调用。但强烈建议加上，加上this的目的可以区分方法的定义来源（在继承中有用）

范例：观察构造方法中存在的问题

```
class Person{
        private String name;
        private int age;

        public Person(){
                System.out.println("********产生一个新的Person对象********");
        }

        public Person(String name){
                System.out.println("********产生一个新的Person对象********");
                this.name = name ;
        }

        public Person(String name,int age){
                System.out.println("********产生一个新的Person对象********");
                this.name = name ;
                this.age = age ;
        }
        public String getPersonInfo(){
                return "姓名：" + name + ",年龄："+age;
        }
}
public class Test{
        public static void main(String[] args) {
                Person per1 = new Person();
                Person per2 = new Person("张三");
                Person per3 = new Person("李四",20);
                System.out.println(per1.getPersonInfo());
                System.out.println(per2.getPersonInfo());
                System.out.println(per3.getPersonInfo());
        }
}
```

在Java中，支持构造方法的互相调用（this）

修改上述代码如下：

```
public Person(){
                System.out.println("********产生一个新的Person对象********");
        }

        public Person(String name){
                this();//调用本类无参构造
                this.name = name ;
        }

        public Person(String name,int age){
                this(name);//调用本类有参构造
                this.age = age ;
        }
```

使用this调用构造方法时请注意：

> 1. this调用构造方法的语句必须在构造方法首行
> 2. 使用this调用构造方法时，请留有出口

4.3 this表示当前对象

范例：this表示当前对象

```
class Person{
        public void print(){
                System.out.println("[PRINT]方法："+this);
        }
}
public class Test{
        public static void main(String[] args) {
                Person p1 = new Person();
                System.out.println("[MAIN]方法："+p1);
                p1.print();
                System.out.println("=================");
                Person p2 = new Person();
                System.out.println("[MAIN]方法："+p2);
                p2.print();
        }
}
```

只要对象调用了本类中的方法，这个this就表示当前执行的对象

5.static关键字（重要）

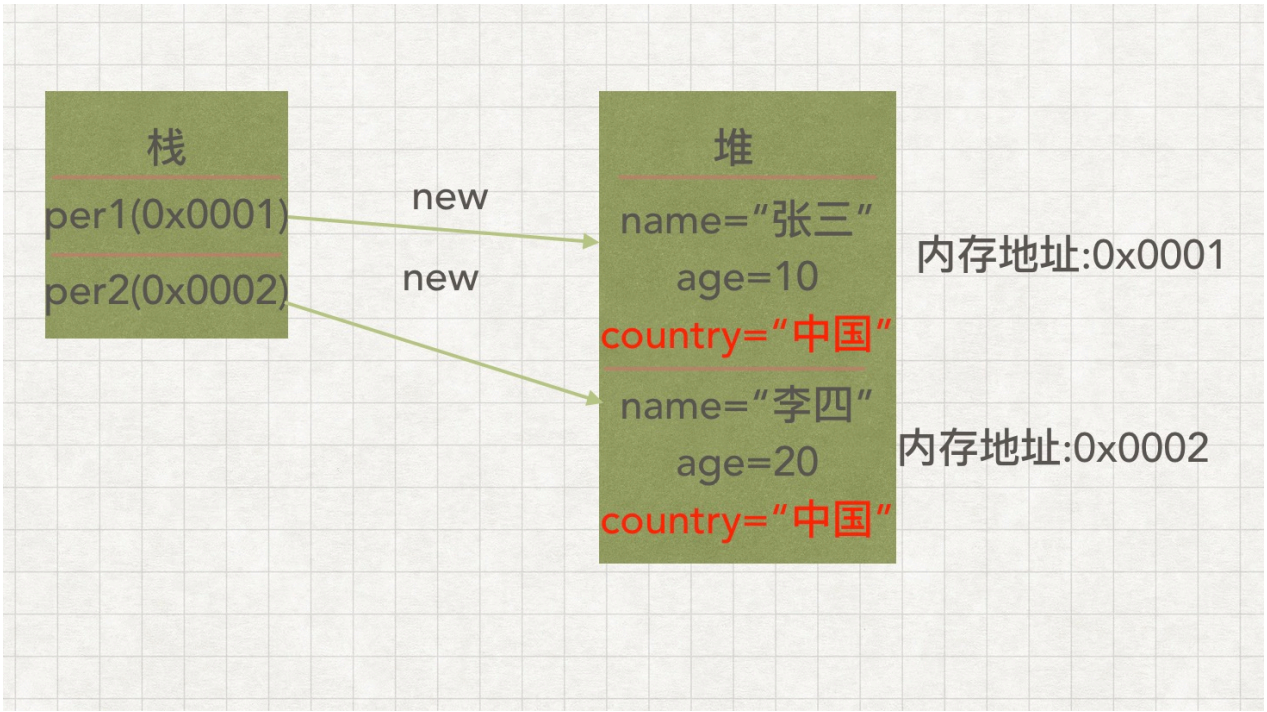
5.1 static类属性

范例：实例属性的内存分析

```
class Person{
        String Country = "中国";
        String name;
        int age;

        public void getPersonInfo(){
                System.out.println("姓名:"+this.name+",年龄: "+this.age+", 国
家: "+this.Country);
        }
}
public class Test{
        public static void main(String[] args) {
                Person p1 = new Person();
                p1.name = "张三" ;
                p1.age = 10 ;
                Person p2 = new Person();
                p2.name = "李四" ;
                p2.age = 20 ;
                p1.getPersonInfo();
                p2.getPersonInfo();
        }
}
```

内存分析图如下：



传统属性所具备的特征：保存在堆内存中，且每个对象独享属性。

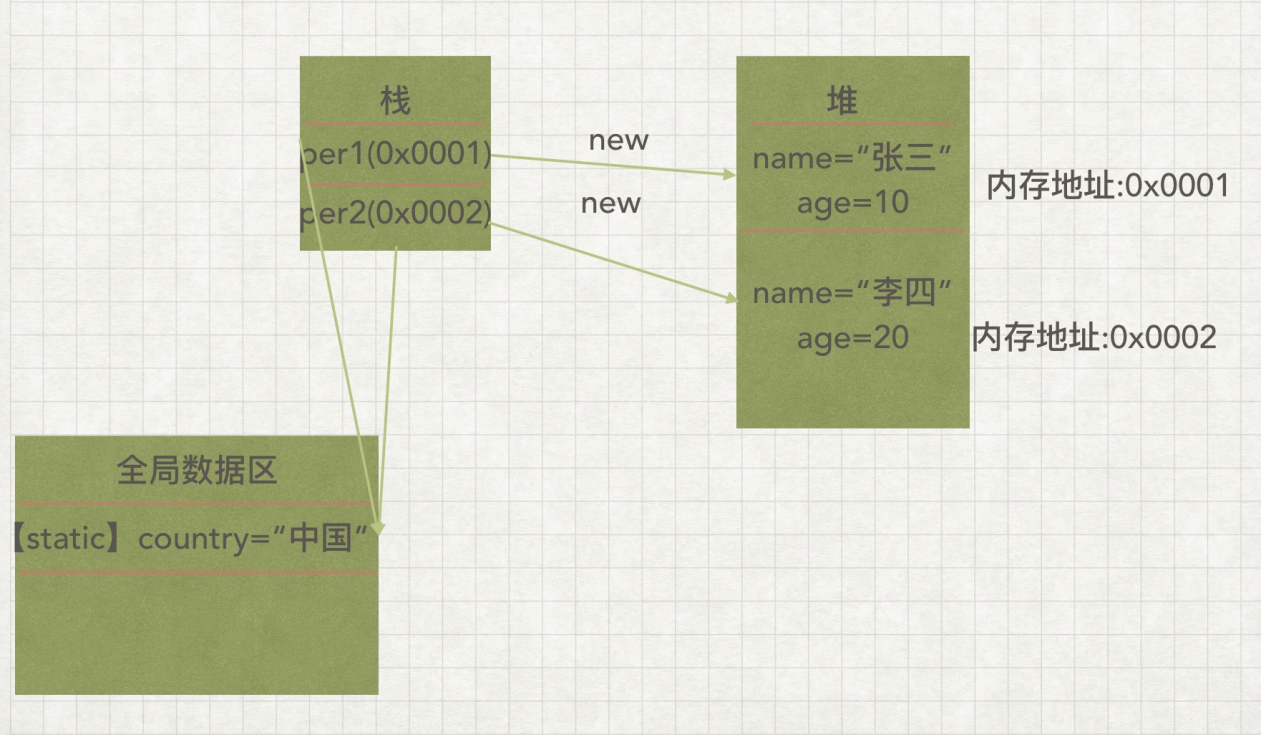
描述共享属性，只需在属性前添加static关键字即可

static属性又称为类属性，保存在全局数据区的内存之中，所有对象都可以进行该数据区的访问

修改上述代码：

```
static String Country = "中国";
```

此时内存分析图如下：



结论：

- 访问static属性（类属性）应使用类名称.属性名
- 所有的非static属性（实例变量）必须在对象实例化后使用，而static属性（类属性）不受对象实例化控制

此时，我们修改country属性，所有对象都同步此属性值：

```
Person.country = "中华民国" ;
```

定义类时，如何选择实例变量和类属性呢?

- 在定义类时，99%的情况都不会考虑static属性，以非static属性（即实例变量）为主
- 如果需要描述共享属性的概念，或者不受对象实例化控制，使用static

5.2 static类方法

使用static定义的方法，直接通过类名称访问。

范例：观察static方法

```
class Person{
        private static String country;
        private String name;
        private int age;

        public Person(String name,int age){
```

```
                this.name = name ;
                this.age = age ;
        }

        public static void setCountry(String c){
                country = c ;
        }

        public void getPersonInfo(){
                System.out.println("姓名:"+this.name+",年龄："+this.age+"，国
家："+this.country);
        }
}
public class Test{
        public static void main(String[] args) {
                Person.setCountry("中国");
                Person person = new Person("张三",20);
                person.getPersonInfo();
        }
}
```

关于static方法有以下两点说明：

- 所有的static方法不允许调用非static定义的属性或方法
- 所有的非static方法允许访问static方法或属性

使用static定义方法只有一个目的：某些方法不希望受到对象的控制，即可以在没有实例化对象的时候执行（广泛存在于工具类中）。