2、具体内容

static是一个关键字主要可以用来定义属性和方法，下面将针对此关键字的使用进行分析。

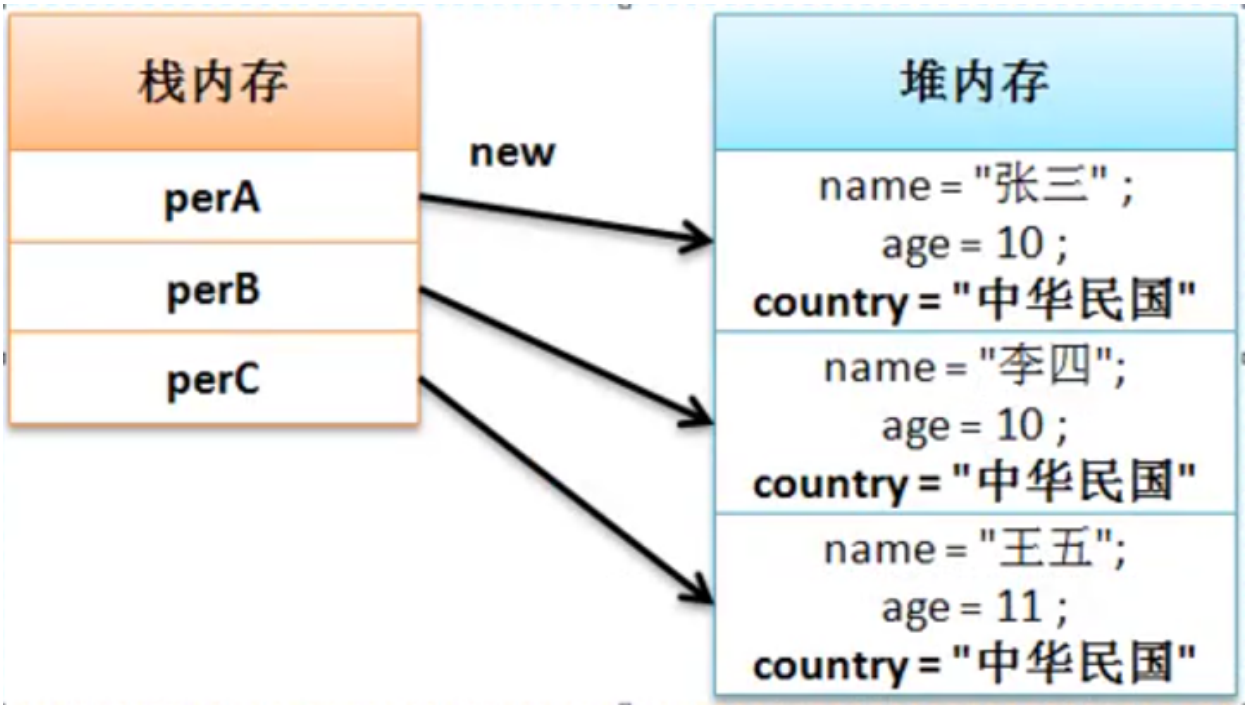
■static定义属性

在一个类之中，所有的属性一旦定义了实际上内容都交由各自的堆内存空间所保存。

范例：定义一个程序类，观察传统操作的问题

```
class Person    //创建所有同一个国家的类
{
	private String name;
	private int age;
	String country = "中华民国";//国家 暂时不封装
	public Person(String name,int age){
		this.name = name;
		this.age = age;
	}
	//setter、getter略
	public String getInfo(){
		return "姓名"+this.name+"、年龄"+this.age+"、国家"+this.country;
	}
}
public class JavaDemo{
	public static void main(String[] args) {
		Person perA = new Person("sanzhang",10);
		Person perB = new Person("sili",10);
		Person perC = new Person("wuwnag",10);
		System.out.println(perA.getInfo());
		System.out.println(perB.getInfo());
		System.out.println(perC.getInfo());
	}
}
```

为了更好的观察出程序的问题，下面对此操作做一个内存的分析。



在正常开发过程之中每一个对象要保存有各自的属性，所以此时的程序没有任何问题，但是如果突然有一天，国家解放了，变为了中华人民共和国。并且你已经产生了5000W个对象，那么此时面对你的将是场噩梦。（重复保存数据并且修改不方便）

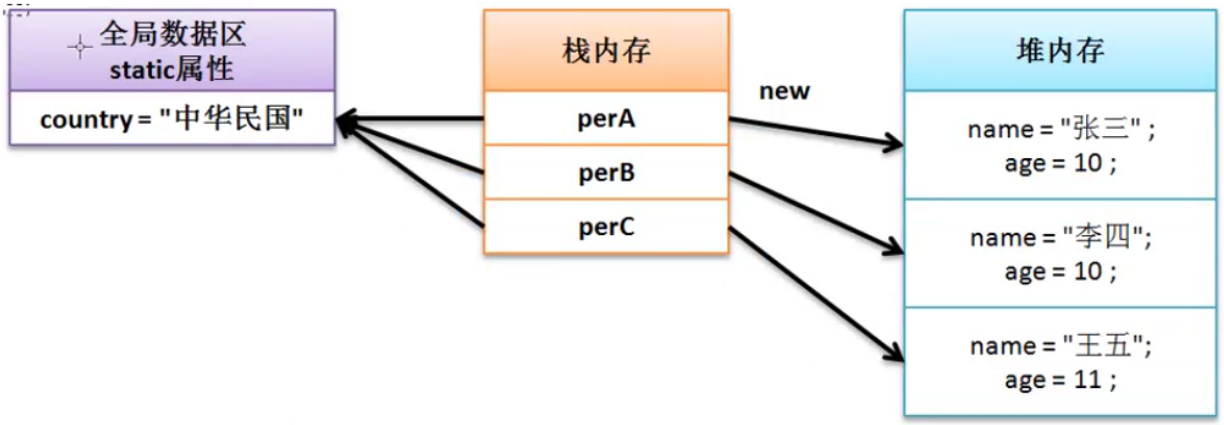
那么这个时候最好的解决方案就是将country修改为公共属性，而这种情况下就必须使用static进行标注。

范例：修改Person类定义，使用static定义公共属性

```
class Person    //创建所有同一个国家的类
{
	private String name;
	private int age;
	static String country = "中华民国";//国家 暂时不封装
	public Person(String name,int age){
		this.name = name;
		this.age = age;
	}
	//setter、getter略
	public String getInfo(){
		return "姓名"+this.name+"、年龄"+this.age+"、国家"+this.country;
	}
}
public class JavaDemo{
	public static void main(String[] args) {
		Person perA = new Person("sanzhang",10);
		Person perB = new Person("sili",10);
		Person perC = new Person("wuwnag",10);
		perA.country = "中华人民共和国";
		System.out.println(perA.getInfo());
		System.out.println(perB.getInfo());
		System.out.println(perC.getInfo());
```

```
    }
}
```

此时会发现所有对象中的country属性的内容都发生了改变，所以这是一个公共属性，而此时的内存关系图如下：



但是对于static属性的访问需要注意一点：由于其本身是一个公共的属性，虽然可以通过对象进行访问，但是最好的做法应该是通过所有对象的最高代表（类）来进行访问，所以static属性可以由类名称直接调用。

```
        Person.country = "中华人民共和国";
```

static属性虽然定义在类之中，但是其并不受到类实例化对象的控制。static属性可以在没有实例化对象的时候使用。

范例：不产生实例化对象调用static属性

```
public class JavaDemo{
    public static void main(String[] args) {
        System.out.println(Person.country);
        Person.country = "中华人民共和国";
        System.out.println(Person.country);
        Person perA = new Person("sanzhang",10);
        System.out.println(Person.getInfo());
    }
}
```

在以后进行类设计的时候首选的一定是非static属性（95%），而考虑到公共信息存储的时候才会使用到static属性（5%），非static属性必须在实例化对象产生之后才可以使用，而static属性可以在没有实例化对象产生的情况下直接通过类名称进行调用。

■static定义方法

static关键字也可以进行方法的定义，static方法主要特点在于，其可以直接由类名称在没有实例化对象的情况下进行调用。

范例：定义static方法

```
class Person    //创建所有同一个国家的类
{
```

```
    private String name;
    private int age;
    private static String country = "中华民国";//国家 暂时不封装
    public Person(String name,int age){
        this.name = name;
        this.age = age;
    }
    public static void setCountry(String c){//static方法
        country = c;
    }
    //setter、getter略
    public String getInfo(){
        return "姓名"+this.name+"、年龄"+this.age+"、国家"+this.country;
    }
}
public class JavaDemo{
    public static void main(String[] args) {
        Person.setCountry("中华人民共和国");
        Person perA = new Person("sanzhang",10);
        System.out.println(Person.getInfo());
    }
}
```

这个时候对于程序而言方法就有了两种：static方法、非static方法，这两个方法之间在调用上就有了限制。

- static方法只允许调用static属性或static方法；
- 非static方法允许调用static属性或static方法；

所有的static定义的属性和方法都可以在没有实例化对象的前提下使用，而所有的非static定义的属性和方法必须要有实例化对象的情况下才可以使用。

如果说现在可以理解这个限制，那么对于之前的方法定义就可以得出新的结论：在最早讲解方法定义的时候强调过：“当前定义的方法都是在主类中定义的，并且由主方法调用的”。

```
public class JavaDemo{
    public static void main(String[] args) {
        print();
    }
    public static void print(){
        System.out.println("xxxxxxx");
    }
}
```

```
public class JavaDemo{
    public static void main(String[] args) {
        new JavaDemo().print();
    }
    public  void print(){
        System.out.println("xxxxxxx");
    }
}
```

static定义的方法或者是属性都不是你代码编写之初所需要考虑的内容，只有在回避实例化对象调用并且描述公共属性的情况下才会考虑static定义的方法或者属性。

■static应用

为了加强理解，下面做两个简单的程序来进行static应用的提示。

范例：编写一个程序类，这个类可以实现实例化对象个数的统计，每一次创建新的实例化对象都可以实现一个统计操作。

• 此时可以单独创建一个static属性，因为所有对象都共享同一个static属性，那么在构造方法中可以实现数据的统计处理。

```
class Book{
	private String title;
	private static int count = 0;
	public Book(String title){
		this.title = title;
		count++;
		System.out.println("第"+count+"本图书创建出来");
	}
}
public class JavaDemo{
	public static void main(String[] args) {
		new Book("java");	new Book("JSP");	new Book("Spring");
	}
}
```

范例：实现属性的自动命名处理

• 如果现在传递了title属性，就使用传递的属性内容，而如果没有传递title属性，则自动采用“NOTITLE-编号”的形式进行该属性内容的定义。

```
class Book{
	private String title;
	private static int count = 0;
	public Book(){
		this("NOTITLE-"+count++);
	}
	public Book(String title){
		this.title = title;
		count++;
	}
	public String getTitle(){
		return this.title;
	}
}
public class JavaDemo{
	public static void main(String[] args) {
		System.out.println(new Book("java").getTitle());
		System.out.println(new Book("JSP").getTitle());
		System.out.println(new Book().getTitle());
	}
}
```

这样处理的好处是可以避免在没有设置title属性时内容为null的重复问题。