Java中的三种代码块

1. **静态代码块、构造代码块、局部代码块**。
2. **面试中经常考到的问题：三者的执行顺序**：需要类加载，就先执行**静态代码块**；

需要创建对象，就先执行**构造代码块**，其次执行构造方法。对于方法体中的**局部代码块**，按照排列的顺序执行。

1. **静态代码块**：（使用的范围不广）作用：给类进行初始化。利用static关键字进行声明。而构造方法是给对象进行初始化的。静态代码块在**类加载**的时候就执行，而**构造方法**在生成对象的时候执行；注意：要想调用某个类生成某个对象时，首先将类加载到java虚拟机上，然后由java虚拟机{JVM}加载这个类生成对象。先加载类再生成对象。由于类只会加载一次，所以静态代码块只会执行一次。注意某些类是不需要生成对象的（这些类一般内部都是静态变量和静态方法，不存在非静态的），所以初始化工作全都有**静态代码块**来完成。如果和主函数在同一类中，优先于主函数执行。
2. **构造代码块（构造对象时首先执行的代码块）**：是与对象有关系的。作用是：**给所有对象进行一样的初始化**，而构造方法是对对象进行针对性的初始化。直接在类中定义的，与构造方法并列存在，区别于**局部代码块**。构造代码块的执行一定在构造方法之前，创建一个对象时，会先执行构造代码块，然后执行相应的构造函数。
3. **局部代码块：**在方法体中直接利用大括号括起来，用来限制变量的作用域，执行完毕就会释放内存空间。**局部代码块是定义在方法体中的**，注意区别于构造代码块。方法体中，局部代码块的执行顺序就是按照代码的排列顺序执行的。
4. 在方法体中，直接利用大括号括起来的代码块，称为局部代码块。
5. 变量必须先定义并赋予初始值，并且注意变量的作用域。
6. 例如：

……

{ int m= 2;

System.out.println(m\*3);

}

执行完局部代码块，立即释放变量m的空间。这样就可以进行**优化**，使用完后立即释放。

（4）局部代码块的作用：可以决定局部变量的生命周期，实现优化。

1. 例子：**public** **class** Test {

**int** age;

**static** String *str*;

**public** **void** speakout() {

System.***out***.println("非静态方法: "+"我的年龄是："+age);

}

**static** {// 静态代码块

System.***out***.println("静态代码块: 只会在类加载的时候，类只会加载一次，所以我只会被执行一次 ");

}

{// 构造代码块

*str* = "我是一个人";

System.***out***.println("构造代码块: 每生成一个对象，我都会先于构造方法，对此对象进行进行初始化，且我对每个对象都是公平的，赋予一样的初始化 " +*str*); //可以声明每个创建的对象都是人

}

Test(**int** age){

**this**.age = age;

System.***out***.println("构造函数：我给每个对象赋予独特的属性");

{

System.***out***.println("**局部代码块**，我在方法体中，老老实实地按照顺序执行，主要作用是限制我的变量的作用域，我消失，他就得消失");

}

**this**.speakout();

}

**public** **static** **void** main(String[] args) {

Test test1 = **new** Test(22);

Test test2 = **new** Test(24);

Test test3 = **new** Test(35);

}

}

1. 面试题目：

**class** HelloA {

**static** {//**静态代码块**

System.***out***.println("static A");}

HelloA() {

System.***out***.println("Hello A");

}

{System.***out***.println("I'm A class."); }//构造代码块

}

**public** **class** HelloB **extends** HelloA {

**static** {//**静态代码块**

System.***out***.println("static B"); }

HelloB() {

System.***out***.println("Hello B"); }

{System.***out***.println("I'm class B.");} //构造代码块

**public** **static** **void** main(String[] args) {

**new** HelloB(); }

}

上面程序的运行结果是什么？

答：（真正操作结果）

static A

static B

I'm A class.

Hello A

I'm class B.

Hello B

分析：知道了三种代码块的执行顺序了，但是对于继承的问题，稍微麻烦了一下，类的加载如同构造函数一样，首先**加载父类**，再**加载子类**，紧接着，子类创建对象，先执行父类的构造代码块和构造方法，其次再执行子类的构造代码块和构造方法。利用子类实例化过程中记录的只是可以解释此现象，见java中的子类实例化过程。