Java中的子类实例化过程

1. **注意类加载，只是把类加载到内存中，并不是加载到栈中，只有在调用方法（包括构造方法和成员方法）和成员变量时才把这些方法和变量压栈，从而进行成员变量的赋值和方法体的执行。**
2. **实例：先加载父类，再加载子类，在加载过程中，只是对成员变量进行隐式初始化，并未显式初始化，紧接着，通过super（\*\*\*）；初始化父类内容，（此时子类的成员变量并未显式地初始化，）等到super（\*\*\*）；初始化父类内容完毕后，才进行子类的成员变量的显式地初始化，然后再执行子类构造方法的第二行内容。**
3. **一个对象实例化过程：例如 Person p = new Person();**

**注意：JVM一启动就会把Object 类加载进来。**

1. **JVM会读取指定路径下的Person.class 文件，如果此类有直接父类，则会先把父类加载进内存，其次加载本子类Person进内存；加载类的时候就会执行静态代码块并使成员变量赋予默认的初始值。**
2. **在堆内存中开辟空间，分配对象地址；**
3. **调用对应的子类构造函数，先执行第一行（super（\*\*\*）；），则调用父类的构造函数，父类（除了Object类）中总也会有super（\*\*\*）；一次调用父类的构造函数；**
4. **从Object类执行完构造函数后返回，次一级的父类（执行构造函数第二句之前），会执行成员变量的显式初始化和构造代码块；执行完毕后，再开始执行构造函数的第二句；（核心：无论父类还是子类，也就是除了Object类之外，在构造方法的第一句和第二句之间有个圈，并不是紧接着的，而是先对成员变量进行显式初始化，然后是构造代码块的执行）如果一个类中连续执行了多个不同的构造方法，其中只插入一次成员变量的显式初始化和构造代码块的执行，应该是在最后一个被调用的此类构造函数的第一句和第二句之间执行的。**
5. **父类的构造函数初始化完成后，再进行子类的变量显式初始化和构造代码块的执行；**
6. **然后再从第二行开始进行子类构造函数的初始化。**
7. **初始化完成后，将地址赋给引用变量。（注意类引用型变量不一定只在栈中，如果是在类中直接生成对象（单例模式时，在类中生成对象），显然引用型变量一开始并不在栈内存中）。**
8. **当子类和父类有相同的属性时，在子类实例化过程中，如果父类构造方法中需要访问该属性，调用的就是父类中的此属性值，如果没有则出错。证明这一点的方法是删去子类中的此属性，输出的是父类中的属性值，保留两个属性，则输出的仍然是父类中的属性值。**
9. 如果子类的成员变量定义为final修饰，则在类加载过程就会显式地赋予此变量，因为final变量只能被赋值一次，不存在默认初始化。 结果是 Son : 10
10. **在子类实例化过程中，父类中的构造方法在访问方法时，如果子类中重写了此方法，则调用的就是子类中重写重写，而且此重写方法中调用的变量就是子类的特有变量，但是若子类中没有此重写方法，则只能调用父类中的方法，此时此方法中调用的变量只是父类中的变量。方法是谁的，里面的变量就是谁的，只不过子类先用自己特有的，若没有再用父类的。**

例子：**class** Father {

// static int num = 10;

{System.***out***.println("父类的构造代码块");}

Father(**int** n){

System.***out***.println("Father int 构造函数");

}

Father() {

**this**(4);// 从结果来看，显然调用的是父类的对应的构造方法。

show();//从结果来看， 这个方法调用的是子类中的show（）方法

**return**;

}

**public** **void** show() {

System.***out***.println("Father :");

}

}

**class** Son **extends** Father {

**int** num = 10;

Son(**int** num){

System.***out***.println("Son int 构造函数");

}

{System.***out***.println("构造代码块");}//构造代码块

Son(){

show();

**return**;

}

**public** **void** show() {

System.***out***.println("Son : " + num);

}

}

**public** **class** FatherSon {

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

Son son = **new** Son();

son.show();

}

}

结果：

父类的构造代码块

Father int 构造函数

Son : 0

构造代码块

Son : 10

Son : 10

1. **实例化过程自我描述（正确）：**
2. **首先，从最高父类往下逐渐加载进内存，（Object类是JVM机运行时便加载进来），加载的时候先对成员变量进行默认初始化然后执行静态代码块，静态初始化时对于final修饰的或者static修饰的成员变量（常量）直接赋予设定值， 故最先是成员变量的默认初始化然后执行静态代码块，此过程过程从最高父类往下执行；这是从上往下。**
3. **然后，利用super（）；以及同一个类中利用this（）；逐步上升，除了这两个语句，其他没有执行任何代码；这是从下往上。**
4. **等到调用到最高父类的构造方法后，最高父类即object类，对每一个构造方法都一样，首先对当前类的成员变量进行显式初始化，紧接着执行当前类的构造代码块，然后执行当前构造方法中的从第二句开始的代码（ 第一句为显式或隐式的super语句或者this语句），就这样从高父类，一级一级地往下执行；这是从上往下走。**
5. **最终，完成了所有的初始化工作，创建了最低的子类的对象。**
6. **注意：（书写代码时，必须把所有的变量定义写在构造代码块和静态代码块的前面，否则出错，这也许就是为什么会先执行默认初始化或者显式初始化的原因吧，其次再执行静态代码块或者构造代码块了）。**
7. **对象的实例化过程中，父类构造方法对成员方法和成员变量的调用要与多态的向上转型后父类引用对成员变量和成员方法的调用类比起来：**

**（1）父类构造方法对成员变量的调用，调用的就是当前构造方法所在的类的成员变量，即父类中的成员变量，而父类中构造方法对成员方法的调用，同多态的一样，编译时看父类中是否有此方法，执行时从子类中寻找重写的方法，若没有重写，直接调用父类的方法；至于调用方法中又调用成员变量，则此成员变量附属此方法的所属，如果执行的方法是属于子类的，则方法中调用的变量就从子类中先寻找，若没有再利用父类的，若执行的方法是父类的，调用的变量也是父类的。**

**（2）多态中，对象的向上转型后，父类引用调用只有在调用非静态成员方法时，需要先观看子类的，如果子类没有，去父类中寻找，但是对于成员变量和静态方法而言，都是只看父类中的，子类中有与没有都没有任何关系。**

**（3）从上面看出父类的构造方法中对成员变量和成员方法的调用是一致的。重点是掌握父类构造方法或者父类引用又调用的方法中又调用成员变量的所属问题。所调用的方法是谁的，方法中调用的变量就是谁的。**

1. **例子说明顺序：**

**Class Student{**

**Super（）；//第一句**

**（显式初始化）**

**（构造代码块）**

**构造方法的其他语句**

**}**

1. **经典案例：必须 透彻，面试经常考察：见单独成章的。**