目录

Java中的构造方法

构造方法简介

构造方法是类的一种特殊方法，用来初始化类的一个新的对象。Java 中的每个类都有一个默认的构造方法，它必须具有和类名相同的名称，而且没有返回类型。构造方法的默认返回类型就是对象类型本身，并且构造方法不能被 static、final、synchronized、abstract 和 native 修饰。

提示：构造方法用于初始化一个新对象，所以用 static 修饰没有意义；构造方法不能被子类继承，所以用 final 和 abstract 修饰没有意义；多个线程不会同时创建内存地址相同的同一个对象，所以用 synchronized 修饰没有必要。

构造方法的init字节码级别的调用和内存分配，请看JVM中对象的创建内存布局和访问定位

构造方法的语法格式如下：

```
class class_name
{
    public class_name(){}    //默认无参构造方法
    public ciass_name([paramList]){}    //定义构造方法
    …
    //类主体
}
```

在一个类中，与类名相同的方法就是构造方法。每个类可以具有多个构造方法，但要求它们各自包含不同的方法参数。

构造方法实例

例 1

构造方法主要有无参构造方法和有参构造方法两种，示例如下：

```
public class MyClass
{
    private int m;    //定义私有变量
    MyClass()
    {
        //定义无参的构造方法
        m=0;
    }
    MyCiass(int m)
```

```
    {
        //定义有参的构造方法
        this.m=m;
    }
}
```

该示例定义了两个构造方法，分别是无参构造方法和有参构造方法。在一个类中定义多个具有不同参数的同名方法，这就是方法的重载。这两个构造方法的名称都与类名相同，均为 MyClass。在实例化该类时可以调用不同的构造方法进行初始化。

注意：类的构造方法不是要求必须定义的。如果在类中没有定义任何一个构造方法，则 Java 会自动为该类生成一个默认的构造方法。默认的构造方法不包含任何参数，并且方法体为空。如果类中显式地定义了一个或多个构造方法，则 Java 不再提供默认构造方法。

例 2

要在不同的条件下使用不同的初始化行为创建类的对象，这时候就需要在一个类中创建多个构造方法。下面通过一个示例来演示构造方法的使用。

(1) 首先在员工类 Worker 中定义两个构造方法，代码如下：

```
public class Worker
{
    public String name;    //姓名
    private int age;    //年龄
    //定义带有一个参数的构造方法
    public Worker(String name)
    {
        this.name=name;
    }
    //定义带有两个参数的构造方法
    public Worker(String name,int age)
    {
        this.name=name;
        this.age=age;
    }
    public String toString()
    {
        return"大家好！我是新来的员工，我叫"+name+"，今年"+age+"岁。";
    }
}
```

在 Worker 类中定义了两个属性，其中 name 属性不可改变。分别定义了带有一个参数和带有两个参数的构造方法，并对其属性进行初始化。最后定义了该类的 toString() 方法，返回一条新进员工的介绍语句。

提示：Object 类具有一个 toString() 方法，该方法是个特殊的方法，创建的每个类都会继承该方法，它返回一个 String 类型的字符串。如果一个类中定义了该方法，则在调用该类对象时，将会自动调用该类对象的 toString() 方法返回一个字符串，然后使用“System.out.println(对象名)”就可以将返回的字符串内容打印出来。

(2) 在 TestWorker 类中创建 main() 方法作为程序的入口处，在 main() 方法中调用不同的构造方法实例化 Worker 对象，并对该对象中的属性进行初始化，代码如下：

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

```
        System.out.println("-----------带有一个参数的构造方法-----------");
        //调用带有一个参数的构造方法，Staff类中的sex和age属性值不变
        Worker worker1=new Worker("张强");
        System.out.println(worker1);
        System.out.println("-----------带有两个参数的构造方法------------");
        //调用带有两个参数的构造方法，Staff类中的sex属性值不变
        Worker worker2=new Worker("李丽",25);
        System.out.println(worker2);
    }
}
```

在上述代码中，创建了两个不同的 Worker 对象：一个是姓名为张强的员工对象，一个是姓名为李丽、年龄为 25 的员工对象。对于第一个 Worker 对象 Worker1，并未指定 age 属性值，因此程序会将其值采用默认值 0。对于第二个 Worker 对象 Worker2，分别对其指定了 name 属性值和 age 属性值，因此程序会将传递的参数值重新赋值给 Worker 类中的属性值。

运行 TestWorker 类，输出的结果如下：

```
-----------带有一个参数的构造方法-----------
大家好！我是新来的员工，我叫张强，今年0岁。
-----------带有两个参数的构造方法------------
大家好！我是新来的员工，我叫李丽，今年25岁。
```

通过调用带参数的构造方法，在创建对象时，一并完成了对象成员的初始化工作，简化了对象初始化的代码。

Java中的几种构造方法详解

普通构造方法

方法名与类名相同

无返回类型

子类不能继承父类的构造方法

不能被static、final、abstract修饰（有final和static修饰的是不能被子类继承的，abstract修饰的是抽象类，抽象类是不能实例化的，也就是不能new）

可以被private修饰，可以在本类里面实例化，但是外部不能实例化对象（注意！！！）

```
public class A{
    int i=0;
    public A(){
        i=2;
    }
    public A(int i){
        this.i=i;
    }
}
```

默认构造方法

如果没有任何的构造方法，编译时系统会自动添加一个默认无参构造方法

隐含的默认构造方法

```
public A(){}
```

显示的默认构造方法

```
public A(){
System.out.print("显示的默认构造方法")
}
```

重载构造方法

比如原本的类里的构造方法是一个参数的，现在新建的对象是有三个参数，此时就要重载构造方法

当一个类中有多个构造方法，有可能会出现重复性操作，这时可以用this语句调用其他的构造方法。

```
public class A{
    private int age;
    private String name;
    public A(int age,String name){
        this.age=age;
        this.name=name;
    }
    public A(int age){
        this(age,"无名氏");//调用 A(int age,String name)构造方法
    }
    public A(){
        this(1);//调用 A(int age)构造方法
    }
    public void setName(String name) {this.name=name;}
    public String getName() {return name;}
    public void setAge(int age) {this.age=age;}
    public int getAge() {return age;}
}

A a=new A(20,"周一");
A b=new A(20);
A c=new A();
String name = a.getName();
String name1 = b.getName();
int age = c.getAge();
System.out.println(name);
System.out.println(name1);
System.out.println(age);
```

java子类构造方法调用父类构造方法

首先父类构造方法是绝对不能被子类继承的。

子类构造方法调用父类的构造方法重点是：子类构造方法无论如何都要调用父类的构造方法。

子类构造方法要么调用父类无参构造方法（包括当父类没有构造方法时。系统默认给的无参构造方法），要么调用父类有参构造方法。当子类构造方法调用父类无参构造方法，一般都是默认不写的，要写的话就是super（），且要放在构造方法的第一句。当子类构造方法要调用父类有参数的构造方法，那么子类的构造方法中必须要用super（参数）调用父类构造方法，且要放在构造方法的第一句。

当子类的构造方法是无参构造方法时，必须调用父类无参构造方法。因为系统会自动找父类有没有无参构造方法，如果没有的话系统会报错：说父类没有定义无参构造方法。

当子类构造方法是有参构造方法时，这时就会有两种情况。 第一种：子类构造方法没有写super，也就是说你默认调用父类无参构造方法，这样的话就和子类是无参构造方法一样。

第二种：子类构造方法有super（参数）时，就是调用父类有参构造方法，系统会找父类有没有参数一致（参数数量，且类型顺序要相同）的有参构造方法，如果没有的话，同样也会报错。

但是这里会遇到和重载构造方法this一样问题，一个参数的构造方法可以调用多个参数构造方法，没有的参数给一个自己定义值也是可以的。

Java中的代码块简介

在java中用{}括起来的称为代码块，代码块可分为以下四种：

一.简介

1.普通代码块：

类中方法的方法体

2.构造代码块：

构造块会在创建对象时被调用，每次创建时都会被调用，优先于类构造函数执行。

3.静态代码块：

用static{}包裹起来的代码片段，只会执行一次。静态代码块优先于构造块执行。

4.同步代码块：

使用synchronized（）{}包裹起来的代码块，在多线程环境下，对共享数据的读写操作是需要互斥进行的，否则会导致数据的不一致性。同步代码块需要写在方法中。

二.静态代码块和构造代码块的异同点

相同点：都是JVM加载类后且在构造函数执行之前执行，在类中可定义多个，一般在代码块中对一些static变量进行赋值。

不同点：静态代码块在非静态代码块之前执行。静态代码块只在第一次new时执行一次，之后不在执行。而非静态代码块每new一次就执行一次。

Java代码块使用

局部代码块

> 位置：局部位置（方法内部）

> 作用：限定变量的生命周期，尽早释放，节约内存

> 调用：调用其所在的方法时执行

```
 public class 局部代码块 {
@Test
public void test (){
    B b = new B();
    b.go();
}
}
class B {
    B(){}
    public void go() {
        //方法中的局部代码块，一般进行一次性地调用，调用完立刻释放空间，避免在接下来的调用过程中占
用栈空间
        //因为栈空间内存是有限的，方法调用可能会会生成很多局部变量导致栈内存不足。
        //使用局部代码块可以避免这样的情况发生。
        {
            int i = 1;
            ArrayList<Integer> list = new ArrayList<>();
            while (i < 1000) {
                list.add(i ++);
            }
            for (Integer j : list) {
                System.out.println(j);
            }
            System.out.println("gogogo");
        }
        System.out.println("hello");
    }
}
```

构造代码块，可以参考第四篇文章，关键字static中初始化顺序

> 位置：类成员的位置，就是类中方法之外的位置

> 作用：把多个构造方法共同的部分提取出来，共用构造代码块

> 调用：每次调用构造方法时，都会优先于构造方法执行，也就是每次new一个对象时自动调用，对对象的初始化

```
class A{
    int i = 1;
    int initValue;//成员变量的初始化交给代码块来完成
    {
        //代码块的作用体现于此：在调用构造方法之前，用某段代码对成员变量进行初始化。
        //而不是在构造方法调用时再进行。一般用于将构造方法的相同部分提取出来。
        //
        for (int i = 0;i < 100;i ++) {
            initValue += i;
        }
    }
    {
        System.out.println(initValue);
        System.out.println(i);//此时会打印1
        int i = 2;//代码块里的变量和成员变量不冲突，但会优先使用代码块的变量
        System.out.println(i);//此时打印2
        //System.out.println(j);//提示非法向后引用，因为此时j的的初始化还没开始。
        //
    }
    {
        System.out.println("代码块运行");
    }
```

```
    int j = 2;
    {
        System.out.println(j);
        System.out.println(i);//代码块中的变量运行后自动释放，不会影响代码块之外的代码
    }
    A(){
        System.out.println("构造方法运行");
    }
}
public class 构造代码块 {
    @Test
    public void test() {
        A a = new A();
    }
}
```

静态代码块

位置：类成员位置，用static修饰的代码块

作用：对类进行一些初始化　只加载一次，当new多个对象时，只有第一次会调用静态代码块，因为，静态代码块是属于类的，所有对象共享一份

调用：new 一个对象时自动调用

```
public class 静态代码块 {

@Test
public void test() {
    C c1 = new C();
    C c2 = new C();
    //结果,静态代码块只会调用一次，类的所有对象共享该代码块
    //一般用于类的全局信息初始化
    //静态代码块调用　类加载阶段
    //代码块调用　new对象阶段的，构造方法之前
    //构造方法调用　最晚执行程序员认定的构造方法
    //代码块调用
    //构造方法调用
}

}
class C{
    C(){
        System.out.println("构造方法调用");
    }
    {
        System.out.println("代码块调用");
    }
    static {
        System.out.println("静态代码块调用");
    }
}
```

Java代码块、构造方法（包含继承关系）的执行顺序

这是一道常见的面试题，要回答这个问题，先看看这个实例吧。

一共3个类：A、B、C 其中A是B的父类，C无继承仅作为输出

A类：

```
public class A {

static {
    Log.i("HIDETAG", "A静态代码块");
}

private static C c = new C("A静态成员");
private  C c1 = new C("A成员");

{
    Log.i("HIDETAG", "A代码块");
}

static {
    Log.i("HIDETAG", "A静态代码块2");
}

public A() {
    Log.i("HIDETAG", "A构造方法");
}

}
```

B类：

```
public class B extends A {

private static C c1 = new C("B静态成员");

{
    Log.i("HIDETAG", "B代码块");
}

private C c = new C("B成员");

static {
    Log.i("HIDETAG", "B静态代码块2");
}

static {
    Log.i("HIDETAG", "B静态代码块");
}

public B() {
    Log.i("HIDETAG", "B构造方法");

}

}
```

C类：

```
public class C {

public C(String str) {
    Log.i("HIDETAG", str + "构造方法");
```

```
}
}
```

执行语句：new B();

输出结果如下：

```
I/HIDETAG: A静态代码块
I/HIDETAG: A静态成员构造方法
I/HIDETAG: A静态代码块2
I/HIDETAG: B静态成员构造方法
I/HIDETAG: B静态代码块2
I/HIDETAG: B静态代码块
I/HIDETAG: A成员构造方法
I/HIDETAG: A代码块
I/HIDETAG: A构造方法
I/HIDETAG: B代码块
I/HIDETAG: B成员构造方法
I/HIDETAG: B构造方法
```

得出结论：

```
执行顺序依次为：
父类的静态成员和代码块
子类静态成员和代码块
父类成员初始化和代码快
父类构造方法
子类成员初始化和代码块
子类构造方法
```

注意：可以发现，同一级别的代码块和成员初始化是按照代码顺序从上到下依次执行

看完上面这个demo，再来看看下面这道题，看看你搞得定吗？

看下面一段代码，求执行顺序：

```
class A {
    public A() {
        System.out.println("1A类的构造方法");
    }
    {
        System.out.println("2A类的构造快");
    }
    static {
        System.out.println("3A类的静态块");
    }
}

public class B extends A {
    public B() {
        System.out.println("4B类的构造方法");
    }
    {
        System.out.println("5B类的构造快");
    }
    static {
        System.out.println("6B类的静态块");
    }
```

```
    public static void main(String[] args) {
        System.out.println("7");
        new B();
        new B();
        System.out.println("8");
    }
}
```

执行顺序结果为：367215421548

为什么呢?

首先我们要知道下面这5点：

每次new都会执行构造方法以及构造块。 构造块的内容会在构造方法之前执行。 非主类的静态块会在类加载时，构造方法和构造块之前执行，切只执行一次。 主类（public class）里的静态块会先于main执行。 继承中，子类实例化，会先执行父类的构造方法，产生父类对象，再调用子类构造方法。 所以题目里，由于主类B继承A，所以会先加载A，所以第一个执行的是第3句。

从第4点我们知道6会在7之前执行，所以前三句是367。

之后实例化了B两次，每次都会先实例化他的父类A，然后再实例化B，而根据第1、2、5点，知道顺序为2154。

最后执行8

所以顺序是367215421548

参考文章

https://blog.csdn.net/likunkun__/article/details/83066062 https://www.jianshu.com/p/6877aae403f7
https://www.jianshu.com/p/49e45af288ea https://blog.csdn.net/du_du1/article/details/91383128
http://c.biancheng.net/view/976.html https://blog.csdn.net/evilcry2012/article/details/79499786
https://www.jb51.net/article/129990.htm