JAVA基础：构造方法的初始化顺序（2） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/145/2021_2022_JAVA_E5_9F_BA_E7_A1_80_c104_145390.htm 在main 中，注意对 A 的构造方法的调用(就是invokespecial 行)，以及A的构造方法中产生的类似的对Object 构造方法的调用。 如果父类没有缺省构造方法，你必须明确使用"super(args)"调用父类的某个构造方法，例如，下面是一个错误的用法： class A { A(int i) {} } class B extends A {} 在上面的情况下， A 没有缺省的构造方法，但是B的构造方法必须调用A的某个构造方法。 让我们来看看初始化的另一个例子： class A { A() { System.out.println("A.A called"). } A(int i) { this(). System.out.println("A.A(int) called"). } } class B extends A { int i = f(). int j. { j = 37. System.out.println("initialization block executed"). } B() { this(10). System.out.println("B.B() called"). } B(int i) { super(i). System.out.println("B.B(int) called"). } int f() { System.out.println("B.f called"). return 47. } } public class CtorDemo3 { public static void main(String args[]) { B bobj = new B(). } } 程序的输出是： A.A called A.A(int) called B.f called initialization block executed B.B(int) called B.B() called 这个例子明确使用super() 和 this() 调用。this()调用是调用同一个类中的另一个构造方法；这个方法被称为“ 显式构造方法调用 ”。当那样的构造方法被调用，它将执行通常的super() 过程以及后续的操作。这意味着A.A 的方法体在A.A(int)之前执行，而这两个都在B.B(int) 和B.B 前执行。 如果返回第一个例子，你就可以回答为什么打印的是2而不是1。B 没有构造方法，因

此生成一个缺省构造方法，然后它调用super()，然后调用A产生的缺省构造方法。 然后A中的成员被初始化，成员a 被设置为方法f()的值，但是因为B 对象正被初始化，f() 返回值2。换句话说，调用的是B中的f()方法。 A产生的构造方法体被执行，然后B的成员被初始化，而b 被赋予值a，也就是2。最后，B的构造方法被执行。 最后一个例子说明了第一个例子的一个小小的变异版本： class A { int a = f(). int f() { return 1. } } class B extends A { int b = 37. int f() { return b. } } public class CtorDemo4 { public static void main(String args[]) { B bobj = new B(). System.out.println(bobj.a). System.out.println(bobj.f()). } } 程序的输出是： 0 37 你可能会期望输出的两个值bobj.a 和bobj.f()是一样的，但是正如你看到的他们不一样。这是正确的，即使是在a是从B的f方法中初始化的并且打印的是a 和 B的 f 方法的值。 这儿的问题是当a通过对B的f方法调用而初始化，而该方法返回成员b的值，而该成员还没有被初始化。因为这个，b的值就是刚开始的初始值0。 这些例子解释了编程中重要的一点——在对象的构造阶段调用可重载的方法是不明智的。 