Java 8 默认方法和多继承

2015/05/22 | 分类： [基础技术](http://www.importnew.com/cat/basic) | [0 条评论](http://www.importnew.com/16013.html#respond) | 标签： [多继承](http://www.importnew.com/tag/%e5%a4%9a%e7%bb%a7%e6%89%bf)

分享到：**1**

原文出处： [colobu-鸟窝](http://colobu.com/2014/11/04/Java-8-default-method-and-multiple-inheritance/)

以前经常谈论的Java对比c++的一个优势是Java中没有多继承的问题。 因为Java中子类只能继承(extends)单个父类， 尽管可以实现(implements)多个接口，但是接口中只有抽象方法，方法体是空的，没有具体的方法实现，不会有方法冲突的问题。

这些都是久远的说法了，自从今年Java 8发布后， 接口中也可以定义方法了(default method)。 之所以打破以前的设计在接口中  
增加具体的方法， 是为了既有的成千上万的Java类库的类增加新的功能， 且不必对这些类重新进行设计。 比如， 只需在Collection接口中  
增加default Stream<E> stream(), 相应的Set和List接口以及它们的子类都包含此的方法， 不必为每个子类都重新copy这个方法。

这是一个折衷的设计，带来的问题就是为Java引入了多继承的问题。 我们知道， 接口可以继承接口， 类可以继承类和实现接口。 一旦继承的类和实现的接口中有相同签名的方法， 会出现什么样的状况呢？ 本文将探讨各种情况的多继承， 以便能清楚的理解Java多继承的规则。

接口继承多个父接口

假定有三个接口Interface A, Interface B, Interface C, 继承关系如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | +---------------+         +------------+  |  Interface A  |         |Interface B |  +-----------^---+         +---^--------+              |                 |              |                 |              |                 |              +-+------------+--+                | Interface C|                +------------+ |

A,B拥有相同签名的默认方法default String say(String name), 如果接口C没有override这个方法， 则编译出错。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | interface A {      default String say(String name) {          return "hello " + name;      }  }  interface B {      default String say(String name) {          return "hi " + name;      }  }  interface C extends A,B{    } |

错误信息：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | C:\Lambda\src>javac -J-Duser.country=US com\colobu\lambda\chap  ter3\MultipleInheritance1.java  com\colobu\lambda\chapter3\MultipleInheritance1.java:17: error: interface C inherits unrelated defaults for say(String) from types A and B          static interface C extends A,B{                 ^  1 error |

我们可以在子接口C中覆盖override这个方法, 这样编译就不会出错了：

|  |  |
| --- | --- |
| 1  2  3  4  5 | interface C extends A,B{      default String say(String name) {          return "greet " + name;      }  } |

注意方法签名不包括方法的返回值， 也就是仅仅返回值不同的两个方法的签名也是相同的。下面的代码编译不会出错，因为A和B的默认方法不同， C隐式继承了两个默认方法。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | interface A {      default void say(int name) {        }  }  interface B {      default void say(String name) {        }  }  interface C extends A,B{    } |

但是有的情况下即使是不同签名的方法也是很难分辨的：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | interface A {      default void say(int a) {          System.out.println("A");      }  }  interface B {      default void say(short a) {          System.out.println("B");      }  }  interface C extends A,B{    }  static class D implements C {    }  public static void main(String[] args) {      D d = new D();      byte a = 1;      d.say(a); //B  } |

Java会选择最适合的方法， 请参看[Java规范 15.12.2.5](http://docs.oracle.com/javase/specs/jls/se8/html/jls-15.html#jls-15.12.2.5)

接口多层继承

下面看一下多层继承的问题。 继承关系如下图， A2继承A1, C继承A2。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | +---------------+  |  Interface A1 |  +--------+------+           |           |           |  +--------+------+  |  Interface A2 |  +-------+-------+          |          |          |  +-------+--------+  |   Interface C  |  +----------------+ |

基于我们以前对类继承的认识， 很容易知道C会继承A2的默认方法，包括直接定义的默认方法， 覆盖的默认方法，以及隐式继承于A1接口的默认方法。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | interface A {      default void say(int a) {          System.out.println("A");      }        default void run() {          System.out.println("A.run");      }  }  interface B extends A{      default void say(int a) {          System.out.println("B");      }        default void play() {          System.out.println("B.play");      }  }  interface C extends A,B{    } |

多层多继承

上面一个例子还是单继承的例子， 如果如下图的多继承呢？

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | 2  3  4  5  6  7  8  9  10  11  12  13  14  15  +---------------+  |  Interface A1 |  +--------+------+           |           |           |  +--------+------+         +---------------+  |  Interface A2 |         |  Interface B  |  +-------+-------+         +---------+-----+          |       +---------+---------^          |       |          |       |  +-------+-------++  |   Interface C  |  +----------------+ |

如果A2和B拥有相同签名的方法，这和第一个例子一样。 如果不想编译出错，可以覆盖父接口的默认方法，还可以调用指定父接口的默认方法：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | interface A1 {          default void say(int a) {              System.out.println("A1");          }      }        interface A2 extends A1 {        }        interface B {          default void say(int a) {              System.out.println("B");          }      }      interface C extends A2,B{          default void say(int a) {              B.super.say(a);          }      } |

更复杂的多层多继承

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | +--------------+   | Interface A1 |   +------+------++          |      ^+-------+          |               |  +-------+-------+       |  |  Interface A2 |       |  +------------+--+       |               ^--++      |                   |      |                +--+------+-----+                |  Interface C  |                +---------------+ |

接口A2继承A1， 接口C继承A2和A1。 代码如下，

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | interface A1 {      default void say() {          System.out.println("A1");      }  }  interface A2 extends A1 {      default void say() {          System.out.println("A2");      }  }  interface C extends A2,A1{    }  static class D implements C {    }  public static void main(String[] args) {      D d = new D();      d.say();  } |

以上代码不会编译出错，运行输出A2。

可以看到接口C会隐式继承子接口的方法， 也就是子接口A2的默认方法。

类继承

如果继承关系类型全部是类， 那么由于类依然是单继承的， 不会有多继承的问题。

类和接口混杂

我们把第一个例子中的其中一个接口换成类，会出现什么现象呢。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | +-------------+       +-----------+  | Interface A |       |  Class B  |  +-----------+-+       +-----+-----+              ^-+    +--+-----^                |    |            +---+----+-+            |  Class C |            +----------+ |

以下代码不会编译出错：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | interface A {      default void say() {          System.out.println("A");      }  }  static class B {      public void say() {          System.out.println("B");      }  }  static class C extends B implements A{    }  public static void main(String[] args) {      C c = new C();      c.say(); //B  } |

结果输出B。

可以看出， 子类优先继承父类的方法， 如果父类没有相同签名的方法，才继承接口的默认方法。

结论

更复杂的继承关系可以简化成以上的继承关系。  
根据以上的例子， 可以得出以下的结论：

* 类优先于接口。 如果一个子类继承的父类和接口有相同的方法实现。 那么子类继承父类的方法
* 子类型中的方法优先于父类型中的方法。
* 如果以上条件都不满足， 则必须显示覆盖/实现其方法，或者声明成abstract。