Java 8 - 默认方法

理解Java 8 默认方法需理解几个问题:

- 为什么会出现默认方法?
- 接口中出现默认方法，且类可以实现多接口的，那和抽象类有啥区别?
- 多重实现的默认方法冲突怎么办?

什么是默认方法，为什么要有默认方法

先上例子

一个接口A，Clazz类实现了接口A。

```
public interface A {
    default void foo(){
       System.out.println("Calling A.foo()");
    }
}

public class Clazz implements A {
    public static void main(String[] args){
       Clazz clazz = new Clazz();
       clazz.foo();//调用A.foo()
    }
}
```

代码是可以编译的，即使Clazz类并没有实现foo()方法。在接口A中提供了foo()方法的默认实现。

什么是默认方法

简单说，就是接口可以有实现方法，而且不需要实现类去实现其方法。只需在方法名前面加个default关键字即可。

为什么出现默认方法

为什么要有这个特性? 首先，之前的接口是个双刃剑，好处是面向抽象而不是面向具体编程，缺陷是，当需要修改接口时候，需要修改全部实现该接口的类，目前的java 8之前的集合框架没有foreach方法，通常能想到的解决办法是在JDK里给相关的接口添加新的方法及实现。然而，对于已经发布的版本，是没法在给接口添加新方法的同时不影响已有的实现。所以引进的默认方法。他们的目的是为了解决接口的修改与现有的实现不兼容的问题。

java 8抽象类与接口对比

这一个功能特性出来后，很多同学都反应了，java 8的接口都有实现方法了，跟抽象类还有什么区别? 其实还是有的，请看下表对比。

相同点	不同点
都是抽象类型	抽象类不可以多重继承，接口可以(无论是多重类型继承还是多重行为继承)
都可以有实现方法(以前接口不行)	抽象类和接口所反映出的设计理念不同。其实抽象类表示的是”is-a”关系，接口表示的是”like-a”关系
都可以不需要实现类或者继承者去实现所有方法，(以前不行，现在接口中默认方法不需要实现者实现)	接口中定义的变量默认是public static final 型，且必须给其初值，所以实现类中不能改变其值；抽象类中的变量默认是 friendly 型，其值可以在子类中重新定义，也可以重新赋值。

多重继承的冲突

由于同一个方法可以从不同接口引入，自然而然的会有冲突的现象，默认方法判断冲突的规则如下:

1.一个声明在类里面的方法优先于任何默认方法(classes always win)

2.否则，则会优先选取路径最短的。

举例子

- Case 1

```
public interface A{
    default void aa() {
        System.out.println("A's aa");
    }
}
public interface B{
    default void aa() {
        System.out.println("B's aa");
    }
}
public static class D implements A,B{

}
```

报错 Duplicate default methods named aa with the parameters () and () are inherited from the types DocApplication.B and DocApplication.A

如果一定要这么写呢，同时实现A,B并且使用A中aa? 可以这么写:

```
public static class D implements A,B{
    @Override
    public void aa(){
        A.super.aa();
    }
}
```

- Case 2

```
public interface A{
    default void aa() {
        System.out.println("A's aa");
    }
}
public interface B{
    default void aa() {
        System.out.println("B's aa");
    }
}
public interface C extends A, B{
    default void aa() {
        System.out.println("C's aa");
    }
}
public static class D implements A,B,C{

}
```

输出 C's aa

- Case 3

```
public interface A{
    default void aa() {
        System.out.println("A's aa");
    }
}
public interface C extends A{
    default void aa() {
        System.out.println("C's aa");
    }
}
public static class D implements C{

}
```

输出 C's aa

> 通过Case1-3可以知道它是找唯一的最短路径的default，如果是多个那么报错。

- Case 4 如果想调用A的默认函数，则用到新语法X.super.m(…),下面修改C类，实现A接口，重写一个hello方法，如下所示:

```
public interface A{
    default void aa() {
        System.out.println("A's aa");
    }
}
public class X implements A{
    @Override
    public void aa(){
        A.super.aa();
    }
}
```

输出: A's aa

- Case 5

```
public interface A{
    default void aa() {
        System.out.println("A's aa");
    }
}
public interface B{
    default void aa() {
        System.out.println("B's aa");
    }
}
public interface C extends A,B{
    default void aa() {
        System.out.println("C's aa");
    }
}
public static class D implements C{
    @Override
    public void aa(){
        C.super.aa();
    }
}
```

输出 C's aa 可见C.super表示的是C接口，同时D无法访问A,B的aa

> 通过Case 5也可以看出，C虽然有同一个两个最短路径的aa, 但是它自己有一个更高优先级的aa，所以不会报错; case 6 会报错

- Case 6

```
public interface A{
    default void aa() {
        System.out.println("A's aa");
    }
}
public interface B{
    default void aa() {
        System.out.println("B's aa");
    }
}
public interface C extends A,B{
}
```

总结

默认方法给予我们修改接口而不破坏原来的实现类的结构提供了便利，目前java 8的集合框架已经大量使用了默认方法来改进了，当我们最终开始使用Java 8的lambdas表达式时，提供给我们一个平滑的过渡体验。也许将来我们会在API设计中看到更多的默认方法的应用。