泛型代码和虚拟机

类型擦除、原始类型、给JVM的指令、桥方法、Java泛型转换的事实

@author ixenos

* 类型擦除(type erasure)
  + Java泛型的处理在编译器中运行，编译生成的字节码bytecode不包含泛型信息，泛型信息在编译处理时被擦除(erasure)，这个过程即类型擦除
* 原始类型(raw type)
  + 定义一个泛型类型，编译器自动提供一个相应的原始类型(raw type)，原始类型就是删除类型参数（包括**泛型类型变量**及其**限定类型**）后的泛型类型名。擦除类型变量，原始类型替换为**限定类型**（无限定则用Object）
  + 泛型变量有限定类型示例：<T extends Comparable & Serializable> ,因为Comparable包含实现方法CompareTo，而Serializable只是个标签（Tagging）接口，若二者对调位置，则原始类型将用Serializable来替换，这样在编译器必要时要向Comparable插入强制类型转换，所以为了提高效率，最好将标签（Tagging）接口放在边界列表的末尾
    - 标签（Tagging）接口：没有方法的接口
* 给JVM的指令：擦除方法返回类型，**编译器**将方法调用处理为两条JVM虚拟机指令
  + 对原始方法的调用（返回Object类型或限定类型）
  + 将返回的Object类型强制转换为泛型类型的具体类型（如Employee）
    - 补充#当存取一个泛型域时也要插入强制类型转换
* 桥方法（bridge method）
  + 擦除前

Class DateInterval extends Pair<Date>

{

Public void setSecond(Date second)

{

If(…)

Super.setSecond(second);

}

}

* 擦除后

Class DateInterval extends Pair

{

Public void setSecond(Date second){…}

}

而这时候还出现了另一个从Pair继承的setSecond方法，形参是Object类型 ： public void setSecond(Object second){…}

* 然后考虑下面的语句（要求多态调用）

DateInterval interval = new DateInterval(…);

Pair<Date> pair = interval; //向上转型(assignment to superclass)

pair.setSecond(aDate);

* 此时出现问题一：类型擦除和多态冲突。

向上转型后pair的setSecond方法可以多态调用了（即调用父类或子类的方法，一般调用子类方法），但是**类型擦除**后只能调用原始类型的方法。

问题一解决方案：编译器在DateInterval类中生成一个**桥方法：**

public void setSecond(**Object second**) {setSecond((Date) second)}

1. 当用pair.setSecond(aDate)调用方法时，将会调用DateInterval.setSecond(Object)方法（**桥方法**）
2. DateInterval.setSecond(Object)将调用DateInterval.setSecond(Date)方法

* 问题二：子类重写无参方法，类中出现两个仅返回类型不同的方法。

返回类型的不同不算多态(重载)，是重写，所以一个类中出现两个仅有返回类型不同的方法在Java语言中是非法的，通不过编译。

假设DateInterval方法也覆盖了getSecond方法（仅改变返回类型）：

Class DateInterval extends Pair<Date>

{

Public Date getSecond()

{

return (Date)super.getSecond().clone();

}

}

我们知道，返回类型的不同不算多态(重载)，是重写，所以出现两个仅有返回类型不同的方法在Java语言中是非法的，通不过编译。

然而，在擦除的类型中，即在DateInterval类中出现了两个getSecond方法（）：

Date getSecond() //在DateInterval中定义

Object getSecond() //**桥方法**！重写定义在Pair类中的方法，将会调用上面的方法Date getSecond()

#而在JVM虚拟机中是用**返回值**和**参数类型**确定同一个方法的，因此如果编译器产生两个仅有返回类型不同的方法字节码，虚拟机能够正确地处理这一情况。

* + - * Java泛型转换的事实：
        + 虚拟机中没有泛型，只有普通的类和方法
        + 所有的类型参数都用他们的限定类型替换
        + 桥方法被合成来保持多态
        + 为保持类型安全性，必要时插入强制类型转换