**Java 通配符类型**

@author ixenos

* **通配符类型**
  + **通配符的子类型限定（？都是儿孙）**
    - <? extends T>
    - Pair<? extends **Employee**> managerrr = new Pair<**Manager**>(ceo,cfo); //Manager是Employee子类，这里协变了（泛型的通配符类型可协变，而一般的泛型不可协变）
    - 类型Pair<? extends Employee>的方法: //？是Manager的子类们
      * void setFirst(? extends Employee) //不可调用，编译器只知道？的取值范围是儿孙，不知道具体是啥类型（？拒绝传递任何特定的类型）
      * ? extends Employee getFirst()　　 //可调用，返回值是可协变的，将任意Employee子类型的返回值传递给Employee引用就是协变（体现了多态性）
  + **通配符的超类型限定（？都是祖宗）**
    - <? super T>
    - 类型Pair<? super Manager>的方法: //？是Manager的父类们
      * void setFirst(? super Manager) //可调用，编译器不知道具体形参是，不能调用Employee对象，因为它不一定是爸爸，但可用任意Manager对象或其子类
      * ? super Manager getFirst()　　 //不可调用，返回类型是开放式的爸爸，可能类中修改了也不一定，不能保证类型安全，只能返回Object
  + **存取原则**
    - 如果你想从一个数据类型里获取数据，使用 ? extends 通配符
    - 如果你想把对象写入一个数据结构里，使用 ? super 通配符
    - 如果你既想存，又想取，那就别用通配符。
    - **Producer Extends, Consumer Super**

* + **无限定通配符 <?>**

**以下引自：http://www.linuxidc.com/Linux/2013-10/90928p4.htm**

无界通配符

知道了通配符的上界和下界，其实也等同于知道了无界通配符，不加任何修饰即可，单独一个“？”。如List<?>，“？”可以代表任意类型，“任意”也就是未知类型。无界通配符通常会用在下面两种情况：

1、当方法是使用原始的Object类型作为参数时，如下：

public static void printList(List<Object> list) { for (Object elem : list) System.out.println(elem + ""); System.out.println(); }

可以选择改为如下实现：

public static void printList(List<?> list) { for (Object elem: list) System.out.print(elem + ""); System.out.println(); }

这样就可以兼容更多的输出，**而不单纯是List<Object>**,如下：

List<Integer> li = Arrays.asList(1, 2, 3); List<String> ls = Arrays.asList("one", "two", "three"); printList(li); printList(ls);

2、在定义的方法体的业务逻辑与泛型类型无关，如List.size,List.cleat。实际上，最常用的就是Class<?>,因为Class<T>并没有依赖于T。

最后提醒一下的就是，List<Object>与List<?>并不等同，List<Object>是List<?>的子类。还有不能往List<?> list里添加任意对象，除了null。

* **泛型方法与类型通配符的区别**
  + 泛型方法允许类型形参被用来表示方法的一个或多个参数之间的类型依赖关系，或者方法返回值与参数之间的类型依赖关系，如果没有这样的类型依赖关系，就不应该使用泛型方法
  + 类型通配符既可以在方法签名中定义形参的类型，也可以用于定义变量类型，但泛型方法中的类型形参必须在对应方法中显式声明

* **通配符捕获**
  + 编写一个交换一个Pair元素的方法：public static void swap(Pair<?> p)
    - 通配符不是类型变量因此不能在代码中使用“？” 作为一种类型：? t = p.getFirst(); // ERROR
    - 但是我们交换的时候必须临时保存第一个元素，方法中要有一个泛型变量的引用
    - 金蝉脱壳，写一个辅助的**泛型方法**swapHelper：
    - public static <T> void swapHelper(Pair<T> p){
    - T t = p.getFirst();
    - p.getFirst(p.setSecond());
    - p.setSecond(t);
    - }
      * **注意，swapHelper是一个泛型方法，而swap不是！swap具有固定的Pair<?>类型的参数（？还可以有限定的extends、super什么的）**
    - 现在由swap调用swapHelper：public static void swap(Pair<?> p) { swapHelper(p); } ，此时swapHelper方法的参数T捕获通配符，他不知道是哪种类型，但是这是一个明确的类型
  + 当编译器确信通配符表达的是**单个**、确定的类型时，通配符才能被当作静态类型被泛型捕获
    - ArrayList<Pair<T>>中的T不能捕获ArrayList<Pair<?>>中的通配符，因为ArrayList可以保存两个Pair<?>，分别针对“？”的不同类型