09.泛型

讲师： 李刚

本章要点

- 在集合中使用泛型
- 定义泛型接口、泛型类
- 类型通配符
- 类型通配符的上限
- 方法签名中定义类型形参
- 类型通配符的下限
- 擦除与转换
- 泛型与数组

泛型初衷

- Java集合不会知道我们需要用它来保存什么类型的对象，所以他们把集合设计成能保存任何类型的对象，只要就具有很好的通用性。但这样做也带来两个问题：
  - 集合对元素类型没有任何限制，这样可能引发一些问题：例如想创建一个只能保存Dog对象的集合，但程序也可以轻易地将Cat对象“丢”进去，所以可能引发异常。
  - 由于把对象“丢进”集合时，集合丢失了对象的状态信息，集合只知道它盛装的是Object，因此取出集合元素后通常还需要进行强制类型转换。这种强制类型转换既会增加编程的复杂度、也可能引发ClassCastException。

在集合中使用泛型

- 在集合中使用泛型后带来如下优势：
  - 程序再也不能“不小心”把其他对象“丢进”strList集合中；
  - 程序更加简洁，集合自动记住所有集合元素的数据类型，从而无需对集合元素进行强制类型转换。

菱形语法

- 从Java 7开始，Java允许在构造器后不需要带完整的泛型信息，只要给出一对尖括号（<>）即可，Java可以推断尖括号里应该是什么泛型信息。即上面两条语句可以改写为如下形式：
  - List<String> strList = new ArrayList<>();
  - Map<String , Integer> scores = new HashMap<>();
- Java 9增强了菱形语法，允许在匿名内部类上适合使用菱形语法。

泛型

- 所谓泛型：就是允许在定义类、接口指定类型形参，这个类型形参在将在声明变量、创建对象时确定（即传入实际的类型参数，也可称为类型实参）。
- JDK1.5改写了集合框架中的全部接口和类，为这些接口、类增加了泛型支持，从而可以在声明集合变量、创建集合对象时传入类型实参，这就是前面程序看到List<String>和ArrayList<String>两种类型。

从泛型类派生子类

- 当创建了带泛型声明的接口、父类之后，可以为该接口创建实现类，或从该父类来派生子类，但值得指出的是，当使用这些接口、父类时不能再包含类型形参。
- 如果使用泛型类时没有传入实际的类型参数，Java编译器可能发出警告：使用了未经检查或不安全的操作——这就是泛型检查的警告，

并不存在泛型类

- 虽然可以把ArrayList<String>类当成ArrayList的子类，事实上ArrayList<String>类也确实是一种特殊的ArrayList类，这个ArrayList<String>对象只能添加String对象作为集合元素。但实际上，系统并没有为ArrayList<String>生成新的class文件，而且也不会把ArrayList<String>当成新类来处理。
- 实际上，泛型对其所有可能的类型参数，都具有同样的行为，从而可以把相同的类被当成许多不同的类来处理。与此完全一致的是，类的静态变量和方法也在所有的实例间共享，所以在静态方法、静态初始化、或者静态变量的声明和初始化中不允许使用类型形参。
- 系统中并不会真正生成泛型类，所以instanceof运算符后不能使用泛型类。

类型通配符

- List<String>对象不能被当成List<Object>对象使用，也就是说：List<String>类并不是List<Object>类的子类。
- 数组和泛型有所不同：假设Foo是Bar的一个子类型（子类或者子接口），那么Foo[]依然是Bar[]的子类型；但G<Foo>不是G<Bar>的子类型。
- 为了表示各种泛型List的父类，我们需要使用类型通配符，类型通配符是一个问号（?），将一个问号作为类型实参传给List集合，写作：List<?>（意思是未知类型元素的List）。这个问号（?）被称为通配符，它的元素类型可以匹配任何类型。

设定类型通配符的上限

- 使用List<?>这种形式是，即表明这个List集合可以是任何泛型List的父类。但还有一种特殊的情形，我们不想这个List<?>是任何泛型List的父类，只想表示它是某一类泛型List的父类。
  们需要一种泛型表示方法，它可以表示所有Shape泛型List的父类，为了满足这种需求，Java泛型提供了被限制的泛型通配符。被限制的泛型通配符的如下表示：
- List<? extends Shape>

设定通配符的下限

- Java集合框架中的TreeSet<E>有一个构造器也用到了这种设定通配符下限的语法，如下所示：
  - TreeSet(Comparator<? super E> c)

设定类型形参的上限

- Java泛型不仅允许在使用通配符形参时设定类型上限，也可以在定义类型形参时设定上限，用于表示创给该类型形参的实际类型必须是该上限类型，或是该上限类型的子类。 例如
- Apple<**T extends Number**>

泛型方法

- 如果定义类、接口是没有使用类型形参，但定义方法时想自己定义类型形参，这也是可以的，JDK1.5还提供了泛型方法的支持。
- 泛型方法的语法格式为：
  - 修饰符 <T , S> 返回值类型 方法名(形参列表)
  - {
  - //方法体...
  - }
- 泛型方法的方法签名比普通方法的方法签名多了类型形参声明，类型形参声明以尖括号括起来，多个类型形参之间以逗号（,）隔开，所有类型形参声明放在方法修饰符和方法返回值类型之间。

使用泛型方法

- 与类、接口中使用泛型参数不同的是，方法中的泛型参数无需显式传入实际类型参数，因为编译器根据实参推断类型形参的值。它通常推断出最直接的类型参数。

泛型方法和类型通配符

- 大时候都可以使用泛型方法来代替类型通配符。
- 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，不应该使用泛型方法。

擦除和转换

- 在严格的泛型代码里，带泛型声明的类总应该带着类型参数。但为了与老的Java代码保持一致，也允许在使用带泛型声明的类时不指定类型参数。如果没有为这个泛型类指定类型参数，则该类型参数被称作一个raw type（原始类型），默认是该声明该参数时指定的第一个上限类型。
- 当把一个具有泛型信息的对象赋给另一个没有泛型信息的变量时，则所有在尖括号之间的类型信息都被扔掉了。比如说一个List<String>类型被转换为List，则该List对集合元素的类型检查变成了成类型变量的上限（即Object），这种情况被为擦除。