泛型

一、泛型入门

1.集合存在的问题

Java集合被设计成能保存任何类型的对象，只要求具有很好的通用性，但是这样会带来两个问题：

- 集合对添加的元素类型没有任何限制
- 集合只知道它盛装的是Object，因此取出元素后通常需要进行强制类型转换

2.使用泛型

使用泛型接口和泛型类 创建泛型接口和泛型类的方法：在接口、类后增加尖括号，尖括号里放一个数据类型，即表明这个接口、类只能保存特定类型的对象

GenericTest

3.改进的泛型语法

改进后的泛型语法：Java允许在构造器后不需要带完整的泛型信息，只要给出一堆尖括号<>即可 Java允许在创建匿名内部类时使用菱形语法

GenericTest

二、深入泛型

1.定义泛型接口、类

- Java允许在定义接口、类时声明泛型形参，泛型形参在整个接口、类体内可当成类型使用，几乎所有可使用普通类型的地方都可以使用泛型形参
- 可以为任何类、接口增加泛型声明

2.从泛型类派生子类

当创建了带泛型声明的接口、父类后，可以为该接口创建实现类，或从该父类派生子类，需要注意的时，但使用这些接口、父类时不能再包含泛型形参，而应该指定明确的泛型实参,如`public class Apple extends Fruit<String>`,或者省略泛型实参，如`publi class Apple extends Fruit`

如果使用Fruit<String>来派生子类，那么子类会继承具有明确泛型类型的方法和成员变量，如果子类要重写父类方法，需要注意这一点。如果使用Fruit来派生子类，此时系统会将Fruit<T>泛型形参当成Object类型处理

3.并不存在的泛型类

- 不管为泛型形参传入哪种泛型实参，它们依然被当成同一个类来处理
- 在静态方法、静态初始化块、静态变量的声明和初始化中不允许使用泛型形参。因为静态变量类型在类初始化时就需要确定，而泛型实参要在对象初始化时才能确定，此时编译器不明白泛型类型，因此不能使用泛型静态成员变量。而静态方法为所有对象共有，而各对象的泛型实参各不相同，此时就相互矛盾了，因此不能使用泛型静态方法

- instanceof运算符后不能使用泛型类

三、类型通配符

数组和泛型有所不同，假设A是B的一个子类型（子类或子接口），那么A[]依然是B[]的子类型，但G<A>不是G<B>的子类型。A[]自动向上转型为B[]的方式被称为型变，也就是说，数组支持型变，但泛型不支持型变

1.使用类型通配符

为了表示各种泛型类型的父类，可以使用类型通配符，类型通配符是一个问号（?），将一个问号作为类型实参传给泛型形参时，它可以匹配任何类型，其类型是Object 但是因为不明确泛型实参，所以在使用通配符的集合中，如`List<?> c = new ArrayList<String>()`,集合不能向其中添加对象，只能添加null对象。另一方面，可以调用get()方法来返回List<?>集合指定索引处的元素，其返回值是一个未知类型，但可以肯定的是，它总是一个Object

2.设定类型通配符的上限

设置类型通配符上限：在问号后面添加`extends Type`,如`List<? extends Fruit>`代表这是一个未知类型，但是这个未知类型一定是Fruit的子类型（也可以是Fruit本身），因此可以把Fruit看作通配符上限

与类型通配符(?)一样，指定通配符上限的集合不能向其中添加元素，只能从集合中取元素（取出的元素总是上限类型）。对于更广泛的泛型类来说，指定通配符上限就是为了支持类型型变，比如A是B的子类，那么G<A>就是G<? extends B>的子类，这种型变方式称为协变

3.设定类型通配符的下限

设置类型通配符下限：在问号后面添加`super Type`, 如`List<? super Fruit>`代表这是一个未知类型，但是这个未知类型一定是Fruit的父类型（也可以是Fruit本身），因此可以把Fruit看作通配符下限

与通配符上限相反，指定通配符下限的集合可以向其中添加元素，且添加的元素只能是Fruit类或其子类的实例。从集合取元素时只能被当成Object类型处理。指定类型通配符下限是为了支持类型逆变，比如A是B的子类，那么G<B>是G<? super A>的子类

协变与逆变

4.设定泛型形参的上限

泛型不仅允许在使用通配符形参时设定上限，而且可以在定义泛型形参时设定上限，用于表示传给该泛型形参的实际类型要么是该上限类型，要么是该上限类型的子类。

在某些情况下，需要为泛型形参设定多个上限（至多一个父类上限，可以有多个接口上限），表面该泛型形参必须是其父类的子类（或父类本身），并且实现多个上限接口。

四、泛型方法

1.定义泛型方法

泛型方法就是在声明方法时定义一个或多个泛型形参 格式如下：

> 修饰符 <T,s>返回值类型 方法名(形参列表) { //方法体... }

与类和接口中的泛型参数不同,方法的泛型参数无须显式传入实际类型参数

泛型方法

2.泛型方法和类型通配符的区别

如果使用泛型的目的是支持灵活的子类化,那么就使用类型通配符.如果是为了表示方法的一个或多个参数之间的类型依赖关系,或者方法的返回值与参数之间的类型依赖关系,那么就使用泛型方法

如果有需要,也可以同时使用泛型方法和通配符

类型通配符既可以在方法签名中定义形参类型,也可以用于声明变量的类型,而泛型方法中的泛型形参用于约束多个参数间或返回值之间的关系

3.菱形语法与泛型构造器

在签名中声明泛型形参的构造器被称为泛型构造器

一旦定义了泛型构造器,接下来在调用构造器时,就不仅可以让Java根据数据参数的类型来推断泛型形参的类型,而且可以显式地为构造器中的泛型形参指定实际的类型。但如果程序显式制定了泛型构造器中声明的泛型形参的实际类型,则不可以使用菱形语法。

4.泛型方法与法方法重载

略

5.改进的类型推断

类型推断主要有以下两个方面:

- 可通过调用方法的上下文来推断目标类型
- 可在方法调用链中，将推断得到的泛型传递到最后一个方法

五、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实际的类型。如果没有为这个泛型类指定实际的类型，此时被称为原始类型，默认是声明该泛型形参时的上限类型

泛型擦除与转换

六、泛型与数组

Java泛型有一个重要的设计原则——如果代码在编译时没有提出“未经检查的转换”警告，则在运行时不会引发ClassCastException异常。因此，数组元素的类型不能包含泛型变量或泛型形参，除非是无上限类型的通配符，但可以声明元素类型包含泛型变量或泛型形参的数组。