泛型

天津卓讯科技有限公司

第8章 泛型

- 8.1 泛型概述
- 8.2 泛型类和接口的定义及使用
- 8.3 有界类型参数
- 8.4 泛型方法
- 8.5 类型参数的通配符
- 8.6 擦除
- 8.7 泛型的局限

8.1 泛型概述

- 问题：编写一个简易的对象容器类，它可以存放任意数量对象？

使用泛型改写容器类

```
class MyGenericList<E>{
    public void add(E ele){ ...}
    public E get(int index){ ...}
    public int size(){ ...}
}
public class MyGenericListTest{
    public static void main(String[] args){
        MyGenericList<String> list = new MyGenericList<String>();
        list.add("123abc");

    }
}
```

- 只能往这个容器中放置定义时指定类型的对象了，取出容器中的对象时也无须进行强制类型转换了。这样就使得这个容器类具有了更高的通用性。

8.1.1 泛型定义

- 泛型：就是在定义类、接口、方法、方法参数或成员变量时，指定它的操作对象的数据类型为一个参数。
  - 在具体使用类、接口、方法、方法参数或成员变量时，将这个参数用具体的某一数据类型来代替。
- 泛型的好处：它在编译时进行类型安全检查，并且在运行时所有的转换都是强制的、隐式的。提高了代码的重用率。

8.2.1 泛型类和接口的定义

- 定义语法
  - class 类名<类型参数的名称> { ...}
  - interface 接口名<类型参数的名称> { ...}
- 说明：类型参数的名称建议使用单个大写字母。如常用的名有：
  - E：表示集合中的元素类型。
  - K：表示“键值对”中的键的类型。
  - V：表示“键值对”中的值的类型。
  - T：表示其它所有的类型。

8.2.2 从泛型类派生子类

- 当某个类的父类是泛型类时：这个子类要把类型参数传递给父类；也可以把子类定义成特定于指定类型的，这个子类就不再是泛型类了。

```
class SuperClass<T>{
    private T o;
    public SuperClass(T o){        this.o = o;    }
    public String toString(){        return "T:" + o;    }
}
/** 泛型类的继承 */
class SubClass <T> extends SuperClass<T>{
    public SubClass(T o){        super(o);    }
}
class SpecialSubClass extends SuperClass<String>{
    public SpecialSubClass(String o){        super(o);    }
}
```

8.2.3 实现泛型接口

- 具体子类要把类型参数传递给所实现的接口；也可以把具体子类定义成特定于指定类型的。

```
interface MyGenericTypeInterface<T>{ }
class MyImplement<T> implements MyGenericTypeInterface<T>{}
class IntegerImplement  implements MyGenericTypeInterface<Integer>{ }
```

8.3 有界类型参数

- 有界类型参数可以为泛型的类型参数指定一个上界

```
class Statistics<T extends Number> {
    private T[] arrs;
    public Statistics(T[] arrs) {
        this.arrs = arrs;
    }
    public double count() {  //计算数组中元素数值的总和
        double sum = 0.0;
        for (int i = 0; i < arrs.length; ++i) {
           sum += arrs[i].doubleValue();
        }
        return sum;
    }
}
```

8.4 泛型方法

- 泛型类中的任何实例方法本质上都是泛型方法。只有静态方法需要显式定义成泛型方法，定义语法：
  - 访问控制符 [修饰符] <类型参数列表> 返回值类型 方法名(参数列表)

```
public static <T extends Number> void max(T... args){
        T temp = (T)Integer.valueOf(0);
        for(T t : args){
            if(t.doubleValue() > temp.doubleValue()){
                temp = t;
            }
        }
        System.out.println(temp);
}
```

8.5 类型参数的通配符

- 当使用泛型类或接口声明属性、局部变量、参数类型或返回值类型时，可使用通配符来代替类型参数

```
public static void method(MyGenericList<?> ml){
}
```

- 通配符上界：
  - <? extends Number>表示这个类型参数必须是Number类或其子类的实例。
- 通配符下界：
  - <? super Integer>表示这个类型参数必须是Integer类或其父类的实例。

8.6 擦除

- JDK1.5以前的版本中没有泛型，为了保证对以前版本的兼容，Java采用了被称为擦除的方式来处理泛型。
  - 当Java代码被编译成字节码时，泛型类型的定义信息会被删除(擦除)，而使用界限类型或Object来代替泛型参数。在使用泛型类型时，也会用相应的强制转换（由类型参数来决定）以维持与类型参数的类型兼容。

8.7 泛型的局限

- 不能使用基本类型的类型参数
  - 因为，在擦除时基本类型无法用Object类来代替。可以使用基本类型的包装类来代替它们。
- 静态成员无法使用类型参数
  - 因为静态成员独立于任何对象，是在对象创建之前就已经存在了，此时，编译器根本还无法知道它使用的是哪一个具体的类型。
- 不能使用泛型类异常
  - Java代码中不能抛出也不能捕获泛型类的异常。
- 不能使用泛型数组：这是Java语法的规定。
  - Generic<Integer> arr[] = new Generic<Ingeger>[10];
- 不能实例化参数类型对象：也就是说，不能直接使用泛型的参数类型来构造一个对象。
  - public class A<T>{ T a = new T(); //编译报错}

小结

- 泛型类和接口的定义及使用
- 有界类型参数
- 泛型方法
- 类型参数的通配符
- 擦除
- 泛型的局限