泛型

目 录

1 什么是泛型

泛型的本质就是参数化类型，参数化类型重要性在于，允许创建一些类、接口和方法，其所操作的数据类型被指定为参数。例如我们可以使用泛型创建一个类，在这个类中可以自动使用不同类型的数据。

```
class GenericClass<T>{
…
}
```

2 泛型的作用

泛型的作用表现在，在没有泛型之前，可以使用 Object 来表示或者创建通用的类、接口和方法，但是这样作带来后果是需要作强制类型转换，而这种转换的条件是开发人员必须预知实际的参数类型，同时如果强制类型转换错误，编译器在编译时并不检查或提示转换错误，但是一旦在运行时发现错误则会产生异常，这对于一个程序来说是非常的不安全。

而泛型的作用则恰恰体现在程序编译时就对类型进行检查，同时不必再进行强制类型转换，因此处理过程得到简化，代码的重用性高，并且又安全。

3 泛型相关概念

在学习如何使用泛型之前，还必须要了解 2 个重要的概念：

1、参数化类型

2、类型变量

参数化类型：参数化类型包含一个类或者接口，以及实际的类型参数列表。

类型变量：是一种非限定性标识符，用来指定类、接口或者方法的类型

4 如何引入泛型

下面我们就通过一个简单的例子来看看如何在程序中引入泛型

```
class GenericClass<T>{
 private T obj;    //指定了类变量的类型
 public GenericClass(T obj){
     this.obj=obj;
 }
```

```
    public T getObj(){    //指定了方法的返回值类型
        return obj;
    }
}
```

与普通的类不同，使用泛型定义的类可以拥有一个或多个类型的类。下面我们就来看一个应用了泛型的类的结构：

我们首先来看类型变量的使用：使用大写的英文字母 T 作为类型变量的名称，放置在一对尖括号内，并放在类名的后面；

我们还可以使用类型变量指定方法的返回类型以及域和局部变量的类型

在使用类型变量时还要注意一些规范的使用

1、类型变量要使用大写字母，且比较短

2、在 Java 的类库中，使用变量 E 来表示集合的元素类型

3、使用字母 K 和 V 表示关键字和值的类型

4、字母 T 表示任意类型

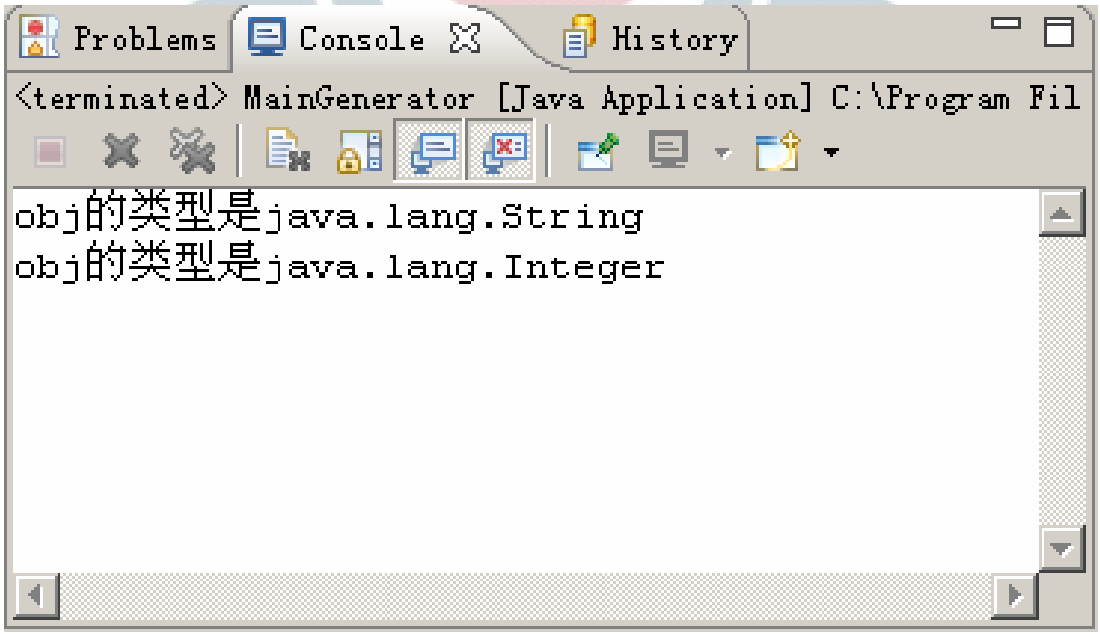
在使用泛型时，使用实际的类型来代替类型变量就可以实例化泛型类型。下面我们来看一个具体的泛型应用

```
public class GenericDemo{
    public static void main(String[] args) {
        GenericClass<String> ge1 = new GenericClass<String>("this is String object");
        ge1.showType();
        GenericClass<Integer> ge2 = new GenericClass<Integer>(100);
        ge2.showType();
    }
}
class GenericClass<T>{
```

```
    private T obj;
    public GenericClass(T obj){
        this.obj=obj;
    }
    public T getObj(){
        return obj;
    }
    public void showType(){
        System.out.println("obj 的类型是"+obj.getClass().getName());
    }
}
```

在 GenericDemo 类中，分别创建了 GenericClass 类的 2 个实例，一个使用 String 类型替代了类型变量 T，并将字符串赋予了成员变量 obj。另一个使用了 Integer 类型代替了类型变量 T，同时将 100 赋值给了类的属性那个 obj。

那么在GenericClass类中的showType()方法能够区分出域变量是String类型还是Integer类型吗，我们通过运行程序的效果来进行检验。



通过运行输出我们可以清楚的发现 GenericClass 类可以识别出，实际传递的数据类型并对 obj 进行赋值。换句话说就是 GenericClass<T>在使用中，能够根据 T 形参的实际类型，对应创

建多个逻辑形式的子类，例如 GenericClass<String >,GenericClass<Integer>等。

5 泛型类

5.1 什么是泛型类

什么是泛型类，简单的说就是具有一个或者多个类型变量的类。

5.2 如何定义泛型类

泛型类的定义比较简单，只要在类名的后面添加一对尖括号，并声明类型参数列表即可。

语法：

```
访问控制符 class className<TypeList>
```

TypelIst:表示类型参数列表，每个类型变量之间以逗号分隔

例如：

```
public class GenericClass<T>{
…
}
```

5.3 创建泛型类的实例

创建泛型类实例时，使用指定的实际类型代替类型参数列表中的类型变量即可。

语法：

```
new className<TypeList>（argList）
```

TypelIst:表示类型参数列表，每个类型变量之间以逗号分隔

argList：表示参数列表，同样以逗号分隔

例如：

```
new GenericClass<String>(“this is String object”)
```

GenericClass<T>是比较常见的泛型类，在定义泛型类时，还可以指定多个类型参数，下面我们就来看一个具有两个类型参数的泛型类

```
class GenericDemo<T,V>{
 private T a;
 private V b;
 public GenericDemo(T a,V b){
     this.a = a;
```

```
        this.b = b;
    }

    public void showType(){
        System.out.println("a 的类型是"+a.getClass().getName());
        System.out.println("b 的类型是"+b.getClass().getName());
    }
}
```

与 GenericClass<T>类不同的是，在 GenericDemo<T,V>类中定义了两个类型参数，分别是形参 T，和形参 V。那么这两个类型变量的具体类型并不知道。注意当在一个泛型中，需要声明多个类型参数时，只需要在每个类型参数之间使用逗号将其隔开即可。

由于 GenericDemo<T,V>设置两个类型参数，因此在实例化泛型类时，就需要传递两个类型参数。下面我们来看一下实例化 GenericDemo<T,V>的代码

```
public class Demo{
    public static void main(String[] args) {
        GenericDemo<String,Integer> ge1 = new
GenericDemo<String,Integer>("Jack",23);
        ge1.showType();
    }
}
```

在创建实例时，分别使用了 String 和 Integer 代替了形参 T 和 V，当然对于 T 和 V 两个形参，我们可以使用同一数据类型进行替代，只不过这样的话就不需要设置两个参数类型了。

6 泛型接口

6.1 什么是泛型接口

什么是泛型接口，泛型接口就是拥有一个或多个类型变量的接口

6.2 如何定义泛型接口

泛型接口的定义方式与定义泛型类类似。

语法：

```
访问控制符 interface interfaceName<TypeList>
```

TypeList：表示由逗号分隔的一个或多个类型参数列表

而实现泛型接口的类，也必须要指定参数类型。

语法：

```
Class className<TypeList> implements interfaceName<TypeList>
```

使用泛型接口时有几点需要注意：

1、一旦建立类型参数，在实现时将它不加修改地直接传递给接口即可

2、如果一个实现了一个泛型接口，那么这个类必须也是泛型

3、当一个实现了一个特定类型的泛型接口，这个类不必指定为泛型

泛型接口的优势：

1、针对不同的数据类型进行实现

2、允许对这些接口的数据类型进行限制

7 泛型方法

7.1 什么是泛型方法

泛型方法实际上就是带有参数类型的方法

需要特别注意的是，定义泛型方法与方法所在的类、或者接口是否是泛型类或者泛型接口没有直接的联系，也就是说无论是泛型类还是非泛型类，如果需要就可以定义泛型方法。

7.2 如何定义泛型方法

语法：

```
访问修饰符〈类型参数〉返回值 方法名（类型参数列表）
```

例如：

```
public 〈String〉 void showName(String s){}
```

注意在泛型方法中，类型变量是放置在访问修饰符与返回值之间。

调用泛型方法的语法是：

```
方法名称（参数列表）
```

7.3 如何使用泛型方法

了解了如何定义泛型方法和调用泛型方法后，我们通过一个实际的泛型方法应用来了解如何使用泛型方法。

```
public class GenericMethod {

 public <Integer> void showSize(Integer o){

     System.out.println(o.getClass().getName());

 }

 public static void main(String[] args) {

     GenericMethod gm = new GenericMethod();

     gm.  showSize(10);

}

}
```