java 泛型的基本介绍和使用

一、泛型的基本概念

泛型的定义：泛型是 JDK 1.5 的一项新特性，它的本质是参数化类型（Parameterized Type）的应用，也就是说所操作的数据类型被指定为一个参数，在用到的时候在指定具体的类型。这种参数类型可以用在类、接口和方法的创建中，分别称为泛型类、泛型接口和泛型方法。

泛型思想早在 C++语言的模板（Templates）中就开始生根发芽，在 Java 语言处于还没有出现泛型的版本时，只能通过 Object 是所有类型的父类和类型强制转换两个特点的配合来实现类型泛化。例如在哈希表的存取中，JDK 1.5 之前使用 HashMap 的 get()方法，返回值就是一个 Object 对象，由于 Java 语言里面所有的类型都继承于 java.lang.Object，那 Object 转型为任何对象成都是有可能的。但是也因为有无限的可能性，就只有程序员和运行期的虚拟机才知道这个 Object 到底是个什么类型的对象。在编译期间，编译器无法检查这个 Object 的强制转型是否成功，如果仅仅依赖程序员去保障这项操作的正确性，许多 ClassCastException 的风险就会被转嫁到程序运行期之中。

泛型技术在 C#和 Java 之中的使用方式看似相同，但实现上却有着根本性的分歧，C#里面泛型无论在程序源码中、编译后的 IL 中（Intermediate Language，中间语言，这时候泛型是一个占位符）或是运行期的 CLR 中都是切实存在的，List<int>与 List<String>就是两个不同的类型，它们在系统运行期生成，有自己的虚方法表和类型数据，这种实现称为类型膨胀，基于这种方法实现的泛型被称为真实泛型。

Java 语言中的泛型则不一样，它只在程序源码中存在，在编译后的字节码文件中，就已经被替换为原来的原始类型（Raw Type，也称为裸类型）了，并且在相应的地方插入了强制转型代码，因此对于运行期的 Java 语言来说，ArrayList<int>与 ArrayList<String>就是同一个类。所以说泛型技术实际上是 Java 语言的一颗语法糖，Java 语言中的泛型实现方法称为类型擦除，基于这种方法实现的泛型被称为伪泛型。（类型擦除在后面在学习）

使用泛型机制编写的程序代码要比那些杂乱的使用 Object 变量，然后再进行强制类型转换的代码具有更好的安全性和可读性。泛型对于集合类来说尤其有用。

泛型程序设计（Generic Programming）意味着编写的代码可以被很多不同类型的对象所重用。

实例分析：

在 JDK1.5 之前，Java 泛型程序设计是用继承来实现的。因为 Object 类是所用类的基类，所以只需要维持一个 Object 类型的引用即可。就比如 ArrayList 只维护一个 Object 引用的数组：

[java] view plain copy

```
1.  public class ArrayList//JDK1.5 之前的
```

```
{
    public Object get(int i){......}
    public void add(Object o){......}
    ......
    private Object[] elementData;
}
```

这样会有两个问题：

1、没有错误检查，可以向数组列表中添加类的对象

2、在取元素的时候，需要进行强制类型转换

这样，很容易发生错误，比如：

[java] view plain copy

```
/**jdk1.5 之前的写法，容易出问题*/
ArrayList arrayList1=new ArrayList();
arrayList1.add(1);
arrayList1.add(1L);
arrayList1.add("asa");
int i=(Integer) arrayList1.get(1);//因为不知道取出来的值的类型，类型转换的时候容易出错
```

这里的第一个元素是一个长整型，而你以为是整形，所以在强转的时候发生了错误。

所以。在 JDK1.5 之后，加入了泛型来解决类似的问题。例如在 ArrayList 中使用泛型：

[java] view plain copy

```
                /** jdk1.5 之后加入泛型*/
        ArrayList<String> arrayList2=new ArrayList<String>();  //限定数组列表中的类型
//      arrayList2.add(1); //因为限定了类型，所以不能添加整形
//      arrayList2.add(1L);//因为限定了类型，所以不能添加整长形
        arrayList2.add("asa");//只能添加字符串
        String str=arrayList2.get(0);//因为知道取出来的值的类型，所以不需要进行强制类型转换
```

还要明白的是，泛型特性是向前兼容的。尽管 JDK 5.0 的标准类库中的许多类，比如集合框架，都已经泛型化了，但是使用集合类（比如 HashMap 和 ArrayList）的现有代码可以继续不加修改地在 JDK 1.5 中工作。当然，没有利用泛型的现有代码将不会赢得泛型的类型安全的好处。

在学习泛型之前，简单介绍下泛型的一些基本术语，以 ArrayList<E>和 ArrayList<Integer>做简要介绍：

整个成为 ArrayList<E>泛型类型

ArrayList<E>中的 E 称为类型变量或者类型参数

整个 ArrayList<Integer>　称为参数化的类型

ArrayList<Integer>中的 integer 称为类型参数的实例或者实际类型参数

·ArrayList<Integer>中的<Integer>念为 typeof　Integer

ArrayList 称为原始类型

二、泛型的使用

泛型的参数类型可以用在类、接口和方法的创建中，分别称为泛型类、泛型接口和泛型方法。下面看看具体是如何定义的。

1、泛型类的定义和使用

一个泛型类（generic class）就是具有一个或多个类型变量的类。定义一个泛型类十分简单，只需要在类名后面加上<>，再在里面加上类型参数：

[java] view plain copy

```java
class Pair<T> {
    private T value;
        public Pair(T value) {
                this.value=value;
        }
        public T getValue() {
        return value;
    }
    public void setValue(T value) {
        this.value = value;
    }
}
```

现在我们就可以使用这个泛型类了：

[java] view plain copy

```java
public static void main(String[] args) throws ClassNotFoundException {
        Pair<String> pair=new Pair<String>("Hello");
        String str=pair.getValue();
        System.out.println(str);
        pair.setValue("World");
        str=pair.getValue();
        System.out.println(str);
    }
```

Pair 类引入了一个类型变量 T，用尖括号<>括起来，并放在类名的后面。泛型类可以有多个类型变量。例如，可以定义 Pair 类，其中第一个域和第二个域使用不同的类型：

public class Pair<T,U>{.....}

注意：类型变量使用大写形式，且比较短，这是很常见的。在 Java 库中，使用变量 E 表示集合的元素类型，K 和 V 分别表示关键字与值的类型。（需要时还可以用临近的字母 U 和 S）表示“任意类型”。

2、泛型接口的定义和使用

定义泛型接口和泛型类差不多，看下面简单的例子：

[java] view plain copy

```java
interface Show<T,U>{
    void show(T t,U u);
}

class ShowTest implements Show<String,Date>{
    @Override
    public void show(String str,Date date) {
        System.out.println(str);
        System.out.println(date);
    }
}
```

测试一下：

[java] view plain copy

```java
public static void main(String[] args) throws ClassNotFoundException {
        ShowTest showTest=new ShowTest();
        showTest.show("Hello",new Date());
    }
```

3、泛型方法的定义和使用

泛型类在多个方法签名间实施类型约束。在 List<V> 中，类型参数 V 出现在 get()、add()、contains() 等方法的签名中。当创建一个 Map<K, V> 类型的变量时，您就在方法之间宣称一个类型约束。您传递给 add() 的值将与 get() 返回的值的类型相同。

类似地，之所以声明泛型方法，一般是因为您想要在该方法的多个参数之间宣称一个类型约束。

举个简单的例子：

[java] view plain copy

```java
public static void main(String[] args) throws ClassNotFoundException {
        String str=get("Hello", "World");
        System.out.println(str);
    }

    public static <T, U> T get(T t, U u) {
        if (u != null)
            return t;
        else
            return null;
    }
```

三、泛型变量的类型限定

在上面，我们简单的学习了泛型类、泛型接口和泛型方法。我们都是直接使用<T>这样的形式来完成泛型类型的声明。

有的时候，类、接口或方法需要对类型变量加以约束。看下面的例子：

有这样一个简单的泛型方法：

[java] view plain copy

```java
public static <T> T get(T t1,T t2) {
        if(t1.compareTo(t2)>=0);//编译错误
        return t1;
    }
```

因为，在编译之前，也就是我们还在定义这个泛型方法的时候，我们并不知道这个泛型类型 T，到底是什么类型，所以，只能默认 T 为原始类型 Object。所以它只能调用来自于 Object 的那几个方法，而不能调用 compareTo 方法。

可我的本意就是要比较 t1 和 t2，怎么办呢？这个时候，就要使用类型限定，对类型变量 T 设置限定（bound）来做到这一点。

我们知道，所有实现 Comparable 接口的方法，都会有 compareTo 方法。所以，可以对<T>做如下限定：

[java] view plain copy

```java
public static <T extends Comparable> T get(T t1,T t2) { //添加类型限定
        if(t1.compareTo(t2)>=0);
        return t1;
    }
```

类型限定在泛型类、泛型接口和泛型方法中都可以使用，不过要注意下面几点：

1、不管该限定是类还是接口，统一都使用关键字 extends

2、可以使用&符号给出多个限定，比如

[java] view plain copy

```java
public static <T extends Comparable&Serializable> T get(T t1,T t2)
```

3、如果限定既有接口也有类，那么类必须只有一个，并且放在首位置

[java] view plain copy

```java
public static <T extends Object&Comparable&Serializable> T get(T t1,T t2)
```