泛型在日常编码过程中经常用到，常用容器List、Set、Map都是支持泛型的，具体怎么使用泛型呢，一起来看下这几个问题。

1、为什么要使用泛型

2、泛型使用过程中有哪些限制

3、为什么说java没有实现真正的泛型

让我们一起探究下泛型吧

一、泛型的定义和设计背景

泛型是JDK5以后出现的特性，即参数化类型，将具体的类型参数化，即在对象创建或者方法调用时才会明确类型。

使用泛型有什么好处呢,看下如下代码：

List list = new ArrayList();  
list.add(new Object());  
list.add(new Object());  
String str = (String) list.get(0);

List没有定义泛型类型，取出String的时候需要做强制类型转换，编译期间是没有问题的，那运行下这段代码会报这个错误



使用泛型就能解决这个问题，在编译期间就只能使用指定类型。

泛型具体有什么好处呢

1、解决类型安全问题，在编译期间就解决强制类型转换的问题

2、减少强制类型转换，提高代码效率（java中没有真的实现泛型，编译后的代码还是强制类型转换）

3、在框架和公共类设计的时候提高代码的复用性

二、日常应用

1、泛型类，类的泛型类型，需要实例化对象时指定

public class Demo2<T,M> {  
   
 private T t;  
 private M m;  
  
 public void setValue(T t,M m)  
 {  
 this.t= t;  
 this.m = m;  
 }  
   
 public T getT()  
 {  
 return t;  
 }  
  
 public static void main(String[] args) {  
 Demo2<Integer,String> demo2 = new Demo2<>();  
 demo2.setValue(1,"1");  
 Integer no =demo2.getT();  
 System.*out*.println(no);  
 }  
}

2、泛型接口，接口的泛型可以在继承的时候指定，也可以在实例化对象是指定

public interface Demo3<T> {  
 T getT();  
}

在继承时指定，如下代码

public class Demo4 implements Demo3<String>{  
 @Override  
 public String getT() {  
 return null;  
 }  
}

在实例化时指定，代码如下

public class Demo5<T> implements Demo3<T> {  
 @Override  
 public T getT() {  
 return null;  
 }  
  
 public static void main(String[] args) {  
 Demo5<String> demo5 = new Demo5<>();  
 }  
}

3、在方法中使用泛型

public class Demo6 {  
 public <T> T getT(T t)  
 {  
 return t;  
 }  
 public <M> void show(M m)  
 {  
 System.*out*.println(m);  
 }  
 public static void main(String[] args) {  
 Demo6 demo6 = new Demo6();  
 Integer no = demo6.getT(1);  
 String str = demo6.getT("mg");  
 demo6.<String>show("mg");  
 }  
}

4、静态方法中使用泛型

静态方法中也是可以使用泛型的，不过不能用类的泛型方法修饰静态方法

public class Demo7 {  
 public static <T> void getT(T t){  
 System.*out*.println(t);  
 }  
 public static void main(String[] args) {  
  
 Demo7.*getT*("1");  
 }  
}

5、泛型集合

ArrayList<String> list = new ArrayList<>();

三、通配符的使用

使用泛型过程中需要限定泛型的类型怎么处理呢，使用通配符

< ? extends E> 上界通配符，即指定的泛型类型只能是E和E的子类

< ? superE>下界通配符，即指定的泛型类型只能是E和E的父类

具体怎么使用呢，先定义三个POJO类

public class Grandpa {}

public class Father extends Grandpa {}  
public class Son extends Father {}

上界限定符

使用上界限定符实现泛型的向上转换,尝试如下代码

public class Demo8<T extends Father> {  
 public static void main(String[] args) {  
 Demo8<Son> demo8 = new Demo8<>();  
 Demo8<Father> demo81 = new Demo8<>();  
 Demo8<Grandpa> demo82 = new Demo8();  
 }  
}

其中 Son和Father可以编译通过，Grandpa会报错，Demo8能限定只能使用Father和子类做泛型类

下界限定符

使用下界限定符表示修饰的类型必须是父类,尝试如下代码

public class Demo9{  
 public void setList(List<? super Father> list){  
 }  
   
 public static void main(String[]args){  
 Demo9 demo9 = new Demo9();  
 demo9.setList(new ArrayList<Grandpa>());  
 demo9.setList(new ArrayList<Son>());  
 }  
}

添加Grandpa的列表可以编译通过，添加son的列表报错。

这样通配符就搞明白了用法了

四、泛型注意事项

1、泛型类型只能是引用类型

2、可以指定多个泛型类型

3、静态方法不能用类的泛型修饰

4、泛型创建具体类型的数组

4、泛型使用过程中，思想即不需要知道泛型类型，所以尽量避免使用反射，如果确实需要参数类型，可以通过在方法中定义Class<T>参数，在使用反射，如下代码

public <T> void setT(T t,Class<T> cl){}

**五、伪泛型之类型擦除**

Demo9去掉demo9.setList(new ArrayList<Son>());这行代码，能正常编译以后，反编译一下class文件代码如下  
**public** **class** Demo9 {  
 **public** **void** setList(List<? **super** Father> list) {}  
   
 **public** **static** **void** main(String[] args) {  
 Demo9 demo9 = **new** Demo9();  
 demo9.setList(**new** ArrayList<>());  
 }  
}

demo9.setList(**new** ArrayList<>());代码中的类型没有了，这是为什么呢。

JDK5之前是没有实现泛型，为了兼容以前的版本，在编译过程中做了类型擦除实现的是伪泛型，如果是<T>这种类型的，会处理成Object，如果使用了上界限定符<? extends T>就会转成上界T。

所以说java中泛型的使用主要是为了类型安全。