1. 为什么要使用泛型编程
2. 未使用泛型编程时

在没有泛型编程的时候，我们要实现同一个功能，但是一个参数可能有不同的类型，就只能通过重复编写类似的类来实现同样的功能。如原来的ArrayList类，只维护了一个Object引用的数组：

public class ArrayList

{

private Object[] elementData;

public Object get(int i){}

public void add(Object o){}

}

当我们想要获取ArrayList中值时，只能通过强制类型转换来获取：

ArrayList files = new ArrayList();

String filename = (String)files.get(0);

同时，当我们想要往ArrayList中添加数据时，是没有错误检查的，任何类的值都可以添加，这就会有很大的风险。这些问题都是原来未使用泛型编程时存在的问题！！！

1. 类型参数
2. 类型参数

泛型编程提供了类型参数，来指示元素的类型，这样，就可以对多种不同类型的对象重用泛型类。且在调用泛型类时，都要指定类型参数的类型，这样代码的可读性非常好，可以直接得知该类中存储的参数类型：

public class Pari<T>{}

编译器可以利用这个类型参数，在调用类中的get方法时，不需要进行强制类型转换，在调用add方法时，编译器也可以进行检查，防止你插入错误类型的对象。

1. 泛型类
2. 概述

泛型类就是有一个或者多个类型变量的类，如下所示：

public class Pair<T>{}

public class Pair<T,U>{}

类型变量在整个类的定义中用于指定方法的返回类型以及字段和局部变量的类型，如：

private T first;

1. 实例
2. 泛型类 Pair

定义一个Pair泛型类，使用一个类型参数T：

public class Pair<T> {  
 private T first;  
 private T second;  
 public Pair(){  
 first = null;  
 second = null;  
 }  
 public Pair(T first, T second){  
 this.first = first;  
 this.second = second;  
 }  
 public T getFirst(){  
 return first;  
 }  
 public T getSecond(){  
 return second;  
 }  
  
 public void setFirst(T newValue) {  
 this.first = newValue;  
 }  
  
 public void setSecond(T newValue) {  
 this.second = newValue;  
 }  
}

1. 测试类PairTest

测试泛型类Pair的使用情况：

public class PairTest1 {  
 public static void main(String[] args) {  
 String[] words = {"Mary", "had", "a", "little", "lamb"};  
 Pair<String> minmax = ArrayArg.*minmax*(words);  
 System.*out*.println("min = " + minmax.getFirst());  
 System.*out*.println("max = " + minmax.getSecond());  
 }  
}  
  
class ArrayArg{  
 public static Pair<String> minmax(String[] a){  
 if(a == null || a.length == 0){  
 return null;  
 }  
 String min = a[0];  
 String max = a[0];  
 for(int i = 1; i < a.length; ++i){  
 if(min.compareTo(a[i]) > 0){  
 min = a[i];  
 }  
 if(max.compareTo(a[i]) < 0){  
 max = a[i];  
 }  
 }  
 return new Pair<>(min, max);  
 }  
}

1. 泛型方法
2. 概述

泛型方法就是将普通方法的参数替换为类型参数，类型变量放在修饰符的后面，返回类型的前面，如下所示：

class ArrayArg{

public static <T> T getMiddle(T… a){

return a{a.length/2};

}

}

泛型类可以定义在普通类中，也可以定义在泛型类中。

1. 调用泛型方法

在调用泛型方法时，可以写出类型参数，包围在尖括号中，如下所示：

String middle = ArrayAlg.<String>getMiddle(“John”, “Q”, “Public”);

但是，通常我们都会省略类型参数，因为编译器有足够的信息可以推断出类型参数，直接将参数的类型与类型参数进行匹配即可。

但是，当传入的参数是不同的类型时，编译器就要首先对参数类型进行自动装箱，然后寻找这些类的共同超类型。如下所示：

double middle = ArrayAlg.getMiddle(3.14, 1729, 0);

编译器会将参数自动装箱为一个Double，二个Integer对象，然后寻找这些类的共同超类型，Number和Comparable接口，可以看到，超类型是不同的，因此补救措施就是将参数都写为double值。

1. 类型参数的限定
2. 概述

类型参数为什么需要被限定呢？

比如我们在PariTest类中实现的minMax类中对类型参数T使用了compareTo方法，我们需要限定每个类型参数T都能够使用compareTo方法，因此需要限定T实现了Comparable接口，需要使用如下的方式实现限定：

public static <T extends Comparable> T min(T[] a){}

为什么使用extends关键字呢？

因为类型参数T和限定类型可能是类也可能是接口，限定主要表达的意思是T限定为限定类型的一个子类型，因此使用extends关键字更接近子类型的概念。

当有多个限定时使用 & 分割， 多个类型参数T使用逗号分隔。注意，最多只有一个限定是类，因为一个类最多只能继承一个类，该限定类必定是限定列表的第一个限定：

public static <T extends Comparable & Serializable, U extends Comparable>

1. 泛型编程与虚拟机
2. 类型擦除

虚拟机没有泛型类型对象，所有的对象都是属于普通类的。无论何时定义一个泛型类型，都会自动提供一个相应的原始类型，该类型会擦除掉类型参数。对于有限定的类型参数，为第一个限定类型，对于没有限定的为Object。

如，Pair<T>泛型类中所有的T擦除为Object，如下所示：

public class Pair{

private Object first;

private Object second;

…

}

有限定的类型参数擦除为限定类型：

public class Inteval<T extends Comparable & Serializable> implements Serializable{

private T lower;

}

擦除为：

public class Inteval implements Serializable{

private Comparable lower;

}

1. 转换泛型表达式

在调用一个泛型方法时，如果该泛型方法的返回类型被擦除了，那么编译器会插入强制类型转换。如下所示：

Pair<Employee> buddies = …;

Employee buddy = buddies.getFirst();

getFirst方法中的返回类型被擦除为Object了，编译器会首先调用getFirst方法，然后将返回的Object类型强制转换为Employee类型。

1. 转换泛型方法
2. 擦除类型参数的问题

泛型方法同样需要擦除类型参数，但是泛型方法的类型参数被擦除后，就会出现多态冲突的问题，如何保持多态的正常运行呢？

如下所示，DateInteval类继承了之前设置的泛型类Pair，且擦除了类型参数:

class DateInteval extends Pair{

public void setSecond(Localdate second){}

}

但是这个类是继承自Pair类的，因此还有一个从Pair类中继承的setSecond方法：

public void setSecond(Object second){}

现在，DateInteval类中有两个setSecond方法，照常来说，我们想要使用的当然是DateInteval中定义的setSecond方法，而不是继承来的setSecond方法，但是类型参数的擦除与多态在此时发生了冲突，它会调用继承来的setSecond方法，偏离了我们的初衷。

1. 桥方法

为了解决这个问题，编译器在DateInteval类中生成了一个桥方法，使用的是从Pair类中继承的方法签名，但是在函数体中代用了当前DateInteval类的setSecond方法，这样就能够间接地调用本类中的setSecond方法了：  
public void setSecond(Object second) {

setSecond((LocalDate) second);

}

为什么会首先调用从Pair类继承的setSecond方法呢？

查看如下调用方式：

var interval = new DateInteval();

Pair<LocalDate> pair = interval;

pair.setSecond(aDate);

最后一句的调用，是在pair引用的对象上调用的，这个对象是DateInteval类型，因而会调用DateInteval.setSecond(Object)方法， 这个方法时合成的巧方法，它会调用DateInterval.setSecond(LocalDate)，这样就达到了我们的目的。

1. 总结Java泛型的转换
2. 虚拟机中没有泛型，只有普通的类和方法。
3. 所有的类型参数都会替换为它们的限定类型。
4. 类会合成桥方法来保持多态。
5. 为保持类型安全性，必要时会插入强制类型转换。