Java中的泛型

1. **泛型**：是JDK1.5 推出来的**安全机制，**是给**编译器**使用的技术，**用于编译时期**，保证了类型的安全。泛型在**集合框架中**使用最为广泛。
2. **Java的每次升级**都是为了三个目的：简化书写、提高效率、提高安全性。泛型的提出是为了提高安全性，基本数据类型的自动装箱和拆箱是为了简化书写。
3. 泛型的擦除和补偿（了解）：
4. 泛型的擦除：泛型只用于编译时，供编译器判断类型正确与否，而到了**程序运行的时候**，会将泛型擦除掉，编译时生成的**class文件**中是不带泛型的，这称为**泛型的擦除**。擦除的目的是：为了兼容旧版本的类加载器，否则**类加载器**还需要进行升级。
5. 泛型的补偿：

如果采用了泛型擦除后，对象仍然被提升为Object类型，还需要进行强制类型转换，为了省略掉此转换，引入了泛型的补偿，即在运行时，通过获取**元素的类型**完成类型的转换工作，不需要再进行强制类型转换了。 （**利用Object的getClass()方法**）。

1. **引入泛型的好处**：
2. 将以前运行时可能出现的类转换异常（**ClassCastException）**转移到了**编译时期**，提高了安全性；
3. 避免了**类型的强制转换**的麻烦。
4. 什么时候使用**泛型**呢？

答：当操作的**引用数据类型**不确定的时候，就使用泛型指定要操作的数据类型；将要操作的引用数据类型名传入即可。其实泛型就是一个用于接收**具体的引用数据类型**的参数范围。

1. 在程序中，只要用到了带有<>的类或者接口，就要明确传入的**具体引用数据类型**，包括**类型、接口名、数组名**（数组名也是引用类型），不能是基本数据类型，泛型中只能传入**引用数据类型**。可以传入**int[] 、String[]**等。
2. **括号符号问题**：a) **小括号()：用于指示方法；b)中括号[]：用于指示数组；**

**c) 大括号{}：用于指示范围；d) 尖括号<>：用于指示泛型**。

1. **E ： element；K ： key；V ： value；T ： type。**
2. **自定义泛型类**：

public class FanXing<T> {

private T type;

void setObject(T type) {

this.type = type;}

T getObject() {

return this.type;

} }

FanXing<Student> fan = new FanXing<Student>();

fan.setObject(new Student());**//只能传入Student对象**

Student s = fan.getObject();

1. 将**泛型**定义在方法上：**泛型放在修饰符后面，返回类型前面。**

**public<W> void show(W w) {**

**System.out.println(w.toString()); }**

**调用： fan.show(new Worker());//就可以打印Worker对象的字符串**

**如果是静态方法，不能使用类定义上的泛型，因为静态方法不依赖于对象，此时只能使用方法自己定义的泛型：**

**public class FanXing<T> {**

**private T type;**

**void setObject(T type) {**

**this.type = type;}**

**T getObject() {**

**return this.type;}**

**public<W> void show(W w) {**

**System.out.println(w.toString()); }**

**public static void staticMethod(T tt){**

**//出现错误，静态方法，不能使用类定义的泛型**

**System.out.println(tt.toString()); }**

**public static <TT> void staticMethod(TT tt){**

**System.out.println(tt.toString()); } }**

1. 泛型接口：

**interface Inter<T> {**

**public void show(T t) ; }**

**（1） 泛型接口的第一种实现方式：实现类仍然采用泛型，在泛型接口上指定实现类采用的泛型，在生成对象时确定具体操作类型。**

class interC1<W> implements Inter<W>{

public void show(W t) {

System.*out*.println(t.toString());

} }

**创建对象： interC1<String> i1 = new interC1<String>();**

（2）**泛型接口的第二种实现方式：实现类不采用泛型，在泛型接口上指定具体类型。  
class interC2 implements Inter<String>{**

**public void show(String t) {**

**System.out.println(t.toString());**

**} }**

**创建对象：interC2 i1 = new interC2();**

1. **泛型的通配符：？，表示任意类型**，用于接收**任意的泛型**。

**private static void printCollection(List<?> list) {**

**//？ 可以接收任何的泛型，相当于Object**

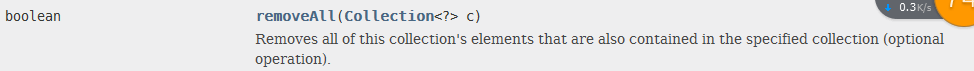
**for(Iterator<?> it = list.iterator();it.hasNext();) {**

**}}**

**使用通配符和自定义泛型T的区别：**

**通配符和自定义泛型都可以接收任何类型，如同Object，但是自定义泛型T可以使某个类或某个方法都统一使用T类型，而且可以为T类型指定具体的某个类，但是比较麻烦；通配符？不可以指定具体某个类。**

1. **通配符的体现：**





由于**containsAll和removeAll**在使用时，需要利用equals方法找出此集合，而**equals方法**是在Object中定义的，且默认接收对象都是**Object的**，所以对**任意类型对象**都可以接收，故用**？**表示**可以接收任意类型的对象**。

1. **泛型的限定**：
2. **泛型的上限**：**使用较多。**

<E>: **只可以接收E类型对象；**

**<? extends E> :　可以接收E类型及其子类对象。上限**

**一般在集合存储元素的时候，都使用上限，因为这样就可以用上限类型E进行接收，不会出现类型安全隐患。**

**其实： 通配符？ 就相当于 ？ extends Object**

1. **泛型的下限**：

**<? super E>：可以接收E类型及其父类型对象。下限。**

通常是在对集合中元素进行**取出操作**时，会使用到**泛型的下限**。

1. 泛型上限和下限的体现：
2. **上限的体现**：通常**存储元素时利用上限**。这样的话，取出元素的时候都是利用上限类型E取出了，不会出现类型安全隐患。

如：接口Collection<E>中的addAll方法：



一个存放E类型对象的集合可以添加另外一个存放**E类型或者其子类型的对象**的集合。如：

**ArrayList<Person> al1 = new ArrayList<Person>();**

**ArrayList<Student> al2 = new ArrayList<Student>();**

**al1.addAll(al2);//Student是Person的子类**

**此外，Map接口中的putAll方法：**



**TreeSet的构造方法：**





1. **下限的体现**：通常**取出元素时**利用**泛型的下限**。

ArrayList、TreeSet的构造方法：



由于**比较器**需要从**TreeSet集合**中取出**对象元素**进行比较，对象元素可以利用**本对象类型引用或者父类型引用**进行接收，所以可以利用指定**本对象的类型的比较器或者指定父类型的比较器**进行排序。例如： TreeSet<Student> 集合可以利用Comparator<Student>比较器也可以利用Comparator<Person>比较器。

即TreeSet<Student>（new ComparatorByName<Student>）合法，TreeSet<Student>（new ComparatorByName<Person>）也合法。

如

TreeSet<Student> ts3 = new TreeSet<Student>(new Comparator<Student>() {

public int compare(Student o1, Student o2) {

return o1.getName().compareTo(o2.getName());}});//是合理的

**TreeSet<Student> ts = new TreeSet<Student>(new Comparator<Person>() {**

**public int compare(Person o1, Person o2) {**

**return o1.getName().compareTo(o2.getName());}});//是合理的**

**TreeSet<Person> ts2 = new TreeSet<Person>(new Comparator<Student>() {**

**public int compare(Student o1, Student o2) {**

**return o1.getName().compareTo(o2.getName());}});//是错误的**

1. **泛型限定的示例**：
2. **泛型的上限**：

private static void printCollection(List<? extends Animal> list) {

//这样之后，只有是Animal类或者其子类的对象才可以作为泛型

for(Iterator<? extends Animal> it = list.iterator();it.hasNext();) {

//进来的都是Animal类或者子类，则即可以利用Animal引用 进行接收

Animal anim = it.next();

System.out.println(anim.getName()+" " + anim.getAge()); }}

1. **泛型的下限**：

private static void printCollection(List<? super Cat> list) {

//这样之后，只有Cat类及其父类可以作为泛型传入，如Dog就不能传进来

for(Iterator<? super Cat> it = list.iterator();it.hasNext();) {

System.out.println(it.next()); }}