**Java泛型－类型擦除**

**一、概述**

[**Java**](http://lib.csdn.net/base/java)泛型在使用过程有诸多的问题，如不存在List<String>.class, List<Integer>不能赋值给List<Number>（不可协变），奇怪的ClassCastException等。 正确的使用Java泛型需要深入的了解Java的一些概念，如协变，桥接方法，以及这篇笔记记录的类型擦除。Java泛型的处理几乎都在编译器中进行，编译器生成的bytecode是不包涵泛型信息的，泛型类型信息将在编译处理是被擦除，这个过程即类型擦除。

**二、编译器如何处理泛型？**

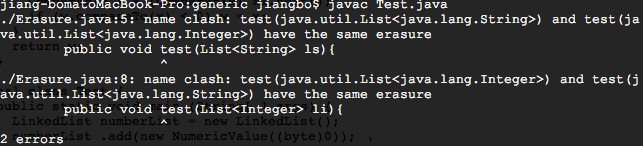
     通常情况下，一个编译器处理泛型有两种方式：  
     1.Code specialization。在实例化一个泛型类或泛型方法时都产生一份新的目标代码（字节码or二进制代码）。例如，针对一个泛型list，可能需要 针对string，integer，float产生三份目标代码。  
     2.Code sharing。对每个泛型类只生成唯一的一份目标代码；该泛型类的所有实例都映射到这份目标代码上，在需要的时候执行类型检查和类型转换。  
     C++中的模板（template）是典型的Code specialization实现。C++编译器会为每一个泛型类实例生成一份执行代码。执行代码中integer list和string list是两种不同的类型。这样会导致代码膨胀（code bloat），不过有经验的C＋＋程序员可以有技巧的避免代码膨胀。  
     Code specialization另外一个弊端是在引用类型系统中，浪费空间，因为引用类型集合中元素本质上都是一个指针。没必要为每个类型都产生一份执行代码。而这也是Java编译器中采用Code sharing方式处理泛型的主要原因。  
     Java编译器通过Code sharing方式为每个泛型类型创建唯一的字节码表示，并且将该泛型类型的实例都映射到这个唯一的字节码表示上。将多种泛型类形实例映射到唯一的字节码表示是通过类型擦除（type erasue）实现的。

**三、 什么是类型擦除？**

     类型擦除指的是通过类型参数合并，将泛型类型实例关联到同一份字节码上。编译器只为泛型类型生成一份字节码，并将其实例关联到这份字节码上。类型擦除的关键在于从泛型类型中清除类型参数的相关信息，并且再必要的时候添加类型检查和类型转换的方法。  
     类型擦除可以简单的理解为将泛型java代码转换为普通java代码，只不过编译器更直接点，将泛型java代码直接转换成普通java字节码。  
     类型擦除的主要过程如下：  
     1.将所有的泛型参数用其最左边界（最顶级的父类型）类型替换。  
     2.移除所有的类型参数。  
     如  
interface Comparable <A> {   
  public int compareTo( A that);   
}   
final class NumericValue implements Comparable <NumericValue> {   
  priva te byte value;    
  public  NumericValue (byte value) { this.value = value; }    
  public  byte getValue() { return value; }    
  public  int compareTo( NumericValue t hat) { return this.value - that.value; }   
}   
－－－－－－－－－－－－－－－－－  
class Collections {    
  public static <A extends Comparable<A>>A max(Collection <A> xs) {   
    Iterator <A> xi = xs.iterator();   
    A w = xi.next();   
    while (xi.hasNext()) {   
      A x = xi.next();   
      if (w.compareTo(x) < 0) w = x;   
    }   
    return w;   
  }   
}   
final class Test {   
  public static void main (String[ ] args) {   
    LinkedList <NumericValue> numberList = new LinkedList <NumericValue> ();   
    numberList .add(new NumericValue((byte)0));    
    numberList .add(new NumericValue((byte)1));    
    NumericValue y = Collections.max( numberList );    
  }   
}  
经过类型擦除后的类型为  
 interface Comparable {   
  public int compareTo( Object that);   
}   
final class NumericValue implements Comparable {   
  priva te byte value;    
  public  NumericValue (byte value) { this.value = value; }    
  public  byte getValue() { return value; }    
  public  int compareTo( NumericValue t hat)   { return this.value - that.value; }   
  public  int compareTo(Object that) { return this.compareTo((NumericValue)that);  }   
}   
－－－－－－－－－－－－－  
class Collections {    
  public static Comparable max(Collection xs) {   
    Iterator xi = xs.iterator();   
    Comparable w = (Comparable) xi.next();   
    while (xi.hasNext()) {   
      Comparable x = (Comparable) xi.next();   
      if (w.compareTo(x) < 0) w = x;   
    }   
    return w;   
  }   
}   
final class Test {   
  public static void main (String[ ] args) {   
    LinkedList numberList = new LinkedList();   
    numberList .add(new NumericValue((byte)0));  ，  
    numberList .add(new NumericValue((byte)1));    
    NumericValue y = (NumericValue) Collections.max( numberList );    
  }   
}  
第一个泛型类Comparable <A>擦除后 A被替换为最左边界Object。Comparable<NumericValue>的类型参数NumericValue被擦除掉，但是这直 接导致NumericValue没有实现接口Comparable的compareTo(Object that)方法，于是编译器充当好人，添加了一个桥接方法。  
第二个示例中限定了类型参数的边界<A extends Comparable<A>>A，A必须为Comparable<A>的子类，按照类型擦除的过程，先讲所有的类型参数 ti换为最左边界Comparable<A>，然后去掉参数类型A，得到最终的擦除后结果。

**四、类型擦除带来的问题**

     正是由于类型擦除的隐蔽存在，直接导致了众多的泛型灵异问题。  
**Q1.用同一泛型类的实例区分方法签名？——NO！**  
    import java.util.\*;  
  
    public class Erasure{  
  
            public void test(List<String> ls){  
                System.out.println("Sting");  
            }  
            public void test(List<Integer> li){  
                System.out.println("Integer");  
            }  
    }  
编译该类，

  
  
参数类型明明不一样啊，一个List<String>，一个是List<Integer>，但是，偷偷的说，type erasure之后，它就都是List了⋯⋯  
**Q2. 同时catch同一个泛型异常类的多个实例？——NO！**  
同理，如果定义了一个泛型一场类GenericException<T>，千万别同时catch GenericException<Integer> 和GenericException<String>，因为他们是一样一样滴⋯⋯  
**Q3.泛型类的静态变量是共享的？——Yes！**  
猜猜这段代码的输出是什么？  
import java.util.\*;  
  
public class StaticTest{  
    public static void main(String[] args){  
        GT<Integer> gti = new GT<Integer>();  
        gti.var=1;  
        GT<String> gts = new GT<String>();  
        gts.var=2;  
        System.out.println(gti.var);  
    }  
}  
class GT<T>{  
    public static int var=0;  
    public void nothing(T x){}  
}  
答案是——2！由于经过类型擦除，所有的泛型类实例都关联到同一份字节码上，泛型类的所有静态变量是共享的。

**五、Just remember**

1.虚拟机中没有泛型，只有普通类和普通方法  
2.所有泛型类的类型参数在编译时都会被擦除  
3.创建泛型对象时请指明类型，让编译器尽早的做参数检查（Effective Java，第23条：请不要在新代码中使用原生态类型）  
4.不要忽略编译器的警告信息，那意味着潜在的ClassCastException等着你。