泛型：是一种把类型明确的工作推迟到创建对象或者调用方法的时候才去明确的特殊的类型。参数化类型，把类型当作参数一样的传递。

格式：

<数据类型>

此处的数据类型只能是引用类型。

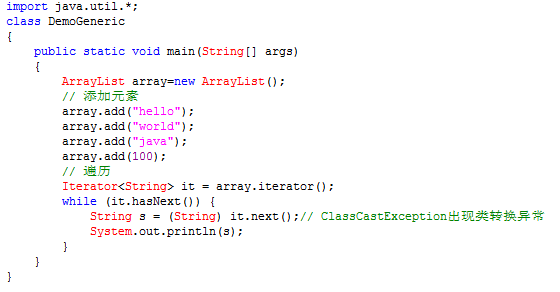
好处：

A:把运行时期的问题提前到了编译期间

B:避免了强制类型转换

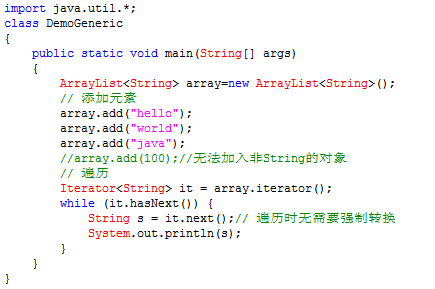
C:增加的数据的安全性，提高了性能

分析：在没有使用泛型前代码如下：



我们按照正常的写法来写这个程序， 结果确出错了。为什么呢?

因为我们开始存储的时候，存储了String和Integer两种类型的数据。而在遍历的时候，我们把它们都当作String类型处理的，做了转换，所以就报错了。但是呢，它在编译期间却没有告诉我们。所以，设计的不好，jdk1.5以后提供了泛型，使用泛型后的代码：



回想一下，我们的数组

String[] strArray = new String[3];

strArray[0] = "hello";

strArray[1] = "world";

strArray[2] = 10;

集合也模仿着数组的这种做法，在创建对象的时候明确元素的数据类型。这样就不会在有问题了。而这种技术被称为：泛型。

早期的时候，我们使用Object来代表任意的类型。向上转型是没有任何问题的，但是在向下转型的时候其实隐含了类型转换的问题。也就是说这样的程序其实并不是安全的。所以Java在JDK5后引入了泛型，提高程序的安全性

早期无泛型定义类代码如下：



定义泛型类，但不指定泛型类中的参数类型与Object的效果等同，代码如下：



定义泛型类，并且指定泛型类中的参数类型，这个时候如果传入的数据不是指定类型那么编译期间就过通不过，代码如下：

泛型方法：



泛型通配？

示例一：



示例二：



泛型上限： 表示参数化的类型可能是所指定类型，或者是其子类

示例一：



示例二

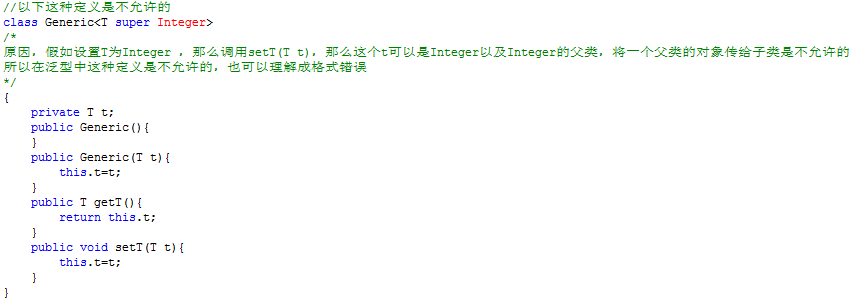


泛型下限：使用的泛型只能是本类及其父类类型上应用的时候，就必须使用泛型的下限。

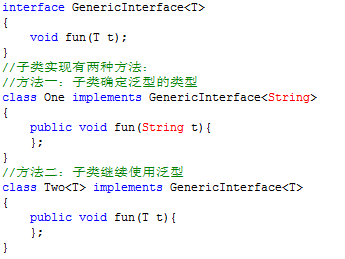
示例一：



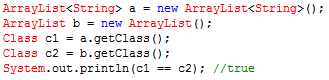
示例二：



泛型接口



泛型只在编译阶段有效



上面程序的输出结果为true。所有反射的操作都是在运行时的，既然为true，就证明了编译之后，程序会采取去泛型化的措施，也就是说Java中的泛型，只在编译阶段有效。在编译过程中，正确检验泛型结果后，会将泛型的相关信息擦出，并且在对象进入和离开方法的边界处添加类型检查和类型转换的方法。也就是说，成功编译过后的class文件中是不包含任何泛型信息的。泛型信息不会进入到运行时阶段。

