泛型的概念：
泛型是一种未知的数据类型，当我们不知道使用什么数据类型的时候，可以使用泛型
泛型也可以用来看成一个变量。接收数据类型。

示例代码：

```
class ArrayList<E>{
    public boolean add(E e){ }

    public E get(int index){ }
    ....
}
```

受限泛型： 泛型的上限：

- **格式**： 类型名称 <? extends 类 > 对象名称
  举例：`Collection<? extends Number>`
- **意义**： 只能接收该类型及其子类

泛型的下限：

- **格式**： 类型名称 <? super 类 > 对象名称
- 举例：`Collection<? super Number>`
- **意义**： 只能接收该类型及其父类型
-

```
public class GenericsDemo {
    public static void main(String[] args) {
        Collection<Integer> list1 = new ArrayList<Integer>();
        Collection<Number> list2 = new ArrayList<Number>();
        Collection<String> list3 = new ArrayList<String>();
        Collection<Object> list4 = new ArrayList<Object>();
        getElement1(list1);
        getElement1(list2);
     // getElement1(list3);//报错
     // getElement1(list4);//报错

     // getElement2(list1);//报错
        getElement2(list2);
     // getElement2(list3);//报错
        getElement2(list4);


    }

    //泛型的上限，此时的泛型？必须是Number类型或者Number类型的子类
    public static void getElement1(Collection<? extends Number> coll){}

    //泛型的下限，此时的泛型？必须是Number类型或者Number类型的父类
    public static void getElement2(Collection<? super Number> coll){}
```

```
}
```

使用泛型的好处：
将运行时期的classCastException,转移到了编译时期，变成编译失败，避免了类型强转的麻烦

泛型擦除

如果没有接触过类型擦除，大家会有疑问,类型擦除到底是什么？其实平常开发中经常会用到泛型，非常常见的一个案例就是sql返回是list，但是返回类型是实体类的时候就会用到泛型，如果在这里你不想用泛型也是可以的，只是你在遍历list的时候强转一下类型就行，但是用泛型会更简单一点，不需要你强制转换。

泛型是Java1.5版本才引进的概念，在这之前是没有泛型的概念的，但显然，泛型代码能够很好地和之前版本的代码很好地兼容。这是因为，泛型信息只存在于代码编译阶段，在进入JVM之前，与泛型相关的信息会被擦除掉，专业术语叫做类型擦除。

通俗的讲，泛型类和普通类在java虚拟机内是没有什么特别的地方。

文章开始先给大家奉上经典的测试题：

```
List<String> list1 = new ArrayList<String>();
List<Integer> list2 = new ArrayList<Integer>();
System.out.println(list1.getClass());//class java.util.ArrayList
System.out.println(list2.getClass());//class java.util.ArrayList
System.out.println(list1.getClass()==list2.getClass());//true
```

从上面输出结果就可看出来list1和list2是一样的,为了再确定一番，我反编译之后，代码如下：

```
List list1 = new ArrayList();//①
List list2 = new ArrayList();//②
System.out.println(list1.getClass());
System.out.println(list2.getClass());
System.out.println(list1.getClass() == list2.getClass());
```

①和②竟然一模一样，这里泛型信息被擦除了，可能有同学会问，那么类型String和Integer怎么办？

答案是泛型转义。

```
public class Erasure<T> {
    T object;

    public Erasure(T object) {
        this.object = object;
    }
}
```

Erasure是一个泛型类，我们查看它在运行时的状态信息可以通过反射。

```
Erasure<String> erasure = new Erasure<String>("hh");
Class clazz = erasure.getClass();
System.out.println("Erasure class: "+clazz.getName());
```

运行结果:

```
Erasure class: com.hh.erasure.Erasure
```

Class的类型是Erasure并不是Erasure<T>这种形式，那我们再看看泛型类中的T的类型在jvm中是什么具体类型。

```
Field[] fs = clazz.getDeclaredFields();
for (Field f : fs) {
    String name = f.getName();
    Class<?> type = f.getType();
    System.out.println("Field name: "+name);
    System.out.println("Field type: "+type);
}
```

运行结果：

```
Field name: object
Field type: class java.lang.Object
```

那我们可不可以说，泛型类被类型擦除后，相应的类型就被替换成Object类型呢？

这种说法，不完全正确。

这与泛型上限和泛型下限有关。。

如果Erasure类变一下。

```
public class Erasure<T extends String> {
    T object;

    public Erasure(T object) {
        this.object = object;
    }
}
```

运行结果则变成：

```
Field name: object
Field type: class java.lang.String
```

由此可见，在泛型类被类型擦除的时候，之前泛型类中的类型参数部分如果没有指定上限，如<T>则会被转译成普通的Object类型，如果指定了上限如<T extends String>则类型会被转译成类型上限。

add()方法对应的Method的签名应该是Object.class.

```
public class Erasure<T> {
    T object;

    public Erasure(T object) {
        this.object = object;
    }

    public void add(T object){

    }

    public static void main(String[] args) throws NoSuchMethodException {
        Erasure<String> erasure = new Erasure<String>("hh");
        Class clazz = erasure.getClass();
        Method[] methods =  clazz.getDeclaredMethods();
        int length = methods.length;
        System.out.println(length);
        for (Method method : methods) {
            System.out.println("method: "+method.toString());
        }
        Method add = clazz.getDeclaredMethod("add", Object.class);
        System.out.println(add);
    }
}
```

运行结果：

```
2
method: public static void com.hh.erasure.Erasure.main(java.lang.String[]) throws
java.lang.NoSuchMethodException
method: public void com.hh.erasure.Erasure.add(java.lang.Object)
public void com.hh.erasure.Erasure.add(java.lang.Object)
```

类型擦除带来的局限性

```
List<String> list = new ArrayList<String>();
list.add("hh");
// list.add(11);//不允许添加非String类型的值
```

上面必须添加同一类型的值，如果不是则报错，但是也可以通过反射来实现添加不同类型的值到list里。

```
  public static void main(String[] args) throws NoSuchMethodException,
InvocationTargetException, IllegalAccessException {
        List<String> list = new ArrayList<String>();
        list.add("hh");
        Method m = list.getClass().getDeclaredMethod("add", Object.class);
        m.invoke(list,11);
        m.invoke(list,"club");
        for (Object s : list) {
            System.out.println(s);
        }
    }
```

运行结果：

```
hh
11
club
```

可以看出，利用类型擦除的原理，用反射的手段就绕过了正常开发中编译不允许的操作限制。

总结

我们可以看到，泛型其实并没有什么神奇的地方，泛型代码能做的非泛型代码也能做。

而类型擦除，是泛型能够与之前的 java 版本代码兼容共存的原因。

可量也正因为类型擦除导致了一些隐患与局限。

但，我还是要建议大家使用泛型，如官方文档所说的，如果可以使用泛型的地方，尽量使用泛型。

毕竟它抽离了数据类型与代码逻辑，本意是提高程序代码的简洁性和可读性，并提供可能的编译时类型转换安全检测功能。

类型擦除不是泛型的全部，但是它却能很好地检测我们对于泛型这个概念的理解程度。