# 四、泛型

Java5开始提供的新特性，表示不确定的类型。

注意：泛型是提供个javac编译器使用的，它用于限定集合的输入类型，让编译器在

源代码级别上，挡住向集合中插入的非法数据。但是编译器编译完带有泛型的java程序后，

生成的class文件中将不在带有泛型信息，以此使程序运行的效率不受到影响，这个过程称为

“擦除”。

## 4.1集合泛型

List<String> list = new ArrayList<String>();

------------------------------

public class Demo1 {

public static void main(String[] args) {

List list = new ArrayList();

list.add("元芳你怎么看");

list.add(123);

for (Object obj : list) {

String str = (String)obj;

System.out.println(str);

}

}

}

----------------------

运行以上程序，出现类型转换异常，添加泛型。

考察一下：

List list1 = new ArrayList();//正确的

List<String> list2 = new ArrayList<String>();//正确的

List<String> list3 = new ArrayList();//?

List list4 = new ArrayList<String>();//?

List<Object> list5 = new ArrayList<String>();//?

List<String> list6 = new ArrayList<Object>();//错误的？

--------------------------------

public class Demo2 {

public static List method\_v3(){

return null;

}

public static void method\_v4(List list){

//。。。。

}

public static void main(String[] args) {

List list1 = new ArrayList();//正确的

List<String> list2 = new ArrayList<String>();//正确的

//以上两种写法没有问题。

List<String> list3 = new ArrayList();//正确的

List<String> list3x = method\_v3();//兼容以前低版本的代码

List list4 = new ArrayList<String>();//正确的

method\_v4(new ArrayList<String>());//兼容以前低版本的代码

List<Object> list5 = new ArrayList<String>();//错误的

List<String> list6 = new ArrayList<Object>();//错误的

}

}

------------------------------

总结：可以两边都没有；也可以一边有，一边没有；

可以两边都有，如果两边都有的话，两边必须一样。

# 2.2自定义泛型

## 2.2.1!!!定义在方法的泛型

--------------------------

class Person{}

class Dog{}

public class God {

public Person kill(Person p){

System.out.println("上帝弄死了"+p);

return p;

}

public Dog kill(Dog d){

System.out.println("上帝弄死了"+d);

return d;

}

public static void main(String[] args) {

God god = new God();

Person p = god.kill(new Person());

Dog dog = god.kill(new Dog());

}

}

-------------------------------

上帝可以“带走”各种类的对象，我们不可能为每一个类型编写一个方法，所以需要使用定义在方法上

的泛型。

定义在方法上的泛型的作用范围是方法内部。

泛型需要先定义，再使用，方法上的泛型需要定义在返回值类型的前面，通常使用一个英文的大写字母

(sun推荐使用T),也可以使用其他的任意字符，但是不要使用java中的关键字和常用类（String）。

public <T> void test(T t){}

-----------------

public <T> T kill(T t){

System.out.println("上帝弄死了"+t);

return t;

}

public <T> T save(T t){

System.out.println("上帝救活了"+t);

return t;

}

public static void main(String[] args) {

God god = new God();

Person p = god.kill(new Person());

Dog dog = god.save(new Dog());

}

--------------------

两个方法上的T不是同一个T。

如果想让两个T表示同一个，需要在类上定义泛型。

## 2.2.2！！！定义在类上的泛型

1.定义在类型的泛型的作用范围是当前类的内部

2.泛型需要先定义在使用，类上的泛型需要定义在类名的后面。通常使用大写的英文字符。

3、类型泛型需要在创建对象的时候，强烈建议指定泛型的具体类型。

4、在类上定义的泛型不能再该类内部的静态方法上使用，如果想在静态方法上使用泛型，需要

先自定义泛型，然后在使用。

public static <T> T sleep(T t){

return t;

}

注意：这个方法上T和God<T>不是同一个T。

5、如果在创建对象时，没有明确执行泛型的具体类型，则默认为泛型的“上边界"。

God pgod2 = new God();

6、在泛型中可以同时使用多个泛型。

Map<K,V>

public V put(K key,V value)

## 2.3！！泛型的通配符

public void print(Collection<String> c){...}

List<Object> list5 = new ArrayList<String>();//错误的

因为泛型没有继承的概念，所有当需要用一个泛型引用引用不同的泛型实现时，泛型中

可以写他们的共同父类是不行的，这时该怎么办？

引入了一个新的概念，叫做泛型的通配符“?”,

注意泛型的通配符只能用在引用的声明处，不能用在实例化中。

public void print(Collection<?> c){...}

发音：Collection of unknown

Collection<?> c:表示一种不确定的类型，因此在方法体重不能调用类型相关的方法.

泛型的边界：

如果没有指定泛型，默认可以接受的任意类型，有时希望进一步限制，此时可以使用方向的边界：

## 1、！！限定泛型的上边界：

extends--用来指定泛型的上边界，用在通配符和泛型定义中，指定具体的泛型实现必须是指定的

类或其子类。

Vector<? extends Number> vts = new Vector<Integer>();//正确 Integer extends Number

Vector<? extends Number> vts1 = new Vector<Number>();//正确

Vector<? extends Number> vts2 = new Vector<String>();//错误的

使用场景见day14/cn.tedu.Demo3

# 2、！限定下边界：

super:用来指定泛型的下边界，使用在泛型通配符中，指定具体的额泛型实现必须是指定类或其

超类。