# 泛型标记注意项

1. 标记不能为基本类型,只能为对象的包装类型
2. 如果没有指定标记,则为Object类型 ,A<String> a = new A<String>(),不指定类型：A a = new A();
3. A<String> p = new Point<>(); JDK1.7后能省略
4. 方法不能根据泛型标记重载

public static void fun(Message<String> msg)

public static void fun(Message<Integer> msg)

# 通配符? ?extends类 ?super类

?:表示不可修改,只能取出

?extends 类:设置泛型上限,可以在声明和方法参数上使用

?extends Number:表示可以设置Number或者是Number的子类

?super 类:设置泛型下限,方法参数上使用

?super String:表示只能设置String或者是它的父类

|  |
| --- |
| public class test{  public static void main(String args[]){  Message<Integer> m = new Message<Integer>();  Message<String> m1 = new Message<String>();  m.setMsg(100);  m1.setMsg("hello,world");  fun(m);  fun(m1);  }  public static void fun(Message msg){//此时没有指定标记类型,则标记  //类型为Object  msg.setMsg("hahah"); //这里有问题  System.out.println(msg.getMsg());  }  }  class Message<T>{  private T msg;  public T getMsg() {  return msg;  }  public void setMsg(T msg) {  this.msg = msg;  }} |

**为了防止在fun函数里修改对象,则使用通配符**

|  |
| --- |
| public static void fun(Message<?> msg){  //不知道是什么类型对属性不能设置,但是能够取出  //msg.setMsg("hahah");  System.out.println(msg.getMsg());  } |

范例:设置泛型的上限?extends 类

|  |
| --- |
| public class test{  public static void main(String args[]){  Message<Integer> m = new Message<Integer>(); //标记只能为Number类或者其子类  //Message<String> m1 = new Message<String>(); String不是Number的子类  m.setMsg(100);  fun(m);  }  public static void fun(Message<?extends Number> msg){ //在方法上设置  //msg.setMsg(200); 不能修改  System.out.println(msg.getMsg());  }  }  class Message<T extends Number>{ //在声明上设置  private T msg;  public T getMsg() {  return msg;  }  public void setMsg(T msg) {  this.msg = msg;  }  } |

范例:设置泛型下限?super 类

|  |
| --- |
| public class test{  public static void main(String args[]){  Message<Integer> m1 = new Message<>();  Message<String> m2 = new Message<>();  m2.setMsg("hello,world");  m1.setMsg(200);  //fun(m1); //报错,只能为String或String父类  fun(m2);  }  public static void fun(Message<?super String> msg){//设置在方法上  //msg.setMsg(200); 不能修改  System.out.println(msg.getMsg());  }  }  class Message<T>{ //不能设置在声明上  private T msg;  public T getMsg() {  return msg;  }  public void setMsg(T msg) {  this.msg = msg;  }  } |

# 泛型接口

子类有两种形式:

1. 子类继续设置泛型
2. 在子类不设置泛型,而为父接口明确定义一个泛型

**范例:子类设置泛型**

|  |
| --- |
| public class test{  public static void main(String args[]){  IMessage<String> msg = new MessageImpl<String>();  msg.print("hello,world");  }  }  //泛型接口  interface IMessage<T>{ //I表示接口  public void print(T t);  }  class MessageImpl<T> implements IMessage<T>{ //这里使用和接口相同的标记T  @Override  public void print(T t) {  System.out.print(t);  }  } |

**范例:在子类不设置泛型,而为父接口明确定义一个泛型**

|  |
| --- |
| public class test{  public static void main(String args[]){  IMessage msg = new MessageImpl();  msg.print("hello,world");  }  }  //泛型接口  interface IMessage<T>{ //I表示接口  public void print(T t);  }  class MessageImpl implements IMessage<String>{ //为父接口明确指定泛型类型  @Override  public void print(String t) {  System.out.print(t);  }  } |

# 泛型方法

范例:

|  |
| --- |
| public class test{  public static void main(String args[]){  String str = fun("hello");  System.out.print(str);  }  //T的类型由传入的参数类型决定  public static <T> T fun(T t){ //注意  return t;  }  } |