# 基础加强

## 1泛型

### 1.1泛型定义

具有一个或多个类型变量的类

class T02<String>{

}

class T03<String,Integer>{

}

class T04<T>{}

class T05<T,E>{}

### 1.2在创建泛型类实例

#### 1.2.1//如果创建实例时，不给类型变量赋值，那么会有一个警告

T02 t=new T02();//这里有警告

T02<String> t2=new T02<String>();

T03 t3=new T03();//这里有警告

T03<String,Integer> t4=new T03<String,Integer>();

### 1.3. 泛型方法

#### 区别:泛型方法与泛型类没有关系，泛型方法不是针对泛型类使用的

#### 1.3.1泛型类的方法

class T06<T>{

//泛型类中的方法

public T method\_set(T t1){

return t1;

}

}

#### 1.3.2泛型方法

class T06<T>{

//泛型方法

//泛型方法没有必要在泛型类中

public <T> T method\_set2(T t1){

return t1;

}

}

##### 1.3.2.1泛型方法的特点：

* 方法的参数中会使用泛型变量；
* 方法的返回值中会使用泛型变量。

|  |
| --- |
| public <T> T get(T[] ts) {  return ts[ts.length / 2];  } |
| String[] names ={“zhangSan”, “liSo”, “wangWu”};  String name = get(names); |

　　调用泛型方法时无需指定泛型变量，编译器会通过实际参数的类型来识别泛型变量的类型，上例中传递的参数为String[]类型，那么相当于给泛型类型T赋值为String

### 1.4. 泛型在类中或方法中的使用

\* 泛型类中使用泛型：

> 成员类型

> 返回值和参数类型

> 局部变量的引用上

/\*

\* 泛型类中使用泛型的位置

\* \*/

class T07<T>{

private T beanT07;//属性上使用

public T method\_get(T t){return t;}//方法上使用

public void method(){//泛型在局部变量上的引用类型上使用

T method\_bean;

//new T();错误

}

}

### 1.5. 泛型的继承和实现

/\*\*

\* 04泛型的继承和实现

\* \*\*/

class T08 extends T07<Stu>{//T08 不是泛型，只有他的父类是泛型

}

#### 1.5.1 继承泛型类

##### 1.5.1.1子类不是泛型类：需要给父类传递类型常量

> 当给父类传递的类型常量为Stu时，那么在父类中所有T都会被stu替换

class T08 extends T07<Stu>{//T08 不是泛型，只有他的父类是泛型}

T08 t08=new T08();

##### 1.5.1.2子类是泛型类：可以给父类传递类型常量，也可以传递类型变量

class T09<T> extends T07<T>{}

T09<Stu> t09=new T09<Stu>();

#### 1.5.2泛型的通配符

##### 1.5.2.1认识通配符

为了说明通配符的作用，我们先看个例子：

|  |
| --- |
| List<Object> list1 = new ArrayList<String>();  List<Object> list2 = new ArrayList<Integer>(); |

上面的调用都是编译不通过的！

这说明想写一个即可以打印list1，又可以打印list2的方法是不可能的！

|  |
| --- |
| public static void fun(List<Object> list) {…} |
| List<String> list1 = new ArrayList<String>();  List<Integer> list2 = new ArrayList<Integer>();  fun(list1);//编译不通过  fun(list2);//编译不通过 |

如果把fun()方法的泛型参数去除，那么就OK了。即不使用泛型！

|  |
| --- |
| public static void fun(List list) {…}//会有一个警告 |
| List<String> list1 = new ArrayList<String>();  List<Integer> list2 = new ArrayList<Integer>();  fun(list1);  fun(list2); |

上面代码是没有错了，但会有一个警告。警告的原因是你没有使用泛型！Java希望大家都去使用泛型。

##### 1.5.2.2　通配符概述

通配符就是专门处理这一问题的。

|  |
| --- |
| public static void fun(List<?> list) {…} |

上面代码中的“?”就是一个通配符，它只能在“<>”中使用。造成不能把它从“<>”中拿出来。

这时你可以向fun()方法传递List<String>、List<Integer>类型的参数了。当传递List<String>类型的参数时，表示给“?”赋值为String；当传递List<Integer>类型的参数给fun()方法时，表示给“?”赋值为Integer

##### 1.5.2.3. 通配符分类

无界通配：?

子类限定：? extends Object

父类限定：? super Integer

##### 1.5.2.4通配符的问题

　　带有通配符的参数不能使用与泛型相关的方法，例如：list.add(“hello”)编译不通过。

上面的问题是处理了，但通配符也有它的缺点。

在上面例子中，List<?> list参数中的通配符可以被赋任何值，但同时你也不知道通配符被赋了什么值。

当你不知道“?”是什么时，会使你不能使用任何与泛型相关的方法。也就是说fun()方法的参数list不能再使用它的与泛型相关的方法了。例如：list.add(“hello”)是错误的，因为List类的add()方法的参数是T类型，而现在你不知道T是什么类型，你怎么去添加String的东西给list呢？如果使用者在调用fun()方法时传递的不是List<String>，而是List<Integer>时，你添加String当然是不可以的。

当然，还可以调用list的get()方法。就算你不知道“?”是什么类型，但它肯定是Object类型的。所以你可以：Object o = list.get(0);

##### 1.5.2.5通配符的限制

　　通配符只能出现在引用的定义中，而不能出现在创建对象中。例如：new ArrayList<?>()，这是不可以的。ArrayList<?> list = null，这是可以的。

##### 1.5.2.6子类限定通配符

List<? extends Number> list;

其中<? extends Number>表示“?”只能被赋值为Number或其子类型。

|  |
| --- |
| public static void fun(List<? extends Number> list) {…} |
| fun(new ArrayList<Integer>());//ok  fun(new ArrayList<Double>());//ok  fun(new ArrayList<String>());//不ok |

当fun()方法的参数为List<? extends Number>后，说明你只能赋值给“?”Number或Number的子类型。

虽然这多了一个限制，但也有好处，因为你可以list的get()方法了。就算你不知道“?”是什么类型，但你知道它一定是Number或Number的子类型。所以：Number num = list.get(0)是可以的。

但是，还是不能调用list.add()方法！

##### 1.5.2.7父类限定通配符

List<? super Integer> list;

其中<? super Integer>表示 “?”只能被赋值为Integer或其父类型。

|  |
| --- |
| public static void fun(List<? super Integer> list) {…} |
| fun(new ArrayList<Integer>());//ok  fun(new ArrayList<Number>());//ok  fun(new ArrayList<Object>());//ok  fun(new ArrayList<String>());//错误 |

这时再去调用list.get()方法还是只能使用Object类型来接收：Object o = list.get(0)。因为你不知道“?”到底是Integer的哪个父类。

但是你可以调用list.add()方法了，例如：list.add(new Integer(100))是正确的。因为无论“?”是Integer、Number、Object，list.add(new Integer(100))都是正确的。

##### 1.5.2.8　通配符小结

1. 方法参数带有通配符会更加通用；

2. 带有通配符类型的对象，被限制了与泛型相关方法的使用；

3. 子类限定通配符：可以使用返回值为泛型变量的方法；

4. 父类限定通配符：可以使用参数为泛型变量的方法。

##### 1.5.2.9泛型父类获取子类传递的类型参数

看一个例子：

|  |
| --- |
| public class A<T> {  } |
| pubilc class B extends A<String> {  } |
| public class C extends A<Integer> {  } |

如果你需要在A类中得到子类给T赋值的类型，那么可以使用下面的方法：

|  |
| --- |
| public class A<T> {  public A() {  ParameterizedType pType = (ParameterizedType)this.getClass().getGenericSuperclass();  Class clazz = (Class)pType.getActualTypeArguments()[0];  System.out.println(clazz.getName());  }  } |

## 2注解

### 2.1　注解的概述

注释你还记得么？开个玩笑而已！

注释是给人看的，而注解是给程序看的！

注释是用来替代配置文件的！你回忆一下，我们以前总是要写一些配置文件，例如web.xml你还记得么？里面要写<servlet>和<servlet-mapping>！谁来读配置文件呢？当然是Tomcat！谁来写配置文件呢？当然是我们来写了！

在Servlet3.0中就可以使用使用注解来代替配置文件，开发者就不用再写配置文件了，而是写注解，然后Tomcat来读取注解。

注解也是类，需要定义了才能使用！

分别在Servlet3.0中有一个注解类为@WebServlet，然后我们就可以在Servlet中使用@WebServlet中使用这个注解了。这个注解就是用来替代<servlet>了。然后Tomcat会通过反射来读取注解中的信息！

### 2.2　Java中的注解

Java中的觉注解：

* @Overrid：作用在方法上的注解。当方法不是重写父类的方法时会报错；
* @Deprecated：作用在方法上。标记该方法为作废方法（已过时）；
* @SuppressWarnings：作用在方法上，压制警告。

### 2.3　定义注解类

定义注解类不能使用class、enum，也不能使用interface，而是使用@interface。

|  |
| --- |
| public @interface MyAnn{} |

### 2.4　使用注解目标

　　注解可以作用在：类（接口或枚举）、属性、方法、构造器、包、参数、局部变量

|  |
| --- |
| @MyAnn  public class MyClass {  @MyAnn  private int a;  @MyAnn  public MyClass() {}  @MyAnn  public void fun1() {}  @MyAnn  public void fun2(@MyAnn String s) {  @MyAnn  int n = 10;  }  } |

### 2.5　注解的属性

定义注解时也可以给出属性

|  |
| --- |
| public @interface MyAnn {  String value();  int value1();  } |

其中value就是属性！你可能会说，它是一个方法！没错，它是一个方法，但我们非要称之为属性，因为把它当做属性更加好理解。

当为注解指定属性后，那么在使用注解时就必须要给属性赋值了：

|  |
| --- |
| @MyAnn(value1=100,value="hello")  public class MyClass {  } |

注解的属性还可以有默认值，在使用注解时就可以不给带有默认值的属性赋值了。但没有给出默认值的属性还是要赋值的。

|  |
| --- |
| public @interface MyAnn {  String value() default "hello world";  int value1();  } |
| @MyAnn(value1=100)  public class MyClass {  } |

在使用注解时，如果只给名为value的属性赋值，那么可以不给出属性的名称直接给出值。

|  |
| --- |
| public @interface MyAnn {  String value() default "hello world";  int value1() default 100;  } |
| @MyAnn()  public class MyClass {  } |
| @MyAnn(value="hello")  public class MyClass {  } |
| @MyAnn(value1=200)  public class MyClass {  } |
| @MyAnn(value="hello",value1=200)  public class MyClass {  } |
| @MyAnn("hello annocation")  public class MyClass {  } |
| @MyAnn(300)  public class MyClass {  } |
| @MyAnn("hello",value1=200)  public class MyClass {  } |

* 注解的属性后面要有一对圆括号，而且圆括号内不能给出东西。就像是无参的方法一样；
* 注解的属性类型只能是：基本类型、String、Enum、Class、注解类型、以上类型的一维数组类型；
* 注解的属性可以有默认值，例如：int a() default 100;
* 数组的属性默认值：int[] arr() default {1,2,3}，这里不能使用new int[]{1,2,3}
* 使用注解时，在给数组属性赋值时的格式：@MyAnn(arr={1,2,3})；

### 2.6　注解的作用目标

在定义注解时可以限制注解的作用目录！例如让注解只能作用在类和方法上。

这需要使用元注解：@Target。该注解有一个属性value，类型为ElementType[]，它是枚举类型。

|  |
| --- |
| public @interface Target {  ElementType[] value();  } |
| public enum ElementType {  TYPE,FIELD,METHOD,PARAMETED,CONSTRUCTOR,LOCAL\_VARIABLE,ANNOCATION\_TYPE,PACKAGE  } |

在定义注解时，可以使用@Target注解来限制注解的作用目标：

|  |
| --- |
| @Target({ElementType.*TYPE*, ElementType.*METHOD*})  public @interface MyAnn {  } |

这样MyAnn就只能作用在类和方法上的！其中ElementType.TYPE表示类和接口。

|  |
| --- |
| @MyAnn()  public class MyClass {  @MyAnn()  private int a;  @MyAnn()  public void fun() {}  } |

### 2.7　注解的保留策略

注解的保留策略是指，注解是只保留在源代码上，还是保留到class文件上，再或者是类在运行时，可以被类加载器加载到内存中。

如果希望注解被反射，那么注解就要保留到运行时，而不是源代码或类文件上。

指定注解的保留策略需要使用元注解@Retention，它有一个value属性，类型为RetentionPolicy类型，RetentionPolicy是枚举类型：

|  |
| --- |
| public @interface Retention {  RetentionPolicy value();  } |
| public enum RetentionPolicy {  SOURCE, CLASS, RUNTIME  } |

下面代码是指定注解保留到运行时

|  |
| --- |
| @Retention(RetentionPolicy.*RUNTIME*)  @Target({ElementType.*TYPE*, ElementType.*METHOD*})  public @interface MyAnn {  String value() default "hello";  int value1() default 100;  } |

### 2.8　通过反射读取注解

读取注解需要使用反射来完成

|  |
| --- |
| @Retention(RetentionPolicy.*RUNTIME*)  @Target({ElementType.*TYPE*, ElementType.*METHOD*})  public @interface MyAnn {  String value() default "hello";  int value1() default 100;  } |
| @MyAnn(value="hello world", value1=200)  public class MyClass {  private int a;    @MyAnn("myMethod")  public void fun() {}  } |
| public class Demo1 {  public static void main(String[] args) throws Exception {  Class clazz = MyClass.class;  MyAnn myAnn = (MyAnn) clazz.getAnnotation(MyAnn.class);  System.*out*.println(myAnn.value());  System.*out*.println(myAnn.value1());    Method method = clazz.getMethod("fun");  MyAnn myAnn1 = method.getAnnotation(MyAnn.class);  System.*out*.println(myAnn1.value());  System.*out*.println(myAnn1.value1());  }  } |

## 3反射

### 3.1class类

获取class对象的3种方法

#### 3.1.1class.forName(“全类名”)

public class ReflectDemo {

@Test

public void test01() {

try {

Class clazz=Class.*forName*("com.haoyu.demo01.ReflectTestDemo");

// Class clazz = Class.forName("java.lang.String");

System.*out*.println(clazz.getSimpleName());

} catch (Exception e) {

}

}

}

class ReflectTestDemo {

}

#### 3.1.2类名.class

@Test

public void test02(){

ReflectDemo rd=new ReflectDemo();

System.*out*.println(ReflectDemo.class);

}

#### 3.1.3getClass方法

@Test

public void test02(){

ReflectDemo rd=new ReflectDemo();

System.*out*.println(rd.getClass().getSimpleName());

}

注意：

1.获取到Class类实例之后，有什么用呢？

首先可以通过Class类实例的getName()和getSimpleName()方法，获取到对应类的全类名或类名。通过getPackage()可以得到一个类所在的包。  
2.通过Class类实例的getSuperclass()方法，获取到对应类的完整继承链关系。

1. // 输出结果：
2. // class java.awt.Window
3. // class java.awt.Container
4. // class java.awt.Component
5. // class java.lang.Object
6. Class c = java.awt.Frame.class;
7. while (c.getSuperclass() != null) {
8. System.out.println(c.getSuperclass());
9. c = c.getSuperclass();
10. }

3.除了java.lang.reflect.ReflectPermission和java.lang.reflect包外，所有类都没有公共的构造方法。为了获得这些类的对象，必须使用Class类中适当的方法，对于不同的数据类型，Class对象的获得方式是不同的。  
技能2：Constructor(构造器对象)

介绍：构造器的作用就是用来创建对象的，Constructor代表的是构造器对象，我们得到它后就可以实例化对象.  
秘籍：要领悟这个技能，需要获取Constructor对象，而获取Constructor对象的方式有一下4种：  
  
                    1.getConstructor()：得到指定的public修饰构造器，参数表示是否具有参数的构造器  
                    2.getConstructors()：得到所有的public构造器，其中包含没有参数和有参数的  
                    3.getDeclaredConstructor()：得到指定的构造器，其中包含由public、private和protected修饰符的。  
                    4.getDeclaredConstructors()：得到所有的构造器，其中包含public、private和protected修饰符的。  
  
            注意：1.如果构造方法不是public。那么要获取其构造器对象使用  getDeclaredConstuctor()方法.  
                      2.对于私有化权限，在使用前必须设置权限检查.  
                            public void setAccessible(boolean flag)  
                            值为 true 则指示反射的对象在使用时应该取消 Java 语言访问检查  
  
                      3.在开发中，我们在得到Class时，一般会直接实例化类的对象.  
  
                    4.Class类的newInstance()方法是使用该类无参的构造函数创建对象, 如果一个类没有无参的构造函数, 就不能这样创建了,可以调用Class类的getConstructor(String.class,int.class)方法获取一个指定的构造函数然后再调用Constructor类的newInstance("张三",20)方法创建对象  
  
技能3：Field(成员变量)  
  
            介绍：Field类代表某个类中的一个成员变量，病提供动态的访问权限。  
  
            秘籍：要领悟这个技能，可以通过以下4个招式获得此技能：  
  
                    1.getFields()方法：获取Class类及父类的所有public成员变量。  
                    2.getDeclaredFields()方法:获取Class类的所有成员变量，其中包含由public、private修饰符的。  
                    3.getField方法():获取Class类及父类的具体成员变量。  
                    4.getDeclaredField()方法：获取Class类的具体成员变量，其中包含由public、private修饰符的。但不能是父类中的方法。  
  
            注意： 1.Class.getField(String)方法可以获取类中的指定字段(可见的), 如果是私有的可以用getDeclaedField("name")方法获取。  
                        2.通过set(obj, "李四")方法可以设置指定对象上该字段的值, 如果是私有的需要先调用setAccessible(true)设置访问权限,用获取的指定的字段调用get(obj)可以获取指定对象中该字段的值。  
  
技能4：Method(方法)  
  
           介绍：Method类代表某个类中的一个成员方法。  
  
           秘籍：要领悟这个技能，可以通过以下4个招式获得此技能：  
  
                    1.getMethods()方法：获取Class类及父类的所有public方法。  
                    2.getDeclaredMethods()方法:获取Class类的所有方法，其中包含由public、private修饰符的。  
                    3.getMethod方法():获取Class类及父类的具体方法。  
                    4.getDeclaredMethod()方法：获取Class类的具体方法，其中包含由public、private修饰符的。但不能是父类中的方法。  
  
           注意： Class.getMethod(String, Class...) 和 Class.getDeclaredMethod(String, Class...)方法可以获取类中的指定方法,调用invoke(Object, Object...)可以调用该方法,Class.getMethod("eat") invoke(obj) Class.getMethod("eat",int.class) invoke(obj,10)  
  
技能5：AccessibleObject       
        
           介绍：AccessibleObject类是Constructor、Method和Field三个类的父类。  
  
           秘籍：要领悟这个技能，可以通过以下2个招式获得此技能：  
  
                    1.isAccessible()方法：判断当前成员是否可访问。  
                    2.setAccessible()方法:设置当前成员是否可访问。  
                  
           注意：当Class类中的构造器、方法和成员变量是私有的时候，如果我们想反射操作的话，就必须先调用setAccessible(true)方法。