主讲人　张龙

Jdk5.0中新特性介绍

- 教学目标
  - 掌握jdk5.0中出现的新特性
    - 泛型（Generics）
    - 增强的“for”循环（Enhanced For loop）
    - 自动装箱/自动拆箱（Autoboxing/unboxing）
    - 类型安全的枚举（Type safe enums）
    - 静态导入（Static import）
    - 可变参数(Var args)

泛型（Generics）

- 泛型是JDK1.5中一个最重要的特征。通过引入泛型，我们将获得编译时类型的安全和运行时更小地抛出ClassCastExceptions的可能。
- 在JDK1.5中，你可以声明一个集合将接收/返回的对象的类型

泛型之前

- 参见程序 BooleanFoo.java
- 参见程序 IntegerFoo.java

泛型之前

- 类别定义时的逻辑完全一样，只是里面成员变量的类型不同
- 如果需要多个相似的类，需要定义多个文件，不同的只是变量的类别，而逻辑是完全一样的

泛型之前

- 对之前代码的一些改写
- 参见程序 ObjectFoo.java

定义泛型类别

- 参见程序 GenericFoo.java
- 参见程序 Generic.java
- 参见程序 Generic2.java
- 参见程序 SimpleCollection.java
- 参见程序 WrapperFoo.java

定义泛型类别

- 如果使用泛型，只要代码在编译时没有出现警告，就不会遇到运行时ClassCastException

限制泛型可用类型

- 在定义泛型类别时,预设可以使用任何的类型来实例化泛型类型中的类型,但是如果想要限制使用泛型类别时,只能用某个特定类型或者是其子类型才能实例化该类型时,可以在定义类型时,使用extends关键字指定这个类型必须是继承某个类,或者实现某个接口
- 参见程序 ListGenericFoo.java

限制泛型可用类型

- 当没有指定泛型继承的类型或接口时,默认使用T extends Object,所以默认情况下任何类型都可以作为参数传入

类型通配声明

```
public class GenericFoo<T>
{
    private T foo;

    public void setFoo(T foo)
    {
            this.foo = foo;
    }

   public T getFoo()
   {
            return foo;
   }
}
```

类型通配声明

- **GenericFoo<Integer> foo1 = null;**
  GenericFoo<Boolean> foo2 = null;
- 那么 foo1 就只接受GenericFoo<Integer>的实例，而foo2只接受GenericFoo<Boolean>的实例。

类型通配声明

- 現在您有這麼一個需求，您希望有一個參考名称foo可以接受所有下面的实例
- **foo = new GenericFoo<ArrayList>();**
  foo = new GenericFoo<LinkedList>();
- 简单的说，实例化类型持有者时，它必须是实现List的类别或其子类别，要定义这样一个名称，您可以使用 ‘?’ 通配字元，并使用“extends”关键字限定类型持有者的型态

类型通配声明

- **GenericFoo<? extends List> foo = null;**
  foo = new GenericFoo<ArrayList>();
  foo = new GenericFoo<LinkedList>();
- 参见程序 GenericTest.java

类型通配声明

- 使用<?>或是<? extends SomeClass>的声明方式，意味著您只能通过该名称來取得所参考实例的信息，或者是移除某些信息，但不能增加它的信息，因为只知道当中放置的是SomeClass的子类，但不确定是什么类的实例，编译器不让您加入信息，理由是，如果可以加入信息的話，那么您就得記得取回的实例是什么类型，然后转换为原來的类型方可进行操作，这就失去了使用泛型的意义。

继承泛型类别 实现泛型接口

- 参见程序 Parent.java
- 参见程序 Child.java
- 参见程序 ParentInterface.java
- 参见程序 ChildClass.java

For-Each循环

- For-Each循环的加入简化了集合的遍历
- 其語法如下
  - **for(type element : array) {**
    System.out.println(element)....
    }
- 参见程序 ForTest.java

自动装箱/拆箱(Autoboxing/unboxing)

- 自动装箱/拆箱大大方便了基本类型数据和它们包装类的使用。
- 自动装箱：基本类型自动转为包装类.(int >> Integer)
- 自动拆箱：包装类自动转为基本类型 .(Integer >> int)
- 参见程序 BoxTest.java
- 参见程序 Frequency.java
- 参加程序 BoxTest2.java

枚举(Enums)

- JDK1.5加入了一个全新类型的“类”—枚举类型。为此JDK1.5引入了一个新关键字enum. 我们可以这样来定义一个枚举类型
- public enum Color
  {
  Red,
  White,
  Blue
  }

- 然后可以这样来使用Color myColor = Color.Red.

枚举(Enums)

- 枚举类型还提供了两个有用的静态方法values()和valueOf(). 我们可以很方便地使用它们，例如
- for (Color c : Color.values())

  System.out.println(c);

枚举(Enums)

- 参见程序 EnumTest.java
- 参见程序 Coin.java

枚举(Enums)

- 定义枚举类型时本质上就是在定义一個类别，只不过很多细节由编译器帮您完成了，所以某些程度上，enum关键字的作用就像是class或interface。

枚举(Enums)

- 当您使用“enum”定义 枚举类型时，实质上您定义出來的类型继承自 java.lang.Enum 类型，而每个枚举的成员其实就是您定义的枚举类型的一個实例（Instance），他们都被预设为 final，所以您无法改变他们，他们也是 static 成員，所以您可以通过类型名称直接使用他们，当然最重要的，它們都是公开的（public）。

枚举(Enums)

- 枚举的比较
  - 参见程序 ShowEnum.java
- 枚举的顺序
  - 参见程序 ShowEnum2.java
- 枚举的方法
  - 参见程序 ShowEnum3.java

EnumSet

- EnumSet的名称说明了其作用，它是在J2SE 5.0后加入的新类别，可以协助您建立枚举值的集合，它提供了一系列的静态方法，可以让您指定不同的集合建立方式
- 参见程序 EnumSetDemo.java
- 参见程序 EnumSetDemo2.java
- 参见程序 EnumSetDemo3.java

EnumMap

- EnumMap是个专为枚举类型设计的类别，方便您使用枚举类型及Map对象
- 参见程序 EnumMapDemo.java

EnumMap

- 与单纯的使用HashMap比较起來的差别是，在上面的程序中，**EnumMap将根据枚举的顺序來维护对象的排列顺序**
- 参见程序 EnumMapDemo2.java
  - 从遍访的结果可以看出，对象的顺序是根据枚举顺序來排列的。

静态导入（Static import）

- 要使用静态成员（方法和变量）我们必须给出提供这个静态成员的类。使用静态导入可以使被导入类的所有静态变量和静态方法在当前类直接可见，使用这些静态成员无需再给出他们的类名
- 参见程序 Common.java
- 参见程序 StaticImport.java
- 参见程序 StaticImport2.java

静态导入（Static import）

- 不过，过度使用这个特性也会一定程度上降低代码的可读性

可变参数(Varargs)

- 可变参数使程序员可以声明一个接受可变数目参数的方法。注意，可变参数必须是方法声明中的最后一个参数
- 参见程序 TestVarargs.java