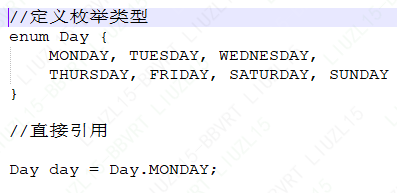
**枚举类型**

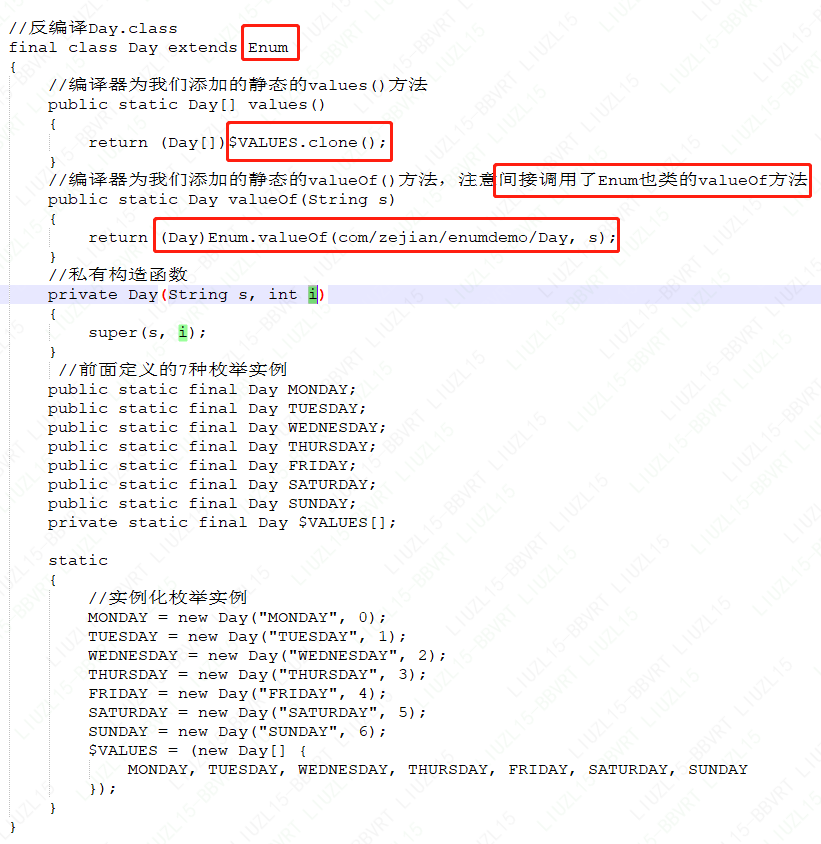
**定义**

使用关键字 enum来定义枚举类型，可定义为一个单独文件，也可定义在其他类内部，更重要的是枚举常量在类型安全性和便捷性都很有保证，如果出现类型问题编译器也会提示我们改进。



**实现原理**

使用关键字enum创建枚举类型并编译后，编译器会为我们生成一个相关的类，这个类继承了Java API中的java.lang.Enum类，也就是说通过关键字enum创建枚举类型在编译后事实上也是一个类类型而且该类继承自java.lang.Enum类。



使用关键字enum定义的枚举类型，在编译期后，也将转换成为一个实实在在的类，而在该类中，会存在每个在枚举类型中定义好变量的**对应实例对象**。

**枚举的常见方法**

|  |  |  |
| --- | --- | --- |
| 返回类型 | 方法名称 | 方法说明 |
| int | compareTo(E o) | 比较此枚举与指定对象的顺序（内部实现是根据每个枚举的ordinal值大小进行比较的） |
| boolean | equals(Object other) | 当指定对象等于此枚举常量时，返回true |
| Class<?> | getDeclaringClass() | 返回与此枚举常量的枚举类型相对应的Class对象 |
| String | name() | 返回此枚举常量的名称，在其枚举声明中对其进行声明 |
| int | ordinal | 返回枚举常量的序数（它在枚举声明中的位置，其中初始常量序数为零） |
| String | toString | 返回枚举常量的名称，它包含在声明中 |
| static<T extends Enum<T>> T | static valueOf(Class<T> enumType,String name) | 返回带指定名称的指定枚举类型的枚举常量 |

**编译器生成的Values方法与 ValuesOf()方法**

values()方法和valueOf(String name)方法是编译器生成的static方法。values()方法的作用就是获取枚举类中的所有变量，并作为数组返回，而valueOf(String name)方法与Enum类中的valueOf方法的作用类似根据名称获取枚举变量。

**枚举与Class对象**

当枚举实例向上转型为Enum类型后，values()方法将会失效，也就无法一次性获取所有枚举实例变量，但是由于Class对象的存在，即使不使用values()方法，还是有可能一次获取到所有枚举实例变量的，在**Class对象**中存在如下方法：

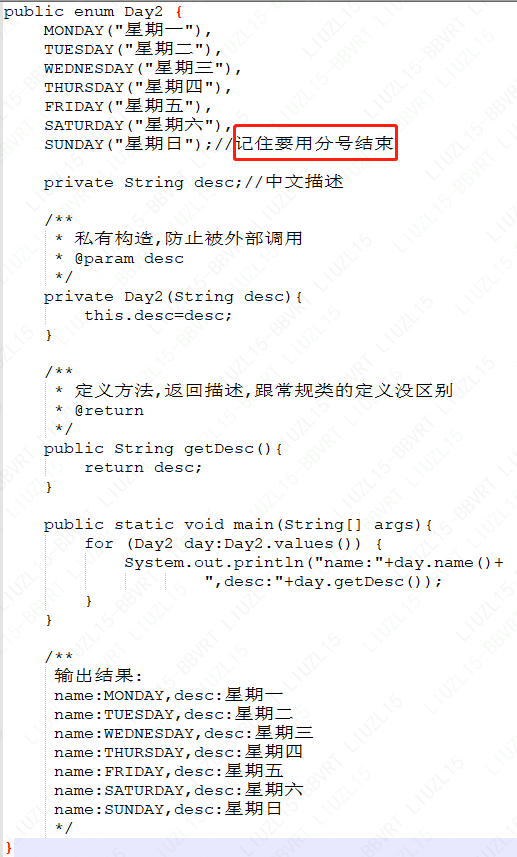
|  |  |  |
| --- | --- | --- |
| 返回类型 | 方法名称 | 方法说明 |
| T[] | getEnumConstants() | 返回该枚举类型的所有元素，如果Class对象不是枚举类型，则返回null |
| boolean | isEnum() | 当且仅当该类声明为源代码中的枚举时返回 true |

Day[] dsz = (Day[]) clasz.getEnumConstants();

**枚举的进阶用法**

**向enum类添加方法与自定义构造函数**

带有desc成员变量描述该日期的对于中文描述，同时定义一个getDesc方法，返回中文描述内容，自定义私有构造函数，在声明枚举实例时传入对应的中文描述。



如果打算在enum类中定义方法，**务必在声明完枚举实例后使用分号分开**，倘若在枚举实例前定义任何方法，编译器都将会报错，无法编译通过，同时即使自定义了构造函数且enum的定义结束，我们也永远无法手动调用构造函数创建枚举实例，毕竟这事只能由编译器执行。

**关于覆盖enum类方法**

父类Enum中的定义的方法只有toString方法没有使用final修饰，因此只能覆盖toString方法。

@Override

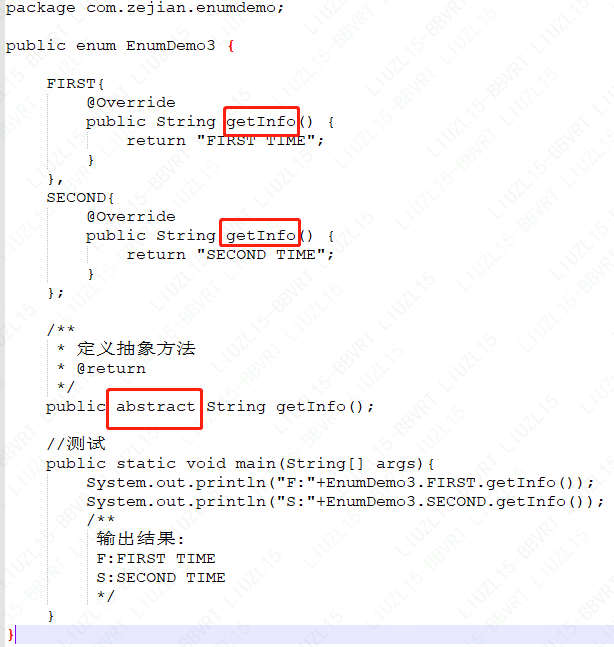
public String toString() {

return desc;

}

**enum类中定义抽象方法**

enum类允许我们为其定义抽象方法，然后使每个枚举实例都实现该方法，以便产生不同的行为方式，注意abstract关键字对于枚举类来说并不是必须的如下：



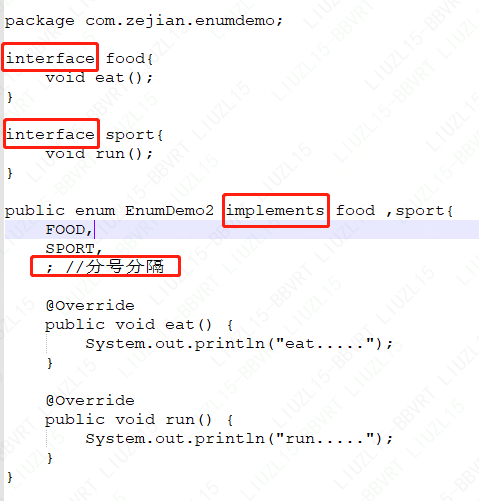
enum类的实例似乎表现出了多态的特性，可惜的是**枚举类型的实例终究不能作为类型传递使用**，就像下面的使用方式，编译器是不可能答应的。

//**无法通过编译**,毕竟EnumDemo3.FIRST是个实例对象

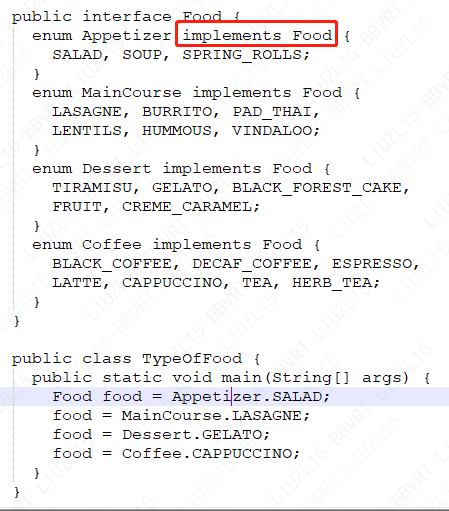
public void text(EnumDemo3.FIRST instance){ }

**enum类与接口**

由于Java单继承的原因，enum类并不能再继承其它类，但并不妨碍它实现接口，因此enum类同样是可以实现多接口的。

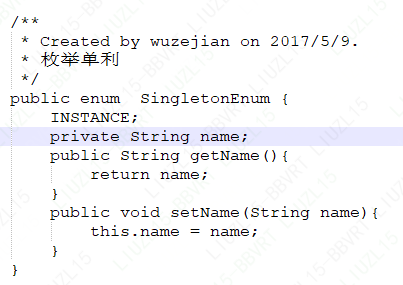


使用场景（分类）



**枚举与switch**

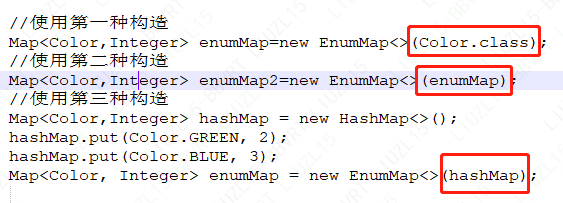
**枚举与单例模式**



枚举序列化是由jvm保证的，每一个枚举类型和定义的枚举变量在JVM中都是唯一的，在**枚举类型的序列化和反序列化上，Java做了特殊的规定**：在序列化时Java仅仅是将枚举对象的name属性输出到结果中，反序列化的时候则是通过java.lang.Enum的valueOf方法来根据名字查找枚举对象。同时，编译器是不允许任何对这种序列化机制的定制的并禁用了writeObject、readObject、readObjectNoData、writeReplace和readResolve等方法，从而保证了枚举实例的唯一性。

**EnumMap**

作为枚举的专属的集合。EnumMap**要求其Key必须为Enum类型**，因而使用Color枚举实例作为key是最恰当不过了，也避免了获取name的步骤，更重要的是EnumMap效率更高，因为其**内部是通过数组实现**的。



注意与HashMap的主要不同在于构造方法需要传递类型参数和EnumMap保证Key顺序与枚举中的顺序一致，但请记住Key不能为null。

**EnumSet**