<https://blog.csdn.net/qq_27093465/article/details/52180865>

JDK1.5引入了新的类型——枚举。在 Java 中它虽然算个“小”功能，却给我的开发带来了“大”方便。

大师兄我又加上自己的理解，来帮助各位理解一下。

## 用法一：常量

在JDK1.5 之前，我们定义常量都是： public static fianl.... 。现在好了，有了枚举，可以把相关的常量分组到一个枚举类型里，而且枚举提供了比常量更多的方法。

**Java代码**

1. **public** **enum** Color {
2. RED, GREEN, BLANK, YELLOW
3. }

## 用法二：switch

JDK1.6之前的switch语句只支持int,char,enum类型，使用枚举，能让我们的代码可读性更强。

**Java代码**

1. **enum** Signal {
2. GREEN, YELLOW, RED
3. }
4. **public** **class** TrafficLight {
5. Signal color = Signal.RED;
6. **public** **void** change() {
7. **switch** (color) {
8. **case** RED:
9. color = Signal.GREEN;
10. **break**;
11. **case** YELLOW:
12. color = Signal.RED;
13. **break**;
14. **case** GREEN:
15. color = Signal.YELLOW;
16. **break**;
17. }
18. }
19. }

## 用法三：向枚举中添加新方法

如果打算自定义自己的方法，那么必须在enum实例序列的最后添加一个分号。而且 Java 要求必须先定义 enum 实例。

**Java代码**

1. **public** **enum** Color {
2. RED("红色", 1), GREEN("绿色", 2), BLANK("白色", 3), YELLO("黄色", 4);
3. // 成员变量
4. **private** String name;
5. **private** **int** index;
6. // 构造方法
7. **private** Color(String name, **int** index) {
8. **this**.name = name;
9. **this**.index = index;
10. }
11. // 普通方法
12. **public** **static** String getName(**int** index) {
13. **for** (Color c : Color.values()) {
14. **if** (c.getIndex() == index) {
15. **return** c.name;
16. }
17. }
18. **return** **null**;
19. }
20. // get set 方法
21. **public** String getName() {
22. **return** name;
23. }
24. **public** **void** setName(String name) {
25. **this**.name = name;
26. }
27. **public** **int** getIndex() {
28. **return** index;
29. }
30. **public** **void** setIndex(**int** index) {
31. **this**.index = index;
32. }
33. }

## 用法四：覆盖枚举的方法

下面给出一个toString()方法覆盖的例子。

**Java代码**

1. **public** **enum** Color {
2. RED("红色", 1), GREEN("绿色", 2), BLANK("白色", 3), YELLO("黄色", 4);
3. // 成员变量
4. **private** String name;
5. **private** **int** index;
6. // 构造方法
7. **private** Color(String name, **int** index) {
8. **this**.name = name;
9. **this**.index = index;
10. }
11. //覆盖方法
12. @Override
13. **public** String toString() {
14. **return** **this**.index+"\_"+**this**.name;
15. }
16. }

## 用法五：实现接口

所有的枚举都继承自java.lang.Enum类。由于Java 不支持多继承，所以枚举对象不能再继承其他类。

**Java代码**

1. **public** **interface** Behaviour {
2. **void** print();
3. String getInfo();
4. }
5. **public** **enum** Color **implements** Behaviour{
6. RED("红色", 1), GREEN("绿色", 2), BLANK("白色", 3), YELLO("黄色", 4);
7. // 成员变量
8. **private** String name;
9. **private** **int** index;
10. // 构造方法
11. **private** Color(String name, **int** index) {
12. **this**.name = name;
13. **this**.index = index;
14. }
15. //接口方法
16. @Override
17. **public** String getInfo() {
18. **return** **this**.name;
19. }
20. //接口方法
21. @Override
22. **public** **void** print() {
23. System.out.println(**this**.index+":"+**this**.name);
24. }
25. }

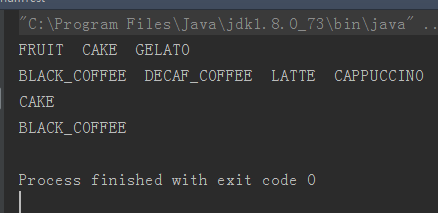
## 用法六：使用接口组织枚举

**Java代码**

1. **public** **interface** Food {
2. **enum** Coffee **implements** Food{
3. BLACK\_COFFEE,DECAF\_COFFEE,LATTE,CAPPUCCINO
4. }
5. **enum** Dessert **implements** Food{
6. FRUIT, CAKE, GELATO
7. }
8. }

**[java]** [view plain](https://blog.csdn.net/qq_27093465/article/details/52180865) [copy](https://blog.csdn.net/qq_27093465/article/details/52180865)

1. /\*\*
2. \* 测试继承接口的枚举的使用（by 大师兄 or 大湿胸。）
3. \*/
4. **private** **static** **void** testImplementsInterface() {
5. **for** (Food.DessertEnum dessertEnum : Food.DessertEnum.values()) {
6. System.out.print(dessertEnum + "  ");
7. }
8. System.out.println();
9. //我这地方这么写，是因为我在自己测试的时候，把这个coffee单独到一个文件去实现那个food接口，而不是在那个接口的内部。
10. **for** (CoffeeEnum coffee : CoffeeEnum.values()) {
11. System.out.print(coffee + "  ");
12. }
13. System.out.println();
14. //搞个实现接口，来组织枚举，简单讲，就是分类吧。如果大量使用枚举的话，这么干，在写代码的时候，就很方便调用啦。
15. //还有就是个“多态”的功能吧，
16. Food food = Food.DessertEnum.CAKE;
17. System.out.println(food);
18. food = CoffeeEnum.BLACK\_COFFEE;
19. System.out.println(food);
20. }

运行结果

## 用法七：关于枚举集合的使用

java.util.EnumSet和java.util.EnumMap是两个枚举集合。EnumSet保证集合中的元素不重复；EnumMap中的 key是enum类型，而value则可以是任意类型。关于这个两个集合的使用就不在这里赘述，可以参考JDK文档。

关于枚举的实现细节和原理请参考：

参考资料：《ThinkingInJava》第四版

<http://softbeta.iteye.com/blog/1185573>

**我的这篇文章，因为是转载的，可能基本就没有变动，导致被某人踩了一脚。觉得不符合我大师兄的性格。下面我把自己的使用理解给整理一下。**

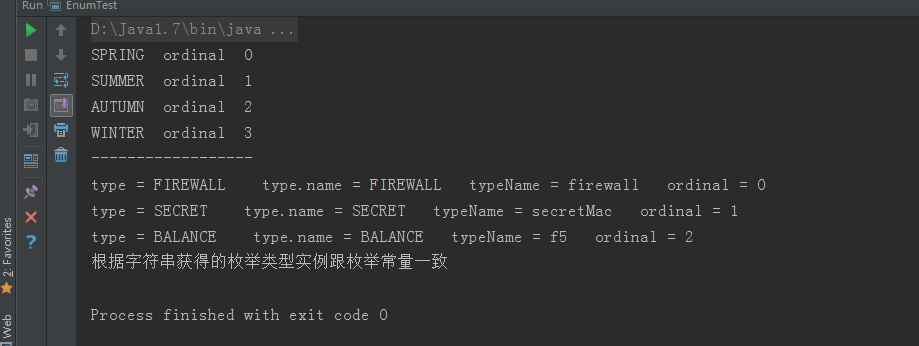
**也是因为因为当时刚刚开始学习吧。把平时自以为了解的东西都只是大概了解了一下，说到底，还是自以为了解了，其实转眼就不知道什么是什么了。  
出来学习，不习惯看代码怎么能行呢？  
下面是我自己的测试代码。**

**[java]** [view plain](https://blog.csdn.net/qq_27093465/article/details/52180865) [copy](https://blog.csdn.net/qq_27093465/article/details/52180865)

1. **package** com.lxk.enumTest;
3. /\*\*
4. \* Java枚举用法测试
5. \* <p>
6. \* Created by lxk on 2016/12/15
7. \*/
8. **public** **class** EnumTest {
9. **public** **static** **void** main(String[] args) {
10. forEnum();
11. useEnumInJava();
12. }
14. /\*\*
15. \* 循环枚举,输出ordinal属性；若枚举有内部属性，则也输出。(说的就是我定义的TYPE类型的枚举的typeName属性)
16. \*/
17. **private** **static** **void** forEnum() {
18. **for** (SimpleEnum simpleEnum : SimpleEnum.values()) {
19. System.out.println(simpleEnum + "  ordinal  " + simpleEnum.ordinal());
20. }
21. System.out.println("------------------");
22. **for** (TYPE type : TYPE.values()) {
23. System.out.println("type = " + type + "    type.name = " + type.name() + "   typeName = " + type.getTypeName() + "   ordinal = " + type.ordinal());
24. }
25. }
27. /\*\*
28. \* 在Java代码使用枚举
29. \*/
30. **private** **static** **void** useEnumInJava() {
31. String typeName = "f5";
32. TYPE type = TYPE.fromTypeName(typeName);
33. **if** (TYPE.BALANCE.equals(type)) {
34. System.out.println("根据字符串获得的枚举类型实例跟枚举常量一致");
35. } **else** {
36. System.out.println("大师兄代码错误");
37. }
39. }
41. /\*\*
42. \* 季节枚举(不带参数的枚举常量)这个是最简单的枚举使用实例
43. \* Ordinal 属性，对应的就是排列顺序，从0开始。
44. \*/
45. **private** **enum** SimpleEnum {
46. SPRING,
47. SUMMER,
48. AUTUMN,
49. WINTER
50. }

53. /\*\*
54. \* 常用类型(带参数的枚举常量，这个只是在书上不常见，实际使用还是很多的，看懂这个，使用就不是问题啦。)
55. \*/
56. **private** **enum** TYPE {
57. FIREWALL("firewall"),
58. SECRET("secretMac"),
59. BALANCE("f5");
61. **private** String typeName;
63. TYPE(String typeName) {
64. **this**.typeName = typeName;
65. }
67. /\*\*
68. \* 根据类型的名称，返回类型的枚举实例。
69. \*
70. \* @param typeName 类型名称
71. \*/
72. **public** **static** TYPE fromTypeName(String typeName) {
73. **for** (TYPE type : TYPE.values()) {
74. **if** (type.getTypeName().equals(typeName)) {
75. **return** type;
76. }
77. }
78. **return** **null**;
79. }
81. **public** String getTypeName() {
82. **return** **this**.typeName;
83. }
84. }
85. }

**然后是测试的结果图：**



简单的例子，大家基本都用过，看不懂的基本都是第二个例子。可以看到，在第二个例子里面，后面带有参数，其实可以这么理解。

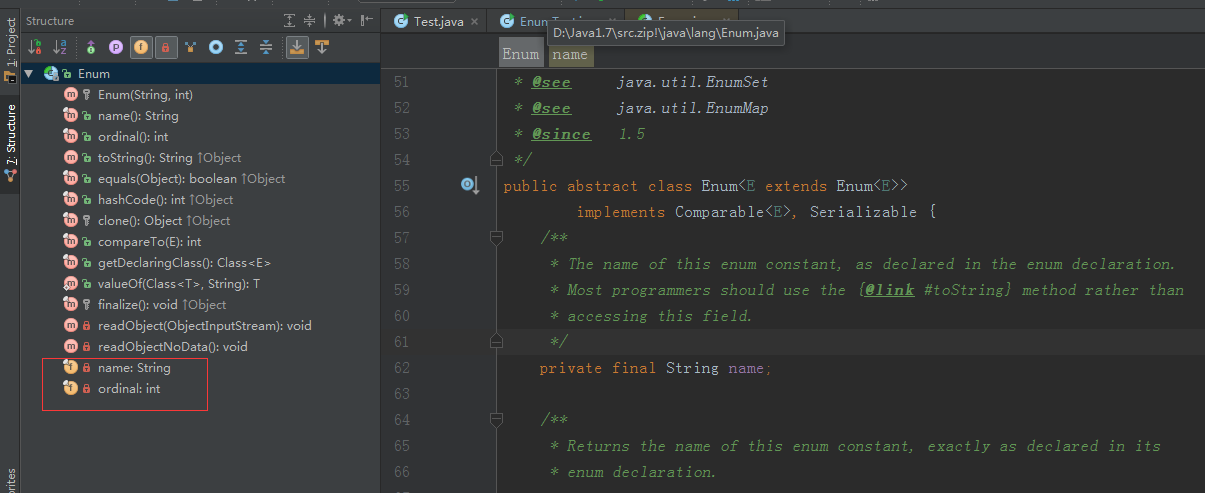
enum这个关键字，可以理解为跟class差不多，这也个单独的类。可以看到，上面的例子里面有属性，有构造方法，有getter，也可以有setter，但是一般都是构造传参数。还有其他自定义方法。那么在这些东西前面的，以逗号隔开的，最后以分号结尾的，这部分叫做，这个枚举的实例。也可以理解为，class  new 出来的实例对象。这下就好理解了。只是，class，new对象，可以自己随便new，想几个就几个，而这个enum关键字，他就不行，他的实例对象，只能在这个enum里面体现。也就是说，他对应的实例是有限的。这也就是枚举的好处了，限制了某些东西的范围，举个栗子：一年四季，只能有春夏秋冬，你要是字符串表示的话，那就海了去了，但是，要用枚举类型的话，你在enum的大括号里面把所有的选项，全列出来，那么这个季节的属性，对应的值，只能在里面挑。不能有其他的。

我上面的例子，就是根据typeName，你可以从那些例子里面挑选到唯一的一个TYPE类型的枚举实例--TYPE.BALANCE。注意方法

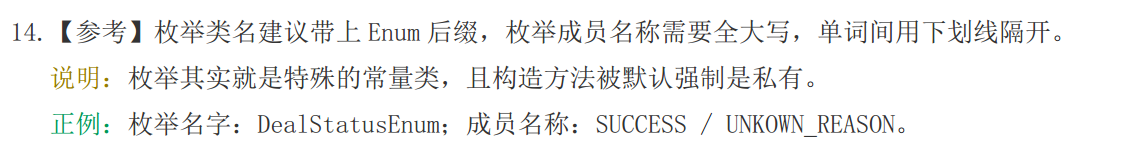
TYPE type = TYPE.fromTypeName(typeName);  
这个方法的返回类型就是这个TYPE枚举类型的。  
这下就好理解，这个枚举是怎么在工作了吧

再补充一下：

上面那个带参数的枚举类型的实例里面实际上是三个属性，除了我自定义的typeName以外，还有2个是系统自带的。看下面源码的图：



看到这里之后，不知道你能不能理解下面图片内说明的话：下面图片主要说明在使用枚举时，的规范和标准。希望可以在实际开发时候用到



https://img-blog.csdn.net/20170112172408793

**最后补充一点：**

也许你知道呢，但是也许你不知道呢？我是真的不知道，测了之后才知道！！！

**枚举类型对象之间的值比较，是可以使用==，直接来比较值，是否相等的，不是必须使用equals方法的哟。**

具体，请参考下面的链接：

[java 枚举类比较是用==还是equals？](http://blog.csdn.net/qq_27093465/article/details/70237349)

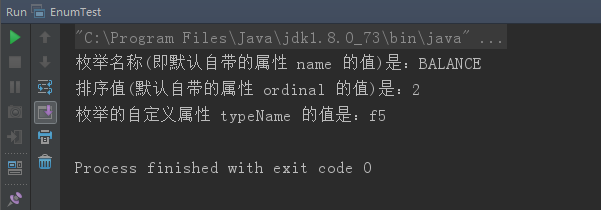
**2017.11.07 更新**

有的老铁，说这个switch case怎么写，我就在下面再啰嗦一下。

**[java]** [view plain](https://blog.csdn.net/qq_27093465/article/details/52180865) [copy](https://blog.csdn.net/qq_27093465/article/details/52180865)

1. **private** **static** **void** testSwitchCase() {
2. String typeName = "f5";
3. //这几行注释呢，你可以试着三选一，测试一下效果。
4. //String typeName = "firewall";
5. //String typeName = "secretMac";
6. TypeEnum typeEnum = TypeEnum.fromTypeName(typeName);
7. **if** (typeEnum == **null**) {
8. **return**;
9. }
10. **switch** (typeEnum) {
11. **case** FIREWALL:
12. System.out.println("枚举名称(即默认自带的属性 name 的值)是：" + typeEnum.name());
13. System.out.println("排序值(默认自带的属性 ordinal 的值)是：" + typeEnum.ordinal());
14. System.out.println("枚举的自定义属性 typeName 的值是：" + typeEnum.getTypeName());
15. **break**;
16. **case** SECRET:
17. System.out.println("枚举名称(即默认自带的属性 name 的值)是：" + typeEnum.name());
18. System.out.println("排序值(默认自带的属性 ordinal 的值)是：" + typeEnum.ordinal());
19. System.out.println("枚举的自定义属性 typeName 的值是：" + typeEnum.getTypeName());
20. **break**;
21. **case** BALANCE:
22. System.out.println("枚举名称(即默认自带的属性 name 的值)是：" + typeEnum.name());
23. System.out.println("排序值(默认自带的属性 ordinal 的值)是：" + typeEnum.ordinal());
24. System.out.println("枚举的自定义属性 typeName 的值是：" + typeEnum.getTypeName());
25. **break**;
26. **default**:
27. System.out.println("default");
28. }
29. }

然后，就是运行结果的截图。



老铁们，看完这个枚举，你要懂个概念，那就是，这个枚举，他是个对象，就像你定义的Student类，Person类，等等一些个类一样。

要有这么个概念。只要是个类，他就可以有构造函数，可以有属性，可以有方法。

对的，老铁，你对这个属性，构造函数啥的，有概念吧，没有的话，我可就郁闷啦。

然后，你就看到，这个地方有2个默认的属性，一个是name，一个是ordinal，这2个属性就像你定义Student类和Person类的name和age一样，

只不过，这2个是系统自带的属性，不用你自己去定义啦。

你也可以给这个枚举类，也就是你自己声明的枚举，随便加属性。

我上面代码例子里面的那个TypeEnum那个枚举，就是这么干的，就简单的添加了个自定义属性typeName，

虽然他有自己的name了，那姑且叫我这个自定义的属性叫别名吧。

可以看到，我例子里面就是通过自己写的那个构造方法给我这个自定义的属性初始化值的。

还有，这个构造方法是不可以，也不被运行public的，不信，你可以试试。

还有，你不能对系统自带的name属性，在构造函数里面赋值，没有为什么。