**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。

public enum Color {

RED, GREEN, BLANK, YELLOW

}

**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。

[IMG_256](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

enum Signal {

GREEN, YELLOW, RED

}

public class TrafficLight {

Signal color = Signal.RED;

public void change() {

switch (color) {

case RED:

color = Signal.GREEN;

break;

case YELLOW:

color = Signal.RED;

break;

case GREEN:

color = Signal.YELLOW;

break;

}

}

}

[IMG_257](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

**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。

[IMG_258](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

public enum Color {

RED("红色", 1), GREEN("绿色", 2), BLANK("白色", 3), YELLO("黄色", 4);

// 成员变量

private String name;

private int index;

// 构造方法

private Color(String name, int index) {

this.name = name;

this.index = index;

}

// 普通方法

public static String getName(int index) {

for (Color c : Color.values()) {

if (c.getIndex() == index) {

return c.name;

}

}

return null;

}

// get set 方法

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getIndex() {

return index;

}

public void setIndex(int index) {

this.index = index;

}

}

[IMG_259](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

**用法四：覆盖枚举的方法**

下面给出一个toString()方法覆盖的例子。

[IMG_260](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

public class Test {

public enum Color {

RED("红色", 1), GREEN("绿色", 2), BLANK("白色", 3), YELLO("黄色", 4);

// 成员变量

private String name;

private int index;

// 构造方法

private Color(String name, int index) {

this.name = name;

this.index = index;

}

// 覆盖方法 @Override

public String toString() {

return this.index + "\_" + this.name;

}

}

public static void main(String[] args) {

System.out.println(Color.RED.toString());

}

}

[IMG_261](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

**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。

[IMG_262](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

public interface Behaviour {

void print();

String getInfo();

}

public enum Color implements Behaviour {

RED("红色", 1), GREEN("绿色", 2), BLANK("白色", 3), YELLO("黄色", 4);

// 成员变量

private String name;

private int index;

// 构造方法

private Color(String name, int index) {

this.name = name;

this.index = index;

}

// 接口方法

@Override

public String getInfo() {

return this.name;

}

// 接口方法 @Override

public void print() {

System.out.println(this.index + ":" + this.name);

}

}

[IMG_263](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

**用法六：使用接口组织枚举**

[IMG_264](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

public interface Food {

enum Coffee implements Food {

BLACK\_COFFEE, DECAF\_COFFEE, LATTE, CAPPUCCINO

}

enum Dessert implements Food {

FRUIT, CAKE, GELATO

}

}

[IMG_265](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

**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

**枚举和常量定义的区别**

**一、 通常定义常量方法**

我们通常利用public final static方法定义的代码如下，分别用1表示红灯，3表示绿灯，2表示黄灯。

[IMG_266](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

public class Light {

/\* 红灯 \*/

public final static int RED = 1;

/\* 绿灯 \*/

public final static int GREEN = 3;

/\* 黄灯 \*/

public final static int YELLOW = 2;

}

[IMG_267](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

**二、 枚举类型定义常量方法**

枚举类型的简单定义方法如下，我们似乎没办法定义每个枚举类型的值。比如我们定义红灯、绿灯和黄灯的代码可能如下：

public enum Light {

RED, GREEN, YELLOW;

}

我们只能够表示出红灯、绿灯和黄灯，但是具体的值我们没办法表示出来。别急，既然枚举类型提供了构造函数，我们可以通过构造函数和覆写toString方法来实现。首先给Light枚举类型增加构造方法，然后每个枚举类型的值通过构造函数传入对应的参数，同时覆写toString方法，在该方法中返回从构造函数中传入的参数，改造后的代码如下：

[IMG_268](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

public enum Light {

// 利用构造函数传参

RED(1), GREEN(3), YELLOW(2);

// 定义私有变量

private int nCode;

// 构造函数，枚举类型只能为私有

private Light(int \_nCode) {

this.nCode = \_nCode;

}

@Override

public String toString() {

return String.valueOf(this.nCode);

}

}

[IMG_269](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

**三、 完整示例代码**

**枚举类型的完整演示代码如下：**

[IMG_270](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

public class LightTest {

// 1.定义枚举类型

public enum Light {

// 利用构造函数传参

RED(1), GREEN(3), YELLOW(2);

// 定义私有变量

private int nCode;

// 构造函数，枚举类型只能为私有

private Light(int \_nCode) {

this.nCode = \_nCode;

}

@Override

public String toString() {

return String.valueOf(this.nCode);

}

}

/\*\*

\*

\* @param args

\*/

public static void main(String[] args) {

// 1.遍历枚举类型

System.out.println("演示枚举类型的遍历 ......");

testTraversalEnum();

// 2.演示EnumMap对象的使用

System.out.println("演示EnmuMap对象的使用和遍历.....");

testEnumMap();

// 3.演示EnmuSet的使用

System.out.println("演示EnmuSet对象的使用和遍历.....");

testEnumSet();

}

/\*\*

\*

\* 演示枚举类型的遍历

\*/

private static void testTraversalEnum() {

Light[] allLight = Light.values();

for (Light aLight : allLight) {

System.out.println("当前灯name：" + aLight.name());

System.out.println("当前灯ordinal：" + aLight.ordinal());

System.out.println("当前灯：" + aLight);

}

}

/\*\*

\*

\* 演示EnumMap的使用，EnumMap跟HashMap的使用差不多，只不过key要是枚举类型

\*/

private static void testEnumMap() {

// 1.演示定义EnumMap对象，EnumMap对象的构造函数需要参数传入,默认是key的类的类型

EnumMap<Light, String> currEnumMap = new EnumMap<Light, String>(

Light.class);

currEnumMap.put(Light.RED, "红灯");

currEnumMap.put(Light.GREEN, "绿灯");

currEnumMap.put(Light.YELLOW, "黄灯");

// 2.遍历对象

for (Light aLight : Light.values()) {

System.out.println("[key=" + aLight.name() + ",value="

+ currEnumMap.get(aLight) + "]");

}

}

/\*\*

\*

\* 演示EnumSet如何使用，EnumSet是一个抽象类，获取一个类型的枚举类型内容<BR/>

\*

\* 可以使用allOf方法

\*/

private static void testEnumSet() {

EnumSet<Light> currEnumSet = EnumSet.allOf(Light.class);

for (Light aLightSetElement : currEnumSet) {

System.out.println("当前EnumSet中数据为：" + aLightSetElement);

}

}

}

[IMG_271](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

**执行结果如下：**

演示枚举类型的遍历 ......

当前灯name：RED

当前灯ordinal：0

当前灯：1

当前灯name：GREEN

当前灯ordinal：1

当前灯：3

当前灯name：YELLOW

当前灯ordinal：2

当前灯：2

演示EnmuMap对象的使用和遍历.....

[key=RED,value=红灯]

[key=GREEN,value=绿灯]

[key=YELLOW,value=黄灯]

演示EnmuSet对象的使用和遍历.....

当前EnumSet中数据为：1

当前EnumSet中数据为：3

当前EnumSet中数据为：2

**四、 通常定义常量方法和枚举定义常量方法区别**

以下内容可能有些无聊，但绝对值得一窥

1. 代码：

[IMG_272](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

public class State {

public static final int ON = 1;

public static final Int OFF= 0;

}

[IMG_273](http://www.cnblogs.com/happyPawpaw/archive/2013/04/09/javascript:void(0);)

有什么不好了，大家都这样用了很长时间了，没什么问题啊。

首先，它不是类型安全的。你必须确保是int

其次，你还要确保它的范围是0和1

最后，很多时候你打印出来的时候，你只看到 1 和0 ，

但其没有看到代码的人并不知道你的企图，抛弃你所有旧的public static final常量

2. 可以创建一个enum类，把它看做一个普通的类。除了它不能继承其他类了。(java是单继承，它已经继承了Enum),

可以添加其他方法，覆盖它本身的方法

3. switch()参数可以使用enum了

4. values()方法是编译器插入到enum定义中的static方法，所以，当你将enum实例向上转型为父类Enum是，values()就不可访问了。解决办法：在Class中有一个getEnumConstants()方法，所以即便Enum接口中没有values()方法，我们仍然可以通过Class对象取得所有的enum实例

5. 无法从enum继承子类，如果需要扩展enum中的元素，在一个接口的内部，创建实现该接口的枚举，以此将元素进行分组。达到将枚举元素进行分组。

6. 使用EnumSet代替标志。enum要求其成员都是唯一的，但是enum中不能删除添加元素。

7. EnumMap的key是enum，value是任何其他Object对象。

8. enum允许程序员为eunm实例编写方法。所以可以为每个enum实例赋予各自不同的行为。

9. 使用enum的职责链(Chain of Responsibility) .这个关系到设计模式的职责链模式。以多种不同的方法来解决一个问题。然后将他们链接在一起。当一个请求到来时，遍历这个链，直到链中的某个解决方案能够处理该请求。

10. 使用enum的状态机

11. 使用enum多路分发