# 枚举的基本使用

自定义类的形式以达到枚举的功能:(JDK 1.4以前的做法,1.5就有枚举了)

|  |
| --- |
| public class Main {  Color a = Color.GREEN;  }  class Color{  private Color() {//不能让用户随意构造  }  public static final Color GREEN = new Color();  public static final Color BLUE = new Color();  } |

使用枚举范例:

|  |
| --- |
| public class test{  public static void main(String args[]){  System.out.print(Color.GREEN);  System.out.print(Color.GREEN.getClass());  }  }  enum Color{  RED,GREEN,BLUE; //这些都是Color类的实例  } |

# Enum类(java.lang)

使用enum关键字定义枚举,相当于是一个类继承了Enum类

public abstract class Enum<E extends Enum<E>>

extends Object

implements Comparable<E>, Serializable

**构造函数**:protected Enum(String name, int ordinal)

**取得枚举的索引:**public final int ordinal()

**取得索引的名字:**public final String name()

取得枚举中所有的实例: 自定义枚举类型[] values()

这个方法是编译器插入的

范例:

|  |
| --- |
| public class test{  public static void main(String args[]){  for(Color c : Color.values()){  System.out.print(c.ordinal() + "-" + c.name());  }  }  }  enum Color{  RED,GREEN,BLUE;  private Color(){  }  } |

# 3.枚举的注意事项

1.枚举中定义的构造方法的访问修饰符不能是public(封装嘛,不然你实例化)

2.枚举对象要放在首行,随后才可以定义属性,构造,普通方法

3.在枚举(实例)后面加括号,里面传递构造函数的参数来初始化枚举对象

范例:

|  |
| --- |
| public class test{  public static void main(String args[]){  for(Color c : Color.values()){  System.out.print(c);  }  }  }  enum Color{  RED("红色"),GREEN("绿色"),BLUE("蓝色"); //最上面,构造传参数  private String title; //属性  private Color(String title){  this.title = title;  }  @Override  public String toString() {  return this.title;  }  } |

# 4.枚举实现接口

## 4.1每个对象实现同一个抽象方法

普通的是实现抽象方法

范例:

|  |
| --- |
| public class test{  public static void main(String args[]){  Message msg = Color.RED;  System.out.print(msg.getTitle());  }  }  enum Color implements Message{  RED("红色"),GREEN("绿色"),BLUE("蓝色"); //最上面,构造传参数  private String title; //属性  private Color(String title){  this.title = title;  }  @Override  public String getTitle() {  return this.title;  }  }  interface Message{  public String getTitle();  } |

## 4.2每个对象实现各自的抽象方法

|  |
| --- |
| public class test{  public static void main(String args[]){  Message msg = Color.RED;  Message msg1 = Color.GREEN;  Message msg2 = Color.BLUE;  System.out.println(msg.getTitle());  System.out.println(msg1.getTitle());  System.out.println(msg2.getTitle());  }  }  enum Color implements Message{  RED("红色"){  @Override  public String getTitle() {  return "自己的" + this.title;  }  },GREEN("绿色"){  @Override  public String getTitle() {  return "自己的" + this;  }  },BLUE("蓝色"){  @Override  public String getTitle() {  return "自己的" + this;  }  }; //最上面,构造传参数  private String title; //属性  private Color(String title){  this.title = title;  }  }  interface Message{  public String getTitle();  } |

# 5.枚举还可以定义抽象方法

此时每一个枚举对象必须覆写每一个抽象方法

|  |
| --- |
| public class test{  public static void main(String args[]){  System.out.print(Color.RED.getTitle());  }  }  enum Color{  RED("红色"){  @Override  public String getTitle() {  return "自己的" + this;  }  },GREEN("绿色"){  @Override  public String getTitle() {  return "自己的" + this;  }  },BLUE("蓝色"){  @Override  public String getTitle() {  return "自己的" + this;  }  }; //最上面,构造传参数  private String title; //属性  private Color(String title){  this.title = title;  }  @Override  public String toString() {  return "Color{" +  "title='" + title + '\'' +  '}';  }  public abstract String getTitle();  } |

# 6.switch能接受枚举

|  |
| --- |
| public class test{  public static void main(String args[]){  Color c = Color.RED;  switch (c){ //接受枚举类型  case RED:  System.out.print("这是红色");  break;  case GREEN:  System.out.print("这是绿色");  break;  case BLUE:  System.out.print("这是蓝色");  break;  }  }  }  enum Color{  RED,GREEN,BLUE;  } |