Talk is cheap, show me the code.

公告

Java 枚举

目录

Java 枚举

知识点

公告

对本博客样式有兴趣的朋友请参考：

详谈如何定制自己的博客园皮肤

关于博客样式的问题恕不答复。

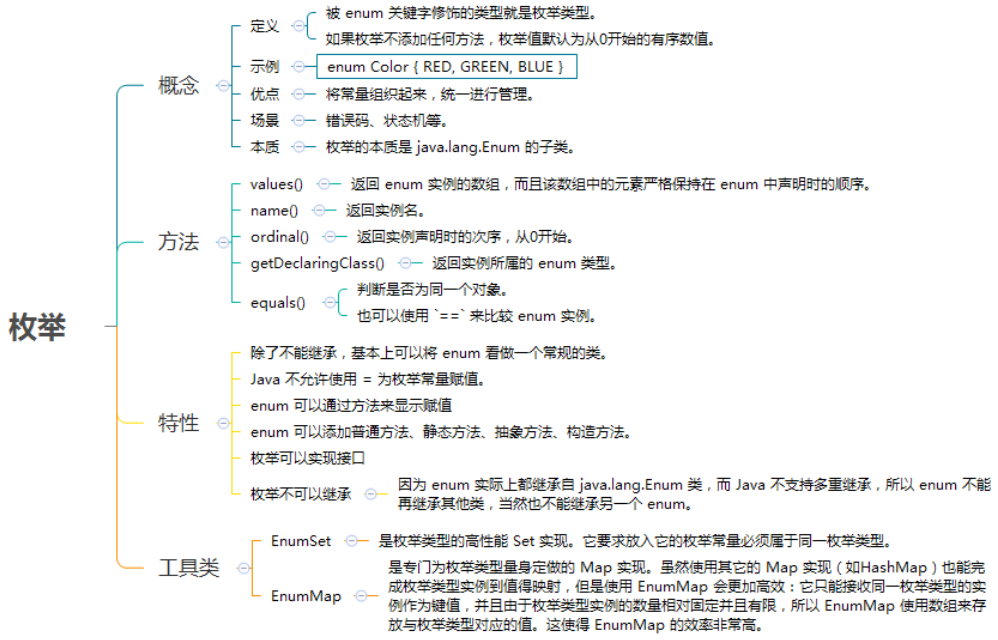
我的文章归档

我的简书

昵称： 静默虚空
园龄： 10年3个月
粉丝： 1844
关注： 10
+加关注

116 13

关注 | 顶部 | 评论



概念

enum 的全称为 enumeration， 是 JDK 1.5 中引入的新特性。

在Java中，被 enum 关键字修饰的类型就是枚举类型。形式如下：

```
enum Color { RED, GREEN, BLUE }
```

如果枚举不添加任何方法，枚举值默认为从0开始的有序数值。以 Color 枚举类型举例，它的枚举常量依次为 RED：0，GREEN：1，BLUE：2。

枚举的好处：可以将常量组织起来，统一进行管理。

枚举的典型应用场景：错误码、状态机等。

枚举类型的本质

尽管 enum 看起来像是一种新的数据类型，事实上，enum是一种受限制的类，并且具有自己的方法。

创建enum时，编译器会为你生成一个相关的类，这个类继承自 java.lang.Enum。

java.lang.Enum类声明

```
public abstract class Enum<E extends Enum<E>>
        implements Comparable<E>, Serializable { ... }
```

枚举的方法

在enum中，提供了一些基本方法：

法 1、选定一个增长量h, 按照增长量对数据分组
2、对划定的每一组(两两一组)进行 >插入排序 例如: 9 1 2 4 5 6 3 ...
--root-bine

3. Re:Java 内存模型
这文真好
--福禄小金刚

4. Re:排序六 堆排序
昨晚上发的言 楼主可以忽略了 现在明白了
--能借我十块钱吗

5. Re:排序六 堆排序
HeapAdjust（）不用while循环
--能借我十块钱吗

values(): 返回 enum 实例的数组，而且该数组中的元素严格保持在 enum 中声明时的顺序。

name(): 返回实例名。

ordinal(): 返回实例声明时的次序，从0开始。

getDeclaringClass(): 返回实例所属的 enum 类型。

equals() : 判断是否为同一个对象。

可以使用 == 来比较enum实例。

此外，java.lang.Enum实现了Comparable和 Serializable 接口，所以也提供 compareTo() 方法。

例：展示enum的基本方法

```
public class EnumMethodDemo {
   enum Color {RED, GREEN, BLUE;}
   enum Size {BIG, MIDDLE, SMALL;}
   public static void main(String args[]) {
      System.out.println("========== Print all Color ==========");
      for (Color c : Color.values()) {
         System.out.println(c + " ordinal: " + c.ordinal());
      }
      System.out.println("========== Print all Size ==========");
      for (Size s : Size.values()) {
         System.out.println(s + " ordinal: " + s.ordinal());
      }

      Color green = Color.GREEN;
      System.out.println("green name(): " + green.name());
      System.out.println("green getDeclaringClass(): " + green.getDeclaringClas
      System.out.println("green hashCode(): " + green.hashCode());
      System.out.println("green compareTo Color.GREEN: " + green.compareTo
      System.out.println("green equals Color.GREEN: " + green.equals(Color.GRE
      System.out.println("green equals Size.MIDDLE: " + green.equals(Size.MIDD
      System.out.println("green equals 1: " + green.equals(1));
      System.out.format("green == Color.BLUE: %b\n", green == Color.BLUE);
   }
}
```

输出

```
========== Print all Color ==========
RED ordinal: 0
GREEN ordinal: 1
BLUE ordinal: 2
========== Print all Size ==========
BIG ordinal: 0
```

116 13

关注 | 顶部 | 评论

```
MIDDLE ordinal: 1
SMALL ordinal: 2
green name(): GREEN
green getDeclaringClass(): class org.zp.javase.enumeration.EnumDemo$Color
green hashCode(): 460141958
green compareTo Color.GREEN: 0
green equals Color.GREEN: true
green equals Size.MIDDLE: false
green equals 1: false
green == Color.BLUE: false
```

枚举的特性

枚举的特性，归结起来就是一句话：

> 除了不能继承，基本上可以将 enum 看做一个常规的类。

但是这句话需要拆分去理解，让我们细细道来。

枚举可以添加方法

在概念章节提到了，枚举值默认为从0开始的有序数值 。那么问题来了：如何为枚举显示的赋值。

Java 不允许使用 = 为枚举常量赋值

如果你接触过C/C++，你肯定会很自然的想到赋值符号 =。在C/C++语言中的enum，可以用赋值符号=显示的为枚举常量赋值；但是，很遗憾，Java 语法中却不允许使用赋值符号 = 为枚举常量赋值。

例：C/C++ 语言中的枚举声明

```
typedef enum{
   ONE = 1,
   TWO,
   THREE = 3,
   TEN = 10
} Number;
```

枚举可以添加普通方法、静态方法、抽象方法、构造方法

Java 虽然不能直接为实例赋值，但是它有更优秀的解决方案：为 enum 添加方法来间接实现显示赋值。

创建 enum 时，可以为其添加多种方法，甚至可以为其

注意一个细节：如果要为enum定义方法，那么必须在e 加一个分号。此外，在enum中，必须先定义实例，不能将 。

否则，编译器会报错。

例：全面展示如何在枚举中定义普通方法、静态方法、抽象方法、构造方法

```
public enum ErrorCode {
    OK(0) {
        public String getDescription() {
            return "成功";
        }
    },
    ERROR_A(100) {
        public String getDescription() {
            return "错误A";
        }
    },
    ERROR_B(200) {
        public String getDescription() {
            return "错误B";
        }
    };

    private int code;

    // 构造方法：enum的构造方法只能被声明为private权限或不声明权限
    private ErrorCode(int number) { // 构造方法
        this.code = number;
    }
    public int getCode() { // 普通方法
        return code;
    } // 普通方法
    public abstract String getDescription(); // 抽象方法
    public static void main(String args[]) { // 静态方法
        for (ErrorCode s : ErrorCode.values()) {
            System.out.println("code: " + s.getCode() + ", description: " + s.getDesc
        }
    }
}
```

注：上面的例子并不可取，仅仅是为了展示枚举支持定义各种方法。下面是一个简化的例子

例：一个错误码枚举类型的定义

本例和上例的执行结果完全相同。

```
public enum ErrorCodeEn {
    OK(0, "成功"),
    ERROR_A(100, "错误A"),
    ERROR_B(200, "错误B");

    ErrorCodeEn(int number, String description) {
```

```
        this.code = number;
        this.description = description;
    }
    private int code;
    private String description;
    public int getCode() {
        return code;
    }
    public String getDescription() {
        return description;
    }
    public static void main(String args[]) { // 静态方法
        for (ErrorCodeEn s : ErrorCodeEn.values()) {
            System.out.println("code: " + s.getCode() + ", description: " + s.getDesc
        }
    }
}
```

枚举可以实现接口

enum 可以像一般类一样实现接口。

同样是实现上一节中的错误码枚举类，通过实现接口，可以约束它的方法。

```
public interface INumberEnum {
    int getCode();
    String getDescription();
}

public enum ErrorCodeEn2 implements INumberEnum {
    OK(0, "成功"),
    ERROR_A(100, "错误A"),
    ERROR_B(200, "错误B");

    ErrorCodeEn2(int number, String description) {
        this.code = number;
        this.description = description;
    }

    private int code;
    private String description;

    @Override
    public int getCode() {
        return code;
    }

    @Override
    public String getDescription() {
        return description;
```

```
    }
}
```

枚举不可以继承

enum 不可以继承另外一个类，当然，也不能继承另一个 enum 。

因为 enum 实际上都继承自 java.lang.Enum 类，而 Java 不支持多重继承，所以 enum 不能再继承其他类，当然也不能继承另一个 enum。

枚举的应用场景

组织常量

在JDK1.5 之前，在Java中定义常量都是public static final TYPE a;这样的形式。有了枚举，你可以将有关联关系的常量组织起来，使代码更加易读、安全，并且还可以使用枚举提供的方法。

枚举声明的格式

注：如果枚举中没有定义方法，也可以在最后一个实例后面加逗号、分号或什么都不加。

下面三种声明方式是等价的：

```
enum Color { RED, GREEN, BLUE }
enum Color { RED, GREEN, BLUE, }
enum Color { RED, GREEN, BLUE; }
```

switch 状态机

我们经常使用switch语句来写状态机。JDK7以后，switch已经支持 int、char、String、enum 类型的参数。这几种类型的参数比较起来，使用枚举的switch代码更具有可读性。

```
enum Signal {RED, YELLOW, GREEN}

public static String getTrafficInstruct(Signal signal) {
    String instruct = "信号灯故障";
    switch (signal) {
        case RED:
            instruct = "红灯停";
            break;
        case YELLOW:
            instruct = "黄灯请注意";
            break;
        case GREEN:
            instruct = "绿灯行";
            break;
        default:
```

```
            break;
        }
        return instruct;
    }
```

组织枚举

可以将类型相近的枚举通过接口或类组织起来。

但是一般用接口方式进行组织。

原因是：Java接口在编译时会自动为enum类型加上public static修饰符；Java类在编译时会自动为 enum 类型加上static修饰符。看出差异了吗？没错，就是说，在类中组织 enum，如果你不给它修饰为 public，那么只能在本包中进行访问。

例：在接口中组织 enum

```
public interface Plant {
    enum Vegetable implements INumberEnum {
        POTATO(0, "土豆"),
        TOMATO(0, "西红柿");

        Vegetable(int number, String description) {
            this.code = number;
            this.description = description;
        }

        private int code;
        private String description;

        @Override
        public int getCode() {
            return 0;
        }

        @Override
        public String getDescription() {
            return null;
        }
    }

    enum Fruit implements INumberEnum {
        APPLE(0, "苹果"),
        ORANGE(0, "桔子"),
        BANANA(0, "香蕉");

        Fruit(int number, String description) {
            this.code = number;
            this.description = description;
        }
```

```
        private int code;
        private String description;

        @Override
        public int getCode() {
            return 0;
        }

        @Override
        public String getDescription() {
            return null;
        }
    }
}
```

例：在类中组织 enum

本例和上例效果相同。

```
public class Plant2 {
    public enum Vegetable implements INumberEnum {...}  // 省略代码
    public enum Fruit implements INumberEnum {...} // 省略代码
}
```

策略枚举

EffectiveJava中展示了一种策略枚举。这种枚举通过枚举嵌套枚举的方式，将枚举常量分类处理。

这种做法虽然没有switch语句简洁，但是更加安全、灵活。

例：EffectvieJava中的策略枚举范例

```
enum PayrollDay {
    MONDAY(PayType.WEEKDAY), TUESDAY(PayType.WEEKDAY), WEDNESDAY(
            PayType.WEEKDAY), THURSDAY(PayType.WEEKDAY), FRIDAY(PayType.W
            PayType.WEEKEND), SUNDAY(PayType.WEEKEND);

    private final PayType payType;

    PayrollDay(PayType payType) {
        this.payType = payType;
    }

    double pay(double hoursWorked, double payRate) {
        return payType.pay(hoursWorked, payRate);
    }

    // 策略枚举
    private enum PayType {
        WEEKDAY {
```

```
        double overtimePay(double hours, double payRate) {
          return hours <= HOURS_PER_SHIFT ? 0 : (hours - HOURS_PER_SHIFT)
              * payRate / 2;
        }
      },
      WEEKEND {
        double overtimePay(double hours, double payRate) {
          return hours * payRate / 2;
        }
      };
      private static final int HOURS_PER_SHIFT = 8;

      abstract double overtimePay(double hrs, double payRate);

      double pay(double hoursWorked, double payRate) {
        double basePay = hoursWorked * payRate;
        return basePay + overtimePay(hoursWorked, payRate);
      }
    }
}
```

测试

```
System.out.println("时薪100的人在周五工作8小时的收入： " + PayrollDay.FRIDAY.pa
System.out.println("时薪100的人在周六工作8小时的收入： " + PayrollDay.SATURDA
```

EnumSet和EnumMap

Java 中提供了两个方便操作enum的工具类——EnumSet 和 EnumMap。

EnumSet 是枚举类型的高性能 Set 实现。它要求放入它的枚举常量必须属于同一枚举类型。
EnumMap 是专门为枚举类型量身定做的 Map 实现。虽然使用其它的 Map 实现（如 HashMap）也能完成枚举类型实例到值得映射，但是使用 EnumMap 会更加高效：它只能接收同一枚举类型的实例作为键值，并且由于枚举类型实例的数量相对固定并且有限，所以 EnumMap 使用数组来存放与枚举类型对应的值。这使得 EnumMap 的效率非常高。

```
// EnumSet的使用
System.out.println("EnumSet展示");
EnumSet<ErrorCodeEn> errSet = EnumSet.allOf(ErrorCodeEn.class);
for (ErrorCodeEn e : errSet) {
    System.out.println(e.name() + " : " + e.ordinal());
}

// EnumMap的使用
System.out.println("EnumMap展示");
EnumMap<StateMachine.Signal, String> errMap = new EnumMap(StateMachine
```

```
errMap.put(StateMachine.Signal.RED, "红灯");
errMap.put(StateMachine.Signal.YELLOW, "黄灯");
errMap.put(StateMachine.Signal.GREEN, "绿灯");
for (Iterator<Map.Entry<StateMachine.Signal, String>> iter = errMap.entrySet(
    Map.Entry<StateMachine.Signal, String> entry = iter.next();
    System.out.println(entry.getKey().name() + " : " + entry.getValue());
}
```

作者：静默虚空

欢迎任何形式的转载，但请务必注明出处。

限于本人水平，如果文章和代码有表述不当之处，还请不吝赐教。

分类： JavaCore

标签： javase , basics

静默虚空
关注 - 10
粉丝 - 1844
+加关注

« 上一篇： [译] MongoDB Java异步驱动快速指南
» 下一篇： npm 入门

posted @ 2016-11-24 16:30 静默虚空 阅读(103286) 评论(14) 编辑 收藏 举报