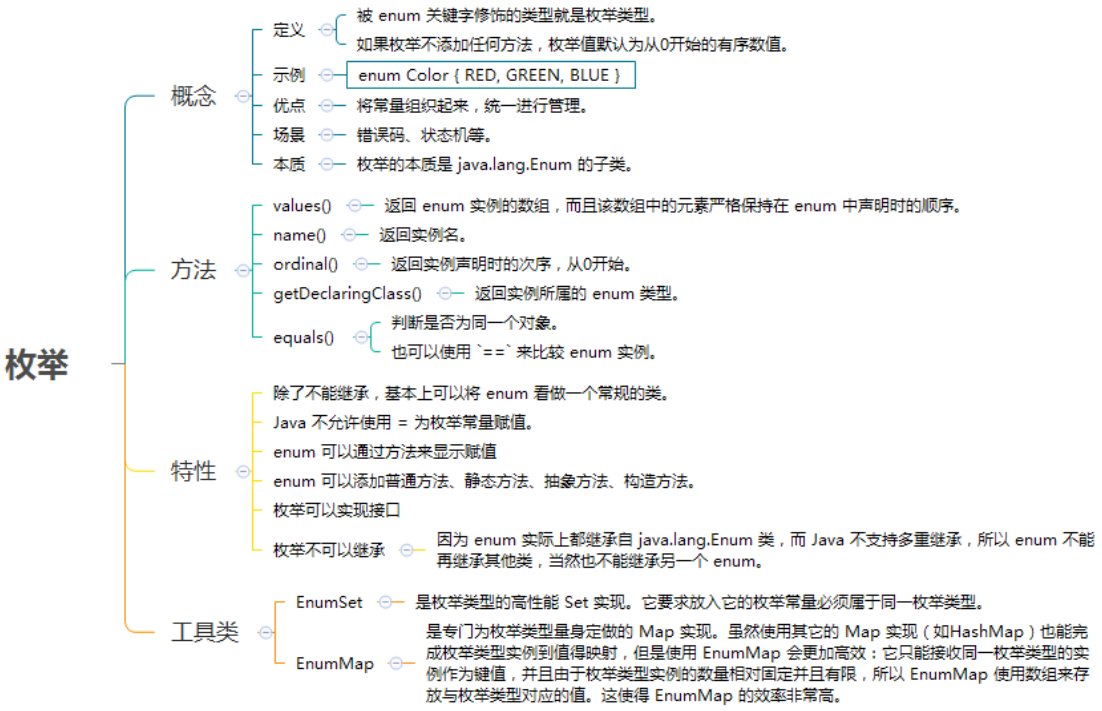
Java枚举

知识点



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：

```
public enum ColorEnum {
    RED, GREEN, BLUE
}
```

通过工具解析class后获得的源代码

```
public final class ColorEnum extends Enum
{

    //返回存储枚举实例的数组的副本。values()方法通常用于foreach循环遍历枚举常量。
    public static ColorEnum[] values()
    {
        return (ColorEnum[])$VALUES.clone();
    }
    //根据实例名获取实例
    public static ColorEnum valueOf(String s)
    {
        return (ColorEnum)Enum.valueOf(ColorEnum, s);
```

```
    }

    //私有构造方法，这里调用了父类的构造方法，其中参数s对应了常量名，参数i代表枚举的一个顺序(这个顺序与枚举的声明顺序对应，用于oridinal()方法返回顺序值)
    private ColorEnum(String s, int i)
    {
        super(s, i);
    }

    //我们定义的枚举在这里声明了三个 ColorEnum的常量对象引用,对象的实例化在static静态块中
    public static final ColorEnum RED;
    public static final ColorEnum BLUE;
    public static final ColorEnum GREEN;
    //将所有枚举的实例存放在数组中
    private static final ColorEnum $VALUES[];

    static
    {
        RED = new ColorEnum("RED", 0);
        BLUE = new ColorEnum("BLUE", 1);
        GREEN = new ColorEnum("GREEN", 2);
        //将所有枚举的实例存放在数组中
        $VALUES = (new ColorEnum[] {
            RED, BLUE, GREEN
        });
    }
}
```

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：

`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，如RED。

`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.getDeclaringClass());
        System.out.println("green hashCode(): " + green.hashCode());
        System.out.println("green compareTo Color.GREEN: " +
green.compareTo(Color.GREEN));
        System.out.println("green equals Color.GREEN: " +
green.equals(Color.GREEN));
        System.out.println("green equals Size.MIDDLE: " +
green.equals(Size.MIDDLE));
        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
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。**

注意一个细节：如果要为enum定义方法，那么必须在enum的最后一个实例尾部添加一个分号。此外，在enum中，必须先定义实例，不能将字段或方法定义在实例前面。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 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.getDescription());
        }
```

```
    }
}
```

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;
    }
}
```

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.Signal.class);
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().iterator(); iter.hasNext();) {
    Map.Entry<StateMachine.Signal, String> entry = iter.next();
    System.out.println(entry.getKey().name() + " : " + entry.getValue());
}
```