**全面掌握java枚举类型(enum type)**

# 背景

在java语言中还没有引入枚举类型之前，表示枚举类型的常用模式是声明一组具有int常量。之前我们通常利用public final static 方法定义的代码如下，分别用1 表示春天，2表示夏天，3表示秋天，4表示冬天。

**public** **class** Season {

**public** **static** **final** **int** *SPRING* = 1;

**public** **static** **final** **int** *SUMMER* = 2;

**public** **static** **final** **int** *AUTUMN* = 3;

**public** **static** **final** **int** *WINTER* = 4;

}

这种方法称作int枚举模式。可这种模式有什么问题呢，我们都用了那么久了，应该没问题的。通常我们写出来的代码都会考虑它的安全性、易用性和可读性。

**首先我们来考虑一下它的类型安全性**。当然这种模式不是类型安全的。比如说我们设计一个函数，要求传入春夏秋冬的某个值。但是使用int类型，我们无法保证传入的值为合法。代码如下所示：

**private** String getChineseSeason(**int** season){

StringBuffer result = **new** StringBuffer();

**switch**(season){

**case** Season.*SPRING* :

result.append("春天");

**break**;

**case** Season.*SUMMER* :

result.append("夏天");

**break**;

**case** Season.*AUTUMN* :

result.append("秋天");

**break**;

**case** Season.*WINTER* :

result.append("冬天");

**break**;

**default** :

result.append("地球没有的季节");

**break**;

}

**return** result.toString();

}

**public** **void** doSomething(){

System.*out*.println(**this**.getChineseSeason(Season*.SPRING*));//这是正常的场景

System.*out*.println(**this**.getChineseSeason(5));//这个却是不正常的场景，这就导致了类型不安全问题

}

程序getChineseSeason(Season*.SPRING*)是我们预期的使用方法。可getChineseSeason(5)显然就不是了，而且编译很通过，在运行时会出现什么情况，我们就不得而知了。这显然就不符合Java程序的类型安全。

**接下来我们来考虑一下这种模式的可读性**。使用枚举的大多数场合，我都需要方便得到枚举类型的字符串表达式。如果将int枚举常量打印出来，我们所见到的就是一组数字，这是没什么太大的用处。我们可能会想到使用String常量代替int常量。虽然它为这些常量提供了可打印的字符串，但是它会导致性能问题，因为它依赖于字符串的比较操作，所以这种模式也是我们不期望的。

从**类型安全性**和**程序可读性**两方面考虑，int和String枚举模式的缺点就显露出来了。幸运的是，从Java1.5发行版本开始，就提出了另一种可以替代的解决方案，可以避免int和String枚举模式的缺点，并提供了许多额外的好处。那就是枚举类型（enum type）。接下来的章节将介绍枚举类型的定义、特征、应用场景和优缺点。

# 定义

枚举类型（enum type）是指由 一组固定的常量 组成合法的类型。Java中由关键字enum来定义一个枚举类型。下面就是java枚举类型的定义。

**public**  **enum** Season {

*SPRING*, *SUMMER*, *AUTUMN*, *WINER*;

}

# 特点

Java定义枚举类型的语句很简约。它有以下特点：

1. 使用关键字enum
2. 类型名称，比如这里的Season
3. 一串允许的值，比如上面定义的春夏秋冬四季
4. 枚举可以单独定义在一个文件中，也可以嵌在其它Java类中

除了这样的基本要求外，用户还有一些其他选择

1. 枚举可以实现一个或多个接口（Interface）
2. 可以定义新的变量
3. 可以定义新的方法
4. 可以定义根据具体枚举值而相异的类

# 应用场景

以在背景中提到的类型安全为例，用枚举类型重写那段代码。代码如下：

**public** **enum** Season {

*SPRING*(1), *SUMMER*(2), *AUTUMN*(3), *WINTER*(4);

**private** **int** code;

**private** Season(**int** code){

**this**.code = code;

}

**public** **int** getCode(){

**return** code;

}

}

**public** **class** UseSeason {

/\*\*

\* 将英文的季节转换成中文季节

\* **@param** season

\* **@return**

\*/

**public** String getChineseSeason(Season season){

StringBuffer result = **new** StringBuffer();

**switch**(season){

**case** *SPRING* :

result.append("[中文：春天，枚举常量:" + season.name() + "，数据:" + season.getCode() + "]");

**break**;

**case** *AUTUMN* :

result.append("[中文：秋天，枚举常量:" + season.name() + "，数据:" + season.getCode() + "]");

**break**;

**case** *SUMMER* :

result.append("[中文：夏天，枚举常量:" + season.name() + "，数据:" + season.getCode() + "]");

**break**;

**case** *WINTER* :

result.append("[中文：冬天，枚举常量:" + season.name() + "，数据:" + season.getCode() + "]");

**break**;

**default** :

result.append("地球没有的季节 " + season.name());

**break**;

}

**return** result.toString();

}

**public** **void** doSomething(){

**for**(Season s : Season.*values*()){

System.*out*.println(getChineseSeason(s));//这是正常的场景

}

//System.out.println(getChineseSeason(5));

//此处已经是编译不通过了，这就保证了类型安全

}

**public** **static** **void** main(String[] arg){

UseSeason useSeason = **new** UseSeason();

useSeason.doSomething();

}

}

[中文：春天，枚举常量:SPRING，数据:1]

[中文：夏天，枚举常量:SUMMER，数据:2]

[中文：秋天，枚举常量:AUTUMN，数据:3]

[中文：冬天，枚举常量:WINTER，数据:4]

这里有一个问题，为什么我要将域添加到枚举类型中呢？目的是想将数据与它的常量关联起来。如1代表春天，2代表夏天。

# 总结

那么什么时候应该使用枚举呢？每当需要**一组固定的常量**的时候，如一周的天数、一年四季等。或者是在我们编译前就知道其包含的所有值的集合。Java 1.5的枚举能满足绝大部分程序员的要求的，它的简明，易用的特点是很突出的。