如何定义Enum类--代表TimeUnit

# 枚举类Enum定义方法

枚举类的枚举元素之间用**逗号**隔开，每个枚举类型名字后可以跟大括号{}，在大括号内部可以对枚举类定义的方法进行**重写**Override。

示例：**MyTimeUnit枚举类型**：



# 利用枚举类Enum实现单例模式

使用Enum实现**单例模式**是最简单的方法。

public enum SingletonByEnum {

**INSTANCE;//**枚举类型定义单例十分简单，只需要定义一个枚举元素就可以

//其他方法

private String name;

public void setName(String name){

this.name = name;

}

public String getName(){

return name;

}

}

这种方式是Effective Java作者Josh Bloch 提倡的方式，它**不仅能避免多线程同步问题，而且还能防止反序列化重新创建新的对象，可谓是很坚强的壁垒啊**，在深度分析Java的枚举类型----枚举的线程安全性及序列化问题中有详细介绍枚举的线程安全问题和序列化问题，不过，个人认为由于1.5中才加入enum特性，用这种方式写不免让人感觉生疏，在实际工作中，我也很少看见有人这么写过。

# TimeUnit枚举类

**TimeUnit枚举类**中有7个枚举元素，枚举类内部定义了很多方法，特别是toXxx转换方法，这些**方法体内部**大多数都是通过**throw new AbstractMethodError();抽象实现；目的就是让各个枚举元素自己去实现这些方法。**

源代码分析：

package java.util.concurrent;

public enum **TimeUnit** {

**NANOSECONDS** {

public long toNanos(long d) { return d; }

public long toMicros(long d) { return d/(C1/C0); }

public long toMillis(long d) { return d/(C2/C0); }

public long toSeconds(long d) { return d/(C3/C0); }

public long toMinutes(long d) { return d/(C4/C0); }

public long toHours(long d) { return d/(C5/C0); }

public long toDays(long d) { return d/(C6/C0); }

public long convert(long d, TimeUnit u) { return u.toNanos(d); }

int excessNanos(long d, long m) { return (int)(d - (m\*C2)); }

},

**MICROSECONDS** {

public long toNanos(long d) { return x(d, C1/C0, MAX/(C1/C0)); }

public long toMicros(long d) { return d; }

public long toMillis(long d) { return d/(C2/C1); }

public long toSeconds(long d) { return d/(C3/C1); }

public long toMinutes(long d) { return d/(C4/C1); }

public long toHours(long d) { return d/(C5/C1); }

public long toDays(long d) { return d/(C6/C1); }

public long convert(long d, TimeUnit u) { return u.toMicros(d); }

int excessNanos(long d, long m) { return (int)((d\*C1) - (m\*C2)); }

},

**MILLISECONDS** {

public long toNanos(long d) { return x(d, C2/C0, MAX/(C2/C0)); }

public long toMicros(long d) { return x(d, C2/C1, MAX/(C2/C1)); }

public long toMillis(long d) { return d; }

public long toSeconds(long d) { return d/(C3/C2); }

public long toMinutes(long d) { return d/(C4/C2); }

public long toHours(long d) { return d/(C5/C2); }

public long toDays(long d) { return d/(C6/C2); }

public long convert(long d, TimeUnit u) { return u.toMillis(d); }

int excessNanos(long d, long m) { return 0; }

},

**SECONDS** {

public long toNanos(long d) { return x(d, C3/C0, MAX/(C3/C0)); }

public long toMicros(long d) { return x(d, C3/C1, MAX/(C3/C1)); }

public long toMillis(long d) { return x(d, C3/C2, MAX/(C3/C2)); }

public long toSeconds(long d) { return d; }

public long toMinutes(long d) { return d/(C4/C3); }

public long toHours(long d) { return d/(C5/C3); }

public long toDays(long d) { return d/(C6/C3); }

public long convert(long d, TimeUnit u) { return u.toSeconds(d); }

int excessNanos(long d, long m) { return 0; }

},

**MINUTES** {

public long toNanos(long d) { return x(d, C4/C0, MAX/(C4/C0)); }

public long toMicros(long d) { return x(d, C4/C1, MAX/(C4/C1)); }

public long toMillis(long d) { return x(d, C4/C2, MAX/(C4/C2)); }

public long toSeconds(long d) { return x(d, C4/C3, MAX/(C4/C3)); }

public long toMinutes(long d) { return d; }

public long toHours(long d) { return d/(C5/C4); }

public long toDays(long d) { return d/(C6/C4); }

public long convert(long d, TimeUnit u) { return u.toMinutes(d); }

int excessNanos(long d, long m) { return 0; }

},

**HOURS** {

public long toNanos(long d) { return x(d, C5/C0, MAX/(C5/C0)); }

public long toMicros(long d) { return x(d, C5/C1, MAX/(C5/C1)); }

public long toMillis(long d) { return x(d, C5/C2, MAX/(C5/C2)); }

public long toSeconds(long d) { return x(d, C5/C3, MAX/(C5/C3)); }

public long toMinutes(long d) { return x(d, C5/C4, MAX/(C5/C4)); }

public long toHours(long d) { return d; }

public long toDays(long d) { return d/(C6/C5); }

public long convert(long d, TimeUnit u) { return u.toHours(d); }

int excessNanos(long d, long m) { return 0; }

},

**DAYS** {

public long toNanos(long d) { return x(d, C6/C0, MAX/(C6/C0)); }

public long toMicros(long d) { return x(d, C6/C1, MAX/(C6/C1)); }

public long toMillis(long d) { return x(d, C6/C2, MAX/(C6/C2)); }

public long toSeconds(long d) { return x(d, C6/C3, MAX/(C6/C3)); }

public long toMinutes(long d) { return x(d, C6/C4, MAX/(C6/C4)); }

public long toHours(long d) { return x(d, C6/C5, MAX/(C6/C5)); }

public long toDays(long d) { return d; }

public long convert(long d, TimeUnit u) { return u.toDays(d); }

int excessNanos(long d, long m) { return 0; }

};

// Handy constants for conversion methods

static final long C0 = 1L;

static final long C1 = C0 \* 1000L;

static final long C2 = C1 \* 1000L;

static final long C3 = C2 \* 1000L;

static final long C4 = C3 \* 60L;

static final long C5 = C4 \* 60L;

static final long C6 = C5 \* 24L;

static final long MAX = Long.MAX\_VALUE;

static long x(long d, long m, long over) {

if (d > over) return Long.MAX\_VALUE;

if (d < -over) return Long.MIN\_VALUE;

return d \* m;

}

// To maintain full signature compatibility with 1.5, and to improve the

// clarity of the generated javadoc (see 6287639: Abstract methods in

// enum classes should not be listed as abstract), method convert

// etc. are not declared abstract but otherwise act as abstract methods.

public long convert(long sourceDuration, TimeUnit sourceUnit) {

throw new AbstractMethodError();

}

public long toNanos(long duration) { **throw new AbstractMethodError();}**

public long toMicros(long duration) {throw new AbstractMethodError();}

public long toMillis(long duration) {throw new AbstractMethodError();}

public long toSeconds(long duration) {throw new AbstractMethodError();}

public long toMinutes(long duration) { throw new AbstractMethodError(); }

public long toHours(long duration) { throw new AbstractMethodError();}

public long toDays(long duration) {throw new AbstractMethodError(); }

}