TimeUnit枚举类源码解读-如何定义一个枚举类

# 枚举类Enum定义方法

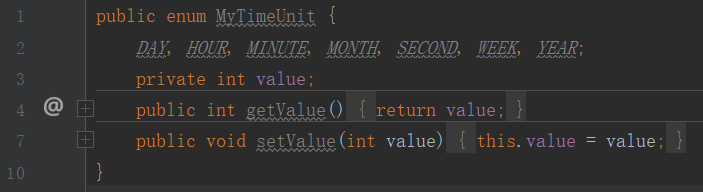
## 如何写一个Enum枚举类？

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{}，**最后一个枚举元素以分号结尾；在枚举元素之后可以定义一些共享的属性以及方法；

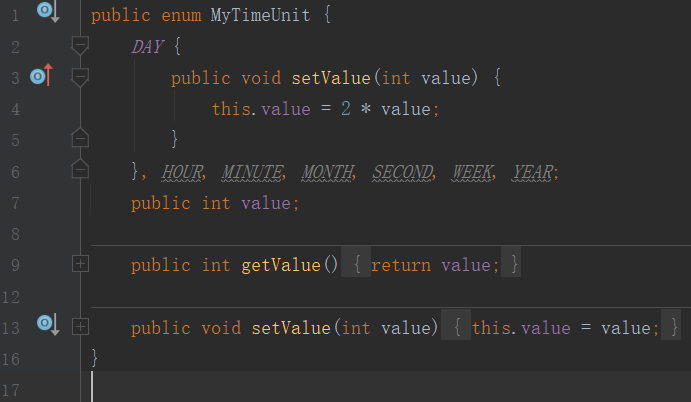
每个枚举元素后跟大括号，可以在在大括号内部可以对枚举类定义的共享属性和方法进行**重写**Override。

## 枚举类MyTimeUnit的示例

### 示例1



### 示例2



### 示例3



## 枚举类Enum的好处

有待于研究。

## 枚举类Enum的应用

实现单例；

在并发编程中，**TimeUnit**时间单位工具类；

# 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类介绍

## 包及继承实现体系

public enum TimeUnit **extends Enum**<**TimeUnit**>

**TimeUnit类**存在于**java.util.concurrent包**中，属于并发包中的一个类，直接继承java.lang.Enum<TimeUnit>（枚举）。TimeUnit没有构造方法，只能通过枚举元素获取**TimeUnit**对象。一共有七种TimeUnit时间单位对象：**天、时、分、秒、毫秒、微妙、纳秒**

**java.lang.Object**

**java.lang.Enum<TimeUnit>**

**java.util.concurrent.TimeUnit**

All Implemented Interfaces:

**Serializable, Comparable<TimeUnit>**

**Enum枚举类：**

public abstract class Enum<E extends Enum<E>> extends **Object** implements **Comparable<E>, Serializable**

其中，java.util.concurrent包详见单独文件介绍。

## TimeUnit类的描述

A **TimeUnit** represents **time durations** at a given unit of granularity and provides utility methods to convert across units, and to perform timing and delay operations in these units. **A TimeUnit does not maintain time information, but only helps organize and use time representations that may be maintained separately across various contexts.**

## 使用示例

A TimeUnit is mainly used to **inform time-based methods** how a given timing parameter should be interpreted. For example, the following code will timeout in 50 milliseconds if the lock is not available:

Lock lock = ...;

if (lock.tryLock(50L, **TimeUnit.MILLISECONDS**)) ...

while this code will timeout in 50 seconds:

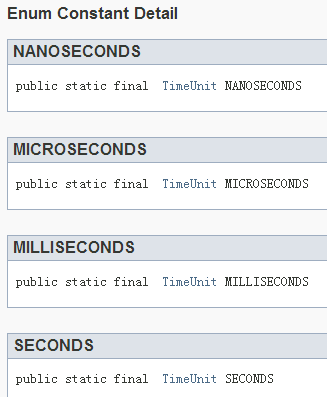
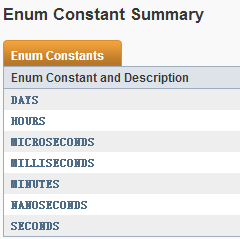
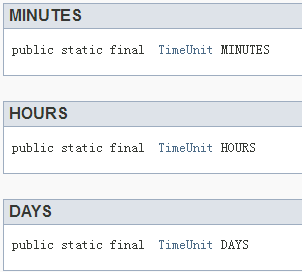
Lock lock = ...;

if (lock.tryLock(50L, **TimeUnit.SECONDS**)) ...

**Note** however, that **there is no guarantee that** a particular timeout implementation will be able to notice the passage of time at the same granularity as the given TimeUnit.

## Enum Constants

一共七个时间单位元素：**天、时、分、秒、毫秒、微妙、纳秒**

## 方法介绍

### **convert(long l,TimeUnit tu)**

**long** convert(long sourceDuration, TimeUnit sourceUnit)

功能：Convert **the given time duration in the given unit** to this unit.

参数:sourceDuration表示给定时间长度，sourceUnit表示给定时间单位，将给定的时间转换为当前时间单位下的时间长度。如：

TimeUnit.SECONDS.convert(8000,TimeUnit.MILLISECONDS);//8

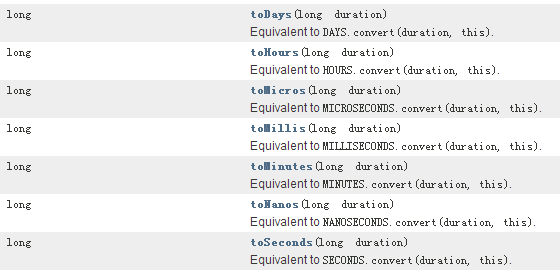
TimeUnit.DAYS.convert(20,TimeUnit.HOURS);//0

TimeUnit.MINUTES.convert(59,TimeUnit.SECONDS);//0

注意：**返回类型为long，只返回结果值的整数部分，并不是四舍五入**。

### toXxx(long duration)

一共7个时间单位，所以有7个toXxx(long duration)方法，其功能相当于**convert(long duration,this)**。



### sleep(long timeout)：睡眠方法，时间单位为当前TimeUnit的类型。

void **sleep(long timeout)**

Performs a Thread.sleep using this time unit.

### Values()、valueOf(String name):两个静态方法。

static TimeUnit **valueOf**(**String name**)

Returns the enum constant of this type with the specified name.

static TimeUnit[] **values()**

Returns an array containing the constants of this enum type, in the order they are declared.

示例：TimeUnit.valueOf("HOURS").toDays(24);//1

TimeUnit.values().length;//7

### **timedJoin(Thread thread, long timeout)**

void **timedJoin**(**Thread thread**, long timeout)

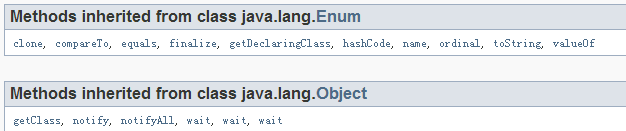
Performs a timed **Thread.join** using this time unit.

### **timedWait(Object obj, long timeout)**

void **timedWait**(Object obj, long timeout)

Performs a timed **Object.wait** using this time unit.

### 其他继承的方法



## 重要的方法介绍

### **timedJoin：等待一定时间，调用线程的join方法**

public void **timedJoin(Thread thread, long timeout)** throws InterruptedException

Performs a timed **Thread.join** using this time unit. This is a convenience method that converts time arguments into the form required by the **Thread.join** method.

Parameters: thread - the thread to wait for

timeout - the maximum time to wait. If less than or equal to zero, do not wait at all.

Throws: InterruptedException - if interrupted while waiting

### **timedWait**:

public void **timedWait(Object obj,long timeout)** throws InterruptedException

Performs a timed **Object.wait** using this time unit. This is a convenience method that converts timeout arguments into the form required by the **Object.wait** method.

For example, you could implement a blocking poll method (see BlockingQueue.poll) using:

public synchronized Object poll(long timeout, TimeUnit unit)

throws InterruptedException {

while (empty) {

unit.timedWait(this, timeout);

...

}

}

**Parameters**: **obj** - the object to wait on

**timeout** - the maximum time to wait. If less than or equal to zero, do not wait at all.

Throws: InterruptedException - if interrupted while waiting.

# TimeUnit枚举类-源码解读

**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。**

7个枚举元素为日、时分秒、毫秒、微妙、纳秒。注意没有定义年月。

源码解读：

在TimeUnit中定义7个枚举元素，定义了一些常量值；然后定义了7个toXxx方法和一个convert方法，只不过内部都是throw异常实现，在7个枚举元素内部，重写了这7个方法。toXxx方法是该时间单位转换成其他时间单位；而convert是其他时间单位转换成该时间单位。

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(); }

}