写在前面：Java SE5提供了一种新的类型-[Java的枚举类型](https://www.hollischuang.com/archives/195)，关键字enum可以将一组具名的值的有限集合创建为一种新的类型，而这些具名的值可以作为常规的程序组件使用，这是一种非常有用的功能。本文将深入分析枚举的源码，看一看枚举是怎么实现的，他是如何保证线程安全的，以及为什么用枚举实现的单例是最好的方式。

**枚举是如何保证线程安全的**

要想看源码，首先得有一个类吧，那么枚举类型到底是什么类呢？是enum吗？答案很明显不是，enum就和class一样，只是一个关键字，他并不是一个类，那么枚举是由什么类维护的呢，我们简单的写一个枚举：

public enum t {

SPRING,SUMMER,AUTUMN,WINTER;

}

然后我们使用反编译，看看这段代码到底是怎么实现的，反编译（[Java的反编译](https://www.hollischuang.com/archives/58)）后代码内容如下：

public final class T extends Enum

{

private T(String s, int i)

{

super(s, i);

}

public static T[] values()

{

T at[];

int i;

T at1[];

System.arraycopy(at = ENUM$VALUES, 0, at1 = new T[i = at.length], 0, i);

return at1;

}

public static T valueOf(String s)

{

return (T)Enum.valueOf(demo/T, s);

}

public static final T SPRING;

public static final T SUMMER;

public static final T AUTUMN;

public static final T WINTER;

private static final T ENUM$VALUES[];

static

{

SPRING = new T("SPRING", 0);

SUMMER = new T("SUMMER", 1);

AUTUMN = new T("AUTUMN", 2);

WINTER = new T("WINTER", 3);

ENUM$VALUES = (new T[] {

SPRING, SUMMER, AUTUMN, WINTER

});

}

}

通过反编译后代码我们可以看到，public final class T extends Enum，说明，该类是继承了Enum类的，同时final关键字告诉我们，这个类也是不能被继承的。当我们使用enmu来定义一个枚举类型的时候，编译器会自动帮我们创建一个final类型的类继承Enum类,所以枚举类型不能被继承，我们看到这个类中有几个属性和方法。

我们可以看到：

public static final T SPRING;

public static final T SUMMER;

public static final T AUTUMN;

public static final T WINTER;

private static final T ENUM$VALUES[];

static

{

SPRING = new T("SPRING", 0);

SUMMER = new T("SUMMER", 1);

AUTUMN = new T("AUTUMN", 2);

WINTER = new T("WINTER", 3);

ENUM$VALUES = (new T[] {

SPRING, SUMMER, AUTUMN, WINTER

});

}

都是static类型的，因为static类型的属性会在类被加载之后被初始化，我们在[深度分析Java的ClassLoader机制（源码级别）](https://www.hollischuang.com/archives/199)和[Java类的加载、链接和初始化](https://www.hollischuang.com/archives/201)两个文章中分别介绍过，当一个Java类第一次被真正使用到的时候静态资源被初始化、Java类的加载和初始化过程都是线程安全的。所以，**创建一个enum类型是线程安全的**。

**为什么用枚举实现的单例是最好的方式**

在[[转+注]单例模式的七种写法](https://www.hollischuang.com/archives/205)中，我们看到一共有七种实现单例的方式，其中，**Effective Java**作者Josh Bloch 提倡使用枚举的方式，既然大神说这种方式好，那我们就要知道它为什么好？

**1. 枚举写法简单**

写法简单这个大家看看[转+注]单例模式的七种写法里面的实现就知道区别了。

public enum EasySingleton{

INSTANCE;

}

你可以通过EasySingleton.INSTANCE来访问。

**2. 枚举自己处理序列化**

我们知道，以前的所有的单例模式都有一个比较大的问题，就是一旦实现了Serializable接口之后，就不再是单例得了，因为，每次调用 readObject()方法返回的都是一个新创建出来的对象，有一种解决办法就是使用readResolve()方法来避免此事发生。但是，**为了保证枚举类型像Java规范中所说的那样，每一个枚举类型极其定义的枚举变量在JVM中都是唯一的，在枚举类型的序列化和反序列化上，Java做了特殊的规定。**原文如下：

Enum constants are serialized differently than ordinary serializable or externalizable objects. The serialized form of an enum constant consists solely of its name; field values of the constant are not present in the form. To serialize an enum constant, ObjectOutputStream writes the value returned by the enum constant’s name method. To deserialize an enum constant, ObjectInputStream reads the constant name from the stream; the deserialized constant is then obtained by calling the java.lang.Enum.valueOf method, passing the constant’s enum type along with the received constant name as arguments. Like other serializable or externalizable objects, enum constants can function as the targets of back references appearing subsequently in the serialization stream. The process by which enum constants are serialized cannot be customized: any class-specific writeObject, readObject, readObjectNoData, writeReplace, and readResolve methods defined by enum types are ignored during serialization and deserialization. Similarly, any serialPersistentFields or serialVersionUID field declarations are also ignored–all enum types have a fixedserialVersionUID of 0L. Documenting serializable fields and data for enum types is unnecessary, since there is no variation in the type of data sent.

大概意思就是说，在序列化的时候Java仅仅是将枚举对象的name属性输出到结果中，反序列化的时候则是通过java.lang.Enum的valueOf方法来根据名字查找枚举对象。同时，编译器是不允许任何对这种序列化机制的定制的，因此禁用了writeObject、readObject、readObjectNoData、writeReplace和readResolve等方法。 我们看一下这个valueOf方法：

public static <T extends Enum<T>> T valueOf(Class<T> enumType,String name) {

T result = enumType.enumConstantDirectory().get(name);

if (result != null)

return result;

if (name == null)

throw new NullPointerException("Name is null");

throw new IllegalArgumentException(

"No enum const " + enumType +"." + name);

}

从代码中可以看到，代码会尝试从调用enumType这个Class对象的enumConstantDirectory()方法返回的map中获取名字为name的枚举对象，如果不存在就会抛出异常。再进一步跟到enumConstantDirectory()方法，就会发现到最后会以反射的方式调用enumType这个类型的values()静态方法，也就是上面我们看到的编译器为我们创建的那个方法，然后用返回结果填充enumType这个Class对象中的enumConstantDirectory属性。

所以，**JVM对序列化有保证。**

**3.枚举实例创建是thread-safe(线程安全的)**

我们在[深度分析Java的ClassLoader机制（源码级别）](https://www.hollischuang.com/archives/199)和[Java类的加载、链接和初始化](https://www.hollischuang.com/archives/201)两个文章中分别介绍过，当一个Java类第一次被真正使用到的时候静态资源被初始化、Java类的加载和初始化过程都是线程安全的。所以，**创建一个enum类型是线程安全的**。

（全文完）