链滴

单例模式详解 -- Java 版

作者：zhengliwei

原文链接：https://ld246.com/article/1601118169580

来源网站：链滴

hello，大家好，欢迎来到银之庭。我是Z，一个普通的程序员。今天我们来以Java语言为例，聊聊单模式。

1. 单例模式简介

单例模式的主要应用场景是希望整个程序进程中某个类只存在一个实例，所有用到这个类的地方都只用这一个实例。这样做的目的主要是节省内存占用和资源频繁创建销毁带来的性能损失。维护创建开比较大，同时使用又很频繁的对象时很常用，比如维护数据库连接池时。在实现方式上，可以分为懒式和饿汉式，懒汉式又可以细分为双重校验锁，枚举类，静态内部类三种不同的实现。

2. 饿汉式

先说比较简单的饿汉式， 即提前创建出实例对象，在后续调用时直接返回该对象。提前一般是指类加时，这时候由classloader保证并发安全，可以防止并发创建实例的问题，缺点是提前创建对象，如果续用不到的话可能导致资源浪费。这种实现方式比较简单可靠，生产环境中大多使用这种方法。代码例如下：

```
public class Singleton {
    private static final Singleton instance = new Singleton(); // 1

    private Singleton() { // 2
    }

    public static Singleton getInstance() { // 3
        return instance;
    }

    public void doSomething() { // 4
        System.out.println("hello");
    }

}
```

1. 单一的实例对象声明为类变量，在类加载时就会初始化，可以防止并发创建问题。
2. 构造方法声明为私有的，防止外部创建实例。
3. 提供一个静态方法返回单例对象。
4. 单例对象的业务逻辑方法。

当然，上面创建实例的代码也可以放到静态代码块里，效果和直接创建一样，不再赘述，代码示例如：

```
public class Singleton {
    private static final Singleton instance; // 1

    static {
        instance = new Singleton();
    }

    private Singleton() { // 2
    }
```

```
    public static Singleton getInstance() { // 3
        return instance;
    }

    public void doSomething() { // 4
        System.out.println("hello");
    }

}
```

3. 懒汉式

懒汉式的实现在面试中用的比较多，因为懒汉式涉及到的知识点比较多，可以和面试官扯上一段时间。

3.1 双重校验锁实现

懒汉式比较出名的实现就是双重校验锁了，代码示例如下：

```
public class Singleton {
    private static volatile Singleton instance; // 1

    private Singleton() {
    }

    public static Singleton getInstance() {
        if (instance == null) { // 2
            synchronized (Singleton.class) { // 3
                if (instance == null) { // 4
                    instance = new Singleton();
                }
            }
        }

        return instance;
    }

    public void doSomething() {
        System.out.println("hello");
    }

}
```

1. 实例对象声明为volatile的，防止指令重排序导致的实例未初始化完成就返回，进而导致使用方拿未初始化完成的对象而报错。关于volatile关键字，细讲起来就要涉及到指令重排序，Java内存模型东西了，后面我会专门写篇文章来讲这部分东西。

2. 第一重校验，也是程序运行中绝大多数时候会被拦截的地方，一旦实例创建完成，会直接返回，不创建，实现了单例模式的定义。

3. 加锁，和下面的第二重检验一起防止多个线程并发调用getInstance，都判断instance为空进而重创建实例。

4. 第二重校验，当有多个线程都通过第一重校验，尝试创建实例时，如果不在synchronized包裹的

界区内再判断一下instance是否存在的话，可能导致第二个及之后的线程重复执行创建逻辑，浪费资
。

3.2 枚举类实现

这种实现也是《Effective Java》推荐的实现方式，利用枚举类的特性保证创建实例的并发安全，代码现也比较简单，只是不太直观，目前为止我在生产环境见到的比较少。代码示例如下：

```
public enum SingletonEnum {

    INSTANCE; // 1

    public void doSomething() { // 2
        System.out.println("hello");
    }

    public static void main(String[] args) {
        SingletonEnum instance = SingletonEnum.INSTANCE;
        instance.doSomething();
    }
}
```

1. 定义一个枚举对象。
2. 声明枚举对象的业务逻辑方法。

3.3 静态内部类实现

这种方式就更少见，基本属于奇技淫巧的范畴了。这种方式可以像饿汉式一样利用classloader保证并安全，同时实现懒加载的效果，代码如下：

```
public class Singleton {
    private static final class InstanceHolder {
        private static final Singleton instance = new Singleton();
    }

    private Singleton() {
    }

    public static Singleton getInstance() {
        return InstanceHolder.instance;
    }

    public void doSomething() {
        System.out.println("hello");
    }

}
```

这种方式通过定义一个静态内部类InstanceHolder保存单例对象，来保证Singleton类加载时不需要

始化实例，只有在调用getInstance方法时，才会使用InstanceHolder类，进而加载InstanceHolder，此时才会初始化实例。

扩展阅读

推荐一篇在单例模式方面讲的比较完美的一篇文章：菜鸟教程关于单例模式的讲解

以上，感谢大家花费生命中宝贵的几分钟看我的自言自语~
um