链滴

设计模式（单例模式）- 如何设计一个启动器

作者：Pleuvoir
原文链接：https://ld246.com/article/1522494408950
来源网站：链滴

单例模式的理论就不再描述了 以下对自己喜欢的单例模式写法做个总结说明：

常见实现

常见的单例模式写法有多种，甄选之后比较喜欢如下两种：

- 静态内部类
- 枚举

静态内部类

来一段静态内部类实现spring容器启动器的代码，如何风骚的初始化spring容器。

```
public class Bootstrap {

    private static final Logger logger = LoggerFactory.getLogger(Bootstrap.class);

    private Bootstrap(){}

    private static class LoaderHelper {

        private static final AnnotationConfigApplicationContext CONTEXT;

        static {
            CONTEXT = initContext();
        }

        //初始化spring容器
```

```
        private static AnnotationConfigApplicationContext initContext() {
            logger.info("开始初始化spring容器。");
            long startTime = Clock.startTime();
            AnnotationConfigApplicationContext context = new AnnotationConfigApplicationCon
ext(TookitConfig.class);
            context.registerShutdownHook();
            logger.info("初始化spring容器完成。耗时：{}", Clock.passed(startTime));
            return context;
        }
    }


    /**
     * 获取spring容器
     * @return
     */
    public static AnnotationConfigApplicationContext getApplicationContext(){
        return LoaderHelper.CONTEXT;
    }


    /**
     * 获取spring中的bean
     * @param beanName
     * @return
     */
    @SuppressWarnings("unchecked")
    public static <T> T getBean(String beanName) {
        return (T) LoaderHelper.CONTEXT.getBean(beanName);
    }


    /**
     * 获取spring中的bean
     * @param clazz
     * @return
     */
    public static <T> T getBean(Class<T> clazz) {
        return LoaderHelper.CONTEXT.getBean(clazz);
    }

}
```

JVM保证了静态内部类被加载时是线程安全的，并且static代码块只会被初始化一次，当该类被初始时，只有调用外部类提供的静态方法时，才会加载内部类，而这些静态方法恰恰需要初始化spring容后调用才有意义，所以这种内部类单例的实现模式可以做到延迟加载。

使用枚举的构造方法实现

```
public enum BootstrapEnum {

    INSTANCE;
```

```
    private static final Logger logger = LoggerFactory.getLogger(BootstrapEnum.class);

    private AnnotationConfigApplicationContext CONTEXT;


    BootstrapEnum() {
        CONTEXT = initContext();
    }


    private AnnotationConfigApplicationContext initContext() {
        logger.info("开始初始化spring容器。");
        long startTime = Clock.startTime();
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContex
(TookitConfig.class);
        context.registerShutdownHook();
        logger.info("初始化spring容器完成。耗时：{}", Clock.passed(startTime));
        return context;
    }


    /**
     * 获取spring容器
     * @return
     */
    public AnnotationConfigApplicationContext getApplicationContext(){
        return CONTEXT;
    }

    /**
     * 获取spring中的bean
     * @param beanName
     * @return
     */
    @SuppressWarnings("unchecked")
    public <T> T getBean(String beanName) {
        return (T) CONTEXT.getBean(beanName);
    }


    /**
     * 获取spring中的bean
     * @param clazz
     * @return
     */
    public <T> T getBean(Class<T> clazz) {
        return CONTEXT.getBean(clazz);
    }

}
```

相比内部类的实现代码简洁了不少，但是容器的初始化是放在枚举的构造方法里完成的，这样当该枚类被初始化时，即会加载spring容器，这可能不是我希望的。

并且看看调用方式：

```
//静态内部类
Bootstrap.getApplicationContext();
//枚举形式 这样语意并不是很好
BootstrapEnum.INSTANCE.getApplicationContext();
```

增加一个静态方法：

```
public static BootstrapEnum getInstance(){
        return INSTANCE;
}
```

这样调用方式就变为了：

```
BootstrapEnum.getInstance().getApplicationContext();
```

这样看着语意更明确了，即使这样方法的调用却依然比静态内部类实现的单例麻烦（代码长了），虽单例的实现代码少了，但是每次调用却要多敲代码，不能忍，怎么办？

枚举实现的变种

```
public enum BootstrapEnum {

    INSTANCE;

    private static final Logger logger = LoggerFactory.getLogger(BootstrapEnum.class);

    private static AnnotationConfigApplicationContext CONTEXT;

    static {
        CONTEXT = initContext();
    }

    private static AnnotationConfigApplicationContext initContext() {
        logger.info("开始初始化spring容器。");
        long startTime = Clock.startTime();
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContex
(TookitConfig.class);
        context.registerShutdownHook();
        logger.info("初始化spring容器完成。耗时：{}", Clock.passed(startTime));
        return context;
    }


    /**
     * 获取spring容器
     * @return
     */
    public static AnnotationConfigApplicationContext getApplicationContext(){
        return CONTEXT;
    }

    /**
```

```
     * 获取spring中的bean
     * @param beanName
     * @return
     */
    @SuppressWarnings("unchecked")
    public static <T> T getBean(String beanName) {
        return (T) CONTEXT.getBean(beanName);
    }


    /**
     * 获取spring中的bean
     * @param clazz
     * @return
     */
    public static <T> T getBean(Class<T> clazz) {
        return CONTEXT.getBean(clazz);
    }

}
```

如上，代码的调用变为了：

```
BootstrapEnum.getApplicationContext();
```

看着和静态内部类实现的效果差不多了，但是有一个问题还是没能得到解决，就是延迟加载的问题，该类的静态属性或者方法被访问时，即使不想初始化spring容器，static代码块也会执行。

还有一个严重的问题，此时只需要把该枚举类改成普通类，并删除掉INSTANCE就会惊奇的发现，这恶汉式单例模式无二。代码这样复杂也失去了用枚举类的意义。

结论

当需要延迟加载时选用静态内部类实现，无延迟加载的需求选用枚举的构造方法实现

