单例模式各种实现及优劣对比(基于Java)

核心过程

1. 私有构造函数
2. 提供获取对象实例的方法

实现

饿汉式静态常量

```
public class Singleton1 {
    private Singleton1() { }
    private final static Singleton1 instance = new Singleton1();
    public static Singleton1 getInstance(){
        return instance;
    }
}
```

饿汉式静态代码块

```
private Singleton2(){}
    static {
        instance = new Singleton2();
    }
    private static Singleton2 instance;
    public static Singleton2 getInstance(){
        return instance;
}
```

懒汉式（线程不安全）

```
public class Singleton3 {
    private Singleton3(){}
    private static Singleton3 instance;
    public static Singleton3 getInstance(){
        if(instance != null)
            instance = new Singleton3();
        return instance;
    }
}
```

线程不安全体现在if的判断条件中

懒汉式（双重判断）

```
public class Singleton4 {
    private static volatile Singleton4 singleton4;
    private Singleton4(){}
    public static Singleton4 getInstance(){
        if(singleton4 == null){
            synchronized (Singleton4.class){
                if (singleton4 == null)
                    singleton4 = new Singleton4();
            }
        }
        return singleton4;
    }
}
```

关于volatile的作用解释：

new操作不是一个原子操作，实际上包含三个过程

1. construct empty resource
2. assign
3. call constructor

使用volatile关键字能够禁止指令重排序，确保对数据进行写操作后在后续的读操作中能反映出来。

静态内部类

```
public class Singleton5 {
    private Singleton5(){}
    private static class SingletonInstance{
        private static final Singleton5 INSTANCE = new Singleton5();
    }
    public static Singleton5 getInstance(){
        return SingletonInstance.INSTANCE;
    }
}
```

枚举类

```
public enum Singleton6 {
    INSTANCE;
    public void function(){}
}
```

其他类调用：Singleton6.INSTANCE.function();

对比

实现方式	优点	缺点
2种饿汉式	逻辑简单	缺点
懒汉式	lazy loading	线程不安全
懒汉式双重检查	线程安全	逻辑复杂
静态内部类	线程安全	可能被反射、反序列化破坏
枚举类	避免反序列化破坏	暂无