链滴

学习笔记 | 单例模式

作者：ellenbboe

原文链接：https://ld246.com/article/1565164748802

来源网站：链滴

防止一个类频繁创建和销毁

提供了访问实例的方法

Lazy Loading

类在第一次调用的时候才初始化,节约空间和提高效率

三个基本组成部分

一个私有构造方法

一个静态成员变量

一个获取实例的方法

我瞎总结的

懒汉式

非线程安全

```
package Design;
//懒汉式加载
//非线程安全
public class Singleton_a1 {

    private Singleton_a1(){}//私有构造函数

    private static Singleton_a1 instance;//静态成员变量

    public static Singleton_a1 getInstance()//获取实例方法
    {
        if(instance==null)
        {
            instance=new Singleton_a1();
        }
        return instance;
    }
}
```

线程安全

```
package Design;
//懒汉式加载
//通过给静态方法加sychronized关键字来对类进行加锁确保线程安全
public class Singleton_a2 {
    private Singleton_a2(){};
    private static Singleton_a2 instance;
    public static synchronized Singleton_a2 getInstance()
    {
        if(instance == null)
```

```
        {
            instance = new Singleton_a2();
        }
        return instance;
    }
}
```

饿汉式

```
package Design;
//饿汉式
//由于是依赖classloader的加载机制,所以线程安全
//在类加载的时候就进行初始化,会浪费内存
public class Singleton_b1 {
    private Singleton_b1(){};
    private static Singleton_b1 instance = new Singleton_b1();
    public static Singleton_b1 getInstance()
    {
        return instance;
    }
}
```

变种

```
package Design;
//饿汉式
//类初始化时实例化instance
public class Singleton_b2 {
    private Singleton_b2(){}

    private static Singleton_b2 instance;
    static {
        instance = new Singleton_b2();
    }

    public static Singleton_b2 getInstance()
    {
        return instance;
    }
}
```

双重检验

```
package Design;

//双重检验
//第一层普通检验
//当多个线程都进入第一层instance == null后,synchronized确保只有一个线程进行操作
//并且在第一个线程退出后,后面的线程在进入进行第二层判断instance == null,此时instance不是null
```

```
public class Singleton_c1 {
    private Singleton_c1(){}
    //volatile确保内存可见性
    private volatile static Singleton_c1 instance;

    public static Singleton_c1 getInstance()
    {
        if(instance == null)
        {
            synchronized(Singleton_c1.class)
            {
                if(instance == null)
                {
                    instance = new Singleton_c1();
                }
            }
        }
        return instance;
    }

}
```

静态内部类

```
package Design;
//使用静态内部类
//Jvm只有在使用内部类的时候才会去加载它,是懒加载
//利用classloader加载机制达到线程安全
public class Singleton_d1 {
    private Singleton_d1(){
    }
    //加载内部类的时候只有一个线程,所以线程安全
    private static class innerClass
    {
        private static final Singleton_d1 instance= new Singleton_d1();
    }

    public static Singleton_d1 getInstance()
    {
        return innerClass.instance;
    }
}
```

枚举

```
package Design;
//枚举
enum Singleton_e1 {
    INSTANCE;
    public void anymethod(){}
}
```

```
//这样似乎是单例的,然后在通过实例调用方法就好了
public Singleton_e1 a = Singleton_e1.INSTANCE;
```

网上说建议使用饿汉式(加载的时候就初始化)和静态内部类,考虑到反序列化时使用枚举,其他特殊需求考虑用双重检验.然而我并没有使用单例模式的经验 co
d_sweat