03单例模式Singleton

确保一个类只有一个实例,并提供一个全局访问点。

主要优点:

1、提供了对唯一实例的受控访问。

2、由于在系统内存中只存在一个对象，因此可以节约系统资源，对于一些需要频繁创建和销毁的对象单例模式无疑可以提高系统的性能。

3、允许可变数目的实例。

主要缺点:

1、由于单利模式中没有抽象层，因此单例类的扩展有很大的困难。

2、单例类的职责过重，在一定程度上违背了“单一职责原则”。

3、滥用单例将带来一些负面问题，如为了节省资源将数据库连接池对象设计为的单例类，可能会导致共享连接池对象的程序过多而出现连接池溢出；如果实例化的对象长时间不被利用，系统会认为是垃圾而被回收，这将导致对象状态的丢失。

```C#
/// <summary>
    /// 单例模式的实现
    /// </summary>
    public class Singleton
    {
        // 定义一个静态变量来保存类的实例
        private static Singleton uniqueInstance;

        // 定义一个标识确保线程同步
        private static readonly object locker = new object();

        // 定义私有构造函数，使外界不能创建该类实例
        private Singleton()
        {
        }

        /// <summary>
        /// 定义公有方法提供一个全局访问点,同时你也可以定义公有属性来提供全局访问点
        /// </summary>
        /// <returns></returns>
        public static Singleton GetInstance()
        {
            // 当第一个线程运行到这里时，此时会对locker对象 "加锁",
            // 当第二个线程运行该方法时，首先检测到locker对象为"加锁"状态，该线程就
            // lock语句运行完之后（即线程运行完之后）会对该对象"解锁"
            // 双重锁定只需要一句判断就可以了
            if (uniqueInstance == null)
            {
                lock (locker)
                {
                    // 如果类的实例不存在则创建，否则直接返回
                    if (uniqueInstance == null)
                    {
                        uniqueInstance = new Singleton();
                    }
                }
            }
            return uniqueInstance;
        }
    }
```