单例模式

在一个系统中，一些类要求只能有一个对象，比如：线程池、缓存、对话框、日志对象等，如果创造出多个实例，可能会造成程序行为异常、资源使用过量等一些“不良反应”，此时可以采用**单例模式：确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例**。

一、单例模式的实现方式

1、饿汉式

这种方式比较简单常用，并且是线程安全的，没有使用同步机制，执行效率比较高。

代码实现：

```
public class Singleton
{
    private static final Singleton INSTANCE = new Singleton();
    private Singleton(){} //构造器私有

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

2、懒汉式

在饿汉式中，类加载时就初始化，若对象占用较多资源并且不是需要马上使用的话，比较浪费内存。懒汉式采用懒加载方式，只有在第一次使用对象时才初始化，避免了内存浪费。并且采用同步机制解决了线程安全安全问题。

代码实现：

```
public class Singleton
{
    private static Singleton instance;
    private Singleton(){}

    public static synchronized Singleton getInstance()
    {
        if (instance == null)
```

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

3、双重检查加锁

在懒汉式中，其实只有第一次执行 getInstance 方法时才需要同步，也就是说一旦 instance 变量被初始化成功，就不在需要同步 getInstance 方法了，之后每次调用该方法，同步都是一种累赘，不但会消耗时间还毫无用处。使用双重检查加锁，首先检查是否示例已经创建了，若尚未创建，才进行同步，这样的话就只有第一次才会同步，节省了时间资源。（注：在这种方式中，instance 变量需要 volatile 修饰，确保 instance 被初始化时多个线程能够正确地处理改变量，在 jdk1.5 之前 JVM 对 volatile 关键字的实现会导致双重检查加锁的失效，所以这种方式只适用于 jdk1.5 及更高版本）

代码实现：

```
public class Singleton
{
    private volatile static Singleton singleton;
    private Singleton(){}

    public static Singleton getSingleton()
    {
        if (singleton == null)     //第一次检查
        {
            synchronized (Singleton.class)
            {
                if (singleton == null)    //第二次检查
                {
                    singleton = new Singleton();
                }
            }
        }
        return singleton;
    }
}
```

4、登记式

在双重检查加锁中，由于 Java 平台内存模型中的“无序写”（out-of-order writes）机制，所以需要使用 volatile 修饰实例变量，而 volatile 关键字可能会屏蔽掉虚拟机中的一些必要的代

码优化，所以运行效率并不是很高，并且只有在 jdk1.5 之后才支持这种方式。而使用静态内部类的方式，既能达到懒加载的目的，而且实现更加简单，且不需要同步机制节省资源。

代码实现：

```
public class Singleton
{
    private static class SingletonHolder
    {
        private static final Singleton INSTANCE = new Singleton();
    }

    private Singleton(){}

    public static final Singleton getInstance()
    {
        return SingletonHolder.INSTANCE;
    }
}
```

5、建议

一般情况下直接使用饿汉式即可，实现方便且效率高；当确实需要懒加载时，选用登记式来解决。

二、单例模式的优缺点

1、优点

1）单例模式防止其它对象对自己的实例化，确保所有的对象都访问一个实例。
2）在内存中只有一个实例，减少了内存开支。
3）当一个对象的创建需要较多资源时，如读取配置信息，此时采用单例模式产生一个单例对象，使之常驻内存，这样就减少了系统的性能开销。

2、缺点

单例模式与单一职责原则有冲突，单例模式中，一个类即实现了自己内部的业务逻辑，也负责对自己的实例化，并防止其它对象对自己的实例化，职责过重。

3、常见的应用场景

1）windows 的任务管理器、回收站

2）项目中读取配置文件的类
3）网站的计数器
4）应用程序的日志应用
5）数据库连接池
6）操作系统的文件系统
7）servlet 编程中，每个 Servlet 是单例的